





ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA  
GRADO DE INGENIERIA DE COMPUTADORES

**Carnés UMA ad hoc: Interconexión de  
Sistemas y Servicios Web SOAP**

**UMA's CARD ad hoc: Systems Interconnection  
and Web Services SOAP**

Realizado por  
**D. JOSÉ CARLOS BUSTAMANTE TOLEDO**  
Tutorizado por  
**D. MARINO CASTILLO CABEZAS**  
Departamento  
**LENGUAJE Y CIENCIAS DE LA COMUNICACIÓN**

UNIVERSIDAD DE MÁLAGA  
MÁLAGA, NOVIEMBRE DE 2016

Fecha defensa:  
El Secretario del Tribunal



**Resumen:**

Este trabajo fin de grado tiene como finalidad el análisis y estudio de la interconexión de sistemas informáticos diferentes a través del desarrollo de Servicios Web SOAP, y su aplicación en un proyecto concreto. Una Arquitectura Orientada a Servicios (SOA) es una arquitectura que define cómo interoperan funciones de negocios independientes implementados por sistemas autónomos para ejecutar un proceso de negocios. El diseño Web tiende a ser cada vez más modular y las aplicaciones se componen de una serie de componentes reutilizables, que pueden encontrarse distribuidos a lo largo de una serie de máquinas conectadas en red. Los Servicios Web nos permiten distribuir nuestra aplicación a través de Internet, definiendo como clave la interoperabilidad entre las aplicaciones. Los servicios Web SOAP utilizan mensajes XML para comunicarse que siguen el estándar SOAP, un lenguaje XML que define la arquitectura y formato de los mensajes, que deben contener una descripción de las operaciones ofrecidas y escrita en WSDL (Web Services Description Language). El formato de mensaje SOAP y el WSDL se ha extendido bastante y muchas IDE (Integrated Development Environment), por ejemplo NetBeans, Eclipse o JDeveloper, reducen la complejidad de desarrollar aplicaciones de servicios Web.

**Palabras claves:** Servicios Web, SOAP, WSDL, SQL, NetBeans.

**Abstract:**

This final work aims to analyze and study the interconnection of different computer systems through the development of SOAP Web Services and its application in a specific project. A Service Oriented Architecture (SOA) is an architecture that defines how independent business functions implemented by autonomous systems interoperate to execute a business process. Web design tends to be increasingly modular and applications are made up of a series of reusable components, which can be distributed over a series of networked machines. Web Services allow us to distribute our application over the Internet, defining interoperability between applications as a key. SOAP Web services use XML messages to intercommunicate following the SOAP standard, an XML language that defines the architecture and format of the messages, which must contain a description of the operations offered and written in WSDL (Web Services Description Language). The SOAP message format and the WSDL have been quite widespread, and many IDEs, such as NetBeans, Eclipse, or JDeveloper, reduce the complexity of developing Web services applications.

**Keywords:** Web Services, SOAP, WSDL, SQL, NetBeans



# INDICE DE CONTENIDOS

1. Introducción.....	9
2. Motivación.....	11
3. Fundamentos y estado del arte.....	13
3.1. Introducción.....	13
3.1.1. Proceso de diseño de servicios software.....	14
3.1.2. Aspectos de diseño de servicios software.....	16
3.1.3. Características de los Servicios Web.....	20
3.2. Tipos de Servicios Web.....	20
3.2.1. Servicios Web SOAP.....	20
3.2.2. Servicios Web REST.....	21
3.3. Servicios Web SOAP y Java EE.....	22
3.3.1. Arquitectura de servicios Web SOAP.....	23
3.3.2. SOAP.....	25
3.3.3. WSDL.....	26
3.3.4. UDDI.....	28
3.5. Interoperabilidad de los servicios Web.....	31
3.5.1. Los Servicios desde la vista del Cliente.....	33
3.5.2. Los servicios Web desde la vista del Servidor.....	34
3.5.3. NetBeans y ficheros WSDL.....	37
3.6. Composición de servicios y SOA.....	38
3.6.1. Breve Introducción a la Composición de Servicios.....	40
3.6.2. Orquestación vs. Coreografía.....	41
3.6.3. Workflow vs. Dataflow.....	43
4. Metodología.....	46
4.1. Metodología Scrum.....	49
5. Análisis de la interconexión de sistemas planteados.....	52
5.1 Análisis de servicios Web SOAP del sistema.....	52
5.1.1. Consultas de usuarios - searchUsers.....	53
5.1.2. Selección del usuario a emitir tarjeta - •searchUserById.....	56
5.1.3. Envío de datos sobre tarjeta emitida - sendIssuedCard.....	61
5.1.4. Objetos para el intercambio de datos.....	65
5.1.5. Seguridad y despliegue de los Servicios Web.....	66
5.1.6. Tratamiento de errores y excepciones.....	67
5.2. Análisis de diferentes modelos.....	68

<b>5.2.1.</b>	<b>Casos de Uso.</b>	68
<b>5.2.2.</b>	<b>Diagrama de base de datos.</b>	71
<b>5.2.2.1.</b>	<b>Tabla Alumnos.</b>	72
<b>5.2.2.2.</b>	<b>Tabla Expedientes.</b>	72
<b>5.2.2.3.</b>	<b>Tabla Fotos.</b>	73
<b>5.2.2.4.</b>	<b>Vista BusquedaUser.</b>	73
<b>5.2.2.5.</b>	<b>Vista BusquedaUserId.</b>	73
<b>5.2.2.6.</b>	<b>Tabla Tarjetas.</b>	74
<b>5.2.3.</b>	<b>Diagrama de estados.</b>	75
<b>5.3.</b>	<b>Análisis de requisitos de infraestructuras.</b>	75
<b>5.3.1.</b>	<b>Análisis de requisitos del servidor iiServer.</b>	77
<b>5.3.1.1.</b>	<b>Características iiServer</b>	77
<b>5.3.1.2.</b>	<b>Comunicaciones iiServer</b>	77
<b>5.3.1.3.</b>	<b>Aplicaciones iiServer</b>	78
<b>5.3.1.4.</b>	<b>Mantenimiento de iiServer</b>	78
<b>5.3.2.</b>	<b>Análisis de requisitos de los puntos de emisión</b>	78
<b>5.3.2.1.</b>	<b>Requisitos de red y comunicaciones.</b>	79
<b>5.3.2.2.</b>	<b>Equipamiento del punto de emisión.</b>	79
<b>6</b>	<b>Plataformas tecnológicas utilizadas en el desarrollo.</b>	81
<b>6.1.</b>	<b>Desarrollo servicios Web SOAP con NetBeans.</b>	82
<b>6.2.</b>	<b>Gestión de base de datos con TOAD.</b>	92
<b>6.3.</b>	<b>Montaje del servidor iiServer del proyecto.</b>	97
<b>6.3.1.</b>	<b>Drivers VirtIO.</b>	98
<b>6.4.</b>	<b>Codificación de los servicios Web SOAP con NetBeans.</b>	100
<b>6.4.1.</b>	<b>Codificación searchUsers.</b>	101
<b>6.4.2.</b>	<b>Codificación searchUsersById.</b>	103
<b>6.4.3.</b>	<b>Codificación sendIssuedCard.</b>	106
<b>6.5.</b>	<b>Pruebas.</b>	108
<b>6.6.</b>	<b>Caso de éxito.</b>	108
<b>6.6.1</b>	<b>Integración tarjetas y parking UMA.</b>	109
<b>6.6.2</b>	<b>Identificación iDUMA con carné universitario.</b>	110
<b>6.6.2.1.</b>	<b>Obtención certificado digital de FNMT.</b>	111
<b>7.</b>	<b>Conclusiones.</b>	112
<b>8.</b>	<b>Bibliografía.</b>	113



## 1. Introducción.

La finalidad de este trabajo fin de grado es realizar un estudio y análisis de la interconexión de sistemas informáticos diferentes a través del desarrollo de Servicios Web SOAP, así como su aplicación en un proyecto concreto: la emisión instantánea de carnés UMA. Así como sentar las bases de todos los servicios de valor añadido que irán obteniendo los alumnos de la UMA a través de su carné UMA.

El desarrollo tecnológico actual, ha propiciado que el diseño software tienda a ser cada vez más modular.

Los servicios web son un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones diferentes, facilitando que distintas aplicaciones de sistemas de información distintos, desarrollados en lenguajes y entornos de programación diferente y ejecutada sobre cualquier plataforma, puedan utilizar los servicios web para intercambiar datos.

SOAP es un protocolo para el intercambio de mensajes sobre redes y basado en XML. Los mensajes SOAP, son independientes del sistema operativo, y pueden transportarse en varios protocolos de internet como SMTP, MIME y HTTP.

Tras la integración de la UMA en el programa Santander Universidades durante el curso académico 2008 / 2009, el Banco Santander empezó a fabricar los carnés de estudiantes de la UMA. Desde entonces, se viene realizando dicho proceso con la subida a la plataforma Nexusc, Sistema de gestión de Tarjetas Universitarias Inteligentes (TUI), de los ficheros de datos de los alumnos. Pero durante el curso académico 2015 / 2016, se plantea la necesidad de integrar a las universidades en el nuevo Sistema de Emisión Instantánea de carnés del Banco Santander.

El nuevo Sistema de Emisión Instantánea del Banco Santander ha diseñado la integración entre sistemas informáticos diferentes a través de Servicios Web SOAP, proporcionando a las universidades los interfaces WSDL que deben cumplir dichos servicios web a desarrollar. El cumplimiento de estas interfaces debe ser estricto, para que después el Sistema de Emisión Instantánea pueda integrarse sin errores.

Como caso práctico del estudio de este proyecto, se desarrollarán los servicios web necesarios para interconectar los sistemas de información de la UMA y del Banco Santander. El fin último de dicho proyecto, es la emisión instantánea y entrega a los alumnos del carné de la UMA cuando se matriculan.

El trabajo está estructurado en 8 capítulos incluyendo las conclusiones generales y bibliografía consultada.

En el capítulo 2 se explica la motivación del trabajo fin de grado y se exponen las futuras aplicaciones a las que pone la base este proyecto.

En el capítulo 3 se realiza el estado del arte, permitiendo conocer la base teórica que sustenta el trabajo, a través de un estudio teórico del tema, mostrando la motivación que llevó el desarrollo del trabajo.

En el capítulo 4 se explica la metodología de trabajo utilizada para modelar el sistema, definiendo y explicando los pasos que he seguido para desarrollar el proyecto, de tal forma que todo aquel que pueda estar interesado en la integración de sistemas diferentes a través de servicios web, puede tener la misma visión.

En el capítulo 5 se expone el análisis de los sistemas de información que van a interconectarse a través de los servicios web SOAP que vamos a desarrollar, así como se estudian los requisitos funcionales y no funcionales que debe cumplir el sistema y guían el desarrollo hacia el resultado correcto. Estableciendo los casos de usos y modelado de datos del sistema.

En el capítulo 6 se describe el entorno tecnológico utilizado y la sección que relaciona los aspectos vinculados al desarrollo de la aplicación. Explicando el desarrollo realizado.

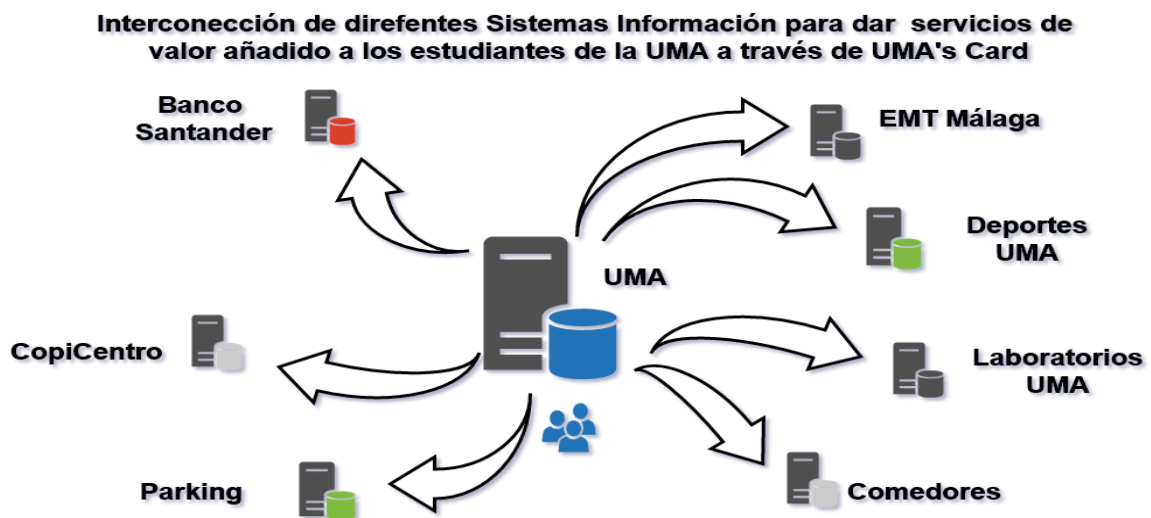
Por último, en el capítulo 7 se redactan las conclusiones del proyecto y en el capítulo 8 se muestra la bibliografía consultada durante todo el proyecto.

## 2. Motivación.

La motivación fundamental para el desarrollo de este trabajo es el estudio de la interconexión de sistemas informáticos diferentes a través de servicios web. Como caso práctico se trabajará en la integración de los sistemas de información del Banco Santander y de la UMA, a través de una aplicación específica que permita la emisión instantánea de los carnés de alumnos de la universidad y la posterior aplicación de dichos carnés en servicios de valor añadido en la misma UMA o fuera de ella.

El Banco Santander en la actualidad emite 7,1 millones de carnés universitarios, desarrollados conjuntamente con más de 340 universidades de 12 países. Las nuevas UMA's CARD son tarjetas inteligentes del Banco Santander (TUI).

Las TUI o UMA's CARD, ofrece entre sus servicios la certificación electrónica para identificación y cifrado en Internet tanto dentro como fuera del campus. Banco Santander emite certificados digitales utilizando su infraestructura de certificación digital llamada Autoridad de Certificación Raíz "WG10 QUALIFIED Identification Root CA". Estos certificados digitales se emiten exclusivamente a personas físicas que mantienen una determinada relación profesional, estudiantil o investigadora con la universidad.



Las UMA's CARD sirven o servirán como:

- Carné universitario. Identifica a su titular y sirve como "clave de acceso" para servicios de valor añadido, incorporándose como medio de autenticación en iDUMA.
- Gestión de préstamos de biblioteca.
- Acceso a la utilización de aulas informáticas.
- Acceso a la utilización de instalaciones deportivas.
- Acceso a la utilización de parkings UMA de control restringido por barrera.

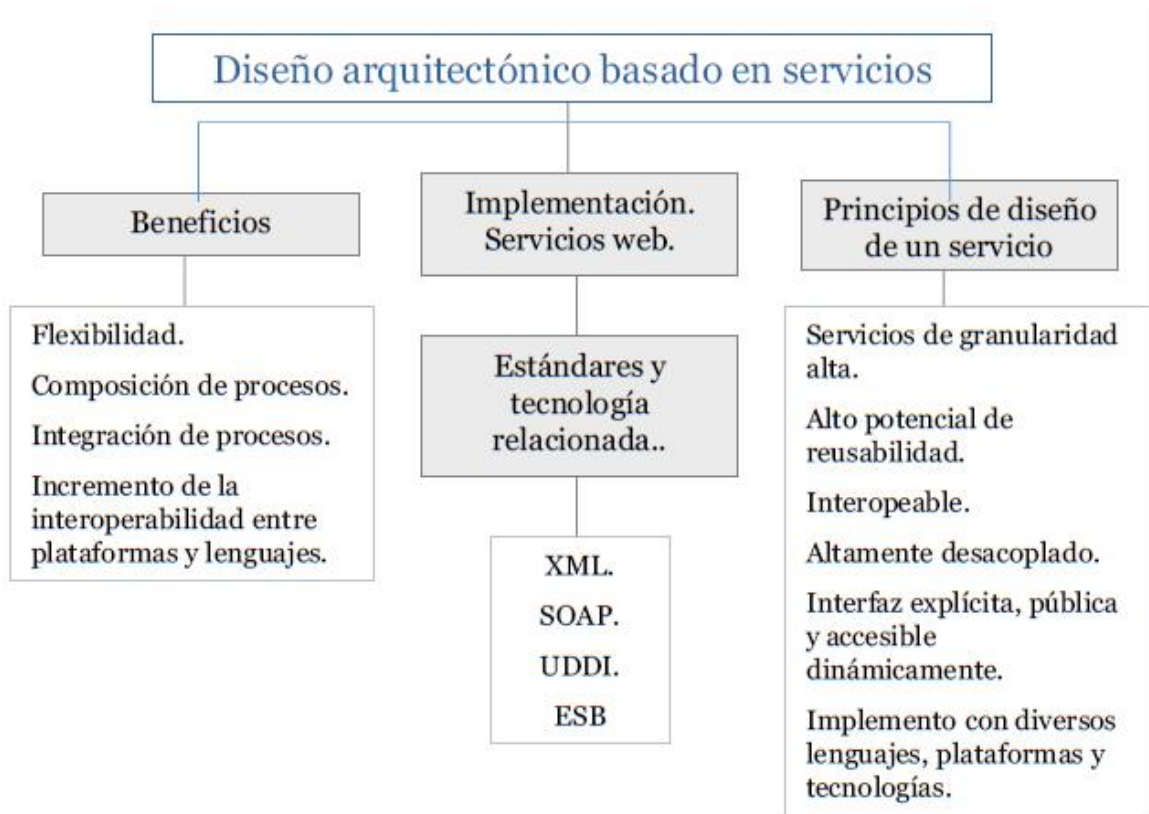
- Identificación como alumno UMA o becario UMA de comedor en los restaurantes de los diferentes centros de la UMA.
- Acceso a los autobuses de la ETM de Málaga.
- Monedero electrónico y servicios de valor añadido en CopiCentro.
- Acceso a la obtención de beneficios derivados de los acuerdos específicos suscritos por la UMA con empresas y entidades.
- Tarjeta débito (opcional) para los titulares de la TUI que así lo deseen, así como monedero electrónico.
- Autoservicio universitario. Funciona a través diferentes canales (cajeros, terminales instalados en el campus, internet, etc.).
- Control de acceso a determinados recintos del campus y ordenadores.
- Obtención de descuentos en comercios, asociados o no con la UMA.
- Aplicación TUI en el móvil.

Y cualquier otra funcionalidad que en un futuro decida la UMA.

### 3. Fundamentos y estado del arte.

El objetivo de este epígrafe es la descripción del estado del arte relacionado con la tecnología utilizada en el desarrollo de los Servicios Web SOAP realizados por parte de la UMA para la integración de sus Sistemas de Información en el Sistema de Emisión Instantánea del Banco Santander.

#### 3.1. Introducción.



En una arquitectura orientada a servicios, SOA o Service Oriented Architecture, el concepto de servicio software se constituye como el bloque básico de construcción. Un servicio desde el punto de vista de este patrón arquitectónico es un componente distribuido, reutilizable y auto contenido con una definición de interfaz explícita, pública y accesible.

Un estilo arquitectónico basado en servicios se enfoca a la integración, reutilización y composición a un nivel funcional de granularidad elevada. Los aspectos comúnmente aceptados sobre la definición del concepto de servicio en SOA son los siguientes:

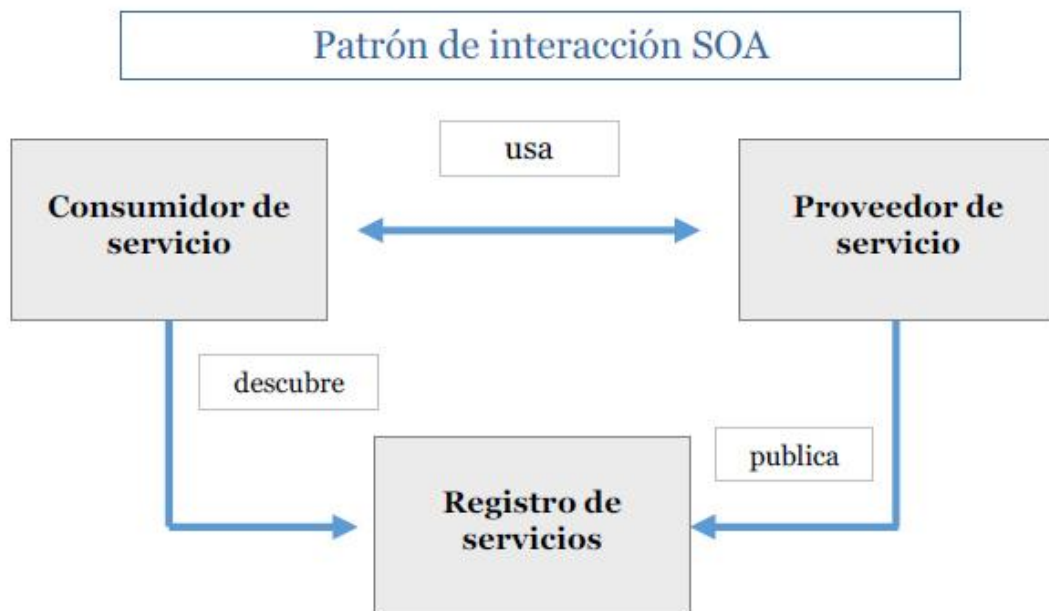
- Los servicios definen interfaces explícitas.
- Los servicios se invocan a través de protocolos de comunicación centrados en proporcionar transparencia respecto a su localización e interoperabilidad.

- Los servicios encapsulan funciones con el objetivo de ser reutilizadas.

El uso de interfaces explícitas para definir servicios encapsulados es de vital importancia. La interfaz encapsula los aspectos de proceso y de comportamiento al tiempo que oculta las particularidades de cada implementación.

El diseño de una arquitectura basada en servicios como estilo arquitectónico está caracterizado por tres componentes arquitectónicos básicos como son:

- **Servicio software**, el cual representa una funcionalidad relevante, reutilizable y expuesta a otros sistemas a través de interfaces basadas en estándares.
- **Consumidor del servicio o cliente del servicio**, el cual utiliza la funcionalidad proporcionada por un servicio o servicios.
- **Infraestructura de comunicaciones** que conecta los consumidores de servicios con los servicios expuestos. Además proporciona un mecanismo de registro y descubrimiento automático.



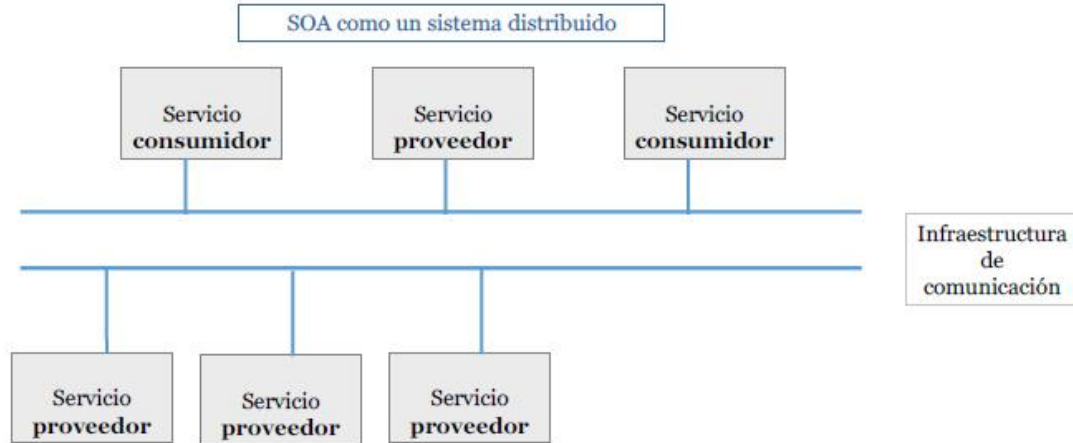
Un componente en el rol de consumidor de servicio descubrirá o buscará un servicio concreto idealmente a través de un registro. El registro proporciona el medio para que el consumidor del servicio pueda interactuar con el servicio.

### 3.1.1. Proceso de diseño de servicios software.

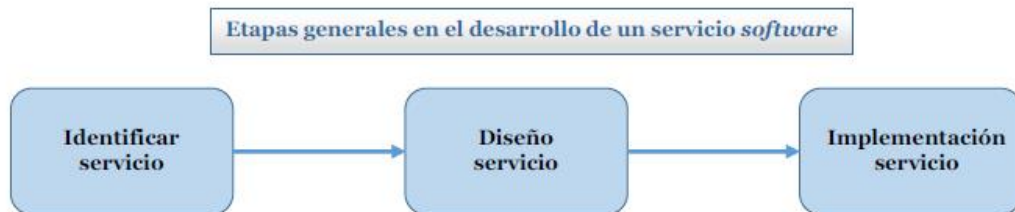
Un diseño arquitectónico basado en servicios constituye en esencia un sistema distribuido, donde los componentes son servicios independientes y físicamente distribuidos. Los servicios desde el punto de vista de un sistema distribuido son abstraídos como unidades autónomas débilmente acopadas dentro de una red formada por proveedores y consumidores siguiendo el patrón de interacción

definido en este paradigma.

En una arquitectura de este tipo existen tres roles fundamentales: consumidor, proveedor y registro de servicios que se pueden combinar arbitrariamente para formar una red de nodos comunicados por una infraestructura general.



El proceso de diseño de un servicio se divide en etapas generales y son:



- **Identificación del servicio.** Se trata de la selección de la funcionalidad que será expuesta como un servicio, siendo una de las claves el identificar la funcionalidad de granularidad alta, que sea fácil de independizar y con potencial de reutilización. La funcionalidad puede ser general o transversal a varios procesos.
- **Diseño de servicio.** Define la interfaz del servicio así como sus parámetros. Lo esencial es minimizar el número de mensajes que deben ser compartidos, intercambiando quizás mayor cantidad de información.
- **Implementación.** Comprende la selección de una tecnología capaz de desplegar y ejecutar la funcionalidad seleccionada como un servicio software.

Existe otra alternativa en el diseño de servicios software que es exponer una funcionalidad existente como un servicio software.

---

**Etapas generales en el desarrollo de un servicio software**

Este tipo de situación de funcionalidad que debe ser expuesta como un servicio puede darse en sistemas legados o *legacy systems*, que por su importancia funcional no pueden ser reconstruidos. Sin embargo, deben ser usados en un escenario de interoperabilidad e integración con otros sistemas.

### 3.1.2. Aspectos de diseño de servicios software.

La encapsulación de servicios a través de interfaces y su invocación transparente, usando protocolos interoperables, son los medios fundamentales para alcanzar una mayor flexibilidad y reutilización en el uso de una arquitectura de servicios.

#### Servicios débilmente acoplados

Una de las claves para entender el diseño de una arquitectura orientada a servicios es el concepto de acoplamiento, que funciona como una métrica de calidad de diseño y determina muchas de las propiedades de un buen diseño arquitectónico.

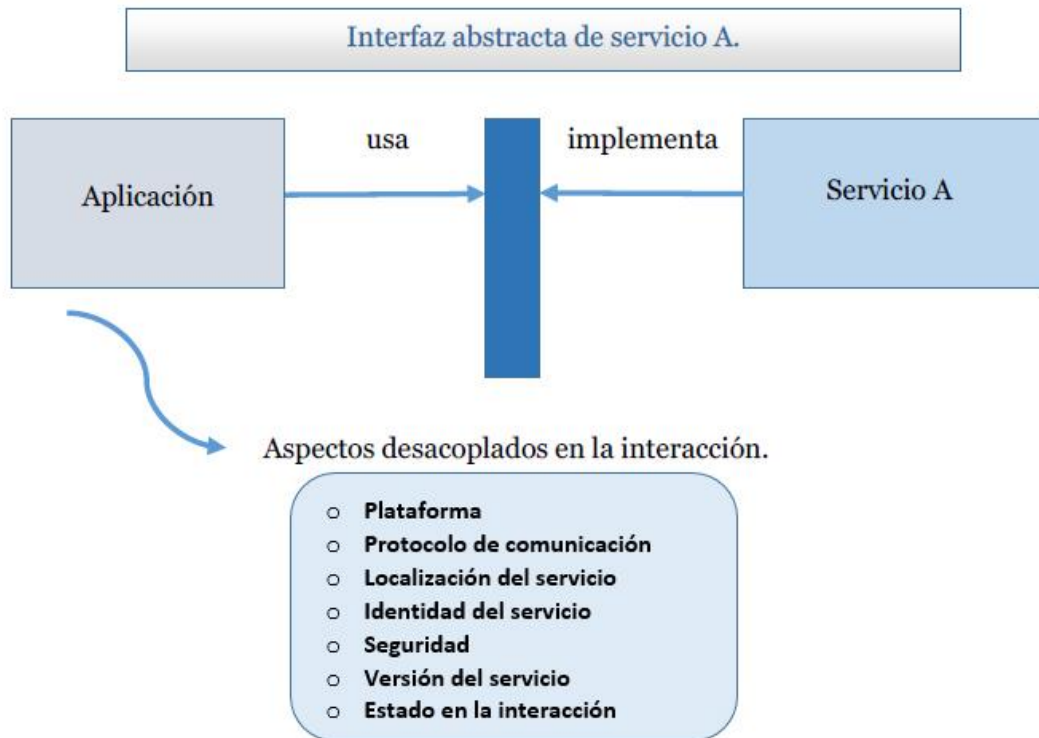
Uno de los objetivos de todo diseño estructural de elementos interrelacionados es alcanzar el mínimo acoplamiento posible entre sus componentes. Siempre existirá un grado de acoplamiento entre módulos, aunque se persigue que este sea mínimo, para tratar de lograr el menor impacto directo en la mantenibilidad del futuro sistema.

En general un módulo funcional está acoplado si los cambios internos en su implementación se propagan a otros módulos que interactúan con él.

Cuando hablamos de cambios y acoplamientos nos referimos a aspectos de implementación interna. Tales como, un cambio de estructura de datos, de variables o de programación en el contexto de servicios de plataforma lenguaje de programación. Idealmente, un servicio se construye para que no exista acoplamiento de este tipo.

El acoplamiento puede tener su origen en diversos aspectos. El objetivo de SOA es aislar a un servicio de siguientes aspectos: la plataforma (sistema operativo, hardware o lenguaje), el protocolo de comunicación, la localización del servicio, la identidad del servicio, la seguridad, la versión y el estado en la interacción. Esto permite diseñar unidades funcionales con un acoplamiento mínimo.

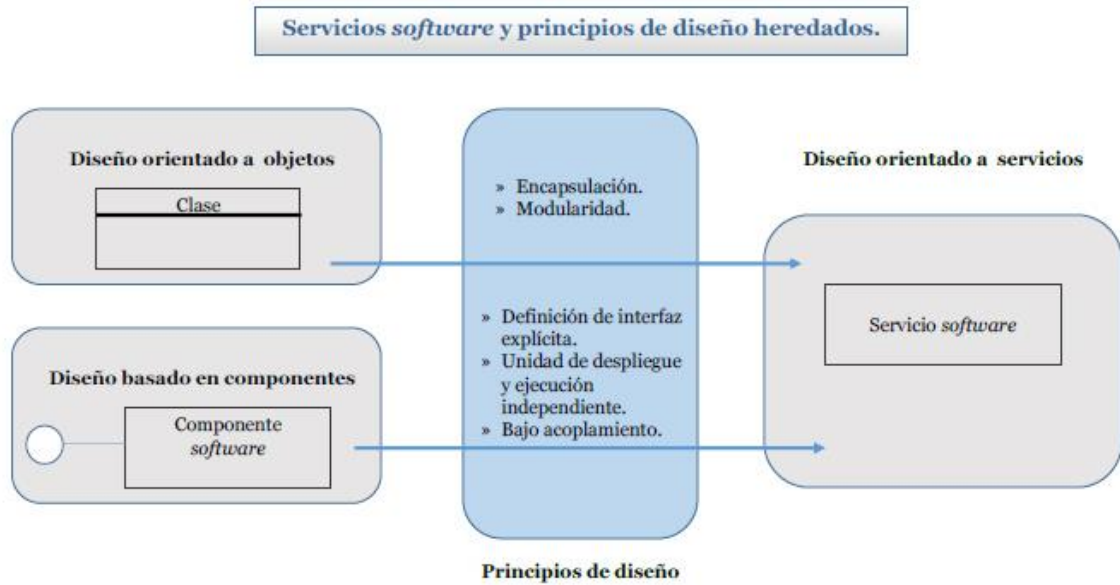




La orientación a servicios persigue disminuir el acoplamiento entre las partes a partir de las definiciones de servicios, mediante una interfaz explícita y abstracta. Las interfaces abstractas además simplifican la interacción.

### **Servicios software, clases y componentes software**

Las arquitecturas de servicios se basan en el concepto de unidades funcionales independientes, desacopladas y pensadas para ser reutilizables. Estas características también se encuentran en los componentes software. Según esto, es posible pensar que no existen diferencias significativas entre los componentes software y servicios software. Con todo esto, llegamos a la conclusión de que la realidad es que tanto los principios de diseño del paradigma de orientación a objetos como los del diseño basado en componentes son aplicables a un servicio.



Sin embargo, un servicio software es distinto a un componente software. Las diferencias entre un enfoque orientado al servicio y un enfoque orientado a componentes son los siguientes:

- Los servicios pueden ser ofrecidos por varios proveedores y un componente suele ser exclusivo de uno.
- Los servicios están pensados para ser independientes respecto del contexto de ejecución. Esto implica que no deben definir interfaces requeridas por el entorno.
- Los servicios software son independientes de la plataforma donde se ejecutan: sistema operativo, lenguaje de programación, etc.
- Un servicio está diseñado para ser publicado, buscado y usado en tiempo de ejecución.
- Es posible el reconocimiento dinámico de nuevos servicios.
- El grado de desacoplamiento es mayor que el de los componentes.

Aunque, como hemos visto, existen diferencias entre un servicio software y un componente, los servicios software comparten principios de diseño propios de los basados en componentes y orientados a objetos como el desacoplamiento, la definición de interfaces y el encapsulamiento de una funcionalidad y sus detalles internos.

Los componentes por sus características de diseño (desacoplamiento, encapsulación, interfaces explícitas, unidades independiente) se pueden transformar en servicios.

### **Servicios independientes**

Existe una cuestión fundamental a la hora de diseñar servicios. Se trata de determinar si mantendrán algún tipo de información de estado en la interacción o no. Esta cuestión determinará gran parte del grado de acoplamiento que tendrá el servicio respecto a su interacción con otros componentes.

En general, los servicios software se diseñan para ser sin estado. Es decir, los servicios a partir de los parámetros de entrada de su interfaz serán capaces de desarrollar el aspecto funcional para el que está diseñado, sin guardar internamente ninguna información de la interacción. Esto tiene la consecuencia de que el servicio ofrecerá un grado de acoplamiento menor, aumentando su potencial de reutilización.

La clave para implementar con éxito la interacción sin estado se basa en:

- El uso de tecnología que impida que una interacción sea retenida por una instancia particular.
- El diseño de interfaces de servicio que no dependan de ningún tipo de conocimiento implícito, compartido y creado a través de una secuencia de interacciones entre un solicitante y un proveedor del servicio.

La primera cuestión puede ser abordada fácilmente por protocolos que permiten comunicaciones asíncronas. Mecanismos para guardar información de sesión tales como cookies o tokens de sesión no deberían usarse.

La segunda consideración solo puede ser alcanzada tomando estas restricciones como un principio de diseño.

### **Tamaño del servicio**

Para referirse al término servicio se usa la palabra large granularity. Es un servicio que se concibe como un aspecto funcional grande, completo e independiente.

Aunque normalmente los servicios se conciben como aspectos funcionales de alto nivel y de un tamaño funcional grande, nada impide que un servicio esté descrito por una única función de granularidad considerada pequeña.

Existen diversos niveles de funcionalidad que podrían ser desarrollados por un servicio, siendo además un hecho lo difícil que es de precisar un nivel de funcionalidad o granularidad de diseño común.

No se debe confundir un servicio software en SOA con SAAS (Software as a Service). SAAS es una forma que proporciona funcionalidad sobre un servidor remoto donde acceden múltiples clientes y donde el servicio ofrecido mantendrá largas transacciones y el estado de la interacción. Un servicio software en SOA se diseña de manera para soportar una interacción corta y sencilla, y sin guardar estado en la interacción

---

### 3.1.3. Características de los Servicios Web.

Las características deseables de un Servicio Web son:

- Un servicio debe poder ser accesible a través de la Web. Para ello debe utilizar protocolos de transporte estándares como HTTP, y codificar los mensajes en un lenguaje estándar que pueda conocer cualquier cliente que quiera utilizar el servicio.
- Un servicio debe contener una descripción de sí mismo. De esta forma, una aplicación podrá saber cuál es la función de un determinado Servicio Web, y cuál es su interfaz, de manera que pueda ser utilizado de forma automática por cualquier aplicación, sin la intervención del usuario.
- Debe poder ser localizado. Deberemos tener algún mecanismo que nos permita encontrar un Servicio Web que realice una determinada función. De esta forma tendremos la posibilidad de que una aplicación localice el servicio que necesite de forma automática, sin tener que conocerlo previamente el usuario.

### 3.2. Tipos de Servicios Web.

A nivel técnico, los servicios pueden implementarse de varias formas, podemos distinguir dos tipos de servicios Web: SOAP y REST.

#### 3.2.1. Servicios Web SOAP.

Los servicios Web SOAP, Simple Object Access Protocol, utilizan mensajes XML para comunicarse que siguen el estándar SOAP, un lenguaje XML que define la arquitectura y formato de los mensajes. Dichos sistemas normalmente contienen una descripción legible por la máquina de la descripción de las operaciones ofrecidas por el servicio, escrita en WSDL (Web Services Description Language), que es un lenguaje basado en XML para definir las interfaces sintácticamente.

El formato de mensaje SOAP y el lenguaje de definición de interfaces WSDL se ha extendido bastante, y muchas herramientas de desarrollo, por ejemplo NetBeans, JDeveloper o Eclipse, pueden reducir la complejidad de desarrollar aplicaciones de servicios Web.

El diseño de un servicio basado en SOAP debe establecer un contrato formal para describir la interfaz que ofrece el servicio Web. WSDL puede utilizarse para describir los detalles del contrato, que pueden incluir mensajes, operaciones, bindings, y la localización del servicio Web. También deben tenerse en cuenta los requerimientos no funcionales, como por ejemplo las transacciones, necesidad de mantener el estado, seguridad y coordinación

---

### 3.2.2. Servicios Web REST.

Los servicios Web REST (Representational State Transfer Web Services) son adecuados para escenarios de integración básicos ad-hoc. Dichos servicios Web se suelen integrar mejor con HTTP que los servicios basados en SOAP, ya que no requieren mensajes XML o definiciones del servicio en forma de fichero WSDL.

Los servicios Web REST utilizan estándares muy conocidos como HTTP, SML, URI, MIME, y tienen una infraestructura "ligera" que permite que los servicios se construyan utilizando herramientas de forma mínima. Gracias a ello, el desarrollo de servicios REST es barato y tiene muy pocas "barreras" para su adopción.

REST, es un estilo de arquitectura de software dirigidos a sistemas hipermedias distribuidos como lo es la Web y se refiere específicamente a una colección de principios (los cuales resumen la forma en que los recursos son definidos y diseccionados) para el diseño de arquitecturas en red.

Este término es utilizado en su mayoría para describir a cualquier interfaz que transmite datos específicos de un dominio sobre HTTP sin una capa adicional, como lo hace SOAP, en tal sentido éstos dos significados pueden chocar o incluso solaparse.

La arquitectura de REST tiene que cumplir con estos 6 principios.

- Cliente-servidor Esta define que deben de estar separados el cliente del servidor a través de interfaces uniformes, es decir el cliente no sabe nada de cómo se almacena la información, ni como se está obteniendo. Por otro lado los servidores no saben la manera en la que se está presentado la información.
- No manejan estado El servidor no debe de contener ningún contexto sobre el cliente que está haciendo la solicitud. La solicitud del cliente debe tener toda la información necesaria para poder procesar la solicitud en el servidor, esto permite crear aplicaciones más escalables sin que tener preocupación sobre cómo debe de responder el servidor a la pérdida de la sesión del cliente por pérdida de conectividad.
- Capaces de almacenarse en caché En el WWW los clientes no tienen un mecanismo de almacenar las respuestas en caché. Las respuestas deben de estar implícitas o en su defecto explícitamente deben definirse a sí mismas como almacenables en caché o no, para evitar que los clientes hagan uso inapropiado de información regresada por una solicitud.
- Sistemas en capas El cliente no debe de saber si está conectado directamente a un servidor final o a un intermediario. un servidor intermediario te puede ayudar a balancear las cargas y la escalabilidad de la aplicación.
- Código bajo demanda Los servidores pueden ser capaces de extender la funcionalidad de un cliente transfiriéndole lógica que puedan ejecutar, por ejemplo Java Applets o JavaScript.

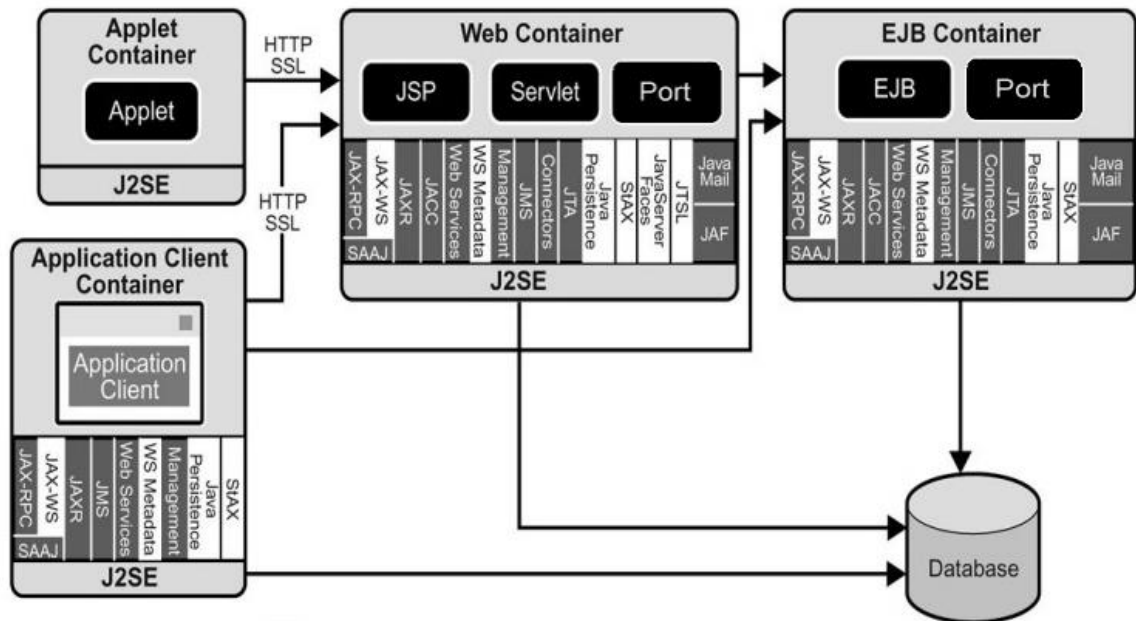
- Interface Uniforme Son recursos individuales que deben de estar incluidos dentro de la solicitud.

### **3.3. Servicios Web SOAP y Java EE.**

Nos centraremos en los servicios Web SOAP desarrollados en Java, siendo componentes con las siguientes características:

- Implementa los métodos de una interfaz descrita mediante un WSDL. Dichos métodos se implementan utilizando un EJB de sesión de tipo Stateless/Singleton o bien un componente web JAX-WS.
- Un servicio Web puede tener publicada su interfaz en uno o más "registros" durante su despliegue.
- La implementación de un Servicio Web, la cual utiliza solamente la funcionalidad descrita por su especificación, puede desplegarse en cualquier servidor de aplicaciones que cumple con las especificaciones Java EE.
- Los servicios requeridos en tiempo de ejecución (run-time), tales como atributos de seguridad, se separan de la implementación del servicio. Se utilizarán herramientas adicionales que pueden definir dichos requerimientos durante el ensamblado o despliegue.
- Un contenedor actúa como mediador para acceder al servicio

La especificación de Java EE para servicios Web define una serie de relaciones arquitectónicas requeridas para dichos servicios, que se pueden observar en el siguiente esquema:



Arquitectura Java EE

Se trata de relaciones lógicas que no imponen requerimiento alguno para el proveedor del contenedor sobre cómo estructurar los contenedores y los procesos. Como añadido para la plataforma Java EE se incluye un componente port que depende de la funcionalidad de contenedor proporcionada por los contenedores web y EJB, y del transporte SOAP/HTTP.

Los servicios Web para Java EE requieren que un componente Port pueda ser referenciado desde un cliente, así como desde los contenedores web y EJB. No se requiere que haya un Port accesible desde un contenedor de applets.

Los servicios Web para Java EE pueden implementarse de dos formas: como una clase Java que se ejecuta en un contenedor Web o como un EJB de sesión stateless o singleton en un contenedor EJB.

El contenedor del servicio Web debe proporcionar la gestión del ciclo de vida de la implementación del servicio, además de proporcionar soporte adicional para la gestión de concurrencia de la invocación de los métodos del servicio, y soporte para la seguridad.

### 3.3.1. Arquitectura de servicios Web SOAP.

Aunque un servicio software puede ser implementado de muchas formas y empleando diferentes tecnologías, son los denominados servicios web los que han conseguido hacer efectiva la arquitectura orientada a servicios.

Los servicios web son un conjunto de especificaciones tecnológicas basadas en estándares abiertos como XML, URL y HTTP, y que proporcionan un modelo de interacción sistema a sistema.

Sobre la base de este modelo estándar de interacción existen otras normas relacionadas dirigidas, por ejemplo, al control de transacciones, a la seguridad en las comunicaciones extremo a extremo y relacionadas con la definición de procesos empresariales.



Estas funciones de alto nivel, transversales a muchos sistemas, son necesarias para hacer que procesos empresariales puedan interactuar entre sí. El modelo de interacción de servicios web básico define interacciones entre los solicitantes de servicios, proveedores de servicios y directorios de servicios.

Los solicitantes de un Servicio podrían encontrar los servicios web disponibles en un servicio de directorio UDDI, que han sido publicados previamente por los proveedores de servicio. Desde este servicio recuperan las descripciones de la interfaz del servicio en formato WSDL.

Después de que la descripción de la interfaz WSDL haya sido recuperada, el servicio solicitante se une al proveedor de servicios invocando los servicios a través del protocolo SOAP. La descripción general de los estándares que forman la base tecnológica de servicios web es la siguiente:



---

### 3.3.2. SOAP.

Es un protocolo de intercambio de mensajes basado en XML, que permite la comunicación entre servicios web. Este protocolo es independiente de cualquier protocolo de transporte específico, pudiendo usar varios de forma transparente.

Normalmente usaremos SOAP para conectarnos a un servicio e invocar métodos remotos, aunque puede ser utilizado de forma más genérica para enviar cualquier tipo de contenido. Podemos distinguir dos tipos de mensajes según su contenido:

- **Orientados al documento:** Contienen cualquier tipo de contenido que queramos enviar entre aplicaciones.
- **Orientados a RPC:** servirá para invocar procedimientos de forma remota (Remote Procedure Calls). Podemos verlo como un tipo más concreto dentro del tipo anterior, ya que en este caso como contenido del mensaje especificaremos el método que queremos invocar junto a los parámetros que le pasamos, y el servidor nos deberá devolver como respuesta un mensaje SOAP con el resultado de invocar el método.

Puede ser utilizado sobre varios protocolos de transporte, aunque está especialmente diseñado para trabajar sobre HTTP.

Dentro del mensaje SOAP podemos distinguir los siguientes elementos:

- **Un sobre (Envelope),** que describe el mensaje, a quien va dirigido, y cómo debe ser procesado. El sobre incluye las definiciones de tipos que se usarán en el documento. Contiene una cabecera de forma opcional, y el cuerpo del mensaje.
- **Una cabecera (Header)** opcional, donde podemos incluir información sobre el mensaje. Por ejemplo, podemos especificar si el mensaje es obligatorio (debe ser entendido de forma obligatoria por el destinatario), e indicar los actores (lugares por donde ha pasado el mensaje).
- **El cuerpo del mensaje (Body),** que contiene el mensaje en sí. En el caso de los mensajes RPC se define una convención sobre cómo debe ser este contenido, en el que se especificará el método al que se invoca y los valores que se pasan como parámetros. Puede contener un error de forma opcional.
- **Un error (Fault)** en el cuerpo del mensaje de forma opcional. Nos servirá para indicar en una respuesta SOAP que ha habido un error en el procesamiento del mensaje de petición que mandamos.
- **El anexo (Attachment)** opcional, puede contener cualquier tipo de contenido (incluido el XML). De esta forma podremos enviar cualquier tipo de contenido junto a un mensaje SOAP, sea o no XML. Además, si es necesario, nuestro mensaje podrá tener tantos anexos como se

necesiten.

Un ejemplo de mensaje SOAP es:

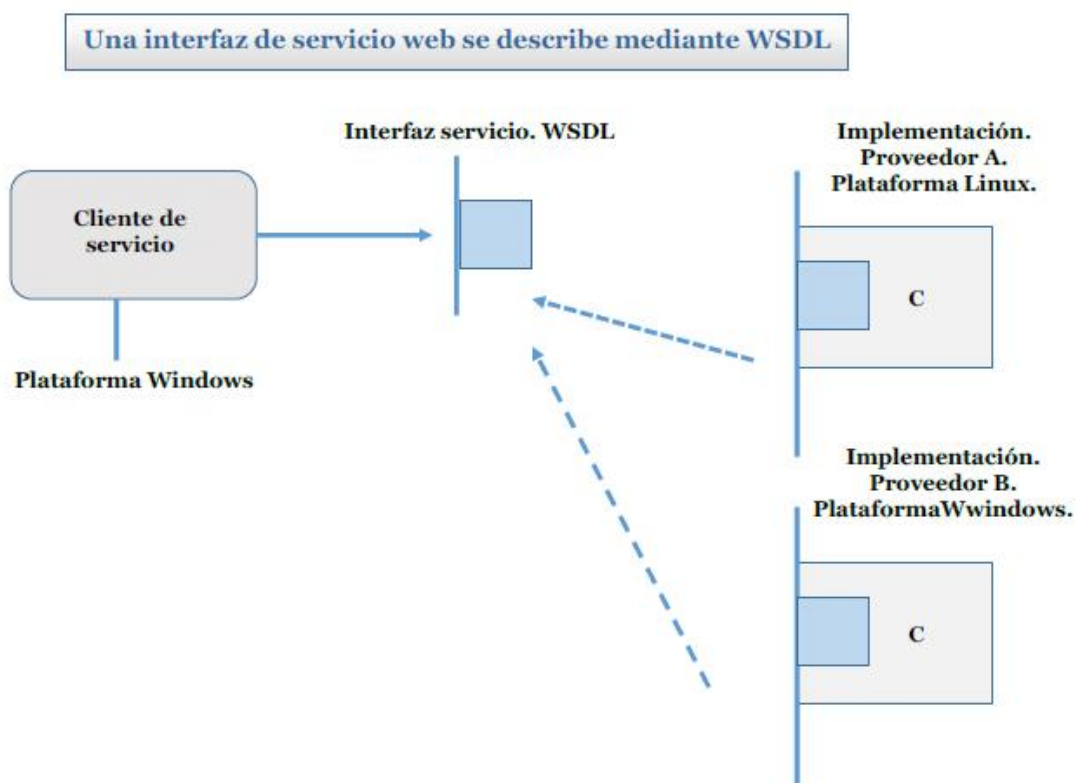
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchUsers xmlns="http://external.issue.datio.es/">
      <data>12345678Z</data>
    </searchUsers>
  </soap:Body>
</soap:Envelope>
```

En dicho mensaje SOAP estamos llamando a nuestro método searchUsers para buscar un alumno cuyo DNI coincida sea 12345678Z.

### 3.3.3. WSDL.

Lenguaje estándar de descripción de servicios web WSDL o web services description language. Es un lenguaje específico XML que permite definir la interfaz de un servicio web. Una descripción WSDL (fichero WSDL) de un servicio web proporciona una descripción entendible por la máquina de la interfaz del servicio Web, indicando cómo se debe llamar al servicio, qué parámetros espera, y qué estructuras de datos devuelve.

La definición WSDL de la interfaz de un servicio es independiente de la implementación específica que se haga de ella. Así, una misma interfaz podría ser implementada por varios equipos de desarrollo de empresas diferentes, usando entornos de programación distintos.



Los elementos de los que consta una definición de interfaz son los siguientes:

- Definición de espacios usados en la descripción del servicio.
- Descripción de la interfaz del servicio web mediante el conjunto de operaciones que definen el servicio y tipos usados.
- Enlace lógico con la implementación concreta de las operaciones de la interfaz.

WSDL describe un servicio utilizando varios elementos (etiquetas XML). Dichos elementos podemos clasificarlos como abstractos o concretos. La parte WSDL abstracta describe las operaciones y mensajes con detalle. En otras palabras, la parte abstracta de un WSDL especifica qué hace el servicio:

- Qué operaciones están disponibles.
- Qué entradas, salidas y mensajes de error tienen las operaciones.
- Cuáles son las definiciones de los tipos para los mensajes de entrada, salida y error.

En el mundo Java, podemos pensar en la parte abstracta de un WSDL como en la definición de una interfaz o una clase abstracta, con la definición de sus métodos, pero no sus implementaciones. La parte abstracta de un WSDL contiene dos componentes principales:

- Las operaciones que forman la definición de la interfaz.
- Los tipos de datos para los parámetros de entrada, salida y error, de las operaciones.

La parte WSDL concreta describe el cómo y dónde del servicio:

- Cómo tiene que llamar un cliente al servicio.
- Qué protocolo debería usar.
- Dónde está disponible el servicio.

En el mundo Java podemos pensar en la parte concreta de un WSDL como en la implementación de la parte abstracta, aunque en términos de Servicios Web, solamente describe dónde se encuentra dicha implementación para utilizarse. La parte concreta de un WSDL contiene dos componentes principales:

- Información de enlazado (binding) sobre el protocolo a utilizar.
- La dirección en donde localizar el servicio.

Los elementos WSDL, tanto la parte abstracta como la concreta, son:

- **definitions:** Es el elemento raíz y permite especificar el espacio de nombres del documento target namespace, el nombre, y otros prefijos utilizados en el documento WSDL. Un ejemplo de definición de prefijo es:

---

xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/".

- **types:** Se utiliza para definir los tipos de datos que se intercambiarán en el mensaje. Podemos definir dichos tipos directamente dentro de este elemento, o importar la definición de un fichero de esquema (fichero xsd). La definición de tipos puede verse como las definiciones Java de clase, con variables que pueden ser tipos primitivos o referencias a otras clases u objetos. Los tipos primitivos se definen en los espacios de nombres del Schema y normalmente nos referimos a ellos como built-in types. Éstos incluyen tipos simples tales como string, int, double,...
- **message:** Define los distintos mensajes que se intercambiarán durante el proceso de invocación del servicio. Se deberán definir los mensajes de entrada y salida para cada operación que ofrezca el servicio. Los mensajes muestran descripciones abstractas de los datos que se van a intercambiar.
- **portType:** Contiene una colección de una o más operaciones. Para cada operación indica cuáles son los mensajes de entrada y salida, utilizando para ello los mensajes definidos en el apartado anterior. Los portTypes son, por lo tanto, colecciones abstractas de operaciones soportadas por un servicio.
- **binding:** Indica el protocolo de red y el formato de los datos para las operaciones de un portType. Los bindings son definiciones concretas de los portTypes. Un portType puede tener múltiples bindings asociados. El formato de datos utilizado para los mensajes de las operaciones del portType puede ser orientado al documento u orientado a RPC. Si es orientado al documento tanto el mensaje de entrada como el de salida contendrán un documento XML. Si es orientado a RPC el mensaje de entrada contendrá el método invocado y sus parámetros, y el de salida el resultado de invocar dicho método, siguiendo una estructura más restrictiva.
- **service:** Define el servicio como una colección de elementos port a los que se puede acceder. Un port se define asociando una dirección de red con un binding, de los definidos en el documento. Dicha dirección de red es la dirección (URL) donde el servicio actúa, y por lo tanto, será la dirección a la que las aplicaciones deberán conectarse para acceder al servicio.

### 3.3.4. UDDI.

El estándar UDDI, Universal Description Discovery Integration, es una especificación de directorio de registro de servicios disponible para la consulta dinámica de los servicios que pueden ser accedidos.

UDDI nos permite localizar Servicios Web. Para ello define la especificación para construir un directorio distribuido de Servicios Web, donde los datos se almacenan en XML. En este registro no sólo se almacena información sobre servicios, sino también sobre las organizaciones que los proporcionan, la categoría en la que se encuentran, y sus instrucciones de uso (normalmente WSDL). Tenemos por lo tanto 3 tipos de información relacionados entre sí:

- Páginas blancas: Datos de las organizaciones (dirección, información de contacto, etc).
- Páginas amarillas: Clasificación de las organizaciones (según tipo de industria, zona geográfica, etc).
- Páginas verdes: Información técnica sobre los servicios que se ofrecen. Aquí se dan las instrucciones para utilizar los servicios. Es recomendable que estas instrucciones se especifiquen de forma estándar mediante un documento WSDL.

Además, UDDI define una API para trabajar con dicho registro, que nos permitirá buscar datos almacenados en él, y publicar datos nuevos.

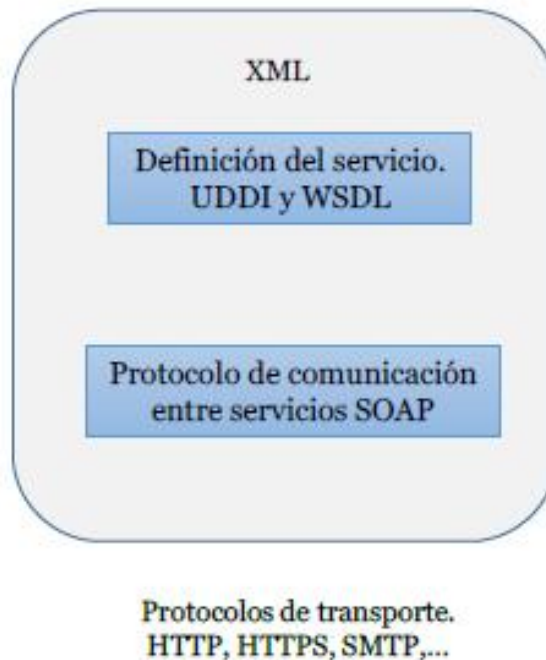
De esta forma, una aplicación podrá anunciar sus servicios en un registro UDDI, o bien localizar servicios que necesitemos mediante este registro.

Esta capacidad de localizar servicios en tiempo de ejecución, y de que una aplicación pueda saber cómo utilizarlo inmediatamente gracias a la descripción del servicio, nos permitirá realizar una integración débilmente acoplada de nuestra aplicación.

La interfaz de UDDI está basada en SOAP. Para acceder al registro se utilizarán mensajes SOAP, que son transportados mediante protocolo HTTP.

Estos registros se utilizan normalmente de forma interna en organizaciones para tener un directorio organizado de servicios.

**Marco de capas funcionales proporcionadas por estándares en la tecnología de servicios web.**



Los estándares SOA se apoyan en una serie de normas adicionales que permiten ampliar la funcionalidad de una implementación SOA, algunas de estas normas son las siguientes:

- **WS-Reliable**: estándar para el intercambio fiable de mensajes.
- **WS-Security**: conjunto de normas para dar soporte a la seguridad en la interacción con servicios web. Incluye normas que especifican la definición de las políticas y normas de seguridad que cubre el uso de firmas digitales.
- **WS-Addressing**: norma que define cómo se debe representar la información de direcciones en un mensaje del protocolo SOAP.
- **WS-Transactions**: define el servicio de transacciones a través de servicios web distribuidos.

Con el fin de alcanzar la estandarización de la interacción se publicó el estándar **WS\_I**, que define una serie de perfiles. El perfil básico WS\_I de escenarios de uso consiste en tres escenarios, explicados a continuación.

- **Comunicación unidireccional**. Es el escenario más simple. En él, un emisor envía una solicitud a un proveedor de servicio. Sólo debería ser usado cuando se puede tolerar la pérdida de información.
- **Solicitud o respuesta síncrona**. Es el escenario más utilizado. Un emisor envía un mensaje de solicitud de servicio a un proveedor que lo procesa y envía la respuesta.

- **Retro llamada básica.** Este escenario se usa para simular una operación asíncrona sobre operaciones síncronas. El escenario se compone de dos interacciones solitud o respuesta síncrona: una iniciada por el emisor y la otra por el receptor.

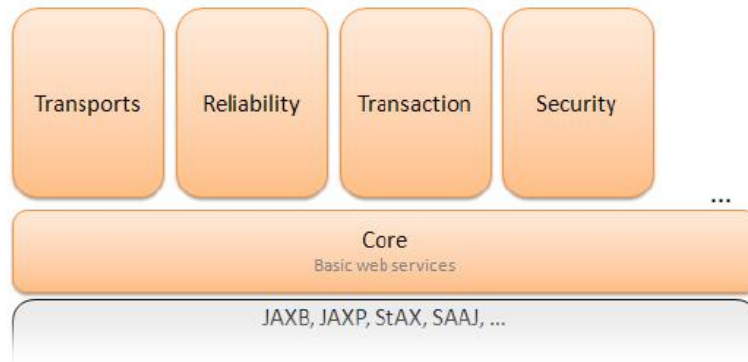
### 3.5. Interoperabilidad de los servicios Web.

La interoperabilidad de los servicios Web es una iniciativa de Sun y Microsoft. El principal objetivo es proporcionar productos que sean capaces de interoperar a través de diferentes plataformas.

Metro (<http://metro.java.net>) es el producto resultante de la iniciativa de Sun para la interoperabilidad de los servicios Web utilizando la plataforma Java. De igual forma WCF (Windows Communication Foundation) es la aportación de Microsoft para la plataforma .NET.

Metro, por lo tanto, constituye la implementación por parte de Sun, de la pila (colección de tecnologías) de servicios Web (web service stack). La versión actual es la 2.1.1 y está formada por tres componentes principales, que podemos ver en la figura que se adjunta:

- Metro/WSIT 2.1.1: WSIT es un conjunto de tecnologías (Web Services Interoperable Technologies) que permiten la interoperabilidad con .NET. A nivel de transporte, Metro permite utilizar HTTP, MTOM, SOAP/TCP. También proporciona fiabilidad en el envío de mensajes (Reliability), implementando las especificaciones WS-ReliableMessaging. Asimismo Metro permite habilitar el uso de transacciones atómicas, para lo cual proporciona una implementación de las especificaciones WS-Coordination y WS-Atomic Transaction. La seguridad también está contemplada en Metro mediante la implementación de las especificaciones WS-Security y WS-Trust. Bootstrapping: WSDL; WS-Policy; WS-MetadataExchange
- JAX-WS RI 2.2.5: Es la implementación de referencia del estándar JAX-WS (especificación JSR 224: Java API for XML-Based Web Services). Proporciona las características básicas para la interoperabilidad de los servicios Web (WS-I Basic Profile: mensajería SOAP, WSDL, publicación de servicios en UDDI; WS-I Attachment Profile: utilización de SOAP con anexos; WS-I Addressing: utilización de espacios de nombres y ficheros de esquema)
- JAXB RI 2.2.4-1: Implementación de referencia del API para la capa de enlazado de datos (JAXB: Java Architecture for XML binding)



### Componentes de Metro

Con JAX-WS, una invocación de una operación a un servicio web se representa con un protocolo basado en XML, como por ejemplo SOAP. La especificación SOAP define la estructura del "envoltorio" (envelope) del mensaje, reglas de codificación, y convenciones para representar invocaciones y respuestas del servicio web. Estas llamadas y respuestas son transmitidas como mensajes SOAP (ficheros XML) a través de HTTP.

Si bien los mensajes SOAP son complejos, la API JAX-WS oculta esta complejidad al desarrollador de la aplicación. En la parte del servidor, el desarrollador especifica las operaciones del servicio web definiendo métodos en una interfaz escrita en lenguaje Java. El desarrollador también codifica una o más clases que implementan dichos métodos. Los programas cliente también son fáciles de codificar. Un cliente crea un proxy (un objeto local que representa el servicio) y a continuación simplemente invoca los métodos sobre el proxy. Con JAX-WS, el desarrollador no necesita generar o "parsear" mensajes SOAP. El runtime de JAX-WS convierte las llamadas y respuestas del API en y desde los mensajes SOAP.

La siguiente figura muestra la interacción entre un cliente y un servicio web a través de JAX-WS.



Comunicación entre un servicio Web y un cliente a través de JAX-WS

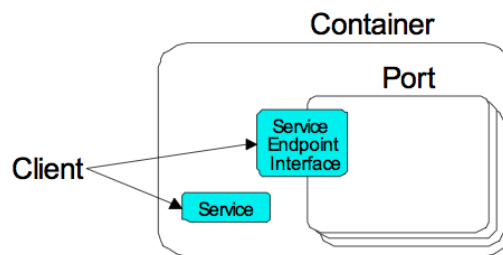
Metro (y consecuentemente, JAX-WS) está incluido en Glassfish 3.x por defecto, por lo que haremos uso de él para implementar nuestros Servicios Web y clientes correspondientes.



### 3.5.1. Los Servicios desde la vista del Cliente.

La vista del cliente de un servicio Web es bastante similar a la de un Enterprise JavaBean. Un cliente de un servicio Web puede ser otro servicio Web, un componente Java EE, incluyendo una aplicación cliente Java EE, o una aplicación Java arbitraria. Una aplicación no Java o un servicio Web para una aplicación no Java EE también podrían ser clientes de un Servicio Web.

Como puede observarse, la vista del cliente de un servicio Web es proporcionada por el proveedor del componente Port y el contenedor en el que éste reside, incluyéndose la clase/interfaz Service (SI: Service Interface), y la interfaz del Endpoint del servicio (SEI: Service Endpoint Interface).



Vista de un cliente de un servicio Web

La clase/interfaz Service (SI) define los métodos que un cliente puede utilizar para acceder a un Port de un servicio Web. Un cliente NO crea o destruye un Port. Utiliza la clase/interfaz Service para obtener el acceso a un Port. La clase/interfaz Service se define en la especificación JAX-WS, pero su comportamiento viene definido en el documento WSDL proporcionado por el proveedor del servicio Web. Las herramientas de despliegue del contenedor proporcionan una implementación de los métodos de la clase/interfaz Service generada por JAX-WS.

El cliente accede a una implementación de un servicio Web utilizando el SEI. Dicho SEI es especificado por el proveedor del servicio. Las herramientas del despliegue y el run-time del contenedor proporciona las clases en la parte del servidor que van a atender las peticiones SOAP sobre la implementación de dicho servicio de los métodos especificados en el SEI.

Un Port no tiene identidad para el cliente, que lo debe considerar como un objeto sin estado.

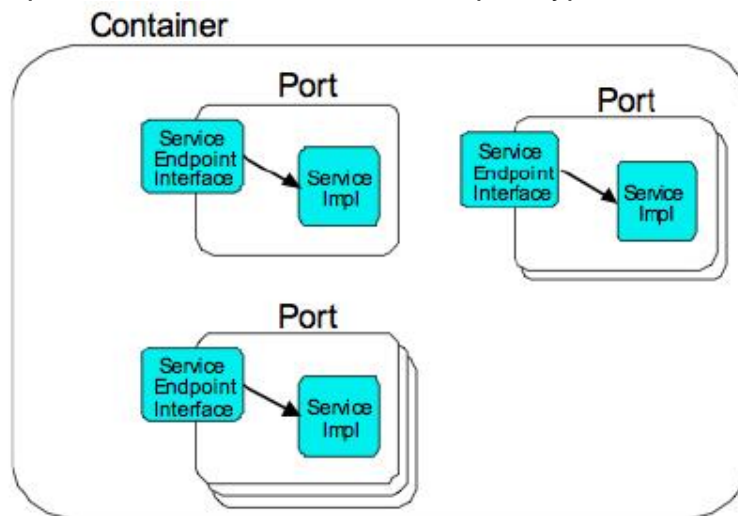
JAX-WS define un modelo de programación en el que se realiza un mapeado de un documento WSDL a Java. Dicho mapeado proporciona una factoría (Service) para seleccionar qué Port (agregado en el servicio) desea usar el cliente. La herramienta JAX-WS que proporciona las clases necesarias en la parte del cliente para poder acceder a un servicio web se denomina wsimport. En general, el transporte, codificación, y dirección del Port son transparentes para el cliente. El cliente solamente necesita realizar llamadas sobre la interfaz del endpoint del servicio (Service Endpoint Interface), utilizando el PortType correspondiente para acceder a dicho servicio.

### 3.5.2. Los servicios Web desde la vista del Servidor.

Un documento WSDL nos proporciona la interoperabilidad de los servicios Web e incluye la especificación sobre requerimientos de transporte y formato de los datos a través de la red. En general, un WSDL no impone ningún requerimiento sobre el modelo de programación del cliente o del servidor. La especificación de servicios Web para Java EE (JSR-109) define tres formas de implementar la lógica de negocio de servicio Web:

- Como un Bean de Sesión sin estado: la implementación del servicio Web (componente Port) se realiza creando un Bean de sesión sin estado, que implementa los métodos del SEI (Service Endpoint Interface) tal y como se describe en la especificación de EJB 3.0.
- Como una clase Java: en este caso se implementa el Port como un Servlet JAX-WS.
- Como un Singleton Session Bean: en este caso se crea un singleton session bean que implementa los métodos de un SEI según la especificación de EJB 3.1

Un componente Port define la vista del servidor de un Servicio Web. Cada Port proporciona un servicio en una dirección física particular definida por el atributo address de la definición <port> de un WSDL. Un componente Port "sirve" una petición de operación definida en un <portType> de un WSDL. La



implementación del servicio (Service Implementation Bean) depende del contenedor del componente Port, pero en general es una clase Java que puede implementar los métodos definidos en el SEI (Service Endpoint Interface). El SEI es un mapeado java del <portType> y <binding> asociado a un <port> de un WSDL. Un servicio Web es un conjunto de Ports que difieren únicamente en su dirección física, y son mapeados en componentes Port separados, cada uno con su potencialmente único, pero probablemente compartido, Service Implementation Bean. La siguiente figura muestra la vista del servidor de un Servicio Web.

Vista del servidor de un servicio Web

El ciclo de vida del Port está completamente controlado por el contenedor, pero

en general sigue el mismo ciclo de vida que el del propio contenedor. Un Port es creado e inicializado por el contenedor antes de que la primera llamada recibida en la dirección del <port> del WSDL sea servida. Un Port es destruido por el contenedor cuando éste considera que sea necesario hacerlo, como por ejemplo cuando el propio contenedor es shutting down. Una implementación de un servicio Web JAX-WS reside en un contenedor Web, y por lo tanto puede desplegarse en un servidor Web o un servidor de aplicaciones, una implementación EJB, en cambio, reside en un contenedor EJB y sólo podrá desplegarse en un servidor de aplicaciones.

Un componente Port asocia una dirección de puerto (port address de un WSDL) con la implementación del servicio (Service Implementation Bean). En general, el componente Port "pospone" la definición de los requerimientos del contenedor del servicio a la fase de despliegue, indicando dichos requerimientos en el descriptor de despliegue del componente. Un contenedor proporciona un listener en la dirección del puerto (port address de un WSDL) y un mecanismo para "enviar" la petición a la implementación del servicio Web. Un contenedor también proporciona servicios en tiempo de ejecución, tales como restricciones de seguridad y mapeados de referencias lógicas a referencias físicas de objetos distribuidos y recursos.

Un desarrollador declara un componente Port en un descriptor de despliegue de servicios Web. El descriptor de despliegue incluye el documento WSDL que describe el PortType y binding del servicio Web. Cuando se usa JAX-WS, no es necesario proporcionar dicho descriptor de despliegue. La mayor parte de la información del descriptor de despliegue se incluye en las anotaciones de la implementación del servicio. Podríamos utilizar el descriptor de despliegue para "sobreescribir" o mejorar la información proporcionada por las anotaciones de la implementación del servicio.

La plataforma Java EE 6, soporta las siguientes implementaciones de servicios Web: como un componente Web JAX-WS en un contenedor de Servlets, y como un componente EJB de sesión stateless o singleton.

El empaquetado de un servicio Web en un módulo Java EE es específico de la metodología de implementación, pero sigue los requerimientos para un fichero EJB-JAR o fichero WAR. Contiene los ficheros de clases java del SEI y los documentos WSDL del servicio Web. Además contiene el descriptor XML de despliegue que define los Ports del servicio y su estructura.

## **El modelo de programación JAX-WS**

Para desarrollar una implementación de un servicio web (web service endpoint) podemos optar por dos puntos de partida: una clase Java que implementa el servicio Web o un fichero WSDL. Cuando comenzamos por una clase java, utilizaremos herramientas para generar los artefactos necesarios, entre ellos el WSDL del servicio. Cuando nuestro punto de partida es un fichero WSDL (junto con los ficheros de esquema que describen los tipos de datos usados en el servicio), utilizaremos herramientas para generar el SEI (Service Endpoint Interface)

---

JAX-WS impone la existencia de un Service Implementation Bean anotado con `javax.jws.WebService` en un componente Port. Como ya hemos indicado en el apartado anterior, la implementación del servicio (Service Implementation Bean) va a depender del contenedor del componente Port, pero en general es una clase Java que puede implementar los métodos definidos en el SEI (Service Endpoint Interface).

Si comenzamos por una clase java, tendremos la seguridad de que la clase que implementa el servicio tiene los tipos de datos java adecuados, pero el desarrollador tiene menos control sobre el esquema XML generado. Si comenzamos por el WSDL y esquemas, el desarrollador tiene un control total sobre qué esquema se está usando, pero menos control sobre el endpoint del servicio generado y de las clases que utiliza.

Cuando nuestro punto de partida es una clase java, tenemos que seguir ciertas restricciones en la implementación de nuestro servicio. Una implementación válida de un servicio web es una clase java que cumple las siguientes restricciones:

- La clase debe estar anotada con `javax.jws.WebService` (o alternativamente con `javax.xml.ws.Provider`, si se desea trabajar directamente a nivel de mensajes XML)
- Podemos anotar cualquiera de sus métodos con `javax.jws.WebMethod`
- Todos sus métodos pueden lanzar la excepción `java.rmi.RemoteException` además de cualquier otra excepción propia del servicio.
- Los parámetros de sus métodos y tipos de retorno deben ser compatible con JAXB (JAXB impone unas reglas para mapear tipos java con tipos de ficheros de esquema XML)
- Ningún parámetro y/o tipo de retorno pueden implementar la interfaz `java.rmi.Remote` ni directa ni indirectamente.

La clase java anotada con `@WebService` define un SEI de forma implícita por lo que no será necesario proporcionar dicha interfaz. Podemos especificar de forma explícita una interfaz añadiendo el atributo `endpointInterface` a la anotación `@WebService`. En ese caso, sí es necesario proporcionar la interfaz que defina los métodos públicos disponibles en la clase que implementa el servicio.

Una implementación de un servicio Web que utiliza la anotación `@WebService` no es necesario que especifique la ubicación del WSDL. Si se utiliza el atributo `wsdlLocation` en la anotación `@WebService`, el fichero WSDL debe ser empaquetado junto con las clases java del servicio web.

### **Creación de servicios a partir del WSDL**

Hemos visto cómo crear con NetBeans servicios web a partir de código Java que ya tenemos implementado. Esta es la forma más inmediata de crear servicios web, sin embargo, si lo que buscamos es una alta interoperabilidad, no resulta la forma más adecuada de hacerlo. Podría darnos problemas sobre todo en el caso

en que nuestras operaciones intercambien tipos de datos complejos, ya que podríamos tener problemas al intentar recomponer dichos tipos desde clientes de diferentes plataformas.

Lo fundamental en un servicio web SOAP es el contrato que existe entre cliente y servicio, es decir, el documento WSDL. Por lo tanto, a la hora de crear servicios web complejos es recomendable empezar definiendo dicho contrato. De esta forma tendremos mayor control sobre los datos que se serializan durante la invocación del servicio, con lo que podremos definir las estructuras de datos que consideremos más adecuadas para el intercambio. Una vez definido el contrato (WSDL), generaremos a partir de él el esqueleto para la implementación del servicio que cumpla con dicho contrato.

Creamos el WSDL y fichero de esquema En NetBeans podremos generar un servicio web a partir de un documento WSDL de forma sencilla. Lo primero que deberemos hacer es escribir el documento WSDL, y el esquema asociado que definirá nuestro servicio. Ya hemos visto cómo hacer esto en la sesión anterior. Recordemos que tenemos que pinchar con el botón derecho sobre nuestro proyecto y seleccionar New > Other .... Nos aparecerá la ventana para crear un nuevo fichero, y dentro de ella seleccionaremos la categoría XML y el tipo XML Schema (para el fichero de esquema), y WSDL Document, para el documento WSDL. Al continuar con el asistente, podremos introducir los datos básicos del documento WSDL , como su nombre, espacio de nombres, puertos, operaciones, mensajes, tipo de codificación, nombre del servicio, etc.

Una vez hayamos terminado de escribir el documento WSDL que actuará como contrato de nuestro servicio, deberemos crear nuestro servicio web. Para crear un servicio web que se ajuste a dicho contrato pincharemos con el botón derecho sobre nuestro proyecto y seleccionaremos New > Web Service from WSDL.... Tras rellenar todos los datos del asistente se generarán una serie de clases con el esqueleto de la implementación de nuestro servicio y los tipos de datos necesarios. Ahora deberemos rellenar el código de cada operación del servicio para darle su funcionalidad, y con esto habremos terminado de implementar nuestro servicio.

### **3.5.3. NetBeans y ficheros WSDL.**

Saber "leer" y/o crear un fichero WSDL es importante para trabajar con servicios Web SOAP. NetBIOS permite trabajar de forma sencilla con ficheros WSDL.

XML Schema es una recomendación del W3C, que proporciona mecanismos para definir la estructura, contenido y semántica de un documento XML. Un fichero WSDL utiliza el "lenguaje de esquemas" (XML Schema) para definir los tipos de datos y/o elementos utilizados por las operaciones de un servicio Web.

El bloque de construcción principal de un documento XML es element. La definición de un elemento element debe contener una propiedad name, que representa su nombre, y una propiedad type para indicar el tipo de elemento.

---

Podemos utilizar alguno de los tipos predefinidos (built-in types), por ejemplo `xs:string`, `xs:integer`, o bien podemos definir nuevos tipos utilizando etiquetas `simpleType` o `complexType`.

NetBIOS nos permite crear ficheros WSDL de forma "visual", simplificando de forma significativa el proceso de edición de la especificación WSDL de nuestro servicio web.

Los servicios Web SOAP tienen una serie de métodos a los que se llama mediante RPC desde cualquier lugar de Internet, por eso mismo debemos poder interpretar en nuestras aplicaciones los mensajes SOAP entrantes de petición para la invocación de un método. Después invocaremos el método solicitado, y con el resultado que nos devuelva, construimos un mensaje SOAP de respuesta y devolvérselo al cliente.

Si tuviésemos que introducir nosotros el código para interpretar este mensaje de entrada, y generar manualmente el mensaje de respuesta, el desarrollo de Servicios Web sería una tarea altamente costosa.

Es más, si se forzase al programador a componer el mensaje SOAP manualmente cada vez que desarrolle un Servicio Web, es muy probable que cometa algún error y no respete exactamente el estándar SOAP. Esto sería un grave problema para la interoperabilidad de los Servicios Web, que es una de las características que perseguimos con esta tecnología.

Para evitar estos problemas, se utilizan librerías que permiten leer o generar mensajes SOAP para la invocación de métodos remotos, como es el caso de la API JAX-WS.

Además, para facilitar aún más la tarea de desarrollar Servicios Web, normalmente contaremos con herramientas que a partir de las clases que implementan nuestro servicio generen automáticamente todo el código necesario para leer el mensaje SOAP de entrada, invocar el método, escribir el mensaje SOAP de salida, y devolverlo al cliente.

De esta manera, logramos centrarnos únicamente en la tarea de programar la funcionalidad que implementan nuestros servicios, olvidándonos del mecanismo de invocación de éstos.

### **3.6. Composición de servicios y SOA.**

Podemos distinguir en una arquitectura SOA las siguientes capas de software:

- Aplicaciones básicas: Son aquellos sistemas que se han desarrollado bajo cualquier tecnología, en cualquier lugar e independiente del autor de las mismas.
- De exposición de funcionalidades: Son aquellas aplicaciones cuyas

funcionalidades son expuestas en forma de servicios (los llamados Servicios Web).

- De integración de servicios: Son aquellos sistemas que facilitan el intercambio de datos entre procesos empresariales internos o en colaboración.
- De composición de procesos: Aquellas aplicaciones que definen el proceso en términos de negocio y sus necesidades y que varían dependiendo de cuál sea dicho negocio.
- De entrega: Son aquellos sistemas en los cuales los diferentes servicios se despliegan directamente a los usuarios finales.

La arquitectura orientada a servicios permite la creación de sistemas que son altamente escalables y que reflejan de manera realista el negocio de la empresa, añadiendo además una manera clara de exposición e invocación de servicios web, lo cual facilita la interacción entre los diferentes sistemas existentes, ya sean propios o ajenos.

Con la arquitectura SOA se obtiene una metodología y un marco de trabajo en el cual se pueden documentar las capacidades de negocio y dar soporte, además, a las actividades de integración y consolidación. Algunos beneficios que tiene adoptar esta arquitectura son:

- Mejora en los tiempos de realización de cambios en los procesos
- Facilidad para evolucionar a modelos de negocio basados en terceros
- Facilidad para abordar modelos de negocio basados en colaboración con otras entidades
- Poder para reemplazar elementos de la arquitectura SOA sin interrupción del proceso de negocio
- Facilidad de integración de tecnologías diferentes.

A diferencia de las llamadas arquitecturas orientadas a objetos, las arquitecturas SOA están formadas por servicios de aplicación débilmente acoplados y altamente interoperables, que para comunicarse se basan en una definición formal independiente de la plataforma y del lenguaje de programación.

La interfaz encapsula las particularidades de la implementación, haciéndola por tanto independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo empleada. Así los componentes software desarrollados son muy reutilizables, ya que la interfaz se define siguiendo un estándar.

---

### 3.6.1. Breve Introducción a la Composición de Servicios.

En algunos casos los requerimientos de información planteados por el usuario no son alcanzados a través de un simple Servicio Web; entonces se hace necesario combinar varios de ellos para generar un servicio que pueda satisfacer los objetivos del usuario.

La Composición de Servicios es una aplicación de la Web Semántica y las tecnologías de planificación automática que describe la manera de enlazar los diferentes servicios Web. Nace de la necesidad de disponer de metodologías que permitan desarrollar sistemas orientados a servicios y la dificultad de utilizar los Servicios Web en las etapas más tempranas de su desarrollo.

Con base en los fundamentos de la programación orientada a servicios (SOC) y su arquitectura (donde se emplean los servicios web como los elementos fundamentales), se identifican posibles roles que los servicios web juegan dentro del proceso de composición. Tales roles pueden ser un repositorio de servicios, un servicio ejecutor de servicios o un planificador, dependiendo de la funcionalidad que se le desee proveer al servicio web para llevar a cabo un propósito en común.

Distintas tecnologías han aportado soluciones a la Composición de Servicios Web, entre ellas se encuentran los Sistemas Multiagente (SMA), los cuales gracias a sus características de modularidad y distribución de funcionalidades entre nodos de red pueden acoger fácilmente los roles presentes en un proceso de Composición de Servicios Web.

Para poder realizar la composición de servicios es necesario que el middleware de los servicios Web soporten la composición; la tecnología empleada es muy similar a la que se utiliza en los sistemas Workflow y se denomina "Business Process Execution Language for Web Services", más conocido como BPEL.

El proceso de ejecución debe soportar tanto los servicios simples como compuestos. Los trabajos realizados en la Composición de Servicios Web básicamente enfocan sus esfuerzos en tres áreas del conocimiento, que son, vía procesos de negocios, planificación en IA y síntesis de Programas.

Se pueden distinguir dos maneras diferentes de combinar los distintos servicios, de los cuales hablaremos más adelante con mayor profundidad; estas dos formas se denominan orquestación y coreografía. La composición de servicios puede contener los dos tipos a la vez.

Tanto en la orquestación como en la coreografía la secuencia de ejecución se determina en alguno de los archivos de los servicios. A esto se le denominan formas estáticas de composición, a diferencia de las dinámicas, que también pueden utilizarse para componer servicios, y en las que no está determinada la secuencia de ejecución completa de antemano.

En este tipo de servicios, si la descripción de los mismos contiene su semántica de manera que pueda ser procesada por las máquinas, el propio proceso podría

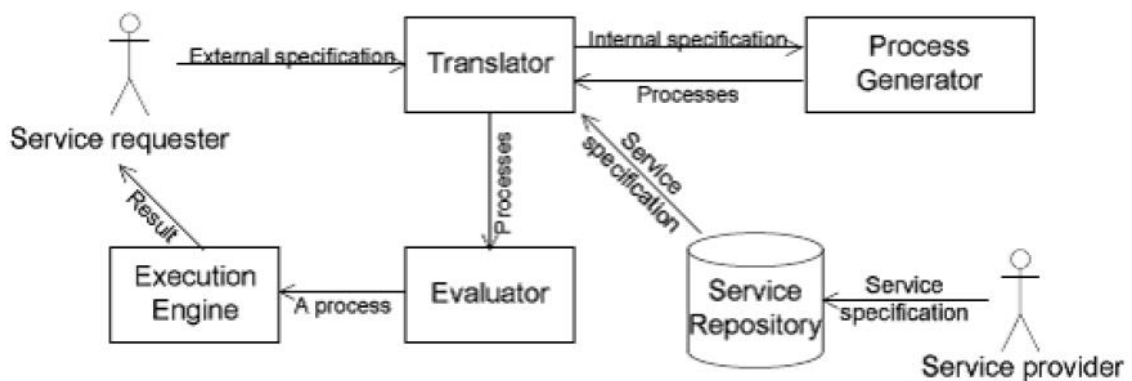


decidir qué servicio invoca después de que haya comenzado.

En el proceso de Composición de Servicios Web, se pueden distinguir dos pasos generales, que materializan un requerimiento de usuario:

- La identificación de los requerimientos del usuario por medio de un método por el cual se pueda generar un plan o flujo de actividades
- La selección de los Servicios Web para su posterior ejecución y monitoreo.

A continuación se puede ver un gráfico que ejemplifica un framework de un sistema de composición de servicios.



### 3.6.2. Orquestación vs. Coreografía.

Dentro de la composición de servicios se pueden distinguir diferentes aspectos. Orquestación y coreografía son dos aproximaciones que definen la composición de servicios, pero que no son lo mismo.

**Orquestación:** Genéricamente, se entiende por Orquestación como el estudio y práctica de ordenar la música para una orquesta; básicamente, orquestación es decidir qué instrumentos deben tocar qué notas en una pieza de música.

**Coreografía:** se define como la ordenación y el movimiento de los bailarines en un determinado escenario; sencillamente consiste en decidir cómo debe efectuarse el movimiento al ritmo de la música.

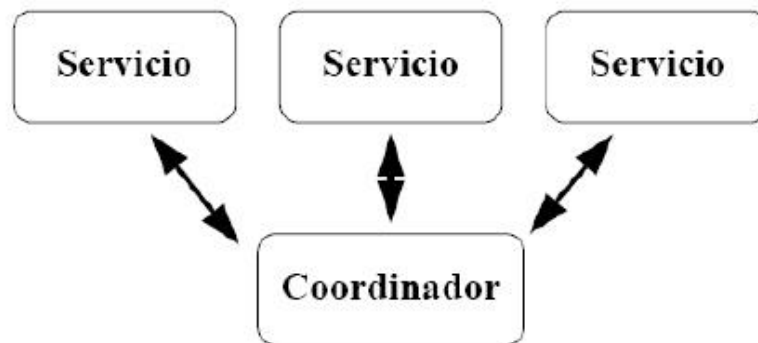
Aplicado a los servicios Web, la Orquestación describe un proceso ejecutable que puede interactuar con servicios propios o ajenos; el flujo del proceso está controlado por un coordinador central y cada servicio tiene solamente conocimiento local y puede tomar decisiones solo para procesos propios.

La orquestación se refiere a la habilidad para conectar servicios Web entre sí para dar lugar a procesos de negocio de alto nivel. Se centra en la composición de nuevos servicios a partir de otros ya existentes y nos proporciona una manera

de describir el comportamiento interno de qué necesitan los servicios para trabajar conjuntamente y así poder crear otro servicio como composición de los primeros.

Para Orquestar Servicios se suele emplear el lenguaje WS-BPEL (del inglés “Web Services Business Process Execution Language”); así, la orquestación podría verse como la ejecución de un proceso de negocio definido en ese lenguaje y que puede ser ejecutado por un motor BPEL.

Básicamente, un diagrama que muestra el esquema de lo que es la orquestación podría ser:



En este diagrama vemos la existencia de un mecanismo de control centralizado que es el encargado de dirigir las diferentes actividades, siendo cada una de ellas una interacción entre servicios.

**Coreografía:** La palabra coreografía se refiere a las diferentes interacciones entre servicios Web, intercambios de mensajes que se efectúan para ejecutar varios procesos particulares o flujos de control. En una coreografía cada servicio tiene su propia tarea en la composición total. No existe un programa central que verifique que la tarea se está cumpliendo de forma correcta.

En la coreografía se define la interoperabilidad necesaria para crear un sistema compuesto de servicios Web y nos permite especificar las reglas de unión y de trabajo colaborativo de diferentes procesos. En resumen es un proceso de negocio global donde se modela el estado de negocio de diversos servicios Web.

El lenguaje empleado para la coreografía de servicios es WS-CDL (del inglés, “Web Services Choreography Description Language”) del cual hablaremos brevemente en el siguiente capítulo de este monográfico.

Se puede ver, por tanto, que la orquestación ofrece una visión más detallada, mientras que la coreografía ofrece una visión mucho más abstracta y descriptiva de los actores que intervienen en el intercambio de mensajes para la ejecución de los procesos.

Coreografía y Orquestación no son lo mismo. Las diferencias entre orquestación y coreografía están basadas en analogías: la orquestación describe un control

central del comportamiento como un director en una orquesta, mientras que la coreografía trata sobre el control distribuido del comportamiento donde participantes individuales realizan procesos basados en eventos externos, como en una danza coreográfica donde los bailarines reaccionan a los comportamientos de sus pares.

La orquestación mantiene el control centralizado de un proceso de negocio, de manera análoga al director de orquesta que mantiene el control sobre sus integrantes; tal es así que si el director realiza un alto o aumenta el ritmo todos reaccionarán de forma instantánea y coordinada. Por el contrario, en una coreografía, donde el comportamiento ya está predefinido por el coreógrafo y cada uno de los miembros solo trata de cumplir con lo previamente pactado, cualquier cambio sobre la marcha sería impensable.

Destacar finalmente que lo que más se está usando en este momento es la orquestación, la coreografía es el aspecto sobre el que más se están centrando la mayor parte de los estudios. Así, la orquestación es la única alternativa fiable, dado que la coreografía se mueve únicamente dentro de los entornos de la investigación.

### 3.6.3. Workflow vs. Dataflow.

“Workflow” y “Dataflow” son dos aspectos diferentes dentro de la composición de servicios.

**Workflow:** Se define como workflow (o flujo de trabajo) como el estudio de todos los aspectos relativos a las operaciones que se llevan a cabo en una actividad de trabajo, por ejemplo, la sincronización de las tareas, cómo se realizan, cuál es su orden correlativo, cómo fluye la información que soporta las tareas y cómo se le hace seguimiento al cumplimiento de las mismas.

El workflow se considera parte esencial del software para trabajo colaborativo.

Una aplicación workflow automatiza la secuencia de acciones, actividades o tareas utilizadas para la ejecución del proceso, incluyendo el seguimiento del estado en cada una de sus etapas y la aportación de las herramientas necesarias para su gestión.

Dentro del workflow podemos distinguir tres tipos de actividad diferentes, en función de si se tratan de tareas colaborativas, cooperativas o coordinadas:

- Actividades colaborativas: Son aquellas realizadas por un conjunto de usuarios que trabajan sobre los mismos datos para obtener un resultado común. Tiene entidad el trabajo de cada uno de ellos en sí mismo.
- Actividades cooperativas: En las actividades cooperativas diferentes usuarios trabajan sobre su propio conjunto de datos particular, estableciendo mecanismos de cooperación entre ellos. No tiene entidad el trabajo de ninguno de ellos si no es visto desde el punto de vista global

del resultado final.

- Actividades de coordinación: En este tipo de actividades, varios usuarios se coordinan entre sí para desarrollar un determinado trabajo.

Los objetivos que persiguen los sistemas de workflow están orientados al acercamiento de las personas, los procesos y las máquinas, independientemente de donde se encuentren, con el fin de conseguir reducir el tiempo y acelerar la realización de un determinado trabajo.

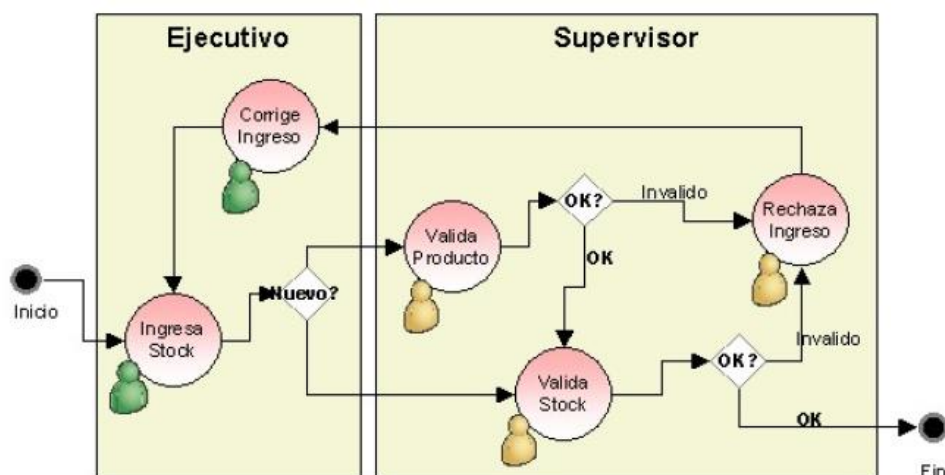
Con estos sistemas se consigue:

- Reflejar, mecanizar y automatizar los métodos y organización en el sistema de información.
- Establecer los mecanismos de control y seguimiento de los procedimientos organizativos.
- Independizar el método y flujo de trabajo de las personas que lo ejecutan.
- Facilitar la movilidad del personal.
- Soportar procesos de reingeniería de negocio.
- Agilizar el proceso de intercambio de información y agilizar la toma de decisiones de una organización, empresa o institución.

La notación gráfica estandarizada que permite el modelado de procesos de negocio con este formato de workflow es el BPMN (“Business Process Modeling Notation”, traducido como Notación para el Modelado de Procesos de Negocio).

Inicialmente, fue desarrollada por la organización Business Process Management Initiative (BPMI) y en la actualidad la mantiene el OMG (Object Management Group).

La imagen siguiente muestra un ejemplo de Workflow que define un proceso de ingreso de nuevo stock:



**Dataflow:** La perspectiva del Dataflow va más enfocada hacia el paso de información o el alcance de las variables, a diferencia del Workflow, más orientado a capturar aspectos relacionados con las dependencias del control del flujo entre diferentes tareas.

Los lenguajes de dataflow promueven los datos como el principal concepto detrás de cualquier programa. Esta perspectiva tiene su mayor exponente en el sistema operativo Unix.

Los programas que emplean lenguajes de dataflow suelen comenzar con una entrada para después ilustrar como se deben utilizar y modificar los datos. Los datos se convierten así en algo explícito, a menudo ilustrados físicamente en pantalla como una línea o tubo mostrando dónde fluye la información.

Las operaciones suelen consistir en una especie de cajas negras, con entradas y salidas explícitamente definidas. Estas operaciones se lanzan tan pronto reciben una entrada válida. Cabe destacar que estas salidas tendrán un formato y características determinados y será necesaria una mediación de los datos de esta salida para que se ajusten a los requisitos de entrada precisados por el proceso que los recibe. A esta mediación es a lo que se conoce como mapping de datos.

A diferencia de los programas tradicionales, un programa basado en un lenguaje dataflow se puede asemejar a una cadena de montaje, donde cada trabajador comienza a hacer su tarea en el momento en el que le llega el material. Por esta razón los lenguajes dataflow son típicamente paralelos; las operaciones no tienen estados ocultos para monitorizar y están todas preparadas a la vez.

En un sistema tradicional las instrucciones se ejecutan una tras otra. Los programas dataflow se pueden implementar con tablas hash que identifican de manera única las entradas. Cuando una operación finaliza, el programa escanea la lista de operaciones hasta que encuentra la primera que tiene las entradas correctas y comienza a ejecutarla; cuando finaliza, devuelve diferentes valores de salida que hacen que actualizan la tabla preparando otras operaciones para su ejecución.

Esta lista debe ser compartida cuando se realizan operaciones en paralelo. Además, como la lista muestra el estado actual del proyecto, se puede prescindir de la tarea de mantenimiento de estado. En máquinas de un solo procesador donde la implementación diseñada para operaciones en paralelo podría producir sobrecarga el dataflow podría evitarla empleando diferentes tiempos de ejecución.

## 4. Metodología.

Con la metodología utilizada para modelar el sistema, defino y explico los pasos que he seguido para desarrollar el proyecto, de tal forma que todo aquel que pueda estar interesado en la integración de sistemas diferentes a través de servicios web, puede tener la misma visión.

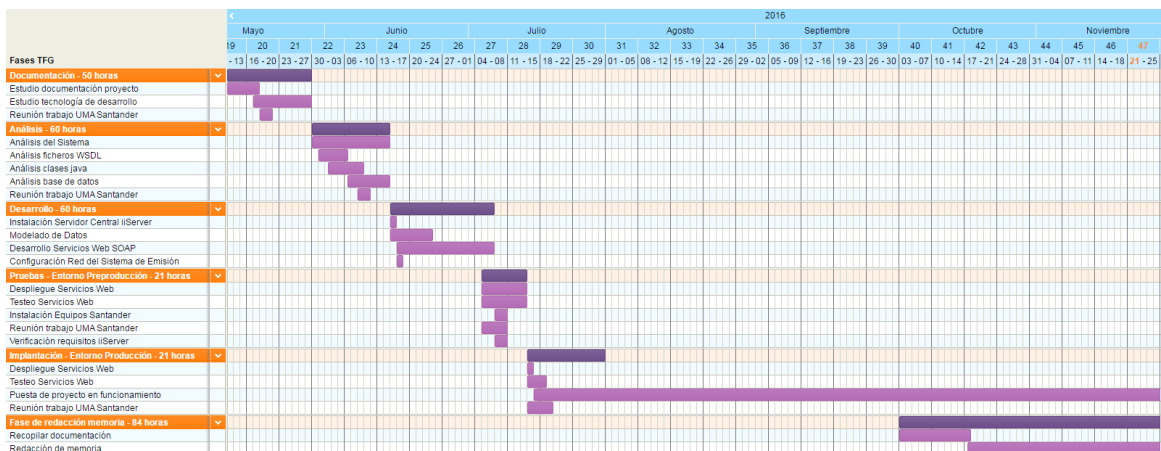
Durante el desarrollo del presente proyecto, se hizo una revisión y recopilación de los distintos materiales y documentación que están relacionados con ambos sistemas informáticos y son necesarios para el buen desempeño en la toma de decisiones.

Se realizó un estudio de los Servicios Wep SOAP con WSDL, junto con algunos entornos IDE de desarrollo software.

El trabajo fue planificado siguiendo seis fases:

- **Documentación - 50 horas:** En la primera fase se recopilará la información y se hará un estudio sobre la materia, consultando las distintas herramientas informáticas y recursos software que se usarán a lo largo del trabajo.
  - a. En este punto se tomó la decisión de desarrollar los servicios web programándolos en Java 1.7 con Librerías JAX-WS utilizando el IDE de NetBeans, frente a desarrollarlo en C# para .NET (4.0) o PHP mediante nusoap, que eran las otras posibilidades de desarrollo habilitadas en el Sistema de Emisión Instantánea.
  - b. Se entendió que el desarrollo de una plataforma “portable” que no requiriese de la instalación de ningún servicio, framework o contenedor en el servidor (véase Apache, Apache- Tomcat, etc...) facilitaría un uso que estaría más cerca al objetivo final del proyecto: su puesta en marcha de forma rápida y sin exceso de requerimientos técnicos.
- **Análisis - 60 horas:** Realización el análisis completo del sistema
  - a. Se decidió los módulos a desarrollar para completar los servicios web.
  - b. También se decidió las consultas, vistas y tablas de base de datos en el sistema UMA que se tenían generar para un desarrollo efectivo de los servicios web.
- **Desarrollo - 60 horas:**
  - a. Modelado de datos
  - b. Desarrollo de los servicios web,
  - c. Instalación del servidor web que es necesario para el proyecto.

- **Pruebas - 21 horas:**
  - a. Realizadas por el soporte técnico del Banco Santander a cada entrega del desarrollo incremental que se fue entregando, siguiendo la filosofía de desarrollo ágil de la metodología Scrum.
  - b. Visita en la UMA de los desarrolladores del proyecto por parte del Banco Santander, para participar en el despliegue de los servicios web en el entorno de preproducción, con el objetivo de que pudieran ser testados y depurados, para garantizar entre ambos equipos el correcto funcionamiento de la integración de ambos sistemas de información.
- **Implantación - 21 horas:**
  - a. Visita en la UMA de los desarrolladores del proyecto por parte del Banco Santander, para participar en el despliegue de los servicios web en el entorno de preproducción, con el objetivo de que pudieran ser testados y depurados, para garantizar entre ambos equipos el correcto funcionamiento de la integración de ambos sistemas de información.
- **Elaboración de la memoria - 84 horas:**
  - a. Reorganización de toda la documentación generada,
  - b. Elaboración de la memoria final del trabajo fin de grado.



Durante la fase de desarrollo de los servicios web, utilice los conceptos de metodología Scrum con los que trabajamos en la UMA.

Hay que señalar que fue un Scrum atípico ya que el proyecto no fue abarcado por un equipo de desarrollo completo, sino que yo fui el único desarrollador asignado. Los equipos de desarrollo están formados por un coordinador o scrum master y tres desarrolladores software. En este proyecto el cliente o product owner fue el responsable del soporte técnico del Banco Santander.

Aun así, siguiendo la filosofía Scrum, realicé hasta cuatro entregas de los servicios web, cumpliendo con la estrategia de desarrollo incremental del

producto, para que el product owner pudiera ir viendo los avances de la integración de los sistemas planteadas.

Además del desarrollo de los servicios web, la integración de sistemas estuvo coordinada por dos otros dos equipos de especialistas del Servicio Central de Informática.

Un equipo de dos especialistas del área de ATDI, que se encargó de montar el servidor iiServer en una máquina virtual Y otro equipo de dos especialistas del área de sistemas, que realizó todo lo concerniente a permisos y configuración de firewall, limitando desde las IP que tenían acceso, los servicios web del sistema desplegados en iiServer. Ambos equipos cumpliendo con los requisitos del product owner al respecto.

Gracias a la metodología SCRUM, desde el principio de la fase de desarrollo se buscó la calidad del resultado, primando el desarrollo ágil y las entregas de productos parciales, dejando de un lado el intercambio de documentación para centrarnos en la obtención de resultados.

Los procesos que se identificaron para abarcar el desarrollo del proyecto fueron:

- Estudio servicios Web SOAP.
- Estudio interfaz WSDL de servicios WEB.
- Análisis requisitos Sistema Emisión Instantánea.
- Creación y configuración iiServer.
- Seguridad a nivel de red para el Sistema Emisión Instantánea.
- Ajustes modelo de datos del sistema.
- Servicio Web búsqueda usuarios.
- Servicios Web usuario seleccionado.
- Servicio Web tarjeta impresa.
- Instalación de los diferentes puntos de emisión del sistema.
- Despliegue y testeo de servicios web en preproducción.
- Despliegue y testeo de servicios web en producción.
- Puesta en producción de la integración de sistemas realizada.
- Acceso a iDUMA con carné UMA.



## Pasando todos por las diferentes fases de desarrollo software en Scrum: **TO DO**, **DOING** y **DONE**

TO DO	DOING	DONE
despliegue pro	ws selección usuario	Estudio ws SOAP.
puesta en producción	ws tarjeta impresa	Estudio interfaz WSDL
Acceso a iDUMA con carné UMA.		Modelo de datos
		Análisis requisitos Sistema
		iiServer
		Seguridad a nivel de red
		ws búsqueda de usuario
		despliegue prepro
		Puntos de emisión

estado de la pizarra Scrum en un momento dado del proyecto

### 4.1. Metodología Scrum.

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto.

### El proceso.

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones que normalmente son de 2 semanas, aunque en algunos equipos son de 3 y hasta 4 semanas, límite máximo de feedback y reflexión). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.

El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente prioriza los objetivos balanceando el valor que le aportan respecto a su coste y quedan repartidos en iteraciones y entregas.



---

Las actividades que se llevan a cabo en Scrum son las siguientes:

- **Planificación de la iteración:** El primer día de la iteración se realiza la reunión de planificación de la iteración. Tiene dos partes:
  - **Selección de requisitos** (4 horas máximo). El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.
  - **Planificación de la iteración** (4 horas máximo). El equipo elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos a que se ha comprometido. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se autoasignan las tareas.
- **Ejecución de la iteración:** Cada día el equipo realiza una reunión de sincronización (15 minutos máximos). Cada miembro del equipo inspecciona el trabajo que el resto está realizando (dependencias entre tareas, progreso hacia el objetivo de la iteración, obstáculos que pueden impedir este objetivo) para poder hacer las adaptaciones necesarias que permitan cumplir con el compromiso adquirido. En la reunión cada miembro del equipo responde a tres preguntas:
  - ¿Qué he hecho desde la última reunión de sincronización?
  - ¿Qué voy a hacer a partir de este momento?
  - ¿Qué impedimentos tengo o voy a tener?Durante la iteración el Facilitador (Scrum Master) se encarga de que el equipo pueda cumplir con su compromiso y de que no se merme su productividad.
  - Elimina los obstáculos que el equipo no puede resolver por sí mismo.
  - Protege al equipo de interrupciones externas que puedan afectar su compromiso o su productividad.Durante la iteración, el cliente junto con el equipo refinan la lista de requisitos (para prepararlos para las siguientes iteraciones) y, si es necesario, cambian o replanifican los objetivos del proyecto para maximizar la utilidad de lo que se desarrolla y el retorno de inversión.
- **Inspección y adaptación:** El último día de la iteración se realiza la reunión de revisión de la iteración. Tiene dos partes:
  - **Demostración** (4 horas máximo). El equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo. En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, replanificando el proyecto.
  - **Retrospectiva** (4 horas máximo). El equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad. El Facilitador se encargará de ir eliminando los obstáculos identificados.

---

## 5. Análisis de la interconexión de sistemas planteados.

El sistema desarrollado se sostiene sobre varios puntos claves que pasamos a analizar.

### 5.1 Análisis de servicios Web SOAP del sistema.

La interconexión de los sistemas de información de la UMA y de emisión instantánea del Banco Santander, se sustentarán en el funcionamiento de los servicios Web SOAP que deben interconectar ambos sistemas de información.

Dichos servicios web han sido especificados a través de sus correspondientes ficheros WSDL. El cumplimiento de las especificaciones de dichos ficheros debe ser estricto para que ambos sistemas puedan integrarse sin errores.

Los servicios web a desarrollar se dividen en la búsqueda de alumnos, la petición de datos del alumno seleccionado para hacer la impresión de su carné y el envío de los datos de la tarjeta impresa a los sistemas de Información de la UMA.

En concreto los servicios web son:

- **searchUsers:** servicio web que busca un usuario o un conjunto de usuarios en el sistema de información la UMA. Es utilizado en el proceso de emisión de carnés para localizar el usuario concreto al cual queremos emitir una tarjeta, ya sean alumnos, profesores y/o personal de administración y servicios. No obstante, en la primera fase de integración de los sistemas, sólo nos centraremos en el colectivo de alumnos de la UMA.
- **searchUserById:** servicio web que solicita los datos para poder emitir y personalizar un carné de la UMA a un usuario concreto. Sirve para recuperar todos los datos que serán utilizados en la personalización y grabación de su tarjeta inteligente.
- **sendIssuedCard:** servicio web que es invocado cada vez que se emite una tarjeta inteligente. Sirve para notificar a la UMA los datos de los carnés que son emitidos de cada usuario.

Todo está preparado para que los servicios web puedan trabajar directamente identificando a los usuarios a través de iDUMA, pero actualmente la integración total del Sistema de Emisión de carnés con iDUMA no es posible, porque iDUMA no contiene de toda la información necesaria para el correcto funcionamiento de los servicios web.

La identificación contra iDUMA está planificada como obligatoria hoy en día para cualquier servicio montado en la UMA, pero todavía hay trasvase de información que no se ha realizado porque se está planificando y analizando que datos son

los necesarios que deben llevarse a IDUMA, por ello la información del colectivo de alumnos de la UMA no está al 100% en iDUMA.

Por todo ello, todos los servicios web se han desarrollado para que accedan a la base de datos Oracle que gestiona los sistemas de información de los alumnos de la UMA. Para los servicios web `searchUsers` y `searchUserById`, habrá que diseñar las vistas de consulta de datos de los alumnos del proyecto. Para el servicio web `sendIssuedCard`, se deberá crear una tabla en la que almacenar los datos de los carnés emitidos de cada usuario.

### 5.1.1. Consultas de usuarios - `searchUsers`.

El sistema de emisión instantánea accederá en tiempo real a los datos de personalización utilizando los servicios web de consulta de usuarios. Con éste mecanismo se garantiza a la UMA un control total sobre los usuarios a los cuales se entrega una tarjeta universitaria inteligente TUI, y los datos utilizados en la personalización.

La consulta de usuarios se realiza con dos funciones de búsquedas distintas para recuperar información. Estos dos métodos se describen a continuación, en las siguientes secciones.

**Búsqueda del usuario - `searchUsers`:** Función que busca un usuario o un conjunto de usuarios en el sistema de información de la UMA. Es utilizada en el proceso de emisión para localizar el usuario concreto al cual queremos emitir una tarjeta.

La búsqueda se hace por una combinación de “Documento de identidad” y “Nombre y Apellidos”. De esta forma el operador del Banco Santander puede localizar a un usuario por su DNI, por su Pasaporte o utilizando algún apellido del usuario en cuestión.

El método de búsqueda recibe un único parámetro de entrada. Es una cadena de texto y se corresponde con la información introducida por el operador del Banco Santander en la `iiTablet` para buscar al alumno, PAS o PDI, que solicita la tarjeta. En la primera fase del proyecto, sólo se buscan alumnos de la UMA.

```
/**
 * Búsqueda inicial de usuarios a través de una cadena de texto.
 *
 * @param data Dato variable utilizado para la búsqueda de usuarios.
 * @return Lista de usuarios con el resultado de la búsqueda.
 */
List<UserData> searchUsers(String data);
```

Al recibir éste método, el servicio web desarrollado implementa una búsqueda de usuarios en el sistema de información de la UMA, utilizando el campo de

---

búsqueda data, y después retorna una lista de usuarios localizados. A la hora de implementar la búsqueda se tuvo en cuenta:

- El listado devuelve siempre cualquier alumno que coincida con el criterio de búsqueda. Aunque ya tenga tarjeta entregada, no haya completado la matrícula, o no tenga derecho a ella. El control de todas estas situaciones se realiza en otro método distinto.  
Es importante que funcione así, para evitar incidencias cuando el operador del Banco no encuentre a un usuario en la iiTablet: si no se encuentra al usuario, debe ser porque realmente no existe, y no porque no tenga derecho a tarjeta. De esta manera, se reduce considerablemente el nivel de incidencias por no localización de usuarios.
- Si el campo data enviado es vacío o nulo, se retorna una lista de usuarios vacía. De esta forma en la iiTablet del operador del Banco Santander se mostrará un mensaje indicando que afine la búsqueda.
- Si en la búsqueda no se produce ninguna coincidencia, se debe retornar una lista vacía. En ningún caso se debe retornar una excepción. De esta forma en la iiTablet del operador del Banco Santander se mostrará un mensaje indicando que afine la búsqueda.
- La función de búsqueda podría devolver muchos usuarios (por ejemplo si usamos un apellido muy común). Por esta razón, se decidió restringir el tamaño de la lista de usuarios resultante a 10 usuarios. De esta forma evitamos un retardo en el intercambio de información con la iiTablet.
- En la búsqueda, se tiene en cuenta que el parámetro data puede tener caracteres especiales (acentos, tildes, diéresis, cedillas, etc.), por lo que se desarrolló ignorando estos caracteres especiales a la hora de hacer consultas en el sistema de información de la UMA.
- La lista de usuarios retornada en éste método, no contiene datos sobre personalización de la tarjeta. Esta lista que es retornada a la iiTablet, se utiliza para informar al operador del Banco Santander de los usuarios que coinciden con el campo de búsqueda. Después el operador debe seleccionar uno de los usuarios de la lista, y en ese momento se invocará otro método distinto.
- La información que debe retornarse para cada usuario de la lista es la siguiente:
  - Nombre y apellidos (**fullname**): Texto con el nombre y apellido(s) juntos.
  - Número de documento (**document**): Número de documento o pasaporte.
  - ID del usuario (**externalId**): Identificador unívoco del usuario dentro del sistema de información de la UMA. Se eligió un identificador único de la tabla de alumnos.
- Para notificar errores graves inesperados se puede lanzar cualquier excepción (**Soap Fault**) sin que sea necesario especificar ningún tipo en concreto. El servicio de emisión capturará las excepciones inesperadas y

---

mostrará un mensaje al operador del Banco Santander indicando que debe contactar con el Soporte Técnico.

Un ejemplo de intercambio XML mediante SOAP en una búsqueda de usuarios:

- Búsqueda de alumno por DNI.

#### PETICIÓN

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchUsers xmlns="http://external.issue.datio.es/">
      <data>71936000A</data>
    </searchUsers>
  </soap:Body>
</soap:Envelope>
```

#### RESPUESTA, EL USUARIO NO EXISTE

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchUsersResponse xmlns="http://external.issue.datio.es/">
      </searchUsersResponse>
    </soap:Body>
</soap:Envelope>
```

#### RESPUESTA, USUARIO ENCONTRADO

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchUsersResponse xmlns="http://external.issue.datio.es/">
      <users>
        <document>71936000A</document>
        <externalId>COD000012345</externalId>
        <fullname>LUIS HERNÁNDEZ MORENO</fullname>
      </users>
    </searchUsersResponse>
  </soap:Body>
</soap:Envelope>
```



- Búsqueda de alumno por apellido.

#### PETICIÓN

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchUsers xmlns="http://external.issue.datio.es/">
      <data>PÉREZ</data>
    </searchUsers>
  </soap:Body>
</soap:Envelope>
```

#### RESPUESTA, VARIOS USUARIOS ENCONTRADOS

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchUsersResponse xmlns="http://external.issue.datio.es/">
      <users>
        <document>120000B</document>
        <externalId>COD000098765</externalId>
        <fullname>MARÍA TERESA PÉREZ DE LA FUENTE</fullname>
      </users>
      <users>
        <document>84025487J</document>
        <externalId>COD000000025</externalId>
        <fullname>ISABEL ISLA RUIPÉREZ</fullname>
      </users>
    </searchUsersResponse>
  </soap:Body>
</soap:Envelope>
```

### 5.1.2. Selección del usuario a emitir tarjeta - •searchUserById.

Selección del usuario concreto a emitir tarjeta: Función que solicita los datos para poder emitir y personalizar una TUI a un usuario concreto de la UMA. Sirve para recuperar todos los datos que serán utilizados en la personalización y grabación de su tarjeta inteligente.

Este método sólo acepta un parámetro de entrada. Es el identificador interno y unívoco del usuario, con el cual el Sistema de Información de la UMA debe buscarlo. Este parámetro se corresponde con el campo recibido externalId en el método de búsqueda del apartado anterior.

Es decir, cuando el operador busca un usuario en la iiTablet, se invoca el método searchUsers y el servicio web retorna una lista de usuarios. Después esa lista de usuarios es mostrada en la iiTablet, y el operador del Banco Santander debe seleccionar uno de ellos para pedir sus datos completos. Al pulsar sobre un usuario de la lista, se invoca el método searchUserById utilizando el ID del usuario que fue retornado en el listado inicial.



```

/**
 * Función que solicita la búsqueda de un usuario concreto.
 *
 * @param userId Identificador unívoco del usuario (campo "externalId").
 * @return Usuario localizado por identificador.
 * @throws UserNotFoundException El usuario solicitado no existe.
 * @throws UserNotAllowedException Usuario no autorizado para emitir tarjeta.
 */
UserData searchUserById(String userId)
    throws UserNotFoundException, UserNotAllowedException;

```

Por tanto, al recibir éste método, se ha implementado una búsqueda de los datos del usuario mediante el ID. Los datos, que se retornan mediante el objeto **UserData**, son los siguientes:

- Los mismos datos que en la búsqueda inicial:
  - Nombre y apellidos (**fullname**): Texto con el nombre y apellido(s) juntos.
  - Número de documento (**document**): Número de documento o pasaporte.
  - ID del usuario (**externalId**): Identificador unívoco del usuario.
- Todos los datos que serán utilizados para la personalización de la tarjeta. Estos datos se almacenan en el objeto **UserData**, dentro de una estructura con formato de mapa (Clave– Valor) que se llama **content**. La estructura **content** es un mapa de valores abierto, que permite a la UMA especificar sus propios datos de personalización en la tarjeta. El mapa de valores para la estructura **content** de la UMA es:
  - **nya**: Nombre y apellidos que son mostrados en el anverso de la tarjeta. Tiene un tamaño máximo de 26 caracteres, ya que si es mayor, será truncado. Además se retorna mayúsculas.
  - **linea1**: Categoría (Estudiante/PAS/PDI) y número de identificación (Expediente, DNI). Debe tener un tamaño máximo de 40 caracteres y estar en mayúsculas.
  - **codbar**: Código de barras único de cada usuario de la UMA, que se corresponde con la acreditación única que se tiene en la UMA y comienza por 061, dicho código de barras irá impreso en el reverso del carné.
  - **nia**: Número de Identificación Académica. Es el número de expediente con el cual se identifica al alumno dentro de la universidad. Debe tener un tamaño máximo de 12 caracteres.
  - **documento**: Documento de identificación con letra (NIF) o pasaporte. Debe tener un tamaño máximo de 9 caracteres.
  - **categoría**: Código de la categoría del usuario (alumno, PAS, PDI, etc.). Es un número comprendido entre 1 y 99 cuya utilización depende de la UMA.
  - **centro**: Código del centro de trabajo o estudios asociado al usuario.

- **curso:** Año actual en formato numérico de 4 dígitos.
- **activa:** Bandera que indica si la tarjeta está activa o no. Se envía con un valor de 1 para que la tarjeta emitida esté activa al entregarse.
- **foto:** si disponemos de la fotografía del usuario, será enviada en este campo para que sea impresa en el anverso de la tarjeta.  
Si la UMA no la tiene, simplemente este atributo no debe incluirse en el mapa, y el operador del Banco deberá tomar una fotografía al usuario con la iiTablet. Sin embargo, si este atributo viene especificado, la iiTablet mostrará la fotografía y el operador puede decidir si desea utilizarla o no.  
La fotografía debe estar en formato JPG (menos de 200 Kb.) y a su vez codificada en Base 64, para que pueda ser transmitida como un texto por el servicio web.
- **interno:** Campo opcional que puede ser utilizado por la UMA para almacenar información interna en la tarjeta. Tiene un tamaño máximo de 8 caracteres. De momento no se usa.
- **reserva1:** Campo opcional utilizado para enviar la fecha de nacimiento del usuario, con el formato yyyy-mm-dd.
- **reserva4:** Campo opcional utilizado para enviar el dominio del usuario: ALU.UMA.ES
- **estudios:** Descripción de los estudios que cursa el alumno propietario de la tarjeta. Contiene el nombre del plan de estudios principal. Para el personal de la universidad se deja vacío.
- **pin:** Número personal PIN (para uso académico y no financiero) que se graba en la tarjeta. Tiene un formato numérico de 4 dígitos., por defecto se envía los 4 siguientes números del código de barras tras el 061.

Es importante tener en cuenta que cada atributo tiene unas restricciones de tamaño y formato, que se aplican a los datos que se grabarán en la TUI, ya que afectan prácticamente a todas las universidades y no sólo a la UMA. Si se superarán los tamaños máximos especificados, los datos se truncarían a la hora de personalizar la tarjeta.

Existen algunos aspectos adicionales a tener en cuenta, a la hora de implementar la búsqueda de los datos: El método de búsqueda **searchUserById** es utilizado para controlar si un usuario puede o no solicitar una tarjeta inteligente. Para restringir la emisión para un usuario concreto, simplemente debe lanzar la excepción **UserNotAllowedException**. Las excepciones incluyen mensajes personalizados que indican como debe actuar el operador del Banco Santander. Estas son las razones más habituales por las cuales impedir que un usuario pueda solicitar otra TUI:

- El usuario ya tiene una tarjeta emitida y activada, registrada en la base de datos de la universidad, por lo que la universidad debería realizar un control de las tarjetas emitidas en sus sistemas.
- El usuario no pertenece al colectivo de emisión instantánea. En ocasiones las universidades deciden que, sólo algunos colectivos, pueden obtener su carné mediante el sistema de I+I (por ejemplo, alumnos de primer año).
- El alumno no cumple con los requisitos para solicitar el carné, aunque está registrado en la base de datos de la universidad. Por ejemplo, porque no completado la matrícula (pago, documentación, etc.).

Y en general, puede establecerse cualquier condición que la universidad desee. En la actualidad, la única restricción que se ha establecido en la UMA es que sólo pueden emitirse hasta un máximo de dos tarjetas por usuario y curso académico.

La excepción **UserNotFoundException** se lanza cuando un usuario buscado por su identificador no existe en la UMA. Este error no debe de producirse nunca, ya que para llegar a éste es necesario pasar siempre por la primera búsqueda general.

Para notificar errores graves inesperados se puede lanzar cualquier excepción (**Soap Fault**) sin que sea necesario especificar ningún tipo en concreto. El servicio de emisión capturará las excepciones inesperadas y mostrará un mensaje al operador del Banco Santander indicando que debe contactar con el Soporte Técnico.

A continuación se muestra un ejemplo de intercambio XML mediante SOAP en una solicitud de los datos de un usuario. El ejemplo utiliza el mismo mapa de datos que hemos visto anteriormente.

- Solicitud de los datos de un usuario existente y con derecho a tarjeta.

```

SOLICITUD DE LOS DATOS DE MARIA SANCHEZ PEREZ
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchUserById xmlns="http://external.issue.datio.es/">
      <userId>653287</userId>
    </searchUserById>
  </soap:Body>
</soap:Envelope>

RESPUESTA
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchUserByIdResponse xmlns="http://external.issue.datio.es/">
      <user>
        <content>
          <entry><key>nombre</key><value>MARIA SANCHEZ PEREZ</value></entry>
          <entry><key>linea4</key><value>ALUMNO</value></entry>
          <entry><key>codbar</key><value>0610123456</value></entry>
          <entry>
            <key>nia</key><value>120120120</value> </entry>
            <key>documento</key><value>12345678Z</value>
          </entry>
          <entry><key>categoria</key><value>01</value></entry>
          <entry><key>centro</key><value>000306</value></entry>
          <entry><key>estudios</key><value>Grado Ingeniería de Computadores</value></entry>
          <entry><key>curso</key><value>2015</value></entry>
          <entry><key>activa</key><value>1</value></entry>
          <entry><key>pin</key><value>0123</value></entry>
          <entry><key>foto</key><value></value></entry>
          <entry><key>reserva1</key><value>0123</value></entry>
          <entry><key>reserva4</key><value>1985-01-25</value></entry>
        </content>
        <document>12345678Z</document>
        <externalId>653287</externalId>
        <fullname>MARÍA SANCHEZ PEREZ</fullname>
      </user>
    </searchUserByIdResponse>
  </soap:Body>
</soap:Envelope>

```

En éste ejemplo de solicitud de datos podemos ver varias cosas:

El XML del ejemplo está codificado en UTF-8 pero podría estarlo en ISO 8859-1. Habitualmente las herramientas que manejan los servicios web y generan estos ficheros XML para el protocolo SOAP se encargan de gestionar automáticamente la codificación.

La búsqueda se hace con el ID recuperado en el ejemplo de searchUsers, 653287, que es un identificador interno de los sistemas de información de la UMA.

El nombre y apellidos de esta estudiante no superan los 26 caracteres. Si los superara, sería necesario que la UMA enviara el campo nombre con una reducción, porque si no el campo se trunca para la impresión. Lo que hacemos si supera los 26 caracteres, es si tiene dos nombres, enviar sólo el primero, pero si aun así, es mayor, pues sólo enviaríamos el primer nombre y el primer apellido. Todo para que a la hora de imprimir el carné, no salga con el nombre truncado.

Por ejemplo, si tenemos un nombre y apellidos largos como "ANTONIO JESUS GARCIA-FIZ FERNANDEZ", primero acortamos el nombre y quedaría "ANTONIO GARCIA-FIZ FERNANDEZ". Como sigue siendo demasiado largo,

eliminamos el segundo apellido y nos quedaría "ANTONIO GARCIA-FIZ" que se ajusta a las especificaciones.

- Solicitud de los datos de un usuario existente pero que no tiene derecho a tarjeta.

```

SOLICITUD DE LOS DATOS DE ISABEL PERDIDA SANCHEZ

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <searchUserById xmlns="http://external.issue.datio.es/">
      <userId>1541254</userId>
    </searchUserById>
  </soap:Body>
</soap:Envelope>

EL USUARIO YA DISPONE DE UNA TARJETA

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Client</faultcode>
      <faultstring>El usuario no puede solicitar una nueva tarjeta,
      porque ya dispone de una en vigor y este curso académico
      ya le han sido impresa dos tarjetas. Para solicitar un
      nuevo duplicado, indique al alumnos que debe comunicarlo
      al Servicio de Alumnos de la UMA.
    </faultstring>
  </soap:Fault>
</soap:Body>
</soap:Envelope>

```

Este es un ejemplo de respuesta con un error, y podemos ver cómo lanzar una excepción (en este caso el usuario no tiene derecho a tarjeta). Se podrían lanzar excepciones para muchos más casos de error, lo que realmente permite a la UMA disponer de un control absoluto y en tiempo real sobre el sistema de emisión de tarjetas.

Además el texto que es devuelto en la excepción en el campo faultstring es mostrado en la iiTablet del operador. Esto es muy útil para indicar al operador del Banco Santander cómo debe proceder e informar al alumno.

### 5.1.3. Envío de datos sobre tarjeta emitida - sendIssuedCard.

El sistema de emisión instantánea envía a través de un servicio web el feedback con los datos de cada una de las tarjetas emitidas. Realmente implementar éste servicio web no es obligatorio para el funcionamiento del sistema de emisión instantánea, pero si es necesario para que los sistemas de información de la UMA puedan controlar en tiempo real las tarjetas emitidas.

Este servicio es desarrollado y desplegado por la UMA, y es invocado por el sistema de emisión instantánea cada vez que se imprime una tarjeta nueva. Por tanto, la UMA dispone de la información en tiempo real, lo que incluye los números de serie y datos de control de las tarjetas emitidas. Esto es muy importante para el resto de servicios de valor añadido que tenga implementado la UMA donde se usen las tarjetas inteligente, como por ejemplo controles de acceso, para lo que necesita controlar el parque de tarjetas y a quién se han asignado en la UMA.

El envío de datos sobre tarjetas emitidas utiliza únicamente una función para notificar la información a la universidad, **sendIssuedCard**. Este método contiene dos parámetros de entrada:

- **CardData**: es el primero y contiene los datos de la tarjeta emitida. En éste caso se utiliza un objeto denominado CardData.
- **UserData**: es el segundo argumento y son los datos del usuario propietario de la tarjeta recién emitida, que deben ser utilizados para localizar en el sistema de la UMA al usuario propietario. Se utiliza la estructura UserData que ya conocemos del servicio web de consulta de usuarios.

```
/**
 * Procedimiento que se ejecuta por cada una de las tarjetas emitidas
 * El procedimiento incluye información de la tarjeta y también del usuario
 * propietario de la misma.
 *
 * @param card Datos de la tarjeta emitida y personalizada.
 * @param user Usuario propietario de la tarjeta.
 */
void sendIssuedCard(CardData card, UserData user);
```

Para recibir éste método, se implementa un servicio web que almacena los datos de la tarjeta recibida en los sistemas de información de la UMA, vinculando el registro creado de tarjeta con el usuario propietario del mismo. Además, los números de serie de la tarjeta se guardarán también en el iDUMA, de esta forma, los sistemas de información de la UMA pueden conocer fácilmente cual es la tarjeta activa y en vigor de cada miembro de la comunidad universitaria.

A la hora de implementar éste método tuve que tener en cuenta algunas consideraciones:

- La estructura con los datos del usuario propietario no se envía completa, sólo se adjuntan los datos básicos para poder localizar al usuario: Nombre y apellidos (**fullname**), Número de documento (**document**) e Identificador del usuario (**externalId**). Los datos utilizados en la personalización de la tarjeta no son enviados de retorno, para mejorar la eficiencia y el rendimiento, ya que son los mismos que proporcionó la universidad antes de la emisión.
- Los datos de la tarjeta emitida se almacenan en el objeto **CardData**. Este objeto tiene algunos atributos fijos, pero también dispone de un atributo



---

con formato de mapa (Clave–Valor) que se llama **content**. Este es el conjunto de datos que se envían en el objeto CardData cada vez que se emite una tarjeta mediante emisión instantánea:

- **id**: Identificador único numérico e incremental que da el sistema de emisión instantánea a la tarjeta. Sólo sirve para su gestión en el sistema de emisión instantánea, así que este dato se puede ignorar.
- **requestDate**: Fecha / hora en la cual el usuario solicitó la tarjeta en el punto de emisión.
- **issueDate**: Fecha / hora en la cual la tarjeta se termina de imprimir y personalizar, y se entrega al usuario.
- **serialNumber**: Número de serie del chip con contactos de la tarjeta. Es el número de serie más importante de la tarjeta (de todos los que tiene) y por eso se incluye en éste atributo fijo.
- **content**: Mapa con el resto de datos de la tarjeta emitida. Tiene una estructura de clave-valor, por tanto los atributos de la tarjeta no están ordenados. Los datos que contiene la estructura son los siguientes.
  - **wg10**: Número de serie del chip con contactos (chip WG10) de la tarjeta emitida. El número de serie está formado por 16 caracteres alfa numéricos. Es un dato que se envía siempre y que permite identificar unívocamente cualquier tarjeta, ya se introduzca en un lector de contactos o en un lector de proximidad.
  - **mifare**: Número de serie del chip de proximidad (chip MiFare). Está formado por 8 caracteres alfanuméricos, y permite identificar unívocamente la tarjeta, pero sólo cuando se utiliza en un lector de proximidad.
  - **pan**: Número de tarjeta asignado por el Banco Santander en el proceso de emisión. Está formado por un número de 16 dígitos, el cual también se imprime en la superficie de la tarjeta. Este número también es único y permite identificar la tarjeta visualmente. Además puede ser recuperado mediante un lector de contactos o un lector de proximidad.
  - **foto**: Fotografía del usuario capturada con la iiTablet, o bien enviada originalmente por la UMA como dato de personalización. Es la foto que ha sido utilizada para su impresión en la superficie de la tarjeta. Está en formato JPEG y codificada en Base64. Por tanto este atributo contiene una cadena de texto de bastante tamaño.
  - **pin**: En caso la UMA asigna un pin inicial de cuatro dígitos a toda tarjeta impresa en de la UMA
- Para notificar errores graves inesperados se lanza cualquier excepción (**Soap Fault**) sin que sea necesario especificar ningún tipo en concreto. El

servicio de emisión capturarán las excepciones inesperadas y las registrará para enviar una alarma al departamento de Soporte Técnico.

A continuación se muestra un ejemplo de intercambio XML mediante SOAP en la notificación de una nueva tarjeta emitida:

- Envío de los datos de una tarjeta emitida.

```
TARJETA EMITIDA A MARIA SANCHEZ PEREZ

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <sendIssuedCard xmlns="http://external.issue.datio.es/">
      <card>
        <content>
          <entry><key>mifare</key><value>FCBDB10B</value></entry>
          <entry><key>pan</key><value>5901000415149888</value></entry>
          <entry><key>wg10</key><value>30600000C7EEEB80</value></entry>
          <entry><key>pin</key><value>7913</value></entry>
          <entry><key>foto</key> <value>/9j/4AAQSkZJRgABAQAAQABAAD/2wBDAAQE...</value></entry>
        </content>
        <issueDate>2016-09-15T11:20:15+01:00</issueDate>
        <requestDate>2016-09-15T11:21:26+01:00</requestDate>
        <serialNumber>30600000C7EEEB80</serialNumber>
      </card>
      <user>
        <id>374</id>
        <document>12345678z</document>
        <externalId>53799</externalId>
        <fullname>MARIA SANCHEZ PEREZ</fullname>
      </user>
    </sendIssuedCard>
  </soap:Body>
</soap:Envelope>

RESPUESTA, SI TODO HA IDO BIEN DEBE SER VACIA

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <sendIssuedCardResponse xmlns="http://external.issue.datio.es/">
      </sendIssuedCardResponse>
    </soap:Body>
</soap:Envelope>
```

El proceso de envío de tarjetas se hace en segundo plano, y nunca se notifican errores a los usuarios o al operador del Banco Santander. En caso de producirse algún error cuando la UMA incorpora los datos de la tarjeta en su sistema, y si lanza una excepción para indicar que hubo un problema, esta excepción nunca se propaga a los usuarios finales.

El error es registrado en el sistema de emisión instantánea del Banco Santander y desde el soporte técnico informarían a la UMA de que se ha producido un error para que se solucione. Además, el sistema de gestión de emisión instantánea permite volver a enviar a la universidad, a través de este servicio web, los datos emitidos de una o un conjunto de tarjetas cualesquiera. Con todas estas medidas se garantiza que no se producirá ninguna pérdida de datos.



---

### 5.1.4. Objetos para el intercambio de datos.

A continuación paso a describir, en lenguaje Java, cómo son los objetos que he debido manejar cuando trabaje con los servicios web que se han desarrollado.

- **UserData:** Objeto para almacenar información del usuario. Los datos del usuario se encapsulan en un objeto UserData que tiene la siguiente estructura.

```
public class UserData
{
    /**
     * Identificador unívoco del usuario en el sistema de la universidad.
     */
    private String externalId;

    /**
     * Nombre y apellidos del usuario.
     */
    private String fullname;

    /**
     * Documento completo e identificativo del usuario.
     */
    private String document;

    /**
     * Mapa de datos del usuario utilizados para la personalización.
     */
    private Map<String, String> content;
}
```

- **CardData:** Objeto para almacenar información de la tarjeta. Los datos de la tarjeta se encapsulan en un objeto CardData que tiene la siguiente estructura.

```
public class CardData
{
    /**
     * Identificador interno en Emisión Instantánea, el dato puede ser ignorado.
     */
    private long id;

    /**
     * Fecha / hora de generación de la petición.
     */
    private Date requestDate;

    /**
     * Fecha / hora de emisión y creación de la tarjeta.
     */
    private Date issueDate;

    /**
     * Número de serie de la tarjeta generada (WG10).
     */
    private String serialNumber;

    /**
     * Mapa de datos resultantes de la emisión (Foto, Mifare, PIN, etc.).
     */
    private Map<String, String> content;
}
```

### 5.1.5. Seguridad y despliegue de los Servicios Web.

Los servicios web desarrollados por la UMA para este proyecto deben desplegar, tanto el endpoint (la implementación) que contiene el servicio, como el WSDL que lo describe, manteniendo el nombre del esquema de los servicios web debe mantenerse como, ya que si hubieran sido cambiados, los clientes de los servicios web del sistema de emisión instantánea no funcionarían, pues están compilados para utilizar éste esquema.

Se incluyó seguridad en el acceso a los servicios web para que los servicios no puedan ser consultados o accedidos por ningún tercero, solo por la APP instalada en las iiTablet del sistema de Emisión Instantánea del Banco Santander a través del iiServer del sistema. Para ello se aplicó las siguientes medidas de seguridad:

- Publicación HTTPS: Los servicios web fueron ser publicados mediante HTTPS. De ésta forma se garantiza que los datos viajen encriptados.
- Restricción mediante IP: se incluyó una regla a los firewall de la UMA para que los servicios web sean restringidos en acceso a una sola DNS creada para dicho propósito para el proyecto. Los servicios web de integración son consultados de forma centralizada únicamente por el servidor central de emisión instantánea llamado iiServer. Por tanto, se puede securizar y

---

restringir esa conexión de forma muy sencilla con ésta técnica, sin necesidad de medidas adicionales.

### 5.1.6. Tratamiento de errores y excepciones.

Otro aspecto importante en los servicios web es la posibilidad de utilizar excepciones, principalmente en el servicio web de consulta de usuarios. Cuando el sistema de emisión instantánea recibe una excepción, intenta extraer el mensaje de la misma, y lo muestra directamente en la iiTablet.

De ésta forma, si un operador del Banco Santander realiza una búsqueda (ya sea el primer filtrado, o la búsqueda de un usuario concreto) y se produce una excepción, se le mostrará un mensaje indicando el error, y muy importante, en el mensaje hay que darle instrucciones de cómo debe proceder.

Para el envío de excepciones utilizaremos las estructuras de excepciones definidas en el interfaz. El mensaje que se muestra en la iiTablet se corresponde con el atributo “message” de estas dos excepciones.

- **UserNotAllowedException** se utiliza para indicar que el usuario no tiene derecho a tarjeta, por la razón que sea oportuna (no está en el colectivo adecuado, tiene demasiadas tarjetas emitidas, etc.). Esta es la excepción controlada que con más frecuencia se utilizan los servicios web desarrollados en el proyecto. La excepción tiene un atributo llamado “message” que permite especificar un texto, el cual será transmitido a la iiTablet para su visualización.
- **UserNotFoundException** se utiliza cuando el usuario no es localizado en el sistema de información de la UMA.
- **Errores inesperados**: en lenguajes como Java o .NET podemos lanzar cualquier tipo de excepción para indicar un error inesperado. Por ejemplo, excepciones de caída de la base de datos de la UMA, un error interno de programación, etc. Las librerías que permiten utilizar los servicios web se encargan automáticamente de transformar esa excepción en una **Soap Fault**, que será enviada al servicio de emisión.

Ejemplo de código en lenguaje Java para error controlado:

```

public es.datio.issue.external.UserData searchUserById(java.lang.String userId)
    throws es.datio.issue.external.UserNotAllowedException,
           es.datio.issue.external.UserNotFoundException {
    UserData userData = new UserData();
    UserData.Content content = new UserData.Content();
    try {
        // Conexión a base de datos
        baseDatos.getInstance().conectar();
        Connection conn = baseDatos.getInstance().getConexion();
        conn.setAutoCommit(false);
        Statement stmt = (Statement)conn.createStatement();
        int cursoAcad = util.dameCursoAcademico();
        if (util.tarjetaActiva(stmt, userId, cursoAcad)) {
            // error ya tiene 2 tarjetas activas.
            throw new UserNotAllowedException("El usuario no puede solicitar una nueva
            tarjeta, porque ya dispone de una en vigor que ha sido emitida dos veces.
            Para solicitar un duplicado debe enviar una instancia a la Sección de Alumnos.");
        }
    } else {

```

Es importante tener en cuenta que en la iiTablet se mostrará el mismo texto que contiene la excepción. Por tanto, es importante que el mensaje de error sea “orientado al usuario de la aplicación”. Es decir, adaptados al operador del Banco Santander encargado de manejar la iiTablet y emitir tarjetas a los usuarios de la universidad.

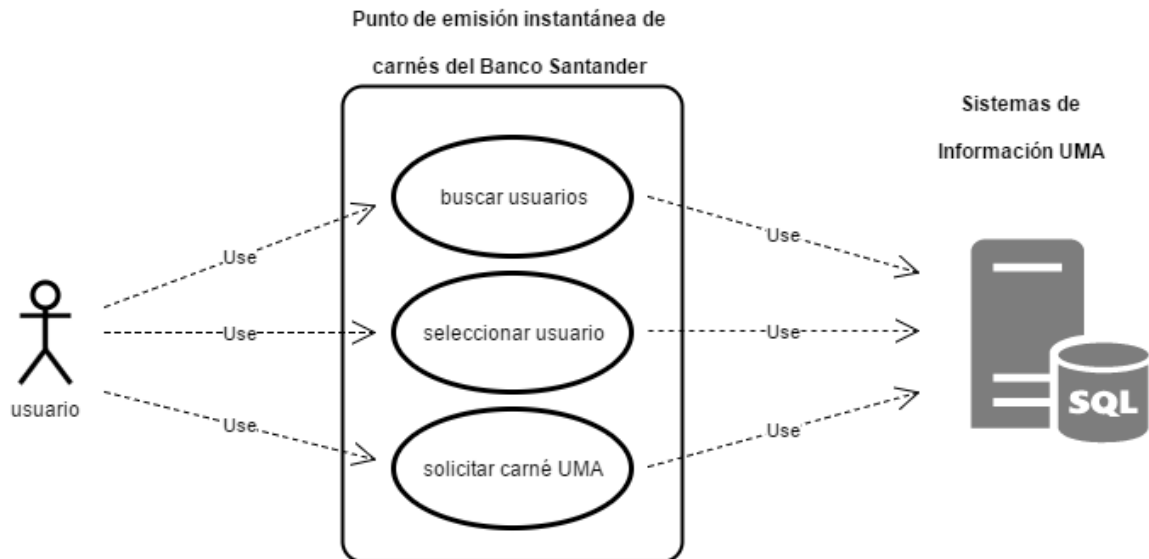
## 5.2. Análisis de diferentes modelos.

### 5.2.1. Casos de Uso.

En el presente capítulo se describen los diagramas de los principales casos de uso del sistema. Los mismos muestran la relación entre los actores y los casos de uso descritos.

Se identificaron dos actores principales:

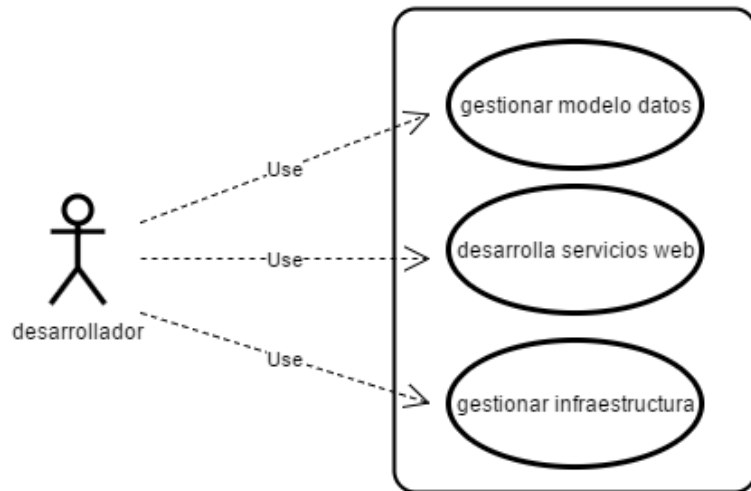
- **Usuario:** Actor genérico que interactúa con el sistema para obtener un resultado concreto. En este caso específico, interactúa a través de las iiTablet a través de la APP del sistema de Emisión Instantánea de carnés.
- **Desarrollador:** Desarrolla los servicios web y gestiona la infraestructura necesaria para que la integración de los sistemas de información de la UMA y el Banco Santander sean todo un éxito y no hay incidencias de mal funcionamiento durante la puesta en producción del proyecto.



<b>Caso de uso</b>	Buscar usuarios
<b>Actor</b>	usuario iiTablet
<b>Propósito</b>	Obtener listado de usuarios en el sistema de información la UMA. Es utilizado para localizar el usuario concreto al cual queremos emitir una tarjeta

<b>Caso de uso</b>	Seleccionar usuario
<b>Actor</b>	usuario iiTablet
<b>Propósito</b>	Obtener los datos para poder emitir y personalizar un carné de la UMA a un usuario concreto, que serán utilizados en la personalización y grabación de su tarjeta inteligente.

<b>Caso de uso</b>	solicitar carné UMA
<b>Actor</b>	usuario iiTablet
<b>Propósito</b>	Tras confirmar datos y foto, ya sea la que tenga la UMA o una nueva que se tome ad hoc, se manda a imprimir el carné UMA del usuario, tras lo cual, se notifica a la UMA los datos del carné que ha sido impreso al usuario.

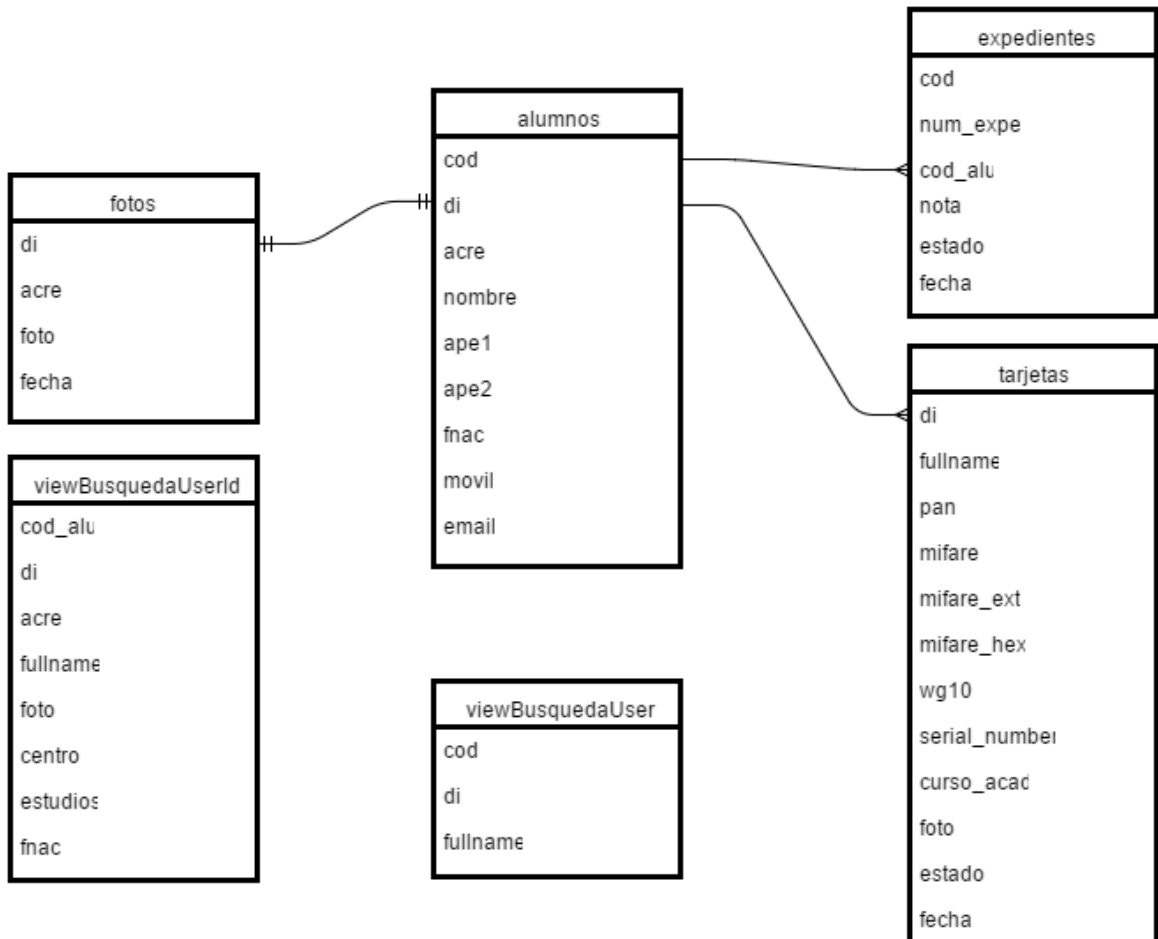


<b>Caso de uso</b>	Gestionar modelo de datos
<b>Actor</b>	Desarrollador
<b>Propósito</b>	Crear las tablas y vistas necesarias en el sistema de base de datos de la UMA, para atender a los servicios web del sistema desarrollado, así como su posterior utilización en otros servicios de valor añadido de la UMA.

<b>Caso de uso</b>	Desarrollar servicios web
<b>Actor</b>	Desarrollador
<b>Propósito</b>	Desarrollar los servicios web SOAP de la UMA que deben integrarse en el sistema de emisión instantánea del Banco Santander

<b>Caso de uso</b>	Gestionar infraestructura
<b>Actor</b>	Desarrollador
<b>Propósito</b>	Crear el servidor iiServer así como solicitar los ajustes de red necesarios para el correcto funcionamiento del sistema de emisión instantánea dentro de la red informática de la UMA y su sistema de emisión instantánea.

### 5.2.2. Diagrama de base de datos.



### 5.2.2.1. Tabla Alumnos.

Almacena los datos de los alumnos del sistema de base de datos de la UMA. Tabla ya existente en la base de datos Oracle de ordenación académica de los sistemas de información de la UMA.

<b>cod</b>	int	identificador único autoincremental
<b>di</b>	varchar	documento identificador: DNI, NIE, pasaporte, etc
<b>acre</b>	varchar	código de barras universitario: 061123121X
<b>nombre</b>	varchar	nombre del alumno
<b>ape1</b>	varchar	primer apellido del alumno
<b>ape2</b>	varchar	segundo apellido del alumno
<b>fnac</b>	date	fecha de nacimiento
<b>movil</b>	varchar	móvil del alumno
<b>email</b>	varchar	correo electrónico

### 5.2.2.2. Tabla Expedientes.

Almacena los datos de todos los expedientes de los alumnos del sistema de base de datos de la UMA. Tabla ya existente en la base de datos Oracle de ordenación académica de los sistemas de información de la UMA.

<b>cod</b>	number	identificador único autoincremental
<b>num_exp</b>	number	número de expediente único
<b>cod_alu</b>	number	código único del alumno del expediente
<b>nota</b>	number	nota del expediente
<b>fecha</b>	date	fecha de estado del expediente
<b>estado</b>	varchar	estado del expediente



### 5.2.2.3. Tabla Fotos.

Almacena las fotos de todos los usuarios de la UMA. Sólo tiene una foto por usuario, asegurando que es la última foto aportada por el usuario. Tabla ya existente en la base de datos Oracle de ordenación académica de los sistemas de información de la UMA.

<b>di</b>	number	documento identificador: DNI, NIE, pasaporte, etc
<b>acre</b>	number	código de barras universitario: 061123121X
<b>foto</b>	blob	foto binaria en formato jpg del usuario
<b>fecha</b>	date	fecha de la foto

### 5.2.2.4. Vista BusquedaUser.

Vista utilizada para localizar a los usuarios de la UMA, ya sea filtrando por documento identificador o nombre del usuario.

<b>cod</b>	number	identificador único autoincremental
<b>di</b>	varchar	documento identificador: DNI, NIE, pasaporte, etc
<b>fullname</b>	varchar	nombre del alumno

### 5.2.2.5. Vista BusquedaUserId.

Vista utilizada para proporcionar al sistema los datos de un usuario concreto al que se le va a hacer el carné.

<b>cod</b>	number	identificador único autoincremental
<b>di</b>	varchar	documento identificador: DNI, NIE, pasaporte, etc
<b>acre</b>	varchar	código de barras universitario: 061123121X
<b>fullname</b>	varchar	nombre completo del alumno
<b>foto</b>	varchar	última foto binaria en formato jpg del usuario que tiene registrada en la UMA
<b>centro</b>	varchar	centro del usuario
<b>estudios</b>	varchar	estudios que está cursando el alumno

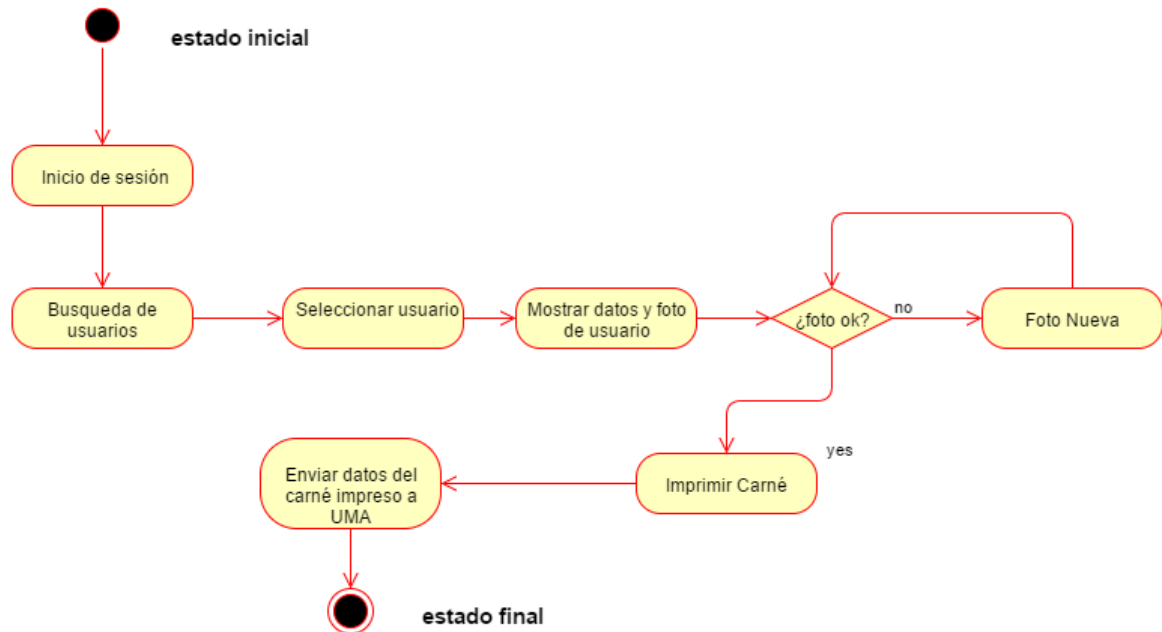
### 5.2.2.6. Tabla Tarjetas.

Almacena los datos carnés que hayan sido impresos para cada usuario, conservando la foto con la que fue impreso el carné. Tabla ya existente en la base de datos Oracle de ordenación académica de los sistemas de información de la UMA, pero a la que hay que añadir nuevos campos, ya que los datos que nos son facilitados a través de la plataforma Nexusc para la gestión de los datos de los carnés universitarios, no es tan completo como el nuevo sistema planteado.

<b>di</b>	number	documento identificador: DNI, NIE, pasaporte, etc
<b>fullname</b>	number	nombre completo del alumno
<b>pan</b>	number	Número de tarjeta asignado por el Banco Santander en el proceso de emisión
<b>mifare</b>	varchar	Número de serie del chip de proximidad (chip MiFare). Está formado por 8 caracteres alfanuméricos.
<b>mifare_ext</b>	varchar	Número de serie del chip de proximidad extendido (chip MiFare). Está formado por 16 caracteres alfanuméricos.
<b>mifare_hex</b>	varchar	mifare en hexadecimal
<b>wg10</b>	number	Número de serie del chip con contactos (chip WG10) de la tarjeta emitida.
<b>serial_number</b>	number	Número de serie del chip con contactos de la tarjeta.
<b>curso_acad</b>	varchar	Curso académico en el que se fabricó el carné
<b>foto</b>	blob	foto binaria en formato jpg del usuario con la que se hizo el carné
<b>estado</b>	varchar	estado de la tarjeta: activa o inactiva
<b>fecha</b>	date	fecha de impresión del carné
<b>fnac</b>	date	fecha de nacimiento del usuario

### 5.2.3. Diagrama de estados.

El siguiente diagrama de estado muestra la secuencia de estados que se van sucediendo en el sistema de emisión instantánea de carnés.

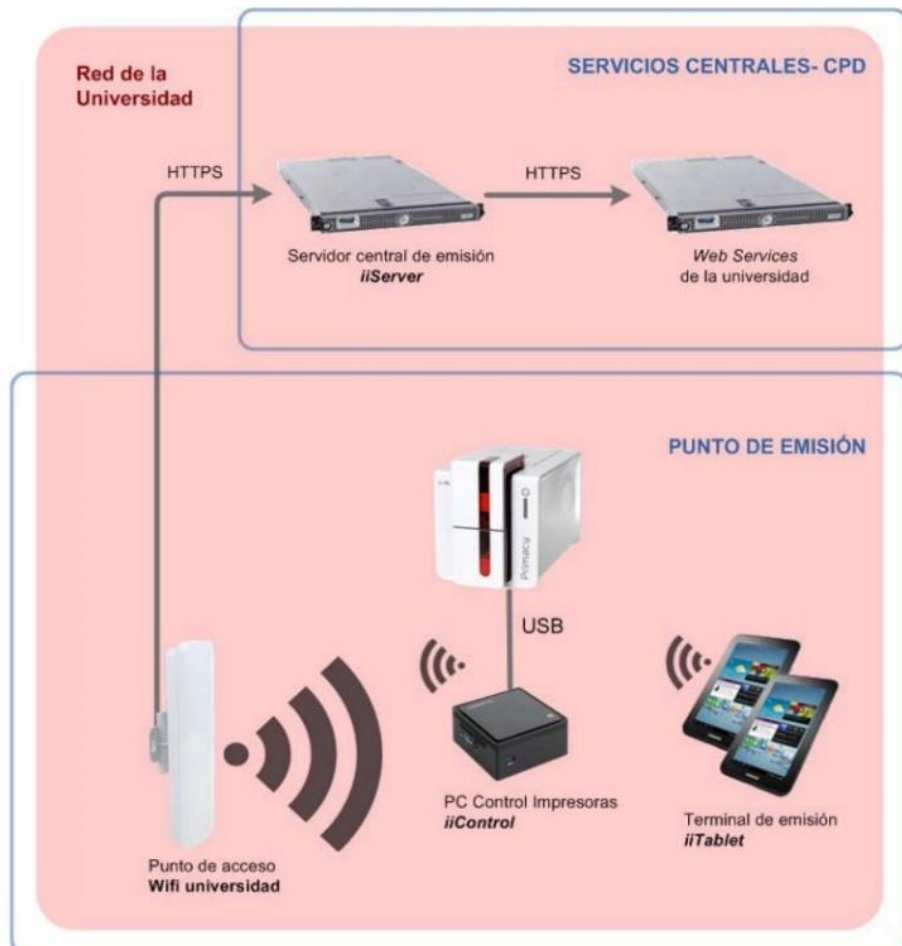


### 5.3. Análisis de requisitos de infraestructuras.

Para llevar a cabo la interconexión de ambos sistemas de información, se debe habilitar la siguiente infraestructura:

- **Servidor central de emisión iiServer:** Es un conjunto de servicios centralizados que permiten el funcionamiento de la emisión instantánea del carnet universitario.
  - Los servicios centrales de iiServer se encargan de controlar las impresoras de tarjetas, tablets y en general, todos los componentes que se despliegan en los puntos de emisión de la tarjeta universitaria.
  - Además, los servicios centrales iiServer son los únicos que accederán a los servicios web desplegados por la UMA para la emisión de la tarjeta universitaria.
  - El servidor central iiServer se debe instalar en un equipo dentro de la red de datos de la UMA, siendo una máquina virtual proporcionada por la UMA.
- **Punto de emisión:** Cada uno de los lugares habilitados dentro de la UMA para la emisión del carnet universitario. Los puntos de emisión deben estar conectados a la red UMA. En cada punto de emisión se desplegarán los siguientes elementos hardware:
  - **Impresora de tarjetas:** encargada de personalizar el carnet UMA.

- **iiControl:** Es un PC que contiene un software que controlar la impresora. Además debe tener conectividad con los sistemas de información de la UMA.
- **iiTablet:** Son cada una de las tablets manejadas por los operadores del Banco Santander. Las tablets disponen de una app que permite comprobar si un alumno tiene derecho a su carnet universitario, y si es así, emitirlo con la impresora de tarjetas. Dicha app es la que consume los servicios web desarrollados por la UMA. Además debe tener conectividad con los sistemas de información de la UMA.
- **Router Wifi:** Cada punto de emisión tiene instalado un router Wifi que crea una red Wifi local privada a la cual se conectan todos los elementos del punto de emisión: impresora de tarjetas, iiControl e iiTablet. Después el router Wifi debe ser conectado a la red de la UMA, pues todos los elementos del punto de emisión necesitan tener conectividad con los sistemas de información de la UMA, accediendo a ellos a través de la Wifi local privada proporcionada por el router wifi.



Esquema de conexión

### **5.3.1. Análisis de requisitos del servidor iiServer.**

#### **5.3.1.1. Características iiServer**

El servidor iiServer deberá ser una máquina virtual suministrada por la UMA, con los siguientes requisitos mínimos.

- Sistema operativo:
  - Windows 2003 Server.
  - Windows 2008 Server (32 o 64 bits).
  - Windows 2012 Server (32 o 64 bits).
- Memoria:
  - 2 GB RAM para Windows 2003 o 2008.
  - 4 GB RAM para Windows 2012
- Disco duro:
  - 50GB de espacio en disco.

#### **5.3.1.2. Comunicaciones iiServer**

Los requisitos generales del servidor iiServer son:

- El servidor debe estar dentro de la red de datos de la UMA.
- Estará conectado permanentemente a la red.
- Deberá tener nombre DNS asignado.

Con respecto a las comunicaciones entrantes, el iiServer deberá:

- HTTPS 443: Comunicaciones con los puntos de emisión en la red UMA.
- HTTPS 443 desde la IP 79.14.247.239: Este puerto del servidor debe estar sólo habilitando el acceso a la IP especificada. Esto es necesario para que el servicio de soporte técnico del Banco Santander pueda acceder a la consola web de gestión y administración del sistema de emisión del servidor iiServer.

Con respecto a las comunicaciones salientes, el iiServer deberá:

- Acceso a los servicios web de emisión instantánea desarrollados en la UMA.
- HTTPS 443 hacia la IP 79.148.247.239: Se envía información de las tarjetas emitidas y disponibles en los puntos de emisión para la gestión del stock de tarjetas.

---

### 5.3.1.3. Aplicaciones iiServer

El servidor central iiServer contiene las siguientes aplicaciones:

- Base de datos MySQL 5.1.
  - Base de datos para controlar los carnés emitidos.
- Servidor web Apache Tomcat 7.
  - Servicios web de emisión instantánea.
  - Servicios de integración con la UMA.
  - Consola web de gestión y administración.
  - Servicio de custodio de claves.

### 5.3.1.4. Mantenimiento de iiServer

- Acceso a iiServer: El soporte técnico del Banco Santander necesita disponer de un acceso permanente al servidor que contiene los servicios web desarrollado, para poder solucionar posibles incidencias (caída del servicio, actualizaciones de software, etc.). Dicha tarea puede realizarse con la herramienta de soporte remoto TeamViewer, pero en su defecto, sería suficiente con disponer de una conexión mediante VPN.
- Copias de seguridad: Se debe hacer una copia diaria nocturna de la base de datos de emisión instantánea instalada en iiServer.
- Mantenimiento del servidor: desde el soporte técnico del Banco Santander, se realizará todo el mantenimiento habitual del iiServer

### 5.3.2. Análisis de requisitos de los puntos de emisión

Los puntos de emisión deben ser espacios habilitados dentro de la UMA para realizar la emisión instantánea de la tarjeta universitaria. Deben tener las siguientes características:

- Debe existir al menos un punto de conexión eléctrica 230V.
- Debe existir un punto de conexión Ethernet a la red UMA o en su defecto, tener la posibilidad de utilizar la red wifi de la UMA con una buena señal.
- Todas las impresoras deben conectarse por USB al equipo que las controla. Por tanto, deben estar cerca de dicho equipo, de forma numerada.
- Debe ser un espacio controlado, para evitar la sustracción del material.
- Entorno adecuado para que la captura de la fotografía sea óptima.
  - El espacio debe estar iluminado.
  - Zona con fondo blanco para la toma de fotografías.
- Debe disponer de espacio y mobiliario para alojar los elementos que componen el punto de emisión.

### 5.3.2.1. Requisitos de red y comunicaciones.

- El soporte técnico del Banco Santander instalará y configurará su propia red local Wifi privada para cada punto de emisión. Las tablets de captura de datos iiTablet y el equipo de control de la impresora iiControl estarán permanentemente conectados a esta red.
- Si no fuera posible utilizar ésta primera opción, que es la recomendada, las tablets y el equipo de control pueden conectarse a la red Wifi de la UMA que disponga de una buena señal.
- En cualquiera de los dos casos anteriores, debe existir conectividad desde el punto de emisión (tablets y equipos de control) hacia los servicios centrales iiServer.
- Los elementos conectados a la red Wifi (tablets y equipos de control) funcionan utilizando DHCP y por tanto no necesitan disponer de una IP fija.
- Comunicaciones salientes de iiControl.
  - HTTPS 443: Comunicación con los servicios centrales de emisión iiServer.
  - Conexión saliente a Internet: Necesario para que el servicio de soporte técnico del Banco Santander pueda gestionar remotamente el PC de control mediante la herramienta TeamViewer.
- Comunicaciones salientes de iiTablet.
  - HTTPS 443: Comunicación con iiServer
  - Conexión saliente a Internet: Necesario para que el servicio de soporte técnico del Banco Santander pueda gestionar remotamente la iiTablet mediante la herramienta TeamViewer.

### 5.3.2.2. Equipamiento del punto de emisión.

El equipamiento del punto de emisión de carnés, que será suministrado por el soporte técnico del Banco de Santander, debe ser:

- Equipo iiControl para control de las impresoras.
  - Sistema operativo Windows 7/8.
  - Interfaz de red WIFI.
  - CPU Dual Core, 2GB memoria y 250 GB disco duro.
- Router / Punto de acceso Wifi.
- Impresora/s de tarjetas Evolis Primacy Duplex RFID.
  - Dispone de un lector/grabador contactless.
  - Conexión USB al equipo de control.
- iiTablets: Terminales de captura de datos, Samsung Galaxy TAB, o similares.

- Incluye cargador y cable de datos USB.
- Fundaprotectora de silicona.
- Funda de transporte semi-rígida.
- Otros materiales.
  - Cable de conexión Ethernet RJ 5.
  - Regletas eléctricas.



## 6 Plataformas tecnológicas utilizadas en el desarrollo.

Tras valorar los posibles IDE con los que realizar los servicios Web SOAP, por su funcionalidad y facilidad de uso, me decanté por el IDE de NetBeans para desarrollar el código de las diferentes funcionalidades de los servicios Web del proyecto.

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE2 es un producto libre y gratuito sin restricciones de uso.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

Modularidad. Todas las funciones del IDE son provistas por módulos. Cada módulo provee una función bien definida, tales como el soporte de Java, edición, o soporte para el sistema de control de versiones. NetBeans contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiéndole al usuario comenzar a trabajar inmediatamente.

Algunas de las características del IDE de NetBeans son:

- Gestión de la interfaz de usuario ( menús y barras de herramientas )
- Gestión de configuración de usuario
- Gestión de almacenamiento (guardar o cargar algún tipo de dato)
- Gestión de ventana
- Marco Asistente (soporta diálogos para a paso)
- Librería visual de NetBeans
- Herramientas de desarrollo integrado
- NetBeans IDE es libre, código abierto, multiplataforma con soporte integrado para el lenguaje de programación Java.

Para la gestión de la base de datos Oracle con la que vamos a trabajar en el desarrollo de los servicios Web SOAP para el sistema de Emisión Instantánea de carnés de la UMA, vamos a hacer uso de la aplicación TOAD.

TOAD, Tool for Oracle Application Developers, es una aplicación informática de desarrollo SQL y administración de base de datos, considerada una herramienta útil para los Oracle DBAs (administradores de base de datos). Actualmente está

disponible para las siguientes bases de datos: Oracle Database, Microsoft SQL Server, IBM DB2, y MySQL.

## 6.1. Desarrollo servicios Web SOAP con NetBeans.

Lo primero fue instalar el IDE elegido, que lo descargué desde la web oficial del IDE ([netbeans.org/downloads/](http://netbeans.org/downloads/)), decantándome por la versión completa, para así tener que evitar la posterior descarga de módulos adicionales que pudieran ser necesarios.

NetBeans IDE 8.2 Download

8.1 | 8.2 | Development | Archive

Email address (optional):

Subscribe to newsletters:  Monthly  Weekly

NetBeans can contact me at this address

IDE Language: Español Platform: Windows

Note: Greyed out technologies are not supported for this platform.

**NetBeans IDE Download Bundles in community contributed languages<sup>1</sup>**

Supported technologies *	Java SE	Java EE	HTML5/JavaScript	PHP	C/C++	All
NetBeans Platform SDK	•	•				•
Java SE	•	•				•
Java FX	•	•				•
Java EE		•				•
Java ME						•
HTML5/JavaScript		•	•	•		•
PHP			•	•		•
C/C++					•	•
Groovy						•
Java Card™ 3 Connected						•
Bundled servers						
GlassFish Server Open Source Edition 4.1.1		•				•
Apache Tomcat 8.0.27		•				•

Download Download Download x86 Download x86 Download x86 Download

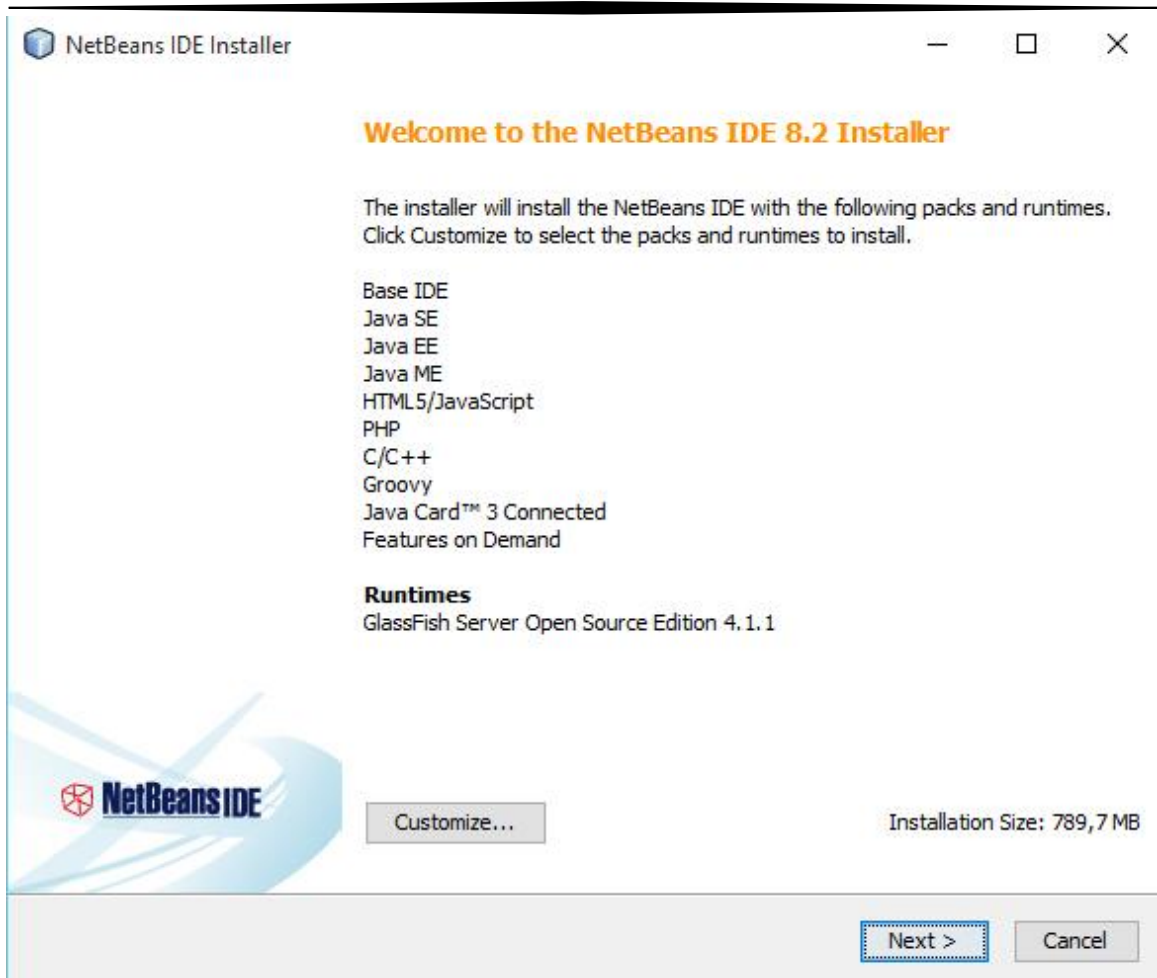
Download x64 Download x64 Download x64

Free, 100 MB Free, 200 MB Free, 111 - 115 MB Free, 111 - 115 MB Free, 109 - 112 MB Free, 222 MB

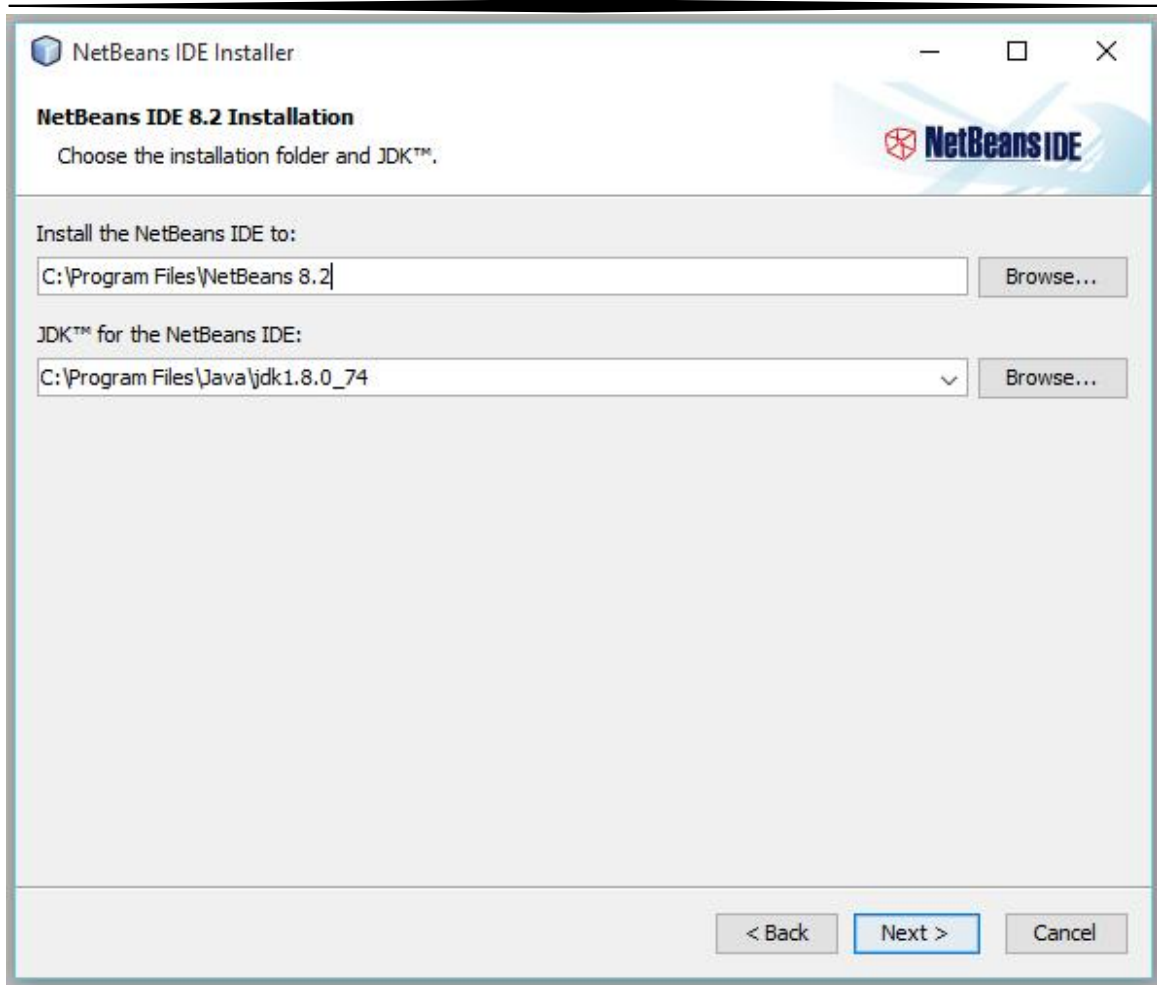
\* You can add or remove packs later using the IDE's Plugin Manager (Tools | Plugins)

Important Legal Information:

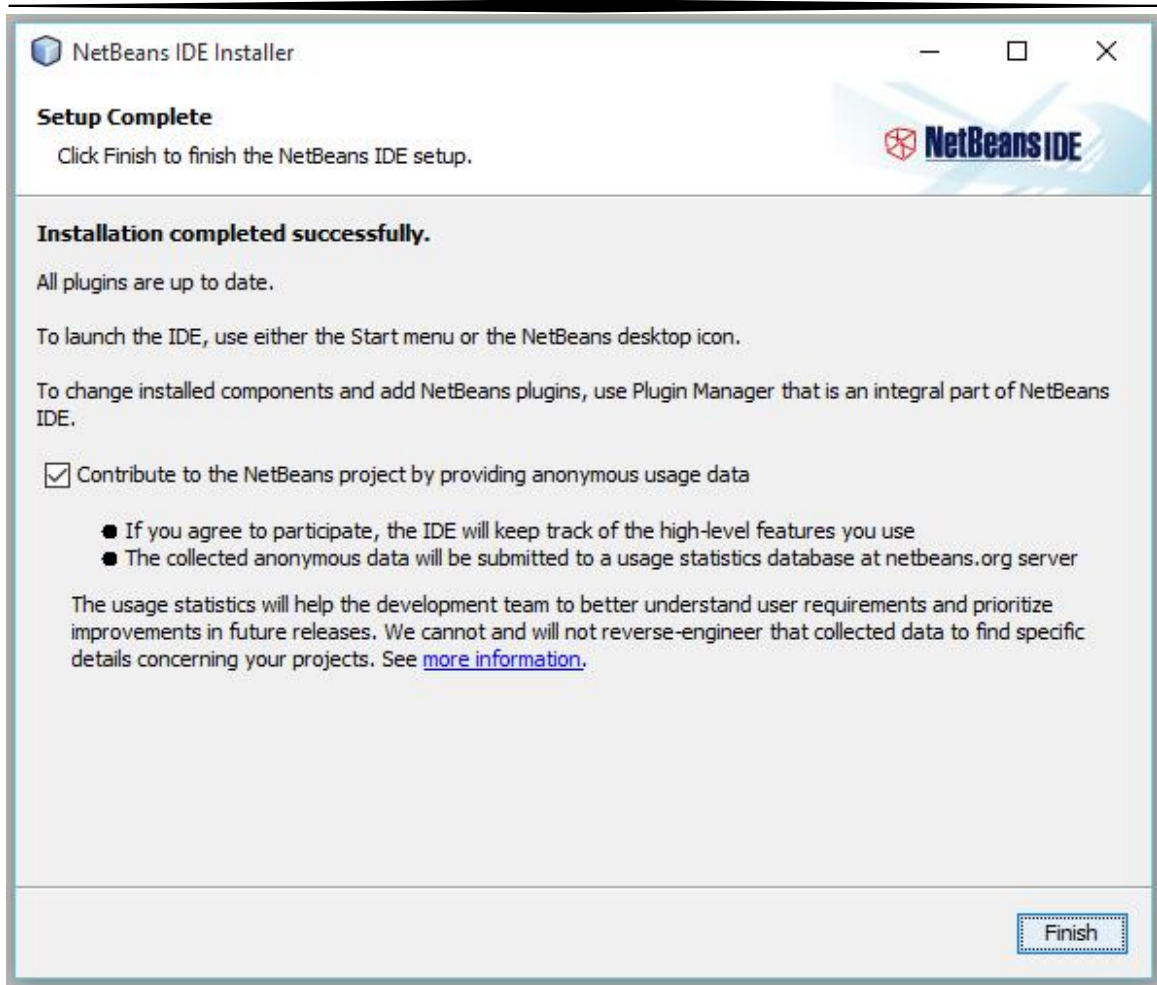
Una vez descargado el fichero de instalación, se procede a su ejecución hasta completar la instalación completa del IDE.



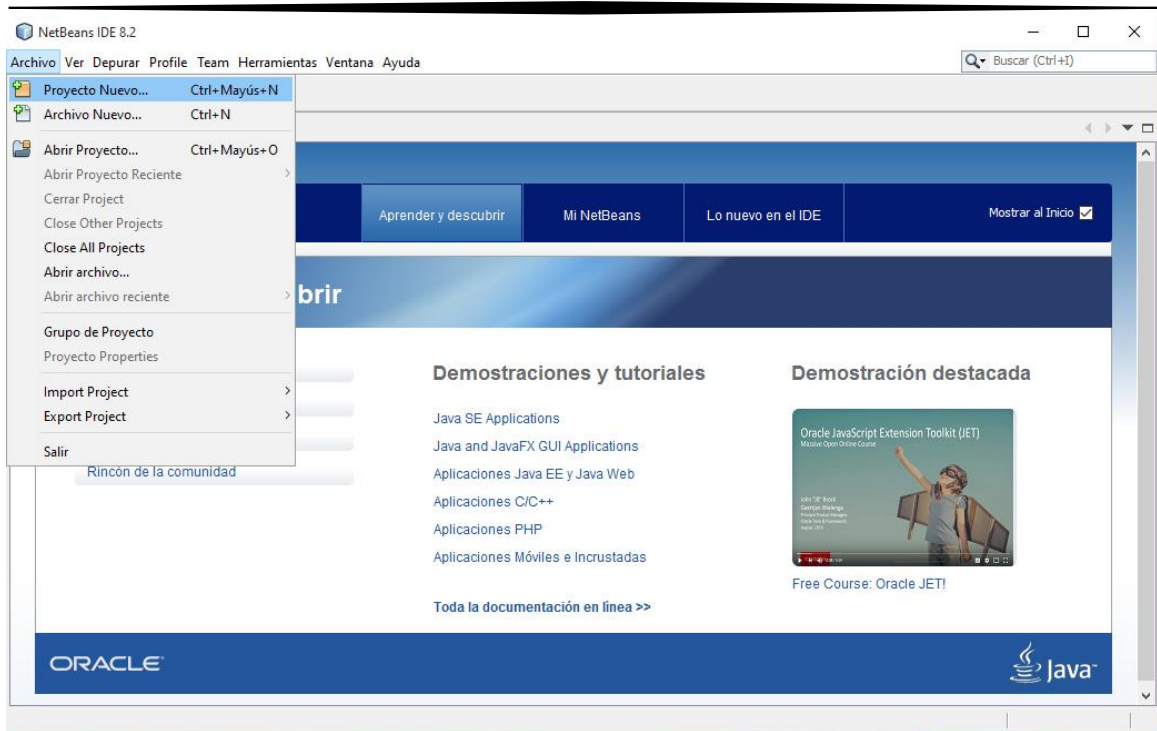
Inicio de la instalación



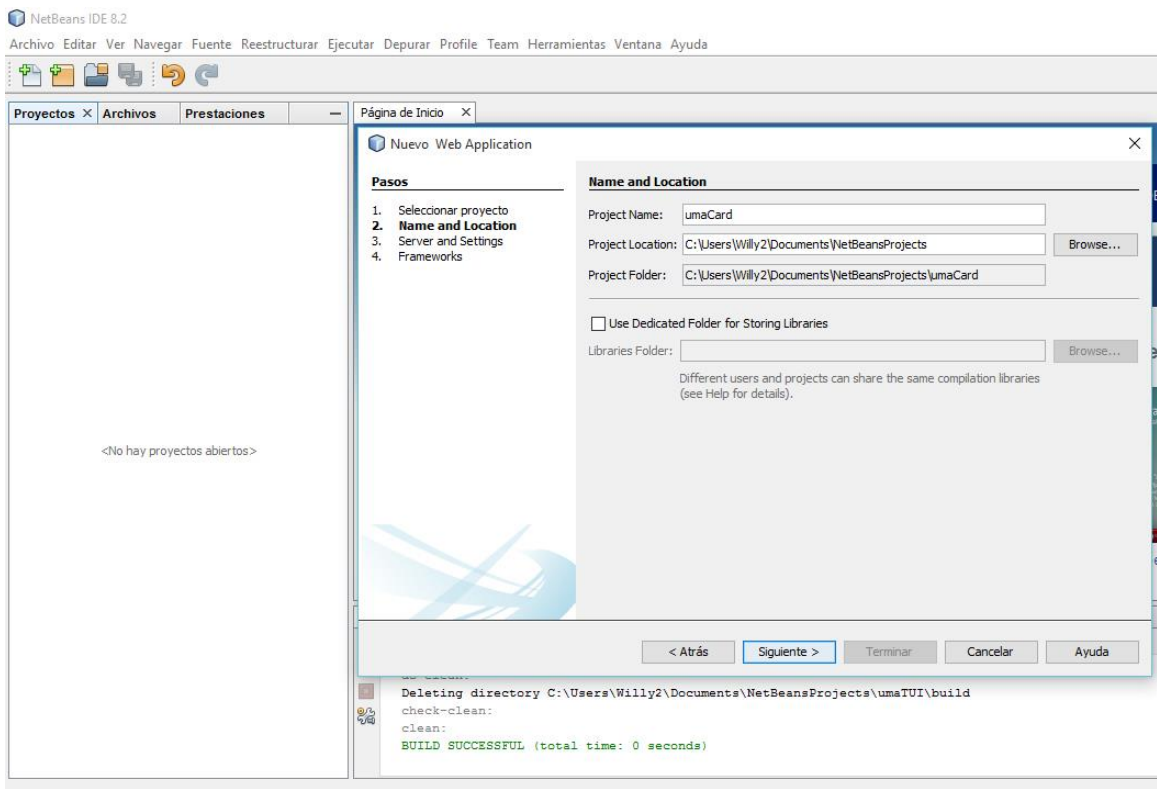
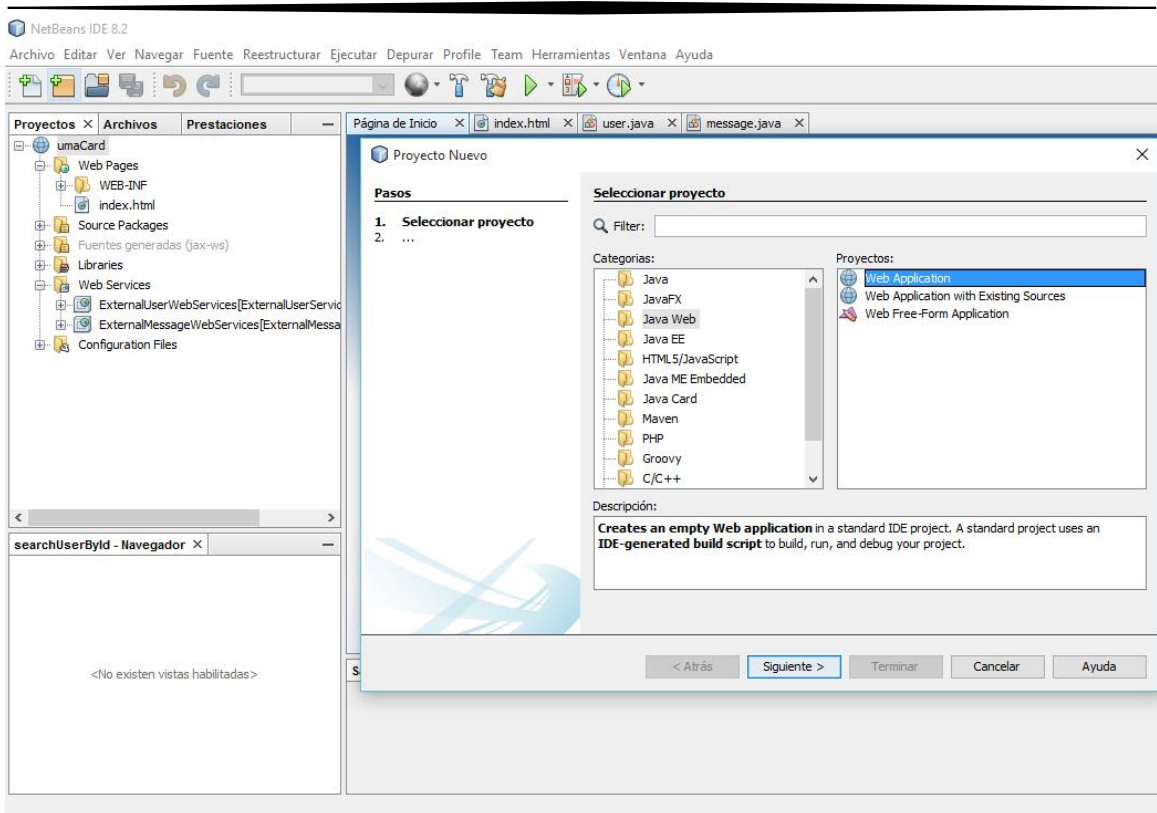
Selección de carpeta de instalación



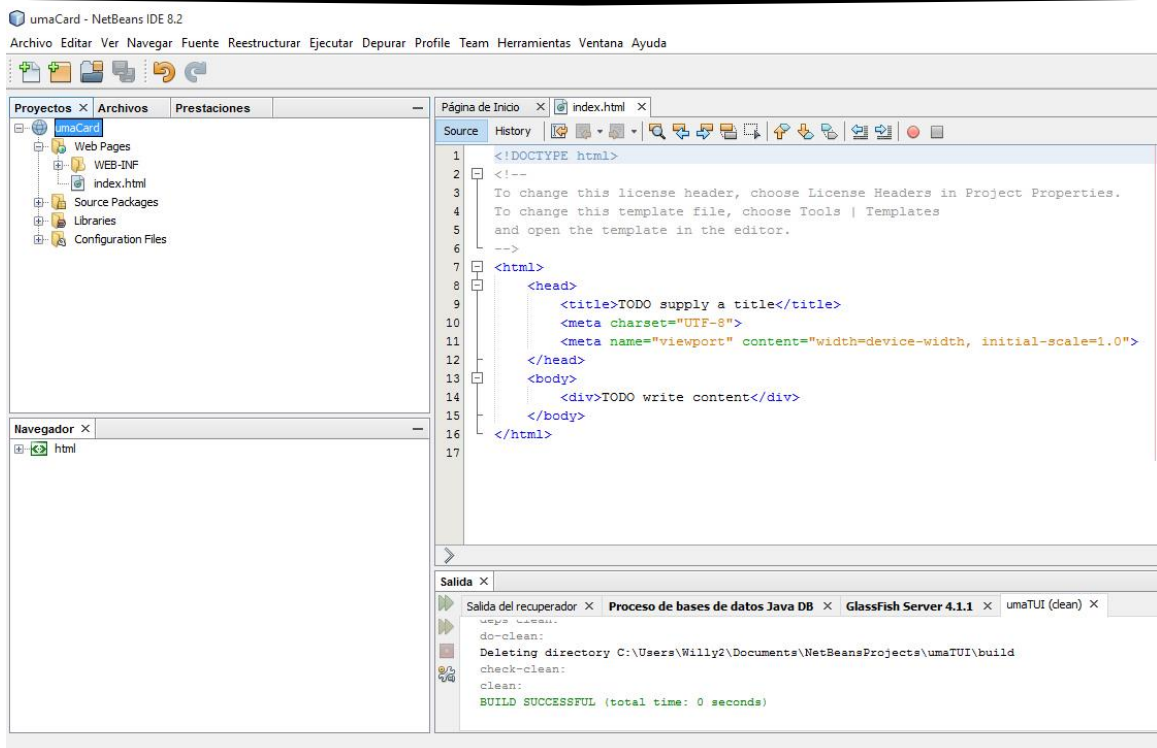
Tras completar la instalación del IDE, abrimos el entorno de desarrollo e iniciamos un proyecto, como vamos a desarrollar servicios web SOAP, lo que necesitaremos es crear un nuevo proyecto web, al que llamaremos **umaCard**.



Debido a que se van a desarrollar un conjunto de servicios web para la interconexión de sistemas de información diferentes, la clase de proyecto a crear a través del IDE NetBeans, debe ser un proyecto java web.



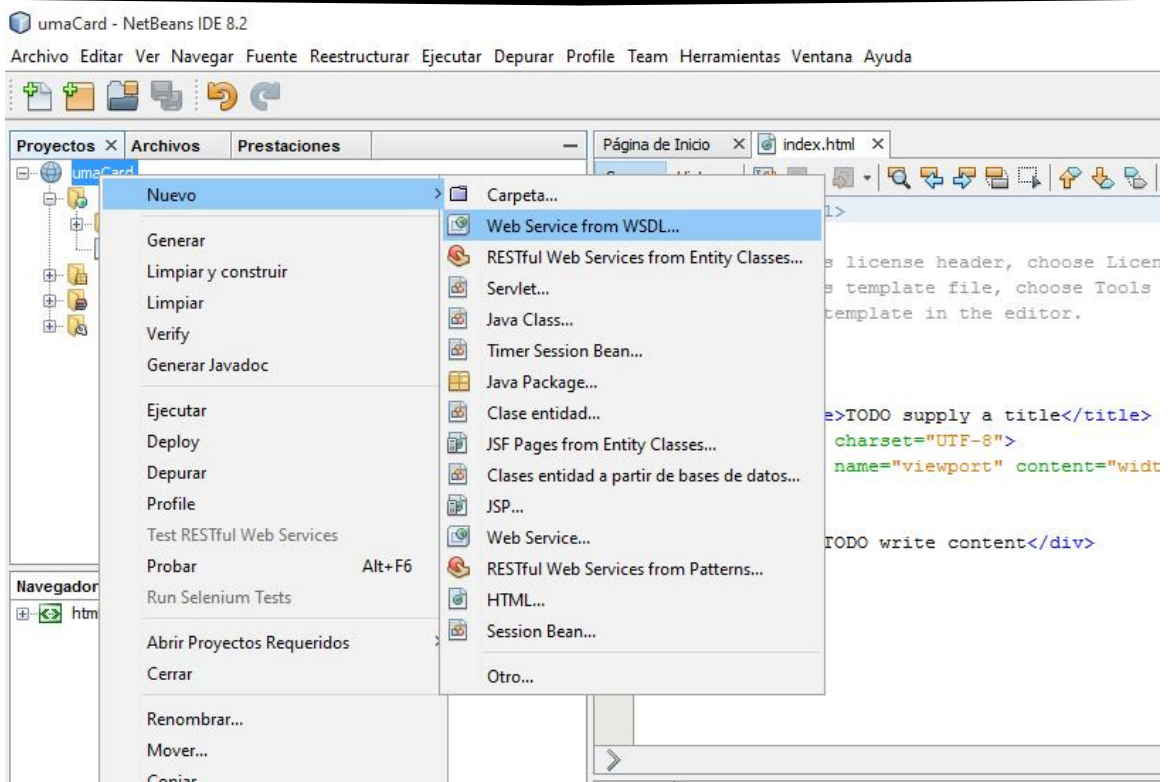
Asignamos un nombre al proyecto



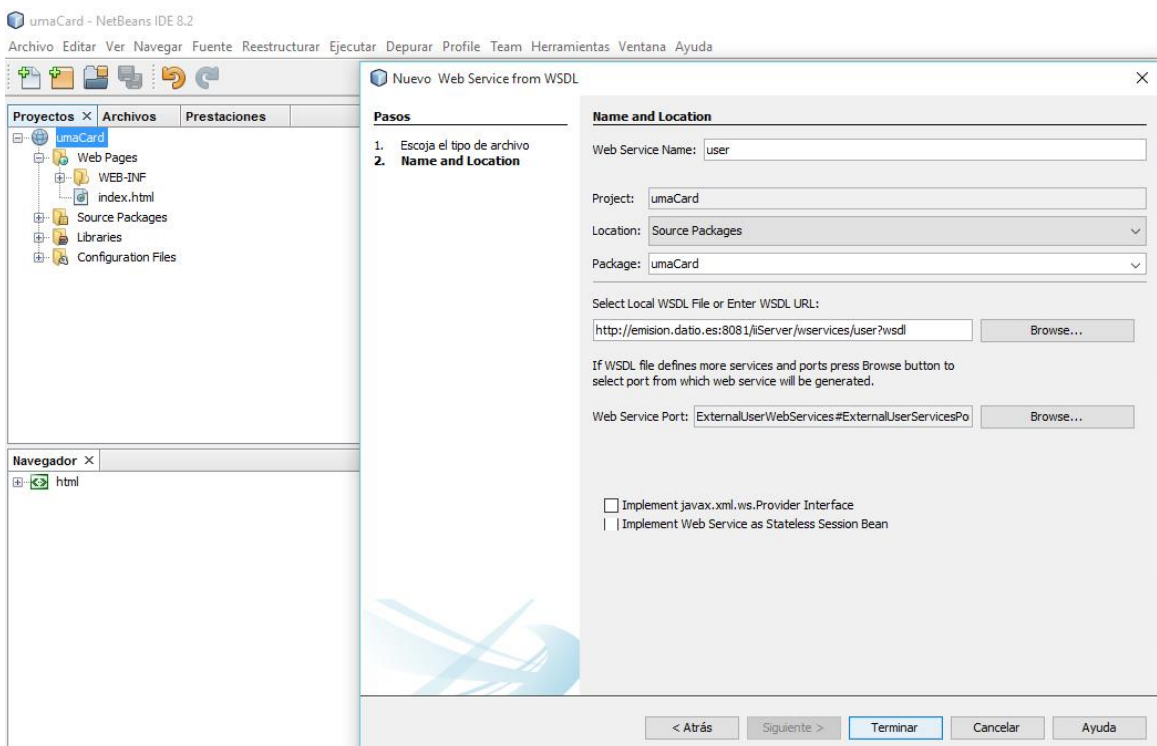
Una vez que NetBeans nos ha generado el proyecto java web, el siguiente paso a realizar será crear en dicho proyecto los servicios web necesarios a través de los ficheros WSDL planteados en el Sistema de Emisión Instantánea de carnés.

Para ello, nos situamos sobre el proyecto y clicando con el botón derecho del ratón, seleccionamos en el menú contextual la opción de Nuevo y Web Services from WSDL.

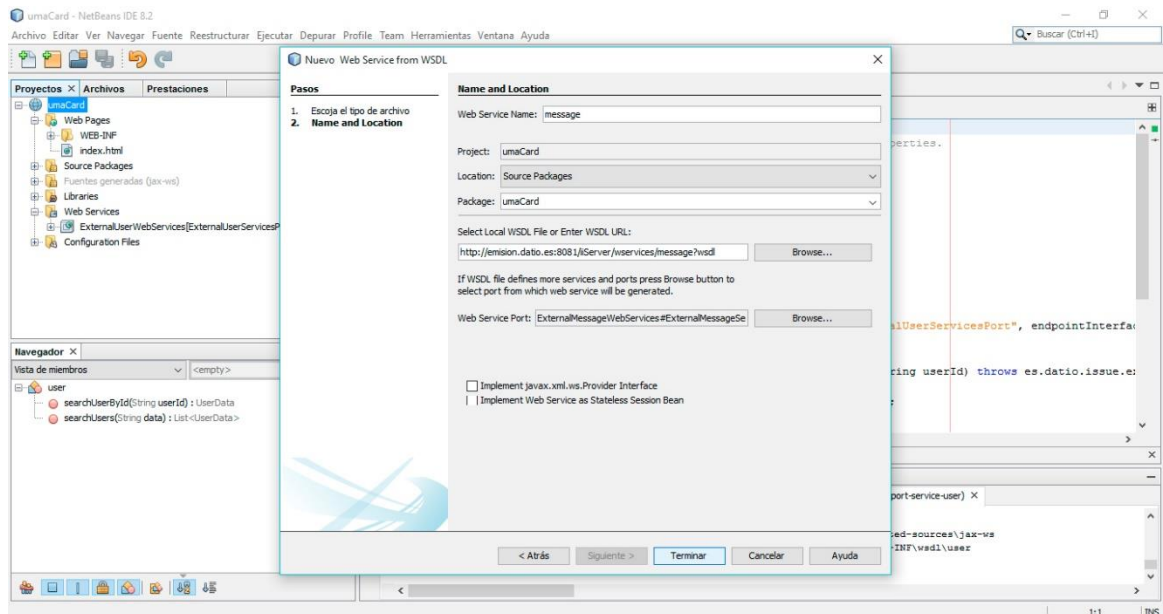




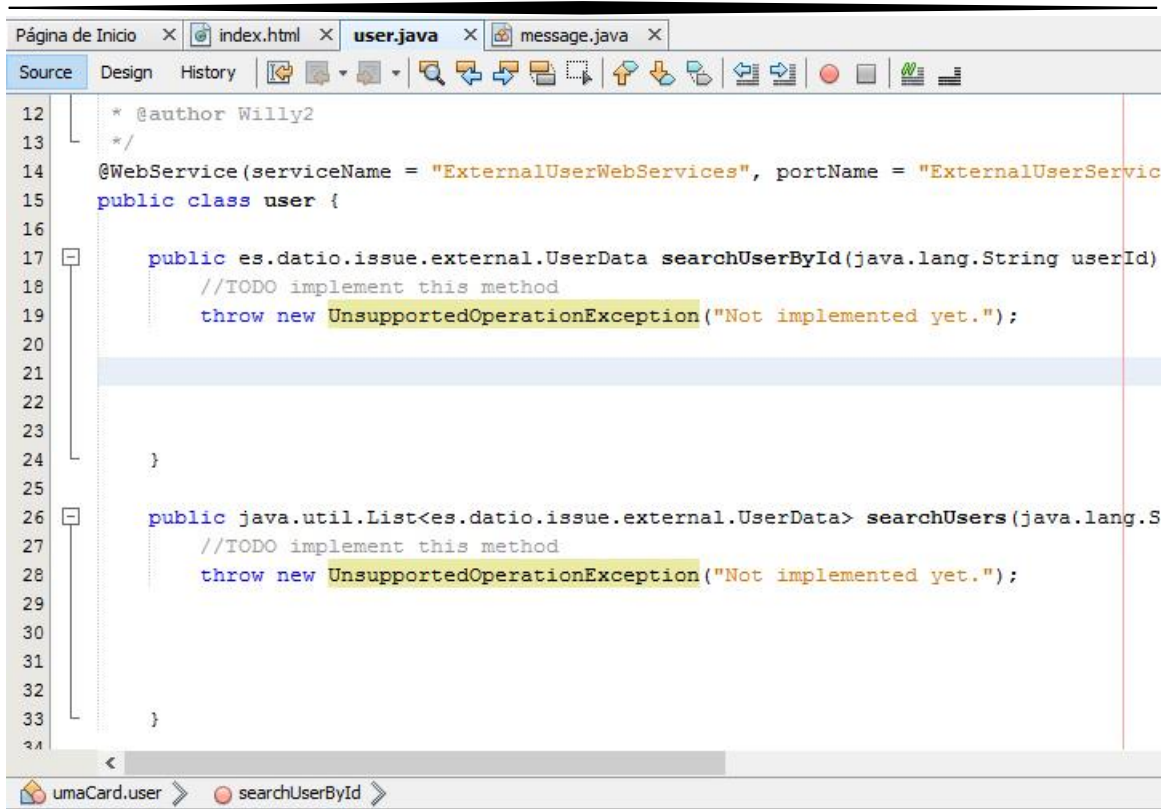
## Seleccionamos nuevo Servicio Web desde WSDL



Asignamos nombre **user** al servicio Web de búsqueda y selección de usuarios y le indicamos al IDE de NetBeans la ruta de su fichero WSDL. Una vez generado hacemos la misma operación para el servicio web **message** que retornará los datos de los carnés impresos en el sistema de emisión instantánea a los sistemas de información de la UMA.



Tras el proceso de creación de ambos servicios Web a través de los ficheros WSDL, NetBeans nos proporciona toda la estructura de directorios y ficheros necesarios sobre los que se sustentan los servicios web. Tras realizar estos pasos, sólo nos quedará escribir todo el código java de los servicios web en las funciones creadas, logrando de dar toda funcionalidad prevista en el sistema.

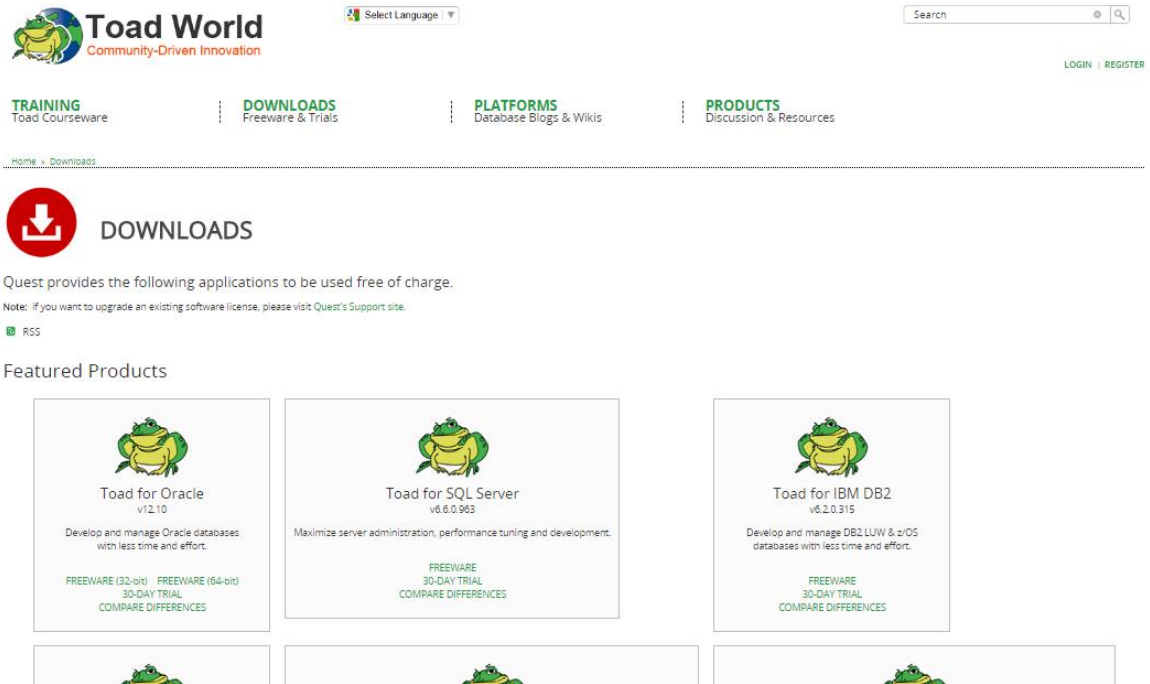


```
12  * @author Willy2
13  */
14  @WebService(serviceName = "ExternalUserWebServices", portName = "ExternalUserServic
15  public class user {
16
17      public es.datio.issue.external.UserData searchUserById(java.lang.String userId)
18          //TODO implement this method
19          throw new UnsupportedOperationException("Not implemented yet.");
20
21
22
23
24  }
25
26  public java.util.List<es.datio.issue.external.UserData> searchUsers(java.lang.S
27      //TODO implement this method
28      throw new UnsupportedOperationException("Not implemented yet.");
29
30
31
32
33  }
34
```

Finalmente, NetBeans nos genera toda la estructura de código de los servicios web SOAP, a partir de aquí sólo nos queda el comportamiento de cada servicio para que haga todo lo que se espera de ellos.

## 6.2. Gestión de base de datos con TOAD.

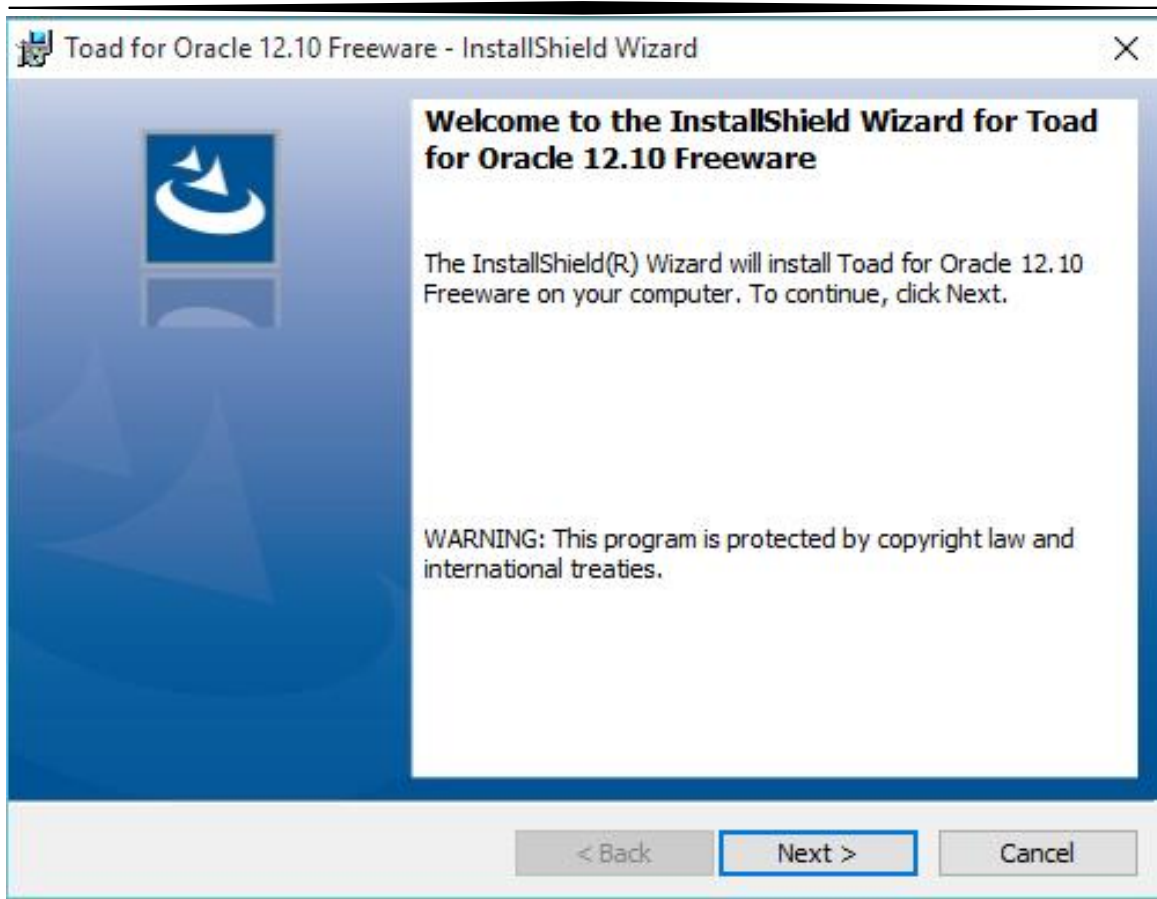
Lo primero fue instalar el TOAD elegido, que lo descargué desde la web oficial de TOAD ([toadworld.com/m/freeware/](http://toadworld.com/m/freeware/)), para lo cual tuve que registrarme en su web. Posteriormente me descargué la versión de prueba para Windows.



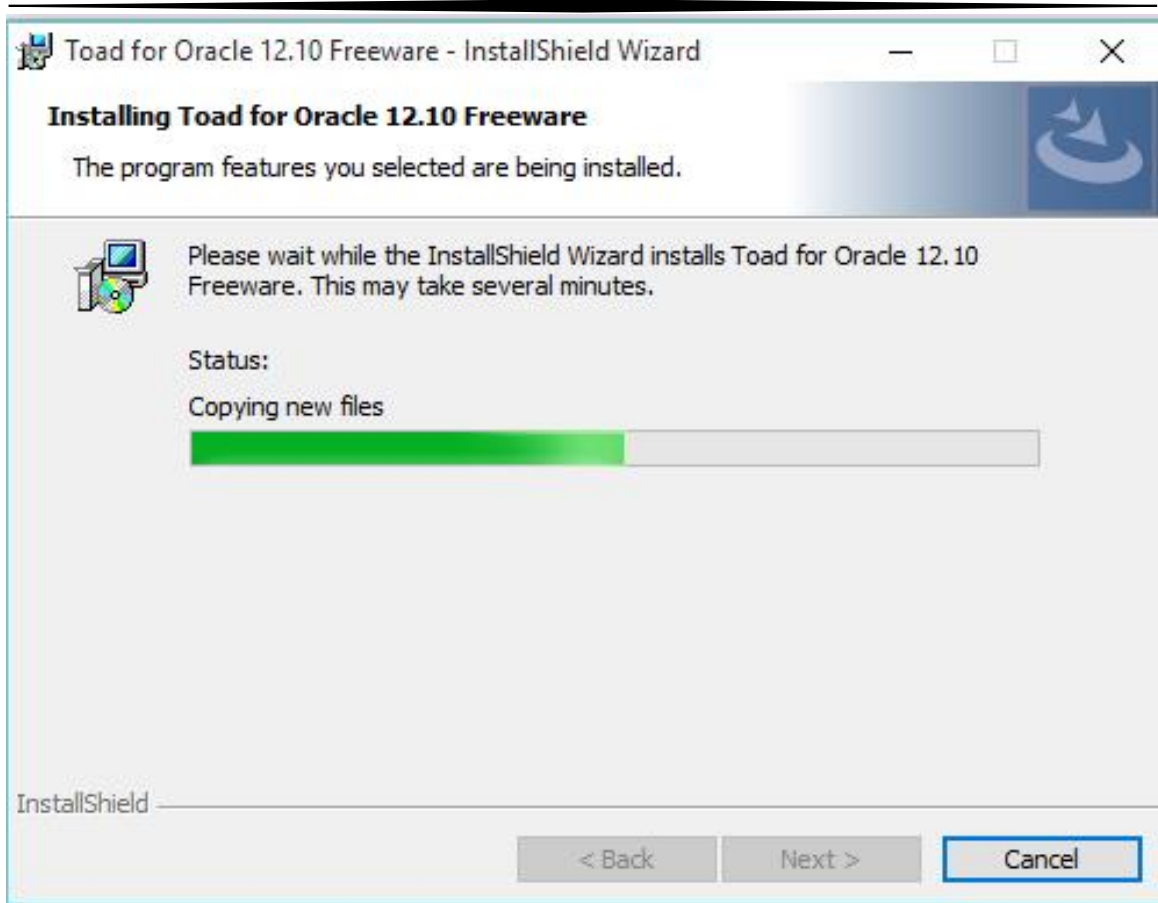
The screenshot shows the 'Downloads' page on the Toad World website. At the top, there is a navigation bar with the Toad World logo, a language selector, a search bar, and links for 'LOGIN' and 'REGISTER'. Below this, there are four main categories: 'TRAINING' (Toad Courseware), 'DOWNLOADS' (Freeware & Trials), 'PLATFORMS' (Database Blogs & Wikis), and 'PRODUCTS' (Discussion & Resources). The 'Downloads' section is highlighted with a red download icon and the text 'DOWNLOADS'. Below this, a note states: 'Quest provides the following applications to be used free of charge. Note: if you want to upgrade an existing software license, please visit Quest's Support site.' There is also an 'RSS' link. The 'Featured Products' section displays three product cards: 'Toad for Oracle v12.10', 'Toad for SQL Server v6.6.0.963', and 'Toad for IBM DB2 v6.2.0.315'. Each card includes a description, a 'FREWARE' label, a '30-DAY TRIAL' label, and a 'COMPARE DIFFERENCES' link. Below the featured products, there are three smaller, partially visible product cards.

El proceso de instalación de la aplicación es bastante sencillo, sólo tenemos que ejecutar el fichero descargado:

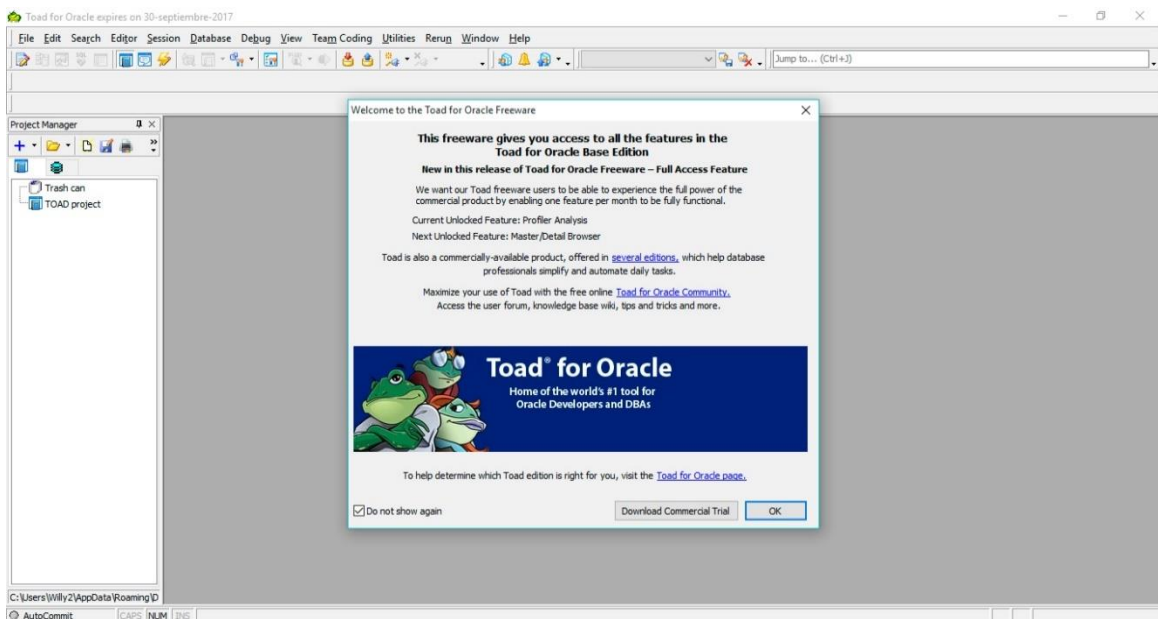
ToadForOracle\_Freeware\_12.10.0.30\_x64\_En.msi



Tras aceptar las condiciones de uso y seleccionar la carpeta donde instalar la aplicación, sólo queda esperar unos minutos mientras se copian e instalan los ficheros de la aplicación.



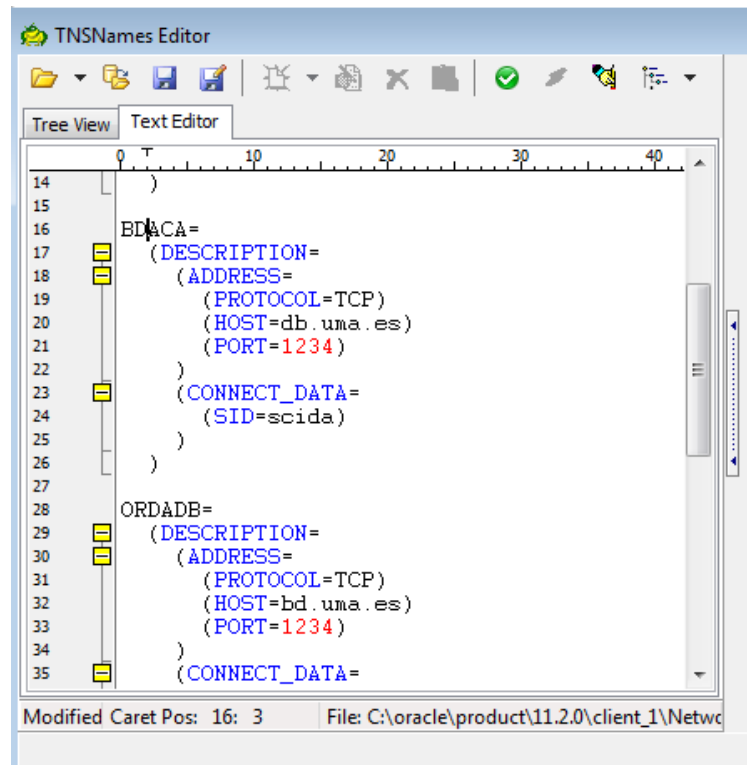
Una vez instalada, sólo tenemos que lanzar la aplicación para configurar TOAD y empezar a trabajar con el sistema de base de datos Oracle de la UMA.



Anteriormente a ponernos a trabajar con el esquema de base de datos de ordenación académica, solicitamos al administrador de base de datos la creación

de un usuario de base de datos para la aplicación, al que habrá que darle permisos de lectura y/o escritura en las diferentes tablas o vistas del modelo de datos.

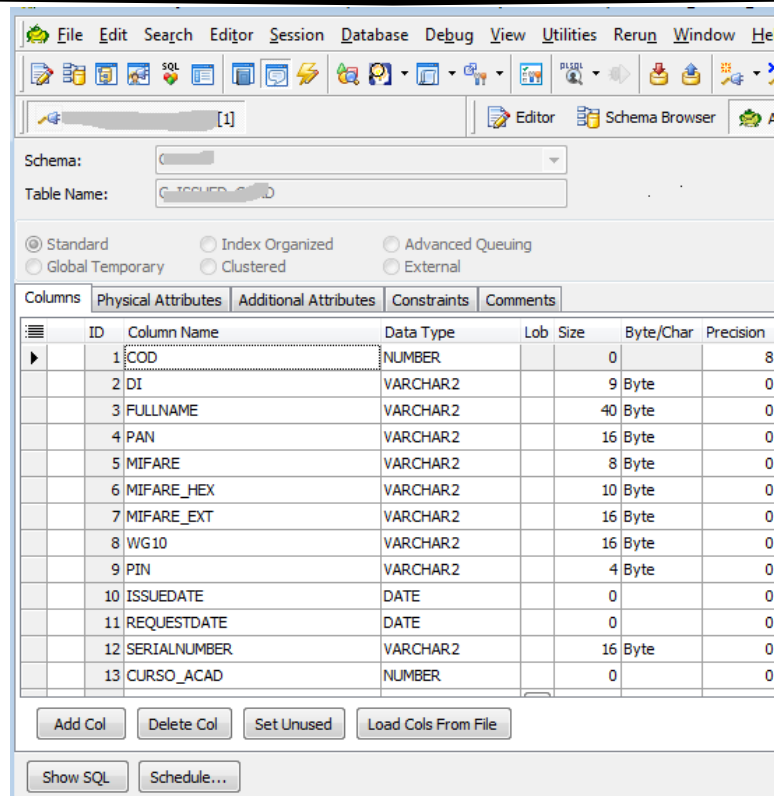
Lo primero que hacemos es configurar el acceso a las bases de datos para poder empezar a trabaja....



Una vez realizado el paso anterior, abrimos una nueva conexión con el esquema de base de datos para crear la tabla y vistas propuestas y necesarias para la posterior conexión a través de la codificación java de los diferentes servicios web desarrollados con NetBeans.

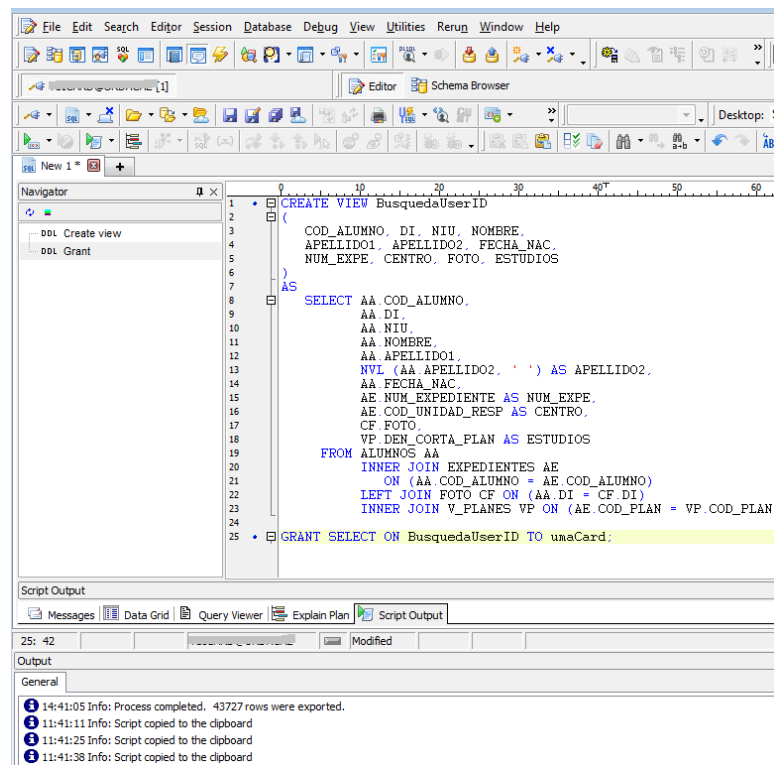
A través del entorno visual de la aplicación, localizamos la tabla de tarjetas para añadir los nuevos campos con los que se trabajará en el nuevo sistema.





ID	Column Name	Data Type	Lob	Size	Byte/Char	Precision
1	COD	NUMBER			0	8
2	DI	VARCHAR2			9 Byte	0
3	FULLNAME	VARCHAR2			40 Byte	0
4	PAN	VARCHAR2			16 Byte	0
5	MIFARE	VARCHAR2			8 Byte	0
6	MIFARE_HEX	VARCHAR2			10 Byte	0
7	MIFARE_EXT	VARCHAR2			16 Byte	0
8	WG10	VARCHAR2			16 Byte	0
9	PIN	VARCHAR2			4 Byte	0
10	ISSUEDATE	DATE			0	0
11	REQUESTDATE	DATE			0	0
12	SERIALNUMBER	VARCHAR2			16 Byte	0
13	CURSO_ACAD	NUMBER			0	0

Posteriormente, nos vamos a la ventana de edición directa de código SQL para lanzar el código de creación de vistas, asignando los permisos de consulta al usuario de la aplicación.



```

1 CREATE VIEW BusquedaUserID
2 (
3   COD_ALUMNO, DI, NIU, NOMBRE,
4   APELLIDO1, APELLIDO2, FECHA_NAC,
5   NUM_EXPE, CENTRO, FOTO, ESTUDIOS
6 )
7 AS
8 SELECT AA.COD_ALUMNO,
9        AA.DI,
10       AA.NIU,
11       AA.NOMBRE,
12       AA.APELLIDO1,
13       NVL(AA.APELLIDO2, ' ') AS APELLIDO2,
14       AA.FECHA_NAC,
15       AE.NUM_EXPEDIENTE AS NUM_EXPE,
16       AE.COD_UNIDAD_RESP AS CENTRO,
17       CF.FOTO,
18       VP.DEN_CORTA_PLAN AS ESTUDIOS
19 FROM ALUMNOS AA
20      INNER JOIN EXPEDIENTES AE
21            ON (AA.COD_ALUMNO = AE.COD_ALUMNO)
22      LEFT JOIN FOTO CF ON (AA.DI = CF.DI)
23      INNER JOIN V_PLANES VP ON (AE.COD_PLAN = VP.COD_PLAN)
24
25 GRANT SELECT ON BusquedaUserID TO umaCard;

```

Script Output

25: 42

Output

General

- 14:41:05 Info: Process completed. 43727 rows were exported.
- 11:41:11 Info: Script copied to the clipboard
- 11:41:25 Info: Script copied to the clipboard
- 11:41:38 Info: Script copied to the clipboard



---

### 6.3. Montaje del servidor iiServer del proyecto.

Los "ocones" son las máquinas preparadas en la UMA sobre las que se montarán las máquinas virtuales con el software open source qemu. En el servicio ATDI de la UMA, hay disponibles varias máquinas en tres ocones, ocon1.sci.uma.es, ocon2.sci.uma.es, ocon3.sci.uma.es, y ocon4.sci.uma.es.

Además, tenemos un Windows 2008 R2, sin activar, que nos sirve de base para replicar los servidores Windows que se necesiten a través del siguiente el proceso:

1. Solicité al área de sistemas que me replicara ese Windows 2008 R2 en una máquina virtual en un ocón y le asignara un puerto por donde conectarme por VNC.
2. Una vez que el área de sistemas ha completó el proceso anterior y me informó de ello, me conecté a la máquina por VNC. Windows no estaba activado, pero antes de activarlo y configurar la red debemos cambiar el SID del sistema operativo, para ello abrí una ventana de comando e hice:
  - a. Ejecutar desde cmd: `c:\windows\system32\sysprep\sysprep.exe`
  - b. Dejamos OOBE, para que cuando reinicie pregunte idioma, etc.
  - c. Marcamos GENERALIZE, para que genere un nuevo SID
3. Cuando reinicie la máquina ya se puede configurar la IP, proxy (proxy.gestion.uma.es:3128), nombre netBIOS, activar RDesktop, etc. y sobre todo activar el producto.
4. Posteriormente, tenemos agregar la máquina a nuestro dominio
  - a. Dominio: servicios.uma.es
  - b. IP DNS: 192.168.92.3
5. El disco duro se queda en 15 Gb en vez de los 50 Gb que necesitamos para el iiServer, por lo que tenemos que extender la partición, para ello abrimos una ventana de comando y ejecutamos:
  - a. `c:\>diskpart`
  - b. `diskpart> list volume`
  - c. Seleccionamos el volumen a extender, en mi caso el de 50 Gb es el volumen 4
  - d. `diskpart> select volume 4`
  - e. `diskpart> extend`
  - f. `diskpart> exit`
6. Después debemos agregar los drivers de VirtIO
7. Una vez llegado a este punto, ya podemos utilizar la máquina para trabajar como iiServer.

---

### 6.3.1. Drivers VirtIO.

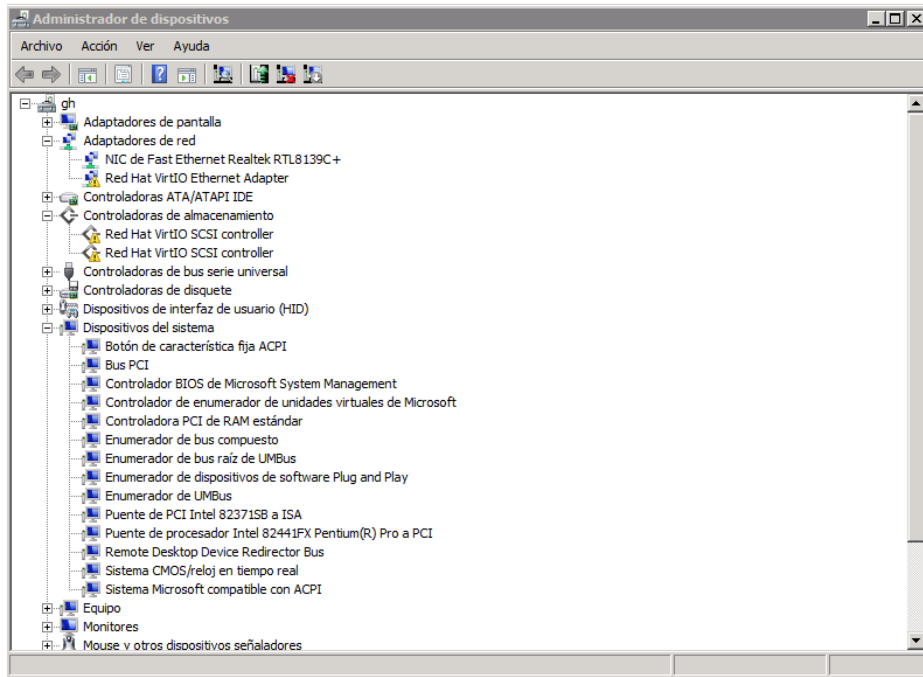
Para optimizar la Entrada/Salida de las máquinas virtuales con Windows 2008 R2 vamos a utilizar los drivers VirtIO, disponibles para Red, Almacenamiento (discos) y Gestión de Memoria (soportando ballooning).

Instalando estos drivers pasamos de trabajar con dispositivos emulados por los Ocones a drivers que son capaces de 'hablar directamente' con el hardware de los Ocones. Debemos tener montada la imagen ISO con los drivers VirtIO en la máquina virtual, o por otro medio.

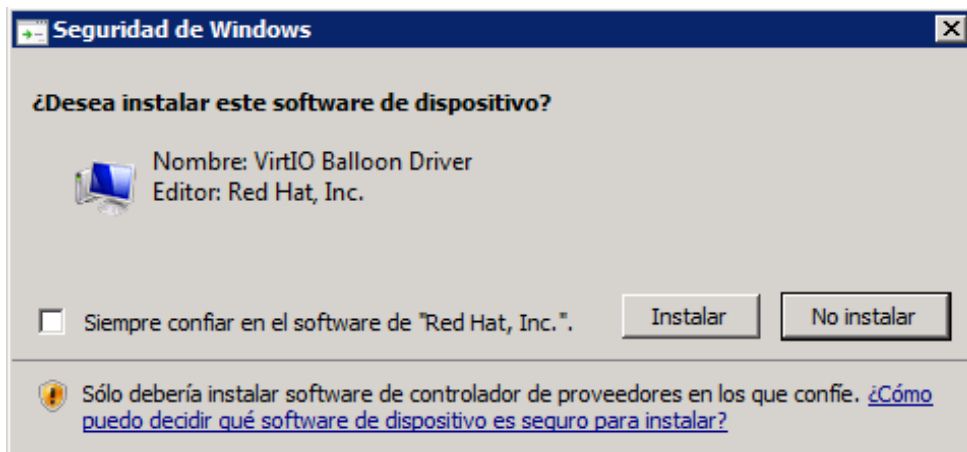
1. Agregamos nuevo Hardware Heredado en Administrador de Dispositivos (botón derecho sobre el nombre de la máquina)
  - a. Agregamos Adaptador de Red usando disco ( se busca en `../Win7/amd64` para 64 bits, o en `../Win7/x86` para 32 bits ) -> Se añade el dispositivo Red Hat VirtIO Ethernet Adapter).
  - b. Agregamos Controladores de Almacenamiento usando disco (se busca en `../Win7/amd64` para 64 bits, o en `../Win7/x86` para 32 bits).

Se deben añadir 2, por seguridad, Red Hat VirtIO SCSI Controller v16.x.xx.xxx y v61.x.xx.xxx.

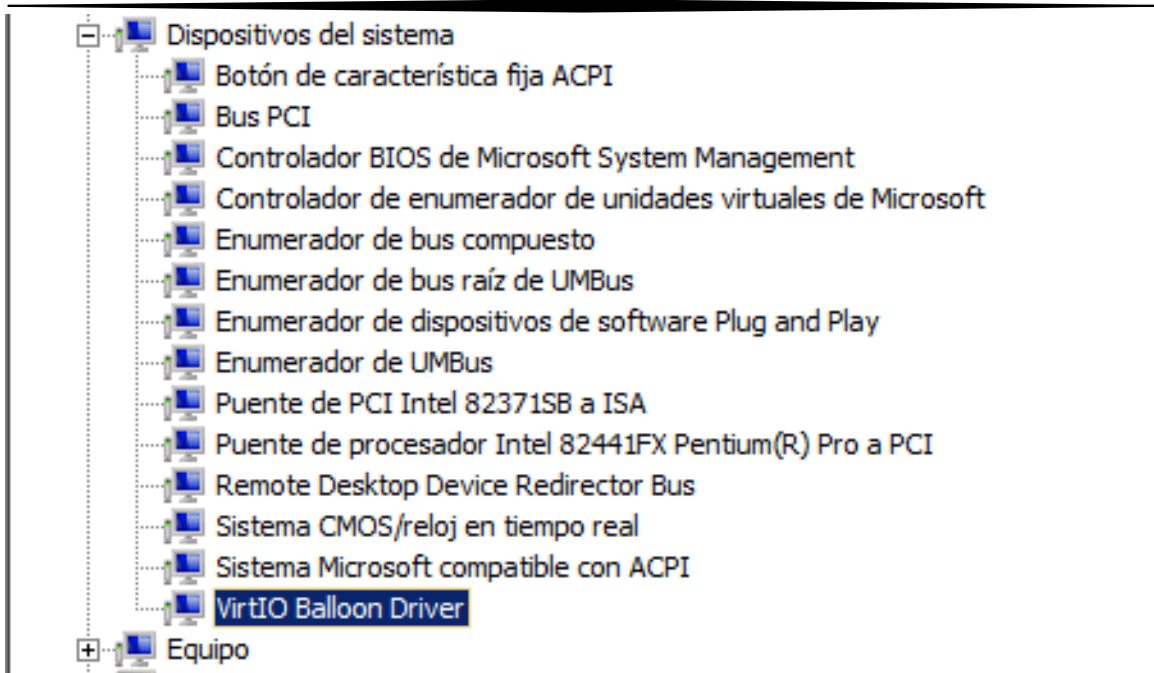
Por ahora, al presentarle el disco tipo VirtIO ha cogido la versión v61.x.xx.xxx, pero no está de más que tenga ambas versiones.
2. Actualizamos drivers
  - a. Actualizamos el controlador del dispositivo del sistema Controladora PCI de RAM estándar, haciendo que busque en disco ( se busca en `../Win7/amd64` ).
  - b. Se sustituirá ese dispositivo por VirtIO Balloon Driver.



Agregar hardware virtual



Memoria



Balloon driver

Una vez hecho esto, es cuando hay que 'presentar' los dispositivos VirtIO a la máquina virtual.

#### 6.4. Codificación de los servicios Web SOAP con NetBeans.

Llegados a este punto, ya disponemos en el sistema del modelo de datos necesario para el desarrollo y los servicios web SOAP, por lo que nos disponemos a entrar en NetBeans para codificar la funcionalidad de los distintos servicios web que debemos desarrollar.

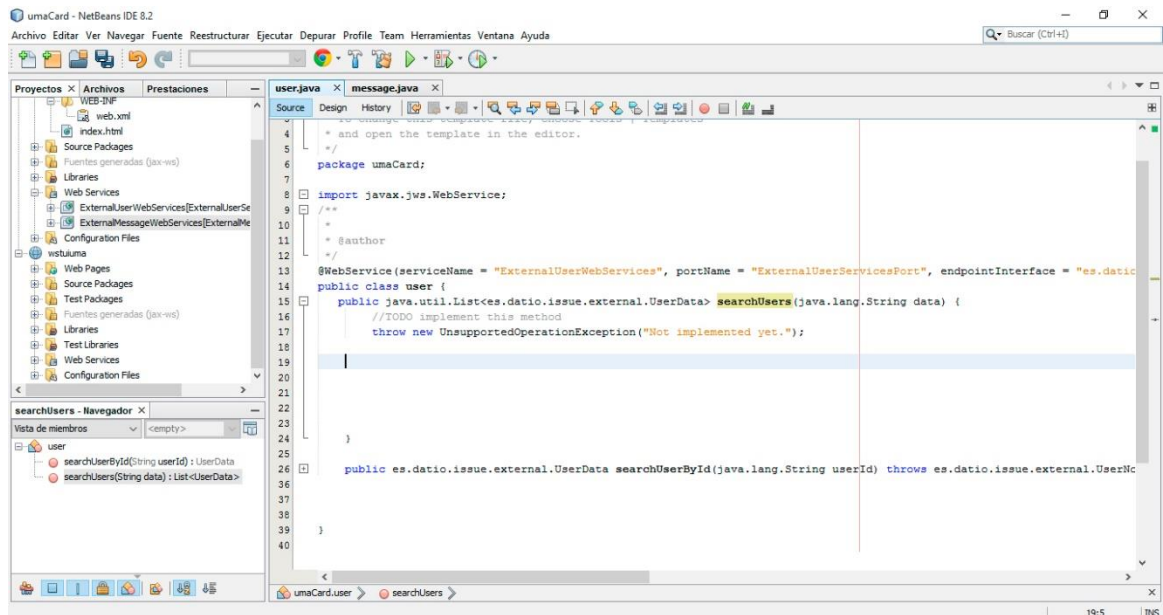
Dichos servicios web han de trabajar conectándose de forma inicial a la base de datos del sistema de información de la UMA, para posteriormente procesar el parámetro de entrada o parámetros de entrada obtenido, lanzar la consulta SQL que cada servicio web precise y procesar la información resultante.

El código fuente del desarrollo, junto con toda la estructura de los ficheros de los servicios web desarrollados, puede descargarse desde el siguiente enlace:

- <https://app.box.com/s/6p9rkye1squ2pzwtldfoo5mirnjd5p3k9>

## 6.4.1. Codificación searchUsers.

Seleccionamos en el explorador de ficheros de NetBeans el servicio web para iniciar su desarrollo.



En la codificación del servicio web, le primer fue la comprobación de que la invocación del servicio no se realizar con el envío de un parámetro vacío.

```

public java.util.List<es.datio.issue.external.UserData> searchUsers(j
//TODO implement this method
//throw new UnsupportedOperationException("Not implemented yet.

List<UserData> listUserData = new ArrayList<UserData>();
data = data.trim().toUpperCase();
if (data.equals(""))
{
    return listUserData;
}
  
```

Para lo cual, primero preparamos una lista del objeto UserData en la que retornar el resultado del servicio web al sistema de emisión instantánea y luego comprobamos que el parámetro enviado no se encuentre vacío, si fuera este el caso, aquí se terminaría el funcionamiento del servicio web, retornando la lista del objeto recién creada que se encuentra en su estado vacío.

Si la ejecución continua es porque el parámetro enviado no iba vacío, por lo que debemos preparar la conexión con la base de datos.

```

// Conexion a base de datos oracle Ordaca
baseDatos.getInstance().conectar();
Connection conn = baseDatos.getInstance().getConexion();
Statement stmt = (Statement)conn.createStatement();
// "alter session set nls_comp=linguistic"
stmt.addBatch("ALTER SESSION SET NLS_COMP = LINGUISTIC");
stmt.addBatch("ALTER SESSION SET NLS_SORT = SPANISH_M");
stmt.executeBatch();
// consulta sql
String sql = " SELECT COD_ALUMNO as ExternalId, DI as Docum
" FROM COMUN.V_C_BUSQUEDA_ALUMNOS_TUI " +
" WHERE ROWNUM <= 10 AND (DI LIKE '%' + data + '%' OR DI
" OR FULLNAME LIKE '%' + data + '%' OR FULLNAME LIKE '%"
ResultSet rset = (ResultSet)stmt.executeQuery(sql);

```

Una vez ejecutada la conexión de la base de datos, preparamos la consulta SQL, limitándola a un máximo de 10 registros, sabiendo que si en el primer envío de resultados el operador no localiza al alumno, deberá refinar la búsqueda y enviar un parámetro más específico que el inicial.

Al ejecutar la consulta, debemos recorrer la colección de resultados e ir mapeándolo en el objeto userData que iremos añadiendo a la lista de respuesta creada inicialmente.

```

while (rset.next()) {
    flag = true;
    UserData userData = new UserData();
    userData.setDocument(rset.getString("Document"));
    userData.setExternalId(rset.getString("ExternalId"));
    userData.setFullname(rset.getString("Fullname"));
    listUserData.add(userData);
}

```

Además, usamos un flag para asegurarnos que el retorno del servicio web no es una lista vacía porque la consulta SQL tuviera algún registro que mostrar.

```

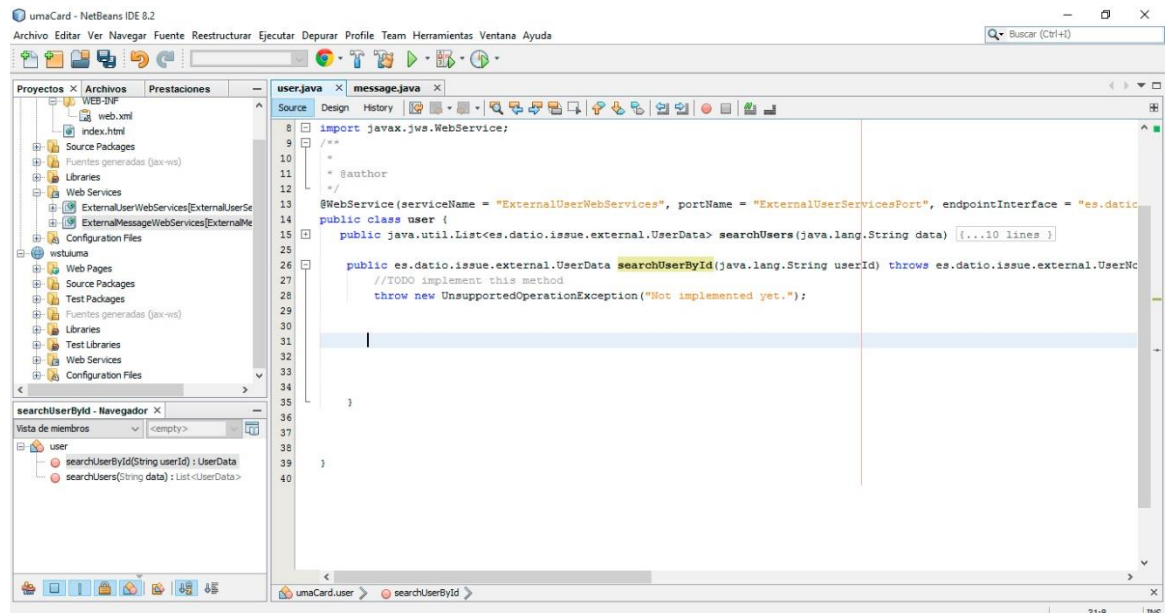
stmt.close();
baseDatos.getInstance().desconectar();
if (!flag) {
    throw new UnsupportedOperationException("Usuario no localizado");
}
return listUserData;

```

Finalmente, tras recorrer toda la colección de resultados de la consulta SQL, cerramos la conexión con la base de datos, comprobamos si hay por lo menos una fila de resultados que enviar (si no la hay, lanzamos la excepción al sistema), para finalizar enviando la lista de objetos userData con los resultados obtenidos en la consulta a base de datos que hemos tratado.

## 6.4.2. Codificación searchUsersByld.

Seleccionamos en el explorador de ficheros de NetBeans el servicio web para iniciar su desarrollo.



A diferencia del servicio web anterior, en la codificación de la funcionalidad de searchUsersByld no debo preocuparme de comprobar si el argumento de entrada viene vacío, ya que dicha comprobación se realiza desde la APP del sistema de emisión instantánea, que sólo invocará al servicio web de un usuario concreto que haya sido seleccionado en la APP. Por todo lo anterior, lo único que debemos hacer al inicio de este servicio web es preparar la estructura de objetos con los que voy a contesta a la petición que ha llegado.

```
UserData userData = new UserData();
UserData.Content content = new UserData.Content();
```

Posteriormente, habilitamos la conexión con la base de datos.

```
// Conexion a base de datos oracle Ordaca
baseDatos.getInstance().conectar();
Connection connOrdaca = baseDatos.getInstance().getConexion();
connOrdaca.setAutoCommit(false);
Statement stmt = (Statement)connOrdaca.createStatement();
```



Y antes de ponernos a ejecutar la consulta SQL que me devuelva los datos del usuario seleccionado, debo comprobar que para dicho usuario no hayan sido impresas ya dos tarjetas universitarias.

```
int cursoAcad = util.dameCursoAcademico();
if (util.tarjetaActiva(stmt, userId, cursoAcad )) {
    // error ya tiene 2 tarjetas activas.
    throw new UnsupportedOperationException("El usuario no pu
}
}
```

Si se da el caso de que para el alumno seleccionado ya han sido impresos dos carnés, se informa de ellos y se da por terminada la ejecución del servicio web.

Por el contrario, si todavía el usuario tiene derecho a obtener una nueva tarjeta universitaria, pues deberemos preparar la consulta SQL que nos permita preparar la información del usuario que ha solicitado el carné.

```
String sql = "SELECT COD_ALUMNO AS ExternalId, DI as Document
"          NUM_EXPE AS NIA, CENTRO, FOTO, ESTUDIOS " +
"          FROM COMUN.V_C_BUSQUEDA_MG_TUI " +
"          WHERE COD_ALUMNO = " + userId ;

rset = (ResultSet)stmt.executeQuery(sql);

if (rset.next()) {
    String document = rset.getString("Document");
    String externalId = rset.getString("ExternalId");
    String nombre = rset.getString("NOMBRE").trim().toUpperCase();
    String ap1 = rset.getString("APELLIDO1").trim().toUpperCase();
    String ap2 = rset.getString("APELLIDO2").trim().toUpperCase();
    String fullname = (nombre + " " + ap1 + " " + ap2).trim();
    String nya = util.nya(nombre, ap1, ap2).trim();
    String lineal = "ESTUDIANTE " + rset.getString("NIA");
```

Tras la ejecución de la consulta SQL en la base de datos, vamos añadiendo los datos extraídos en los objetos preparados inicialmente, para su posterior retorno a la APP del sistema de emisión instantánea a través del servicio web.

Además, antes de la inclusión de los datos en los objetos, debemos tratar la foto del usuario, convirtiéndola a una cadena de texto en base 64, para poder enviarla en el objeto del sistema.



```

String foto = "";
InputStream isFoto = rset.getBinaryStream("FOTO");
if (isFoto != null) {
    foto = util.pasafotoB64(rset.getBinaryStream("FOTO"));
}

public static String pasafotoB64(InputStream foto) {
    String sfoto = "";
    byte[] byteFoto;
    try {
        byteFoto = IOUtils.toByteArray(foto);
        sfoto = Base64.encodeBytes(byteFoto);
    } catch (IOException e) {
        // TODO
        System.out.println("KO: pasafotoB64");
        System.out.println(e.getMessage());
    }
    return sfoto;
}

```

Una vez tenemos todos los datos preparados, los asignamos a la estructura de objetos creada inicialmente, para su posterior envío al sistema de emisión instantánea.

```

userData.setDocument(document);
userData.setExternalId(externalId);
userData.setFullname(fullname);
Entry entry = new Entry();
content.getEntry();
entry.setKey("nya"); entry.setValue(nya); content.entry.add(entry); entry = new
entry.setKey("linea1"); entry.setValue(linea1); content.entry.add(entry); entry
entry.setKey("linea2"); entry.setValue(linea2); content.entry.add(entry); entry
entry.setKey("codBar"); entry.setValue(codBar); content.entry.add(entry); entry
entry.setKey("nia"); entry.setValue(nia); content.entry.add(entry); entry = new
entry.setKey("documento"); entry.setValue(document); content.entry.add(entry); e
entry.setKey("categoria"); entry.setValue(categoria); content.entry.add(entry);
entry.setKey("centro"); entry.setValue(centro); content.entry.add(entry); entry
entry.setKey("curso"); entry.setValue(curso); content.entry.add(entry); entry =
entry.setKey("activa"); entry.setValue(activa); content.entry.add(entry); entry
entry.setKey("interno"); entry.setValue(interno); content.entry.add(entry); entr
entry.setKey("reserva1"); entry.setValue(reserva1); content.entry.add(entry); en
entry.setKey("reserva2"); entry.setValue(reserva2); content.entry.add(entry); en
entry.setKey("

```

Terminamos cerrando la conexión con la base de datos y devolviendo el objeto `userData` que contiene la información solicitada por el sistema.

```

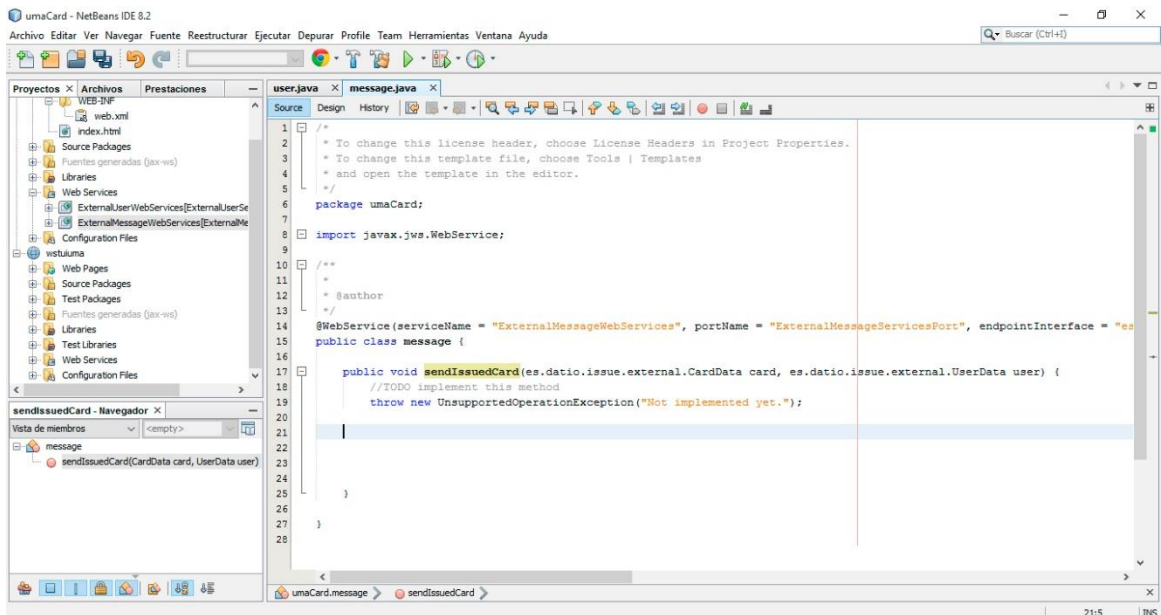
} finally{
    baseDatos.getInstance().desconectar();
}

return userData;

```

### 6.4.3. Codificación sendIssuedCard.

Seleccionamos en el explorador de ficheros de NetBeans el servicio web para iniciar su desarrollo.



Lo primero que debemos prever es la conexión con la base de datos

```
baseDatos.getInstance().conectar();
Connection conn = baseDatos.getInstance().getConexion();
conn.setAutoCommit(false);
```

Posteriormente debemos poder acceder a los distintos datos de la tarjeta impresa.

```
CardData.Content content = card.getContent();
contenedor = new Hashtable<String, String>();
for (CardData.Content.Entry atributo : content.getEntry())
{
    String clave = atributo.getKey(); // Por ejemplo "mifara"
    String valor = atributo.getValue(); // Por ejemplo "ABCDEF12"
    contenedor.put(clave, valor);
}
// paso la foto a InputStream
String foto = contenedor.get("foto");
byte[] bfoto = Base64.decode(foto);
isFoto = new ByteArrayInputStream(bfoto);
```

Primero asignamos el "content", para después poder iterar por los distintos elementos con el método "getEntry" que devuelve una lista con todos los atributos del content:

Posteriormente, decodificamos el contenido de la foto en base64 a binario para su posterior almacenamiento en la base de datos.

Una vez tenemos todos los datos que debemos almacenar en la base de datos, nos basta con codificar la consulta SQL para la inserción en la base de datos. En esta ocasión, insertaremos en la tabla tarjetas, todos los datos de la nueva tarjeta impresa, e insertaremos o actualizaremos en la tabla Foto, la imagen de la foto del usuario del carné.

```
String sql = " INSERT INTO TARJETAS (DI, FULLNAME, MIFARE, PAN, WG10,
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1,user.getDocument());
pstmt.setString(2,user.getFullname());
pstmt.setString(3,contenedor.get("mifare"));
pstmt.setString(4,contenedor.get("pan");
pstmt.setString(5,contenedor.get("wg10"));
pstmt.setString(6,contenedor.get("pin"));
pstmt.setString(7,card.getSerialNumber());
pstmt.setBinaryStream(8, (InputStream)isFoto, (int)foto.length());
pstmt.setDate(9,issueDate);
pstmt.setDate(10, requestDate);
pstmt.setInt(11,Integer.parseInt(user.getExternalId()));
pstmt.setLong(12,card.getId());
pstmt.setString(13,contenedor.get("mifare_ext"));
pstmt.setInt(14, cursoAcad);
pstmt.setString(15, util.mifareToHex(contenedor.get("mifare_ext")));
pstmt.executeUpdate();
```

Finalizamos el servicio web, realizando confirmando la inserción en la base de datos con el comit de la transacción.

```
conn.commit();
stmt.close(); pstmt.close(); pstmtFoto.close();
baseDatos.getInstance().desconectar();
```

## **6.5. Pruebas.**

Se pueden diferenciar dos tipos de prueba durante el desarrollo de los servicios web, las realizadas a cada entrega de desarrollos parciales entregadas, siguiendo las recomendaciones de la metodología Scrum, y la realizada al final del proyecto y que fue programada en el planteamiento inicial de las fases del proyecto que seguían el modelo de desarrollo en cascada.

Las primeras pruebas fueron realizadas por parte del soporte técnico del Santander, siempre que les informáramos de que una nueva versión de los servicios web había sido disponibles en el entorno de preproducción.

Las segundas pruebas fue las que fueron programadas en un inicio, que coinciden con el ciclo de desarrollo en cascada, se realizaron pruebas de los sistemas implementados en las fechas programadas, junto con los compañeros del soporte técnico del Banco Santander que se desplazaron hasta la UMA.

Estas pruebas se centraron en el correcto funcionamiento, en el entorno de preproducción, de los servicios web SOAP desarrollados. Probando desde la APP del sistema de emisión instantánea, todo el ciclo de funcionamiento de los servicios web, realizando múltiples búsquedas y selección de usuarios, así como la impresión de un lote de carnés, para comprobar también la correcta inserción de los datos de las tarjetas en los sistemas de información de la UMA.

El objetivo de estas pruebas, además de comprobar el correcto funcionamiento de los servicios web, era poder detectar algún mal funcionamiento del sistema interconectado, para poder corregir aquello que fuera mal.

Las pruebas fueron superadas con éxito y todo funcionó sin que apareciera ningún error.

## **6.6. Caso de éxito.**

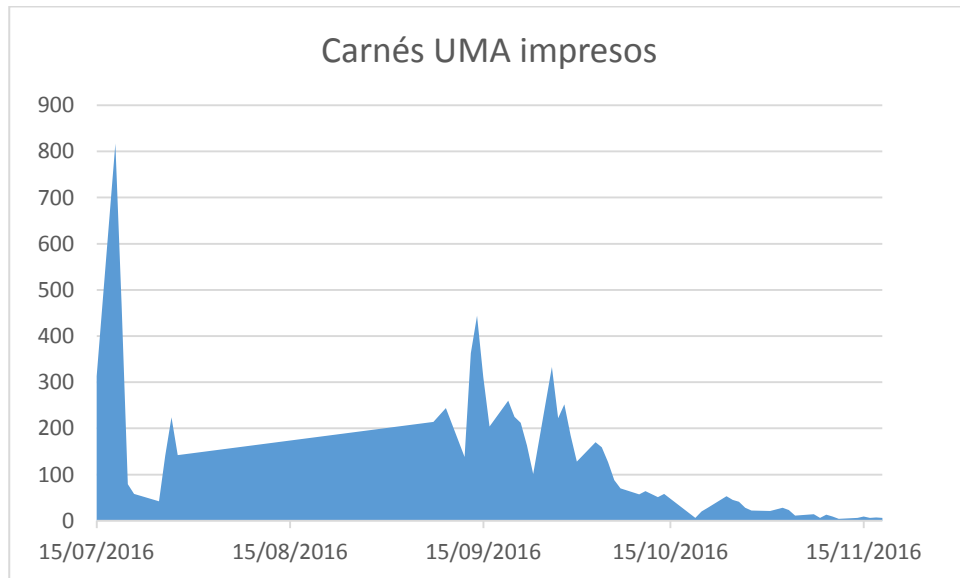
El proyecto fue puesto en producción desde el 15 de Julio coincidiendo con el inicio de la matriculación de los alumnos de primero del curso académico 2016/2017.

Para la puesta en marcha fueron habilitado cinco puntos de emisión de carnés en la UMA, ubicados en los siguientes espacios:

- Facultad de Ciencias Económicas y Empresariales.
- Facultad de Ciencias de la Comunicación y Facultad de Turismo.
- ETSI de Informática y ETSI de Telecomunicación.
- Facultad de Derecho.
- Facultad de Estudios Sociales y del Trabajo.

Desde el 1 de Octubre de 2016, dichos cinco puntos fueron deshabilitados por el Banco Santander, pasando la emisión de carnés a las oficinas bancarias que tiene en los Campus de Teatinos y El Ejido.

Hasta el 18 de noviembre de 2016 se han impreso desde el sistema de emisión instantánea 7587 carnés.



Destacando como el segundo día uso real del sistema, fue cuando más carnés fueron impresos, un total de 817, habiendo unas pocas jornadas que pasaron de las 400 impresiones, pero como término medio, podemos destacar que la media de carnés impresos supera las 150 impresiones.

También se puede comprobar, que los alumnos de la UMA acudieron a obtener sus carnés de la UMA durante los periodos de matriculación, dejando en testimonial, el conteo continuo que se produce diariamente de unas pocas impresiones en las oficinas del Santander, una vez los 5 puestos de impresión del campaña fueron deshabilitados.

### 6.6.1 Integración tarjetas y parking UMA.

Una vez tenemos implantado el sistema de emisión instantánea de carnés UMA y disponemos de los datos de las tarjetas emitidas ad hoc en los sistemas de información de la UMA, ya hemos implantado el uso de los carnés UMA para facilitar el acceso a los estudiantes de la Facultad de Ciencias Económicas al parking de estudiantes del centro.

Dicho parking ha sido provisto de barreras de entrada y salida de vehículos, controladas por lectores de proximidad Evoprox, que son controlados a través del Sistema de Control MODOS.

El sistema MODOS conecta a través de una ip fija, con su software de gestión, el cual consulta las lecturas recibidas en una servidor SQL Server de Microsoft, que almacena los datos de los usuarios habilitados con acceso al sistema de barreras.

Ya ha sido desarrollada la integración de los sistemas de la UMA con el sistema Modos de barreras, para que el trasvase de los datos necesarios de las tarjetas que deben tener acceso a dicho parking, las tarjetas de los alumnos de económicas, sean trasvasados después de la impresión de los carnés.

Los carnés UMA, al ser tarjetas inteligentes, podrán servir para identificarse en las aplicaciones de la UMA a través del iDUMA siempre y cuando se les grabe en el chip su certificado personal de la FNMT. Una vez que la tarjeta tenga el certificado, podrá usarse para identificarse en iDUMA, de la misma manera que se puede autenticar con un DNI electrónico.

## **6.6.2 Identificación iDUMA con carné universitario.**

Las UMA's CARD, ofrece entre sus servicios la certificación electrónica para identificación y cifrado en Internet tanto dentro como fuera del campus, ya que en el chip de la tarjeta pueden grabarse certificados electrónicos.

Por supuesto, lo primero será que el alumno disponga de su certificado personal de la FNMT y exportarlo en un formato de fichero .pfx o .p12, que incluye la clave privada de cada uno, para llevarlo, por ejemplo en un pendrive, de forma temporal, a cualquiera de los Registros de la UMA, que son los servicios responsables de la gestión de las firmas electrónicas de la FNMT en nuestra universidad.

Si el alumno no tiene el certificado, en el Registro de la UMA podrán indicarle los trámites necesarios que debe hacer para obtener su certificado. Cuando tengan el certificado y su carné, sólo tendrán que pasarse por el Registro para que le graben el certificado personal de la FNMT en su carné universitario.

Así, luego, con los lectores de tarjeta y software adecuado instalado en su ordenador, los alumnos podrán usar su carné para identificarse en iDUMA en los servicios electrónicos que lo exigen y, también, para firmar documentos digitalmente.



### 6.6.2.1. Obtención certificado digital de FNMT.

El proceso de obtención del Certificado Digital se compone de tres pasos:

1. **Solicitud vía internet de su Certificado:** A través del siguiente enlace (<https://www.sede.fnmt.gob.es/certificados/persona-fisica/obtener-certificado-software/solicitar-certificado>). Debe introducir su NIF y enviar la petición, al finalizar el procedimiento de solicitud, obtendrá un código que deberá presentar en el momento de acreditar su identidad en el Registro de la UMA y posteriormente en el momento de la descarga del certificado.
2. **Acreditación de la identidad en un Registro de la UMA:** Debe acreditarse presencialmente en alguno de los Registros de la UMA, aportando el DNI y el código obtenido durante la solicitud vía internet. En el registro deberá firmar el contrato de la FNMT que le expedirán, una vez se haya identificado.
3. **Descarga de su Certificado Digital:** Aproximadamente 2 horas después de haber acreditado su identidad, haciendo uso del código de solicitud obtenido en el paso 2 y desde el mismo equipo y navegador desde el que realizó la solicitud, podrá descargar su certificado a través del siguiente enlace (<https://www.sede.fnmt.gob.es/certificados/persona-fisica/obtener-certificado-software/descargar-certificado>).

Recuerde que todo el procedimiento se realiza a través de la web de La Fábrica Nacional de Moneda y Timbre en el apartado certificados.

The screenshot shows the FNMT website interface. At the top, there is a navigation bar with links for FNMT, CERES, MUSEO CASA DE LA MONEDA, SIAEN, ESCUELA DE GRABADO, and TIENDA VIRTUAL. Below this is the 'Sede Electrónica' header for the Real Casa de la Moneda. The main content area is titled 'Persona Física' and includes a sidebar with options: 'Obtener Certificado Software', 'Obtener Certificado con Android', 'Obtener Certificado con DNIe', 'Verificar estado', 'Renovar', 'Anular', and 'Certificado de Representante'. The main content area contains the following text:

**Persona Física**

El Certificado digital FNMT de Persona la FNMT-RCM que vincula a su suscrij confirma su identidad.

Este certificado, también conocido com documento digital que contiene sus d Internet e intercambiar información cor que sólo Ud. y su interlocutor pueden a

**¿Quién puede obtener un Certif**

Cualquier ciudadano español o extran, esté en posesión de su DNI o NIE, pc forma gratuita para firmar y acreditar su

**¿Cómo puedo obtener el Certifi**

Existen 3 formas distintas para obtener:

The right side of the page shows the 'idUMA - Servicio de Identidad de la Universidad de Málaga' login interface. It features a 'Correo en UMA' field, a 'Contraseña' field, and an 'INICIAR SESIÓN' button. Below the login fields are two options: 'Soy usuario pero no puedo entrar' and 'Soy nuevo en la UMA'. To the right, there are 'OTROS MEDIOS DE AUTENTICACIÓN' including 'Certificado digital' and 'STORK'. At the bottom, there is a security warning: 'Nunca atienda las solicitudes de clave que le lleguen por correo electrónico. Para desconectarse, recomendamos que cierre su navegador (cerrando todas las ventanas).'

## 7. Conclusiones.

SOAP permite a los diseñadores encapsular la complejidad del sistema, dando lugar a interfaces generadas automáticamente que permiten facilitar el diseño del sistema.

Lo interesante de SOAP es que permite utilizar lenguajes de alto nivel para llamar y para implementar el servicio. Dado un fichero WSDL, los IDE actuales facilitan enormemente la tarea de programación de los servicios Web necesarios para la interconexión de sistemas informáticos diferentes. Lográndose un desarrollo rápido, limpio y eficaz.

Como ejemplo de este estudio, hemos visto cómo se realizó el desarrollo de los servicios web necesarios para interconectar los sistemas de información de la UMA y el Banco Santander. Para lo cual me decidí por desarrollar los servicios web en NetBIOS que es un IDE o Entorno de Desarrollo Integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo y es gratuito sin restricciones de uso.

NetBIOS es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, que permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos.

Me gustaría recordar que una Arquitectura Orientada a Servicios (SOA) es una arquitectura que define cómo interoperan funciones de negocios independientes implementados por sistemas autónomos para ejecutar un proceso de negocios. El diseño Web tiende a ser cada vez más modular y las aplicaciones se componen de una serie de componentes reutilizables, que pueden encontrarse distribuidos a lo largo de una serie de máquinas conectadas en red. Los Servicios Web nos permiten distribuir nuestra aplicación a través de Internet, definiendo como clave la interoperabilidad entre las aplicaciones.

Hoy en día, la mayoría de las empresas disponen de sistemas de información independientes, que ofrecen y consumen servicios en internet, logrando conectar opciones de negocio implementadas en sistemas de información muy diferentes entre sí. Pero el principal beneficio de los servicios web SOAP recae en ser fuertemente acoplado, lo que permite poder ser testado y depurado antes de poner en marcha la aplicación, lográndose un desarrollo ágil y efectivo. SOAP ha demostrado ser fácil de utilizar, con una depuración simple, donde las operaciones complejas pueden ser escondidas detrás de una fachada simple, incrementando la privacidad. Además de que gracias a los IDE, como hemos demostrado con NetBeans, el desarrollo de servicios web SOAP a partir de ficheros WSDL, es muy sencillo y rápido.



---

## 8. Bibliografía.

- **Programación Java Server con J2EE** Edición 1.3 (Anaya Multimedia/Wrox) de Subrahmanyam Allamaraju, Cedric Beust, John Davies, Tyler Jewell, Rod Johnson, Andrew Longshaw, Ramesh Nagappan, Alex Toussaint, Sammer Tyagi, Gary Watson, Mark Wilcox, Alan Willianson y Daniel O'Connor.
- **Revisión de los Servicios Web SOAP/REST** - [www.albertolsa.com/wp-content/uploads/2009/07/mdsw-revision-de-los-servicios-web-soap\\_rest-alberto-los-santos.pdf](http://www.albertolsa.com/wp-content/uploads/2009/07/mdsw-revision-de-los-servicios-web-soap_rest-alberto-los-santos.pdf)
- **Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI** por Curbera, F; Duftler, M; Khalaf, R; Nagy, W; Mukhi, N; Weerawarana, S. - [ieeexplore.ieee.org/jabega.uma.es/xpls/icp.jsp?arnumber=991449](http://ieeexplore.ieee.org/jabega.uma.es/xpls/icp.jsp?arnumber=991449)
- **Debate entre servicios web basados en REST y SOAP** - [carlosmayta.blogspot.com.es/](http://carlosmayta.blogspot.com.es/)
- **Scrum** - [proyectosagiles.org/que-es-scrum/](http://proyectosagiles.org/que-es-scrum/)
- **Guía Breve de Servicios Web** - [www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb](http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb)
- **Web Services Description Language (WSDL)** - [www.w3.org/TR/wsd120-primer/](http://www.w3.org/TR/wsd120-primer/)
- **Introduction to Web Services** - [netbeans.org/kb/docs/websvc/intro-ws.html](http://netbeans.org/kb/docs/websvc/intro-ws.html)
- **Scrum: metodologías ágiles** - [www.javiergarzas.com/metodologias-agiles](http://www.javiergarzas.com/metodologias-agiles)