DOCUMENTO RESUMIDO - DESCRIPCION DE ARQUITECTURA

Achilles

Junio 2014

Autores:
Agustín Castro - 153652
Ignacio Rognoni – 168334

Docentes:
Gastón Mousques
Damián Moretti

ÍNDICE

1. Introducción

1.1 *Propósito*

El propósito del presente documento es proveer una especificación completa de la arquitectura de la aplicación Achilles.

2. Antecedentes

2.1 *Propósito del sistema*

El propósito del sistema es administrar la compra de diferentes artículos de la empresa, seleccionando entre diversas formas de envío, así como también darle la permitirle a los compradores consultar y trackear sus compras. A lo largo del documento se especificarán con mayor detalle estas funcionalidades.

2.2 *Requerimientos significativos de Arquitectura*

2.2.1 Resumen de Requerimientos Funcionales

ID Requerimiento	Descripción	Actor
RF 1 Comprar artículo	*Permite a un cliente comprar un articulo*	*ClienteWEB*
RF 2 Consultar compras	*Permite a un cliente consultar sus compras anteriores*	*ClienteWEB*
RF 3 Ver historial de estados	*Permite a un cliente consultar los estados de la compra*	*ClienteWEB*
RF 4 Actualizar estado de envío	*Permite a un carrier o cliente cambiar el estado de envío de una compra*	*ClienteWEB, CarrierWEB*
RF 5 Conversor de monedas	*Permite a un usuario del sistema convertir el precio de un artículo en diferentes unidades*	*ClienteWEB*

2.2.2 Resumen de Requerimientos No Funcionales

ID Requerimiento	ID Requerimiento no Funcional	Descripción
RNF1	*Modificabilidad*	*El sistema debe estar preparado para aceptar futuras modificaciones en los diferentes módulos, así como también permitir la inclusión de nuevas funcionalidades*
RNF2	*Interoperabilidad*	*El sistema permite la comunicación entre diferentes módulos del mismo, los cuales trabajan con diferentes tecnologías. Entre estas, se encuentra la comunicación sincrónica y asincrónica*
RNF3	*Seguridad*	*El sistema cuenta con un log que registra todas las operaciones que se realizan, tanto si fracasan como si tienen éxito*
RNF4	*Disponibilidad*	*El sistema cuenta con un manejo de excepciones propias el cual aumenta la disponibilidad del mismo*
RNF5	*JEE*	*El sistema está implementado en lenguaje Java utilizando las tecnologías de JEE*
RNF6	*Glassfish Server*	*El servidor de aplicaciones utilizado por el sistema es Glassfish v4.0*
RNF7	*MySQL*	*La base de datos utilizada es una base de datos relacional MySQL*
RF8	*Web Services REST*	*Los web services implementados en esta aplicación utilizan tecnologías REST*

3. Documentación de la arquitectura

En esta sección se especifica toda la documentación de arquitectura del sistema. Para esto, consideramos las vistas que fueron tratadas en el curso, siendo estas las más relevantes para nuestra arquitectura.

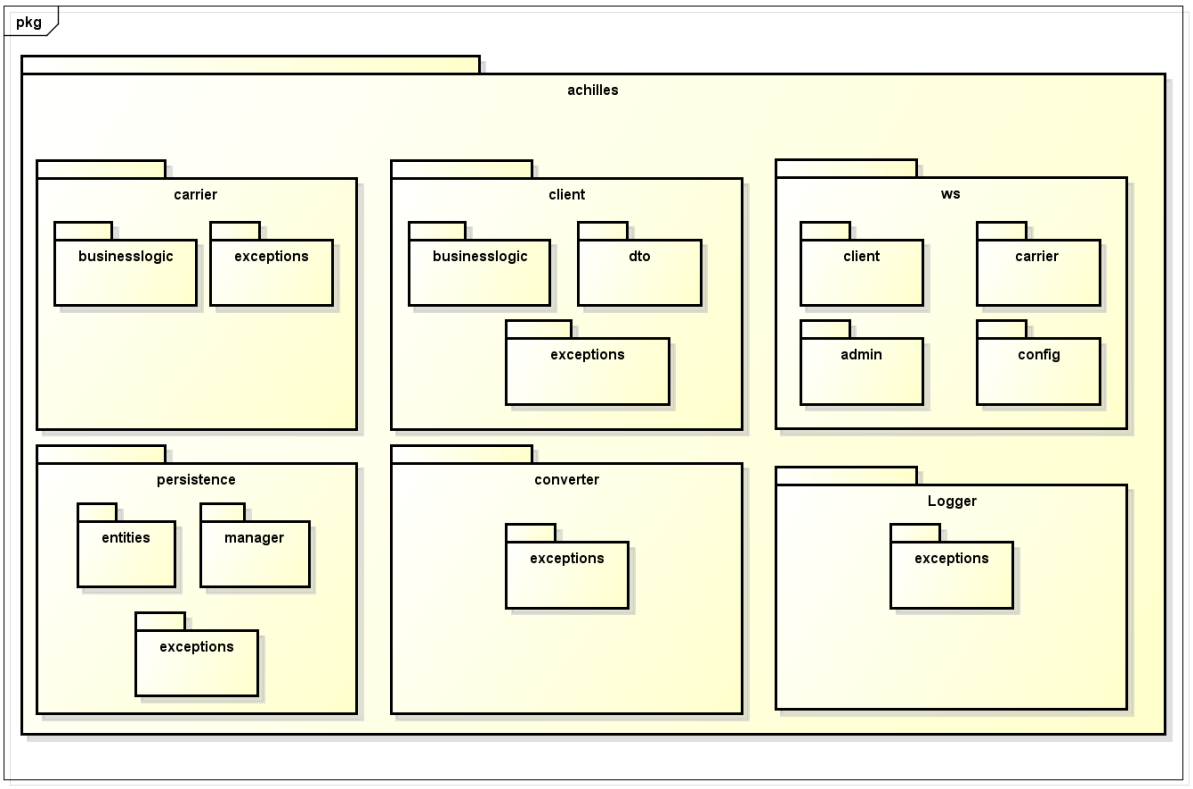
3.1 *Vistas de Módulos*

3.1.1 Vista de Descomposición

La vista de descomposición es la primera aproximación al sistema considerando los diferentes módulos del mismo. En ella podemos ver la estructura total de paquetes del sistema y como los mismos están organizados.

3.1.1.1 Representación primaria

Aclaramos que todos los paquetes representados en esta vista, así como también todos los paquetes involucrados en este proyecto se encuentran dentro del paquete uy.edu.ort. Para no crear confusión, ni tampoco para hacer que el diagrama sea engorroso no se incluye en la notación de los paquetes.



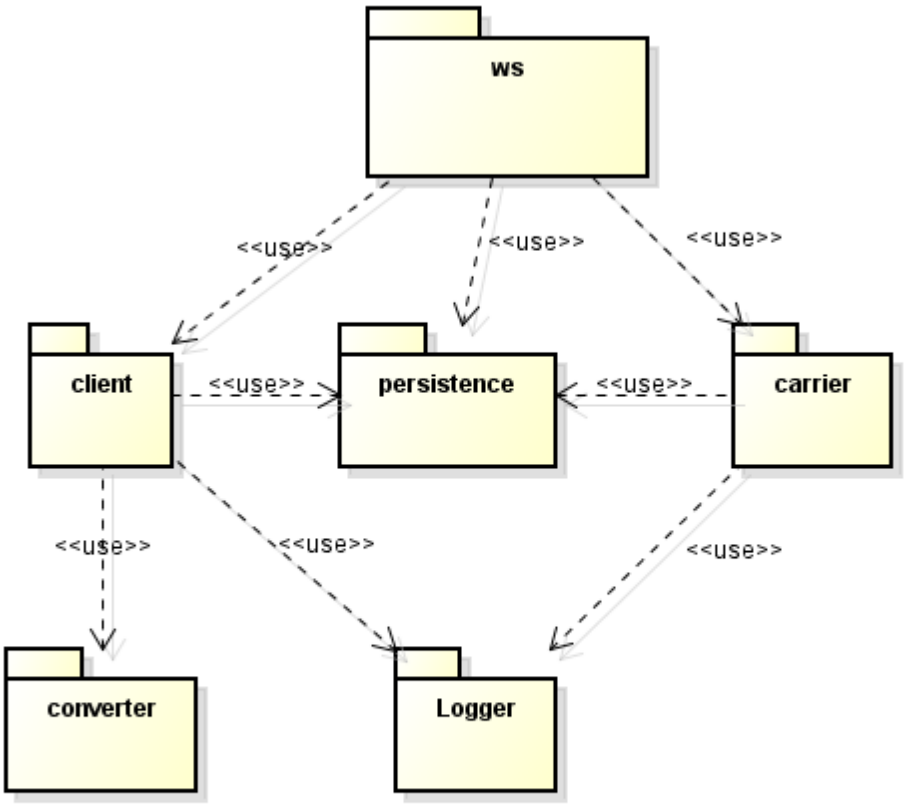
3.1.1.2 Decisiones de diseño

En esta vista se puede observar que dividimos los paquetes de forma de cumplir con el principio de clausura común, donde se establece que los paquetes tienen solo una razón para cambiar.

3.1.2 Vista de Uso

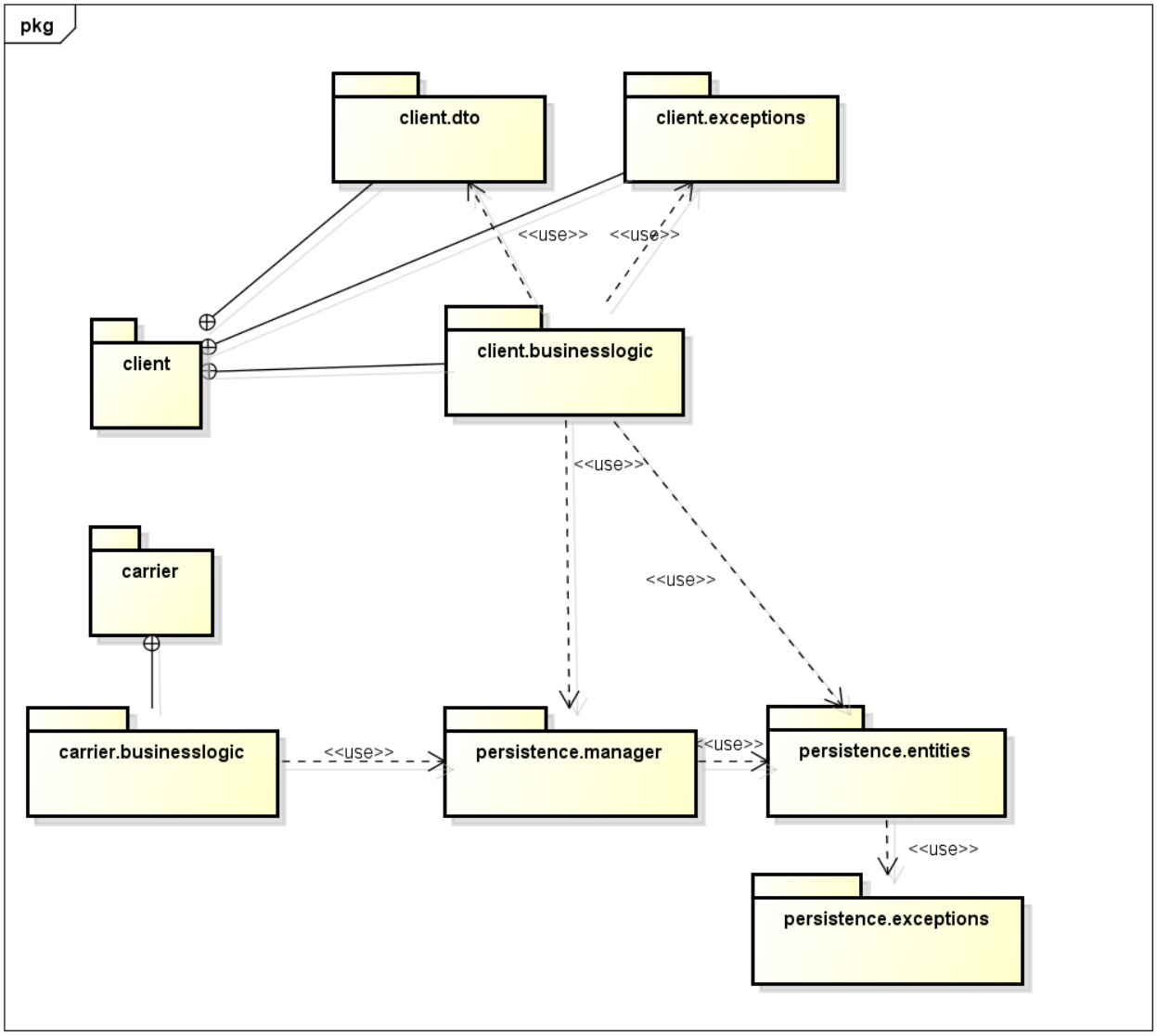
La vista de usos describe la relación entre los paquetes del sistema, así como también las diferentes responsabilidades de los mismos ya que podemos ver las dependencias entre cada uno de ellos.

3.1.2.1 Representación primaria



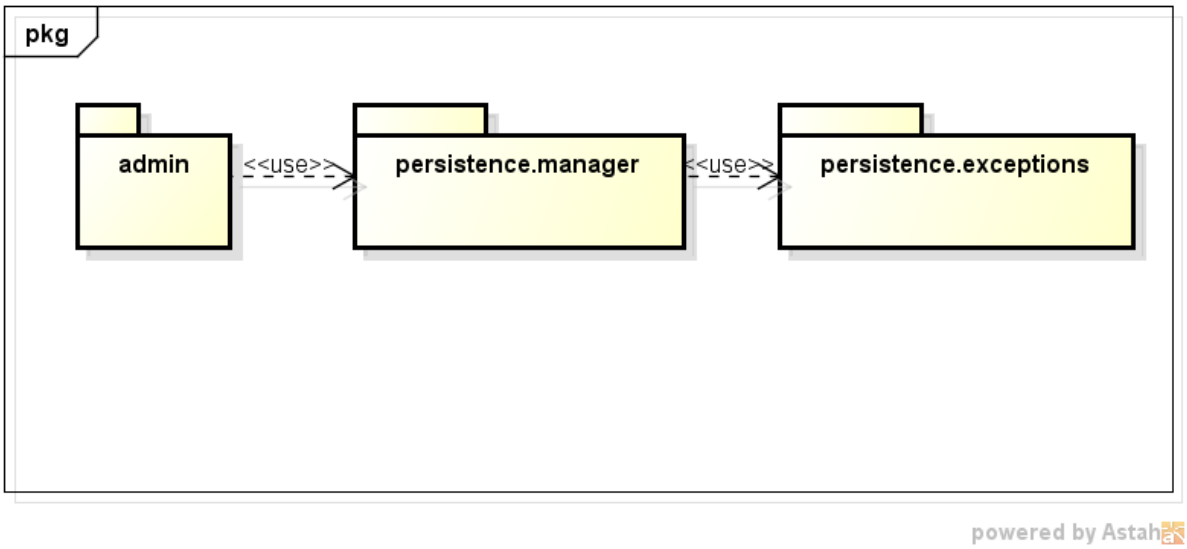
3.1.2.2 Vista de uso – Operaciones del carrier y el cliente

Esta vista de uso representa las dependencias entre los paquetes de cliente y carrier con respecto a la persistencia, de forma de guardar las entidades en la base de datos.



3.1.2.3 Vista de uso – Operaciones de administrador

Como se mencionará en las decisiones de diseño de esta sección, los recursos del administrador (artículo y opción de envío), se comunican directamente con el manejador del acceso a datos. Esta vista de uso muestra la dependencia mencionada.



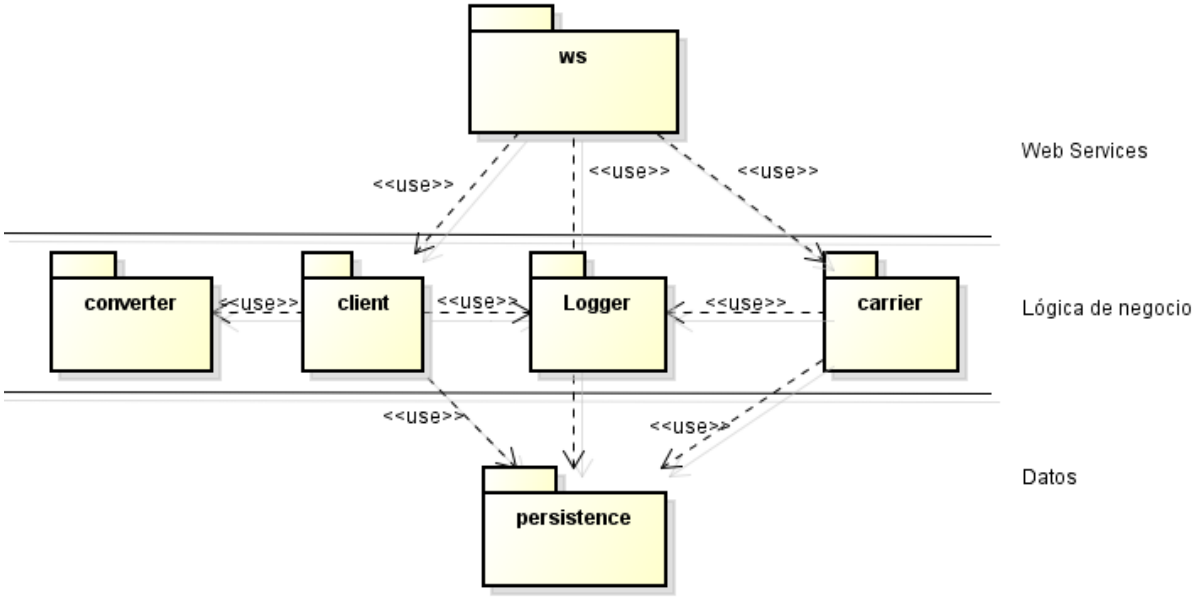
3.1.2.4 Decisiones de diseño correspondientes a la vista de usos

En el diagrama se puede ver que el paquete correspondiente a los Web services utiliza de forma directa al paquete de la persistencia. Esto viola el diseño en capas que planteamos más adelante. La razón por la cual existe esta dependencia es para satisfacer el requerimiento de poder agregar artículos y estados de envío en tiempo de ejecución. Este requerimiento surgió como recomendación para la defensa del obligatorio, y no como requerimiento funcional del sistema. Si hubiese sido de esta forma, agregaríamos una capa intermedia que manejara estas entidades, de forma análoga a como lo hace el cliente y el carrier con sus operaciones. Es por esa razón que, para esta instancia, no nos parece necesario contar con esa capa intermedia.

3.1.3 Vista de Layers

La vista de Layers o capas lógicas agrupa los módulos del sistema por responsabilidades. En nuestro caso, la capa de presentación será implementada por un sistema externo, pero tendremos sí una capa correspondiente a la lógica de negocios y una correspondiente al acceso a datos.

3.1.3.1 Representación primaria



3.1.3.2 Decisiones de diseño correspondientes a la vista de layers

En un principio pensamos agrupar el convertidor y el log en un único paquete y llamarlo utilidades ya que los mismos agrupan operaciones que están por fuera de las requeridas por el sistema, y parecerían ser más de apoyo. Descartamos luego esa idea, ya que mantener la independencia entre el logger y el conversor nos agrega mayor modificabilidad, permitiendo cambiar alguna de éstas por otra implementación.

3.1.4 Catálogo de elementos

Elemento	Responsabilidades
Ws	*Contiene todos los web services del sistema*
Ws.client	*Contiene el web service que utiliza el cliente para realizar las operaciones de compra y listado.*
Ws.carrier	*Contiene el web service que utilizan los carriers para realizar la operación de modificar estado*
Ws.admin	*Contiene el web service que utiliza el administrador para agregar los artículos y los estados de envío de forma dinámica*
Ws.config	*Contiene el archivo de configuración de todos los web service del sistema*
Client	*Contiene toda la lógica correspondiente al manejo del cliente. Esto incluye las validaciones de negocio, los DTO y las excepciones*
Client.businesslogic	*Contiene la lógica del negocio de las operaciones correspondientes al cliente*
Client.dto	*Contiene los DTOs utilizados para enviar información al web service*
Client.exceptions	*Contiene las excepciones relacionadas con el cliente*
Carrier	*Contiene toda la lógica correspondiente al manejo del carrier. Esto incluye las validaciones de negocio y las excepciones.*
Carrier.businesslogic	*Contiene la lógica del negocio de las operaciones correspondientes al carrier*
Carrier.exceptions	*Contiene las excepciones relacionadas con el carrier*
Persistence	*Contiene todas las entidades persistibles del sistema, así como también el manejo de acceso a datos*
Persistence.entities	*Contiene las entidades del sistema*

Persistence.manager	*Contiene el manejo de acceso a datos*
Persistence.exceptions	*Contiene las excepciones relacionadas con el acceso a datos*
Converter	*Contiene la lógica del convertidor de unidades*
Converter.exceptions	*Contiene las excepciones del convertidor de unidades*
Logger	*Contiene la lógica del Log4j.*
Logger.exceptions	*Contiene las excepciones del logger*
Achilles	*Contiene toda la aplicación desarrollada*

3.1.5 Interfaces

En esta vista no se describe ningún detalle referente a interfaces. Todo lo referente a las mismas está documentado en la sección de interfaces de la vista de Componentes y Conectores.

3.1.6 Comportamiento

En esta vista no se describe ningún detalle referente a los diagramas de comportamiento. Todo lo referente a los mismos está documentado en la sección de comportamiento de la vista de Componentes y Conectores.

3.1.7 Decisiones de diseño

A continuación describiremos las decisiones de diseño tomadas que se pueden apreciar en los diferentes diagramas de la vista de módulos.

Modularización del sistema

Como se puede ver en las diferentes vistas presentadas anteriormente, decidimos dividir el sistema en diferentes módulos, cada uno con una única responsabilidad bien definida, con el fin de aumentar principalmente la modificabilidad del sistema. Además, la división de acuerdo a módulos permite que los mismos puedan ser reusables en un futuro. A continuación detallamos cada uno de los módulos.

Achilles-WAR - Web Services (WS)
Este módulo contiene los web services que exponemos para que el cliente pueda, a través del navegador, realizar cualquiera de las operaciones descriptas en los requerimientos funcionales. Como se puede ver en la vista de descomposición, dentro del módulo organizamos los RESTful web services en paquetes de acuerdo a los roles. Por lo tanto, las operaciones del cliente, carrier y administrador caen dentro de cada uno de los paquetes correspondientes.

Client-EJB
Este módulo contiene todo lo relacionado al manejo del cliente desde el punto de vista lógico. En primer lugar, se reciben las peticiones del web service correspondiente al cliente. Luego, dentro del módulo nos encargamos de validar que los datos sean correctos (o lanzamos excepciones en caso contrario). Una vez realizadas las validaciones

correspondientes a la lógica de negocio, se comunica con la persistencia para realizar las operaciones que sean necesarias.

Carrier-EJB
Este módulo es análogo al módulo del Client-EJB. En él se encuentra todo lo relacionado al manejo lógico de las operaciones del Carrier. El mismo recibe las peticiones del web service correspondiente al carrier, realiza las validaciones y se comunica con el acceso a datos para completar las operaciones necesarias.

Persistence-EJB
Este módulo es el encargado del acceso y modificación de datos. El módulo contiene además todas las entidades persistibles de la aplicación. Es el encargado de crear las nuevas instancias de entidades que se vayan generando, así como también de obtener los datos de acuerdo a las consultas que se realizan. Más específicamente, en este módulo se persisten los datos de una nueva compra (RF1) y de un nuevo estado de envío (RF4), además de poder consultar por las compras de un cliente (RF2) y el histórico del mismo (RF3).

CurrencyConverter-EJB
Como su nombre lo indica, este módulo es el encargado de realizar la conversión entre monedas.

Logger-EJB
En este módulo se lleva el registro de las operaciones realizadas durante la ejecución del programa.

Uso de DTOs

Utilizamos Data Transfer Objects (DTOs) para que la capa más alta del sistema (web services) no tenga que conocer la implementación de nuestras entidades. Esto tiene varios beneficios. El primero, es que no consideramos beneficioso que se conozcan las entidades del sistema, ya que estaríamos violando el principio de separación en capas, haciendo que la capa superior tenga una dependencia a la capa inferior. Esto también lleva a que si en algún momento cambian las entidades del negocio habría un mayor impacto en todo el sistema, perdiendo así modificabilidad. Además, desde un punto de vista de performance, los DTOs nos permiten crear objetos más “livianos” que los originales, ya que estamos pasando los datos que obligatoriamente necesitamos, descartando aquellos que no aportan valor a la solución.

Excepciones propias

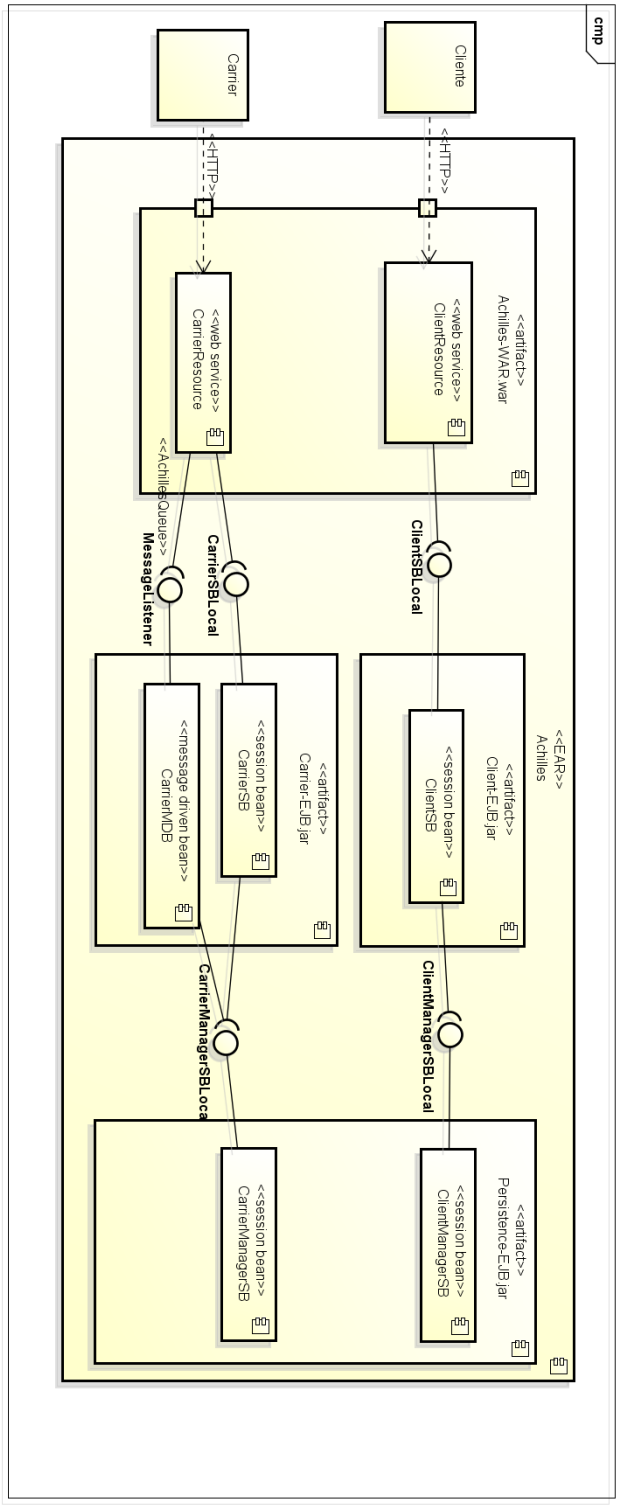
Como se puede ver en los diagramas, creamos un paquete de excepciones para cada módulo. La idea es agrupar la excepción en el mismo módulo que se encuentra el componente que la lanza. Esto nos permite ubicar con mayor precisión donde es que se localiza la excepción. Además, el hecho de ubicar las excepciones de esta forma aumenta de forma considerable la modificabilidad, ya que cuando cambia alguna de las reglas de negocio podemos modificar las excepciones de un módulo sin necesidad de impactar en el resto de los módulos del sistema.

3.2 *Vistas de Componentes y conectores*

Esta sección describe las vistas de componentes y conectores se consideran relevantes para comunicar la visión del sistema en tiempo de ejecución. En particular se describen los componentes, las formas de conexión y la interacción entre los componentes para explicar la implementación de funcionalidades o de mecanismos claves.

3.2.1 Diagrama de componentes y conectores

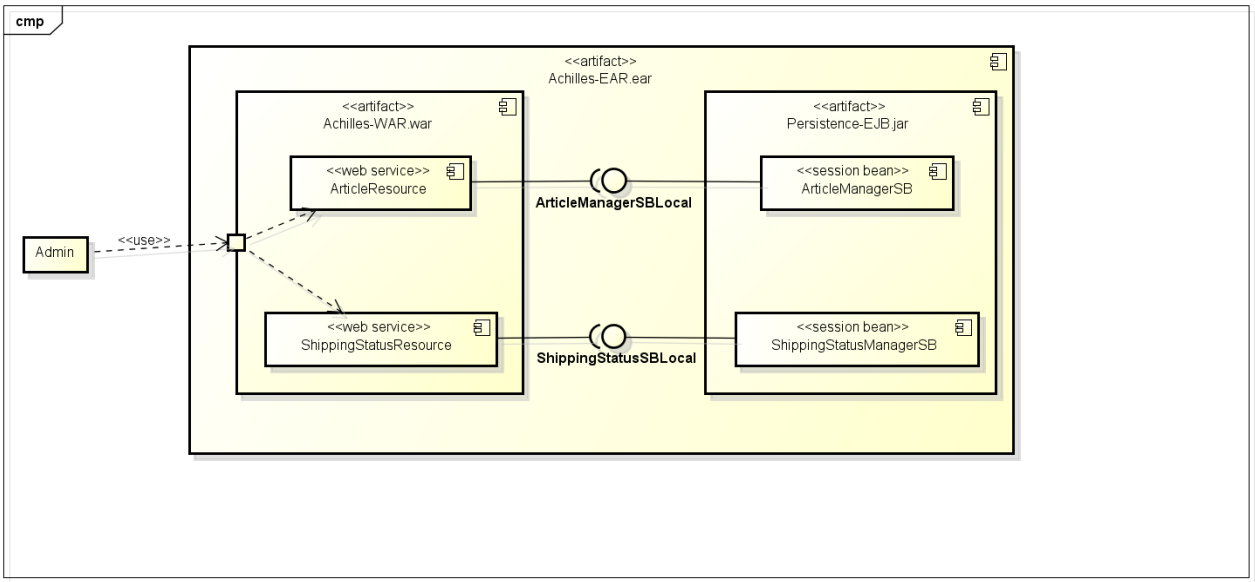
3.2.1.1 Representación primaria



Aclaración: Este diagrama de representación primaria no muestra la interacción entre los EJB de Cliente y Carrier con el logger, así como tampoco la interacción entre el EJB Cliente y el Conversor de monedas. La principal razón fue para no agregar complejidad al diagrama, ya que lo que nos interesa mostrar es como un cliente o carrier, a través de un navegador, se conecta con la aplicación.

Tampoco se muestra como agregamos de forma dinámica los artículos y los estados de envío. Esto, como se explicó anteriormente, no forma parte de las funcionalidades pedidas y por lo tanto para su implementación se violaron ciertos principios. A continuación mostramos el diagrama de componentes y conectores para este caso en particular.

3.2.1.2 Vista de componentes y conectores para artículo y estado de envío



En este diagrama sí se puede apreciar claramente como saltamos intencionalmente la capa de negocio para comunicar el web service directamente con el acceso a datos. Como ya se mencionó y vale la pena reiterarlo, lo hacemos sabiendo que estamos violando ciertos principios de diseño.

3.2.1.3 Catálogo de elementos

Componente/conector	Tipo	Descripción
Cliente	*Artefacto*	*El cliente representa un actor que realiza una operación en el sistema correspondiente al rol del cliente. Decimos que es una caja negra ya que se encuentra por fuera de la frontera de nuestro sistema y no nos importa cómo funciona.*
Carrier	*Artefacto*	*Idem cliente*
ClientResource	*Componente Web service*	*Contiene los servicios correspondientes a las operaciones del cliente.*
CarrierResource	*Componente Web service*	*Contiene los servicios correspondientes a las operaciones del carrier*
Achilles-WAR	*Componente WAR*	*Agrupa todos los web services que expone el sistema*
Client-EJB	*Componente EJB*	*Módulo EJB que agrupa la capa de negocio del cliente*
ClientSB	*Componente Session bean*	*Session bean que implementa la interfaz ClientSBLocal, que permite la comunicación entre el Web Service y la lógica del cliente*
Carrier-EJB	*Componente EJB*	*Módulo EJB que agrupa la capa de negocio del carrier*
CarrierSB	*Componente Session Bean*	*Session bean que implementa la interfaz CarrierSBLocal, que permite la comunicación entre el Web Service y la lógica del carrier*
CarrierMDB	*Componente Message Driven Bean*	*MDB que implementa la interfaz MessageListener, que permite la comunicación asincrónica entre el web service y la lógica del carrier*
Persistence-EJB	*Componente EJB*	*Módulo EJB que agrupa todo lo relacionado a las entidades del sistema y el manejo de acceso a datos*
ClientManagerSB	*Componente Session bean*	*Session bean que implementa la interfaz ClientManagerSBLocal, que permite la comunicación entre la lógica del cliente y el acceso a datos*
CarrierManagerSB	*Componente Session bean*	*Session bean que implementa la interfaz CarrierManagerSBLocal, que permite la comunicación entre la lógica del cliente y el acceso a datos*
ArticleResource	*Componente Web service*	*Contiene los servicios correspondientes al artículo*
ShippingStatusResource	*Componente Web service*	*Contiene los servicios correspondientes al estado de envío*
ArticleManagerSB	*Componente Session Bean*	*Session bean que implementa la interfaz ArticleManagerSBLocal, que permite la comunicación entre el Web service y el acceso a datos*
ShippingStatusManagerSB	*Componente Session Bean*	*Session bean que implementa la interfaz ShippingStatusManagerSBLocal, que permite la comunicación entre el Web Service y el acceso a datos*

3.2.1.4 Interfaces

Interfaz: ClientSBLocal	
Componente que la provee: Client-EJB.jar	
Servicio	***Descripción***
purchaseArticle	*Permite la comunicación entre el web service y la capa de negocio del cliente para comprar un artículo*
showOrders	*Permite la comunicación entre el web service y la capa de negocio del cliente para ver la lista de artículos del cliente*
currencyConverstion	*Permite la comunicación entre el web service y la capa de negocio del cliente para convertir unidades*
getHistory	*Permite la comunicación entre el web service y la capa de negocio del cliente para ver el histórico de una compra*

Interfaz: CarrierSBLocal	
Componente que la provee: Carrier-EJB.jar	
Servicio	***Descripción***
updateShippingStatus	*Permite la comunicación entre el web service y la capa de negocio del carrier para actualizar el estado de una compra.*

Interfaz: ClientManagerSBLocal	
Componente que la provee: Persistence-EJB	
Servicio	***Descripción***
availableArticle	*Permite la comunicación entre la capa lógica del cliente y los datos para comprobar si el artículo está disponible*
validClient	*Permite la comunicación entre la capa lógica del cliente y los datos para comprobar si el cliente es válido*
registerPurchase	*Permite la comunicación entre la capa lógica del cliente y los datos para efectuar una compra*
findOrders	*Permite la comunicación entre la capa lógica del cliente y los datos para obtener las compras de un cliente*
findHistory	*Permite la comunicación entre la capa lógica del cliente y los datos para obtener el histórico de compras de una compra*

Interfaz: CarrierManagerSBLocal	
Componente que la provee: Persistence-EJB	
Servicio	***Descripción***
updateShippingStatus	*Permite la comunicación entre la capa lógica del carrier y los datos para actualizar el estado de envío de una compra*
belongsToEnterprise	*Permite la comunicación entre la capa lógica del carrier y los datos para comprobar si el carrier que realiza la actualización pertenece a la empresa*
validCarrier	*Permite la comunicación entre la capa lógica del carrier y los datos para comprobar si el carrier es válido*
validStatusChange	*Permite la comunicación entre la capa lógica del carrier y los datos para comprobar si se está*

	cambiando a un estado válido
validUserChanging	*Permite la comunicación entre la capa lógica del carrier y los datos para comprobar si el usuario puede realizar el cambio de estado*

Interfaz: ArticleManagerSBLocal	
Componente que la provee: Persistence-EJB	
Servicio	***Descripción***
createArticle	*Permite la comunicación entre el web service y la capa de persistencia para crear un artículo en tiempo de ejecución*
getArticles	*Permite la comunicación entre el web service y la capa de persistencia para obtener los artículos del sistema*

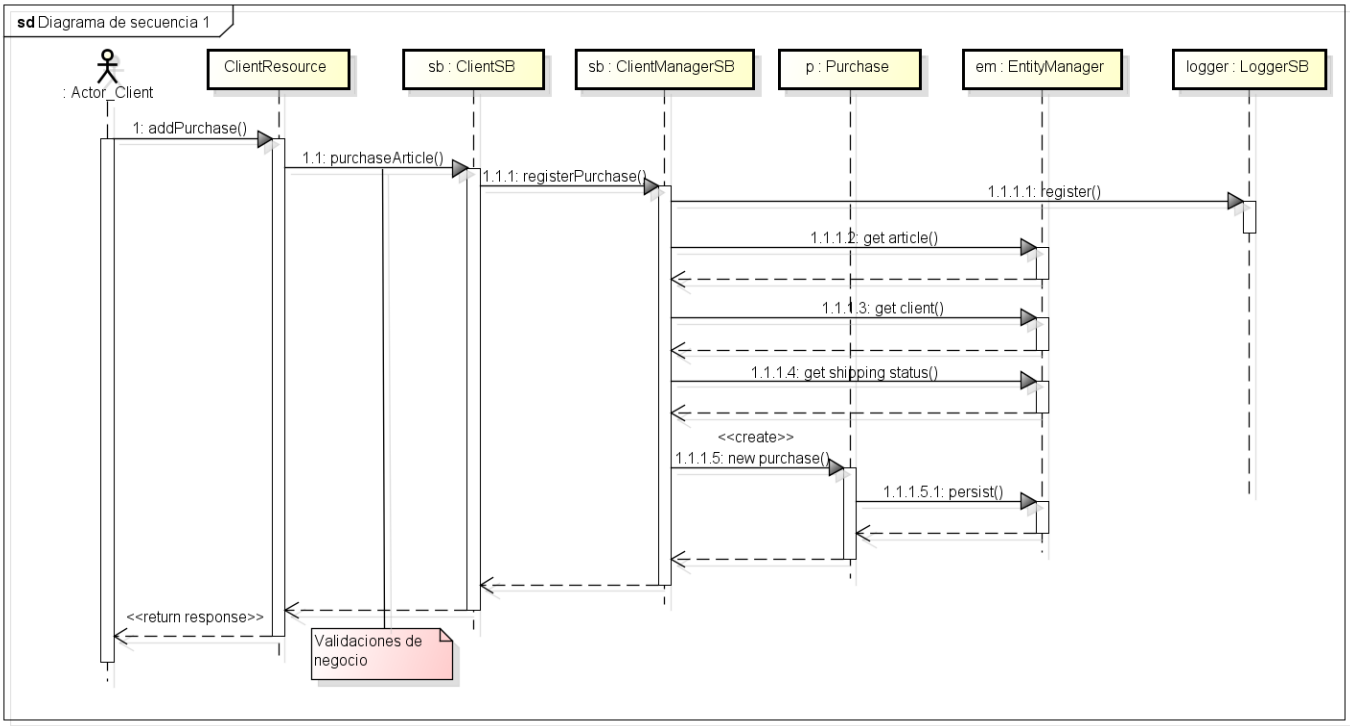
Interfaz: ShippingStatusManagerSBLocal	
Componente que la provee: Persistence-EJB	
Servicio	***Descripción***
createShippingStatus	*Permite la comunicación entre el web service y la capa de persistencia para crear un estado de envío en tiempo de ejecución*
getShippingStatus	*Permite al comunicación entre el web service y la capa de persistencia para obtener los estados de envío del sistema*

3.2.1.5 Comportamiento

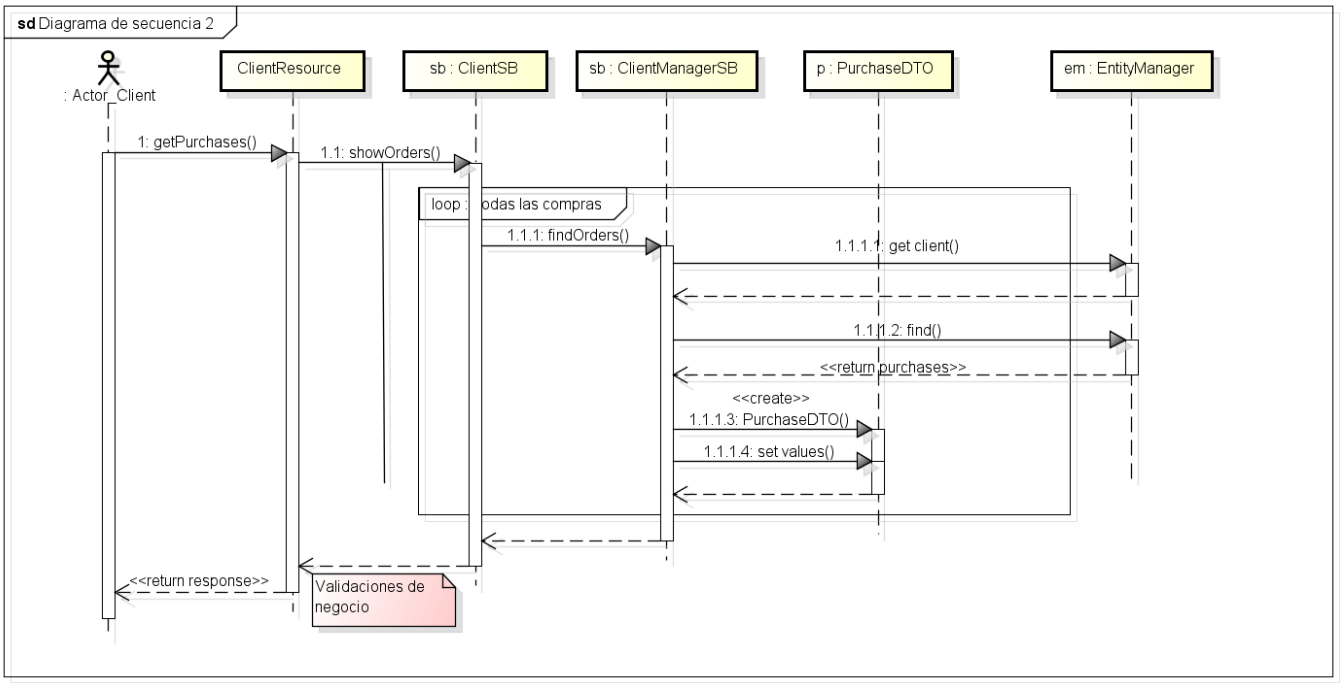
En esta sección se describen los diagramas de secuencia correspondientes a los principales casos de uso descriptos anteriormente.

Caso de uso 1 – Comprar artículo

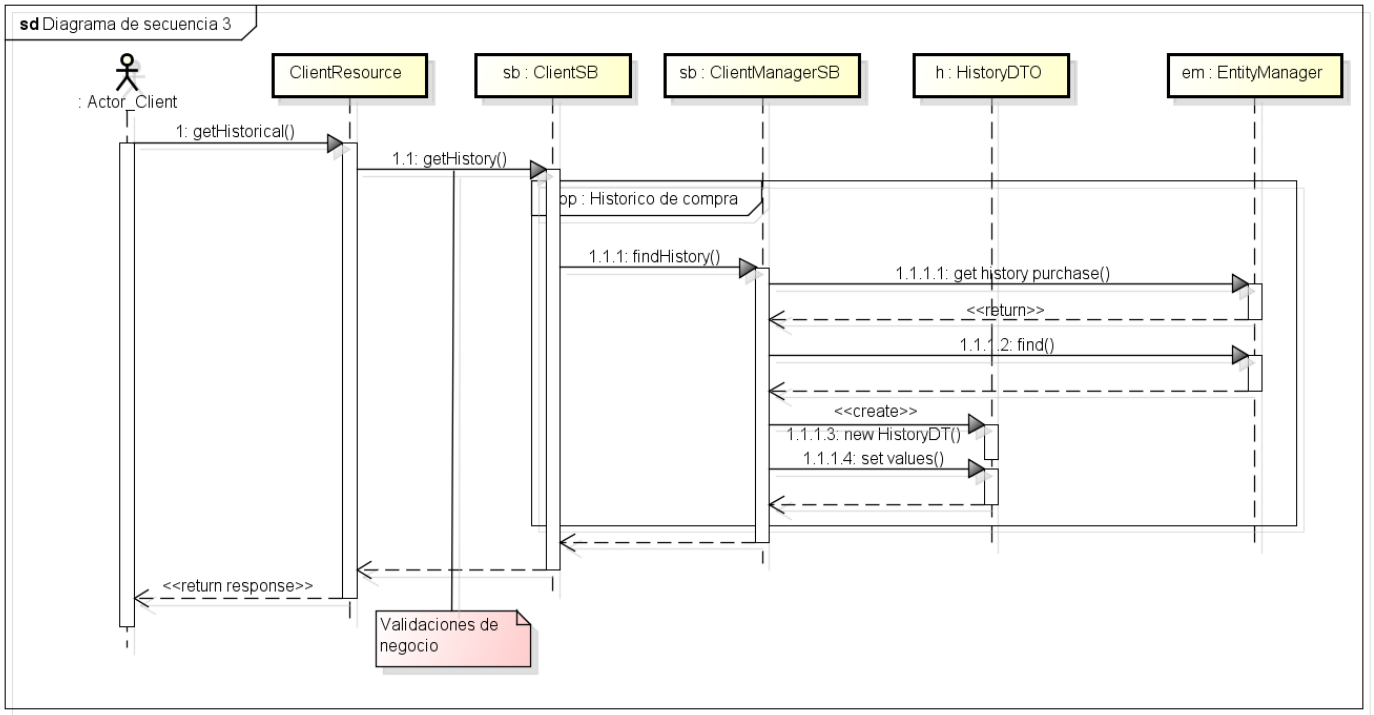
Como se describe en una nota en el diagrama, el session bean correspondiente a la lógica del negocio del cliente realiza validaciones varias. Las mimas no se incluyen en el diagrama ya que no le aportan valor al flujo de la operación, sino que hacen que el diagrama sea más engorroso.



Caso de uso 2 – Consultar compra



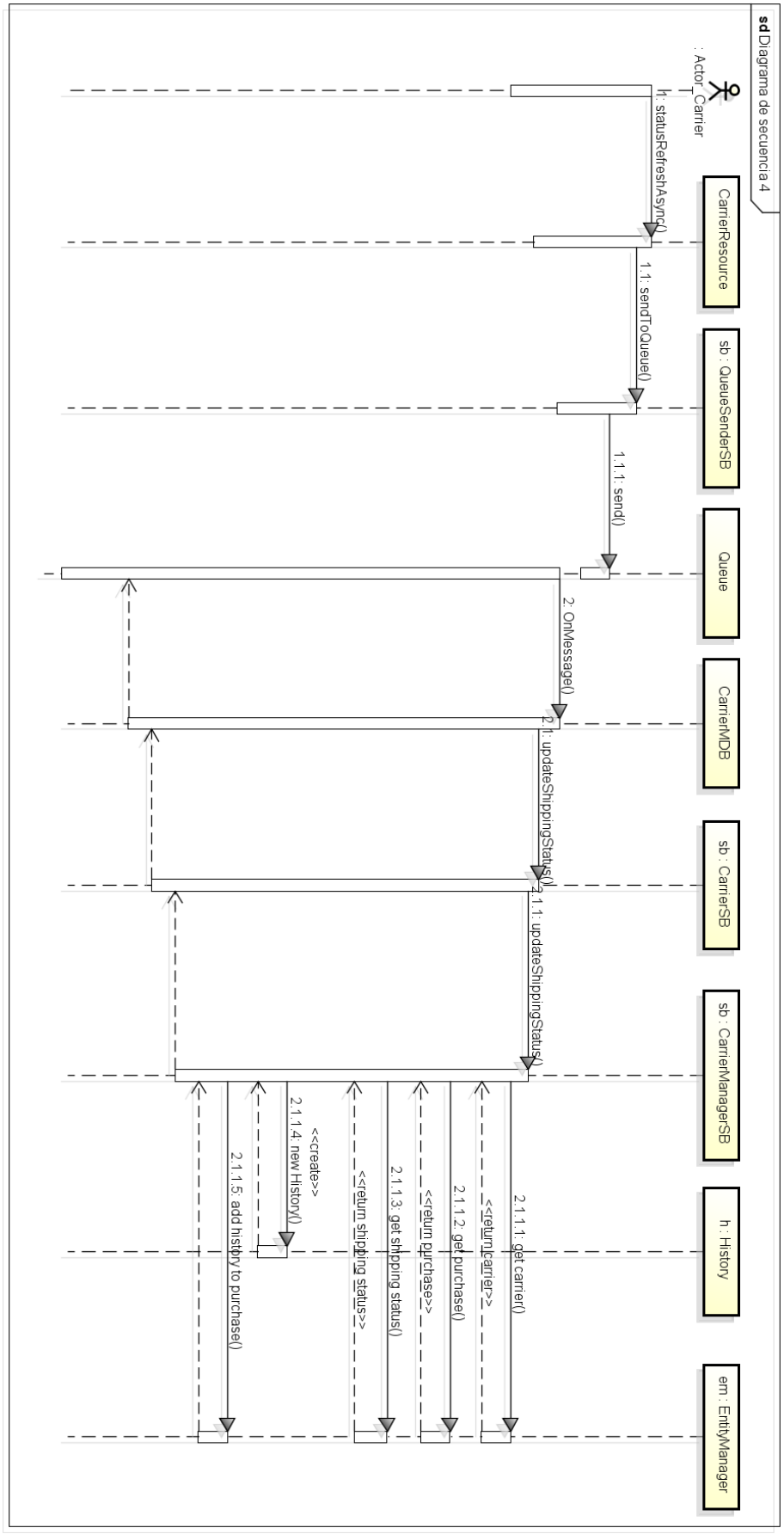
Caso de uso 3 – Ver historial de estados



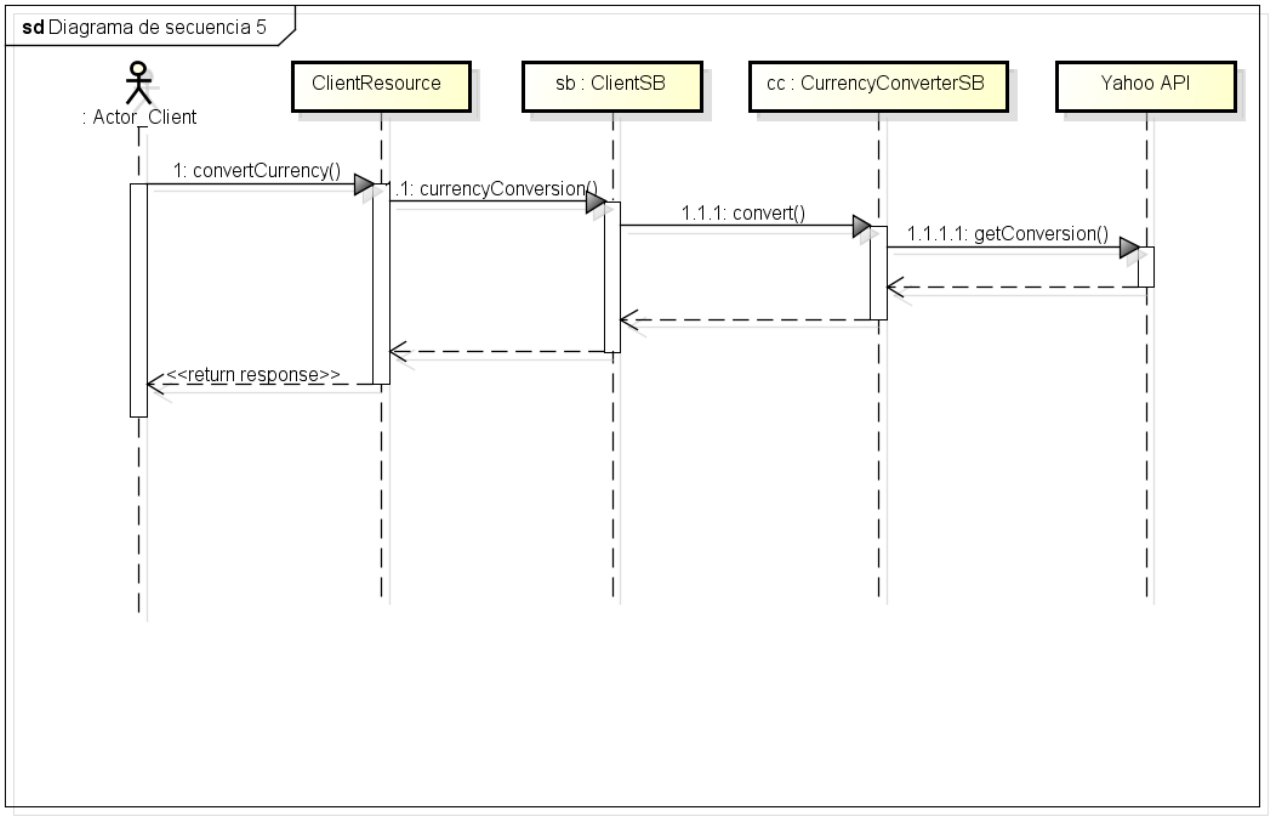
Caso de uso 4 – Actualizar estado de envío

Se muestra, como operación más significativa del cambio de estado el que realiza el carrier. El usuario también está habilitado a cambiar el estado para el caso en el cual acepta la compra. El mismo es prácticamente análogo a este diagrama, salvo que el actor es el cliente y se realizan otro tipo de validaciones.

Además, se muestra el diagrama de secuencia del cambio de estado de forma asincrónica, ya que es la única operación que se realiza de esta forma y es bueno ilustrarla de esta forma. En el caso de la actualización de estado de forma sincrónica, el flujo es similar pero sin pasar por la queue, si no directamente pasa por un sesión bean.


sd Diagrama de secuencia 4
: Actor Carrier
h: statusRefreshAsync()
CarrierResource
1.1: sendToQueue()
sb : QueueSenderSB
1.1.1: send()
Queue
2: OnMessage()
CarrierMDB
2.1: updateShippingStatus()
sb : CarrierSB
2.1.1: updateShippingStatus()
sb : CarrierManagerSB
2.1.1.1: get carrier()
<<return carrier>>
h : History
2.1.1.2: get purchase()
<<return purchase>>
2.1.1.3: get shipping status()
<<return shipping status>>
em : EntityManager
2.1.1.4: new History()
<<create>>
2.1.1.5: add history to purchase()

Caso de uso 5 – Conversor de monedas



3.2.1.6 Decisiones de diseño

Uso de la cola de mensajes

Para resolver el problema de la consumición asincrónica de mensajes por el componente Carrier EJB utilizamos un message driven bean que lee mensajes de una cola y los delega al mismo session bean que implementa la lógica de la consumición sincrónica de cambios de estado de envió por los carriers. Cabe destacar los mensajes de la cola se descartan una vez que son consumidos y se asume que son enviados con un formato especifico (un conjunto de Id´s necesarias por la lógica separadas por un "#") ya que el message driven bean parsea estos mensajes para utilizar esos datos.
Otro punto importante es que en nuestra solución particular tenemos dentro del componente Achilles-WAR tenemos un servicio web REST que se encarga de mandar mensajes a la cola con fines de prueba o control. Cabe destacar que en etapa de producción los mensajes llegaría a la cola por otros medios diversos que nosotros manejaremos de la misma forma con el message driven bean siempre que los mensajes respeten el formato establecido.

Uso de session beans y message driven beans

Un punto que es necesario notar y que consideramos que en la vista de componentes y conectores es donde mejor se observa es el uso de session beans y message driven beans con interfaz para exponer e implementar la lógica de negocio así como las interacciones con la base de datos. La gran ventaja de utilizar session beans y MDB´s es que estos se instancian a demanda escalando horizontalmente para atender los pedidos.
Esta característica juga un buen papel en la disponibilidad del sistema ya que no corremos el riesgo de que una instancia se vea acaparada por un usuario del sistema.
Otra ventaja es la facilidad que tenemos para trabajar con session beans ya que mediante inyección de dependencia podemos referenciarlos utilizando su interfaz.

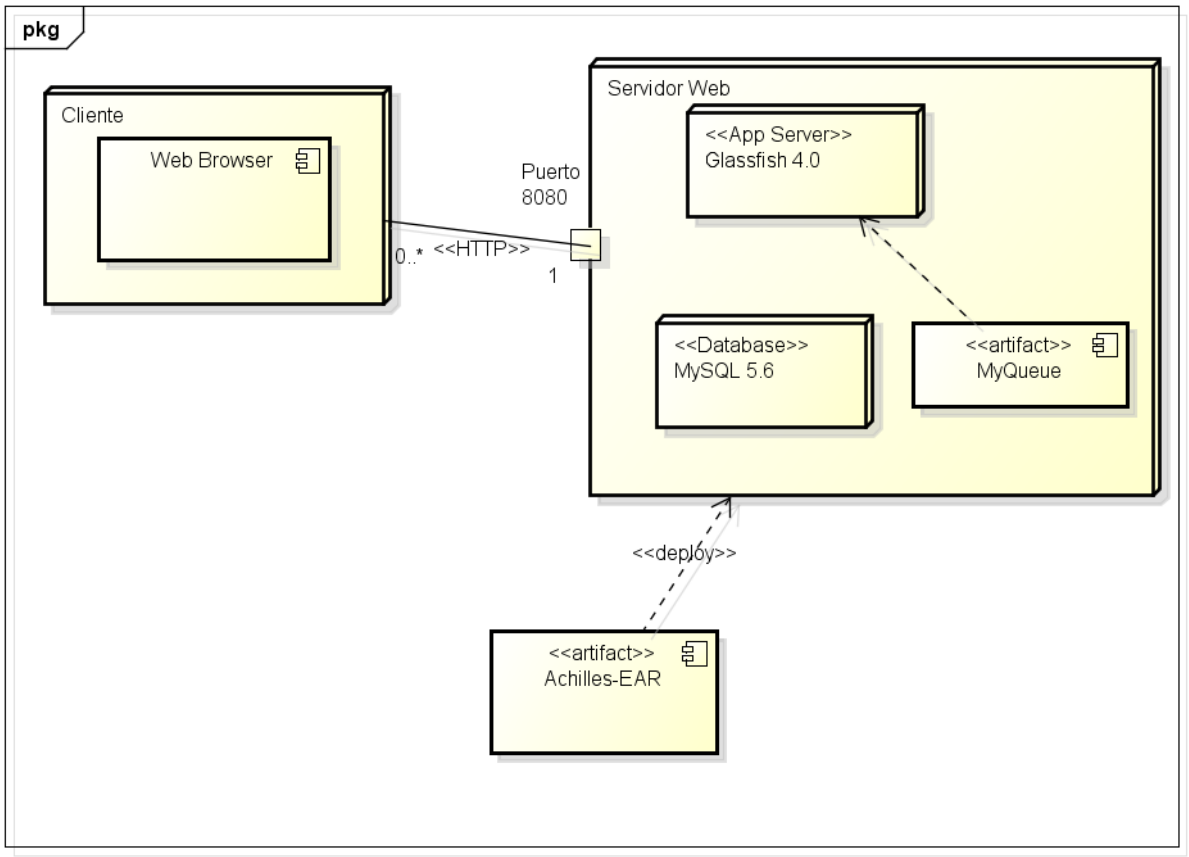
3.3 *Vistas de Asignación*

Esta sección describe como se relacionan los elementos de software con el ambiente en el cual son ejecutados. En particular se muestran los diferentes componentes del sistema y como se despliegan en los nodos físicos.

3.3.1 Vista de Despliegue

En esta vista se muestra como la aplicación EAR se despliega en el nodo del servidor web, mientras que el cliente se conecta a través del navegador mediante el protocolo HTTP.

3.3.1.1 Representación primaria



3.3.1.2 Catálogo de elementos

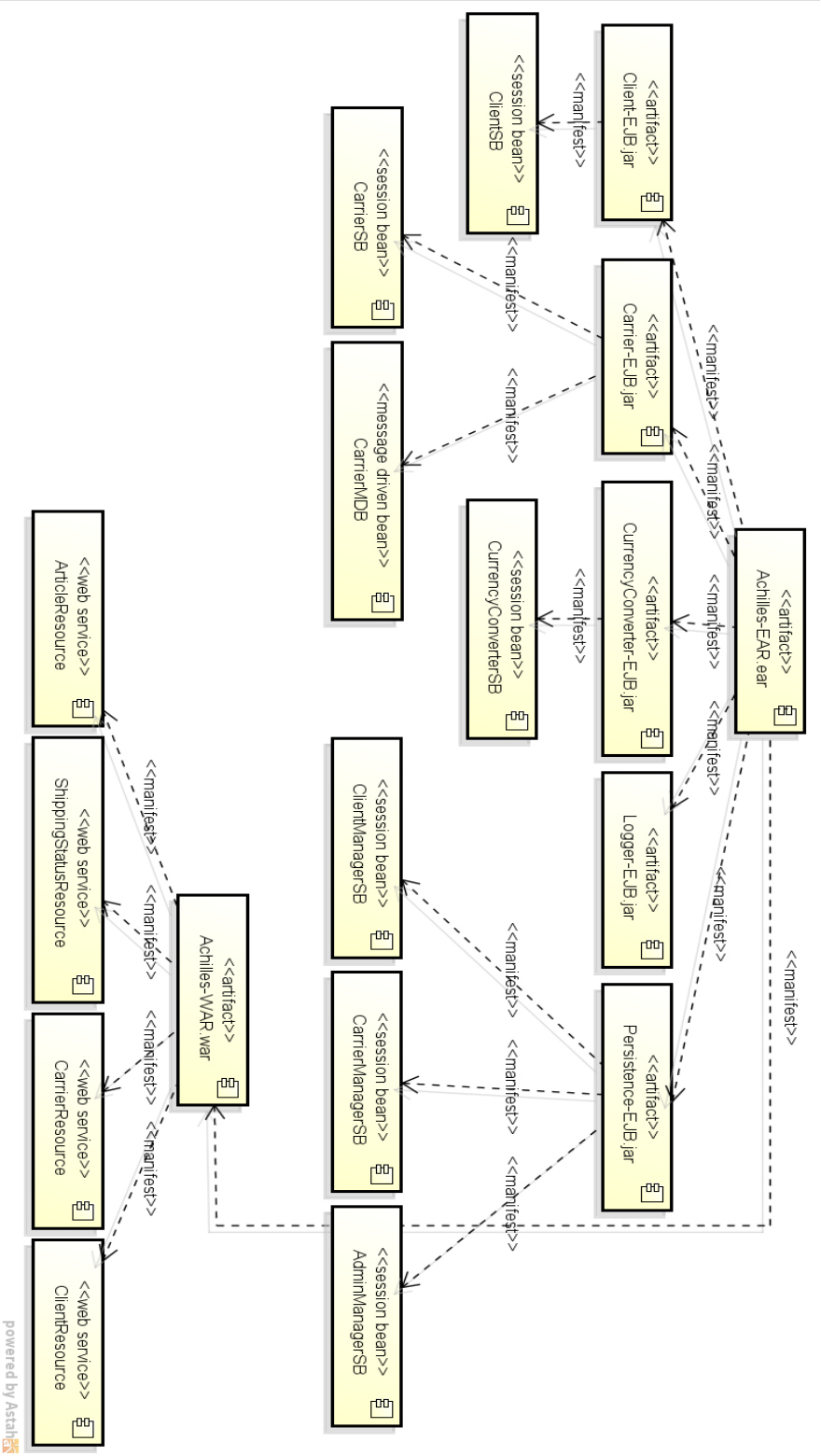
Nodo	Características (velocidad, memoria, etc.)	Descripción
Cliente	*Intel Core Duo, 4 GB RAM.*	*El nodo cliente es donde el usuario ejecuta el navegador de preferencia.*
Servidor Web	*Intel Core Duo, 4 GB RAM*	*El nodo servidor es donde se encuentra el servidor de aplicaciones Glassfish, así como también la BD MySQL*
Servidor Glassfish	*Versión 4.0*	*El nodo de Glassfish es donde se despliega la aplicación*
BD MySQL	*Versión 5.6*	*El nodo de base de datos es donde se almacena la BD y sus tablas con los valores*

Conector	Características (velocidad, etc.)	Descripción
Puerto 8080	*HTTP*	*El puerto 8080 es el configurado para la comunicación entre el cliente y el servidor*

3.3.2 Vista de Instalación

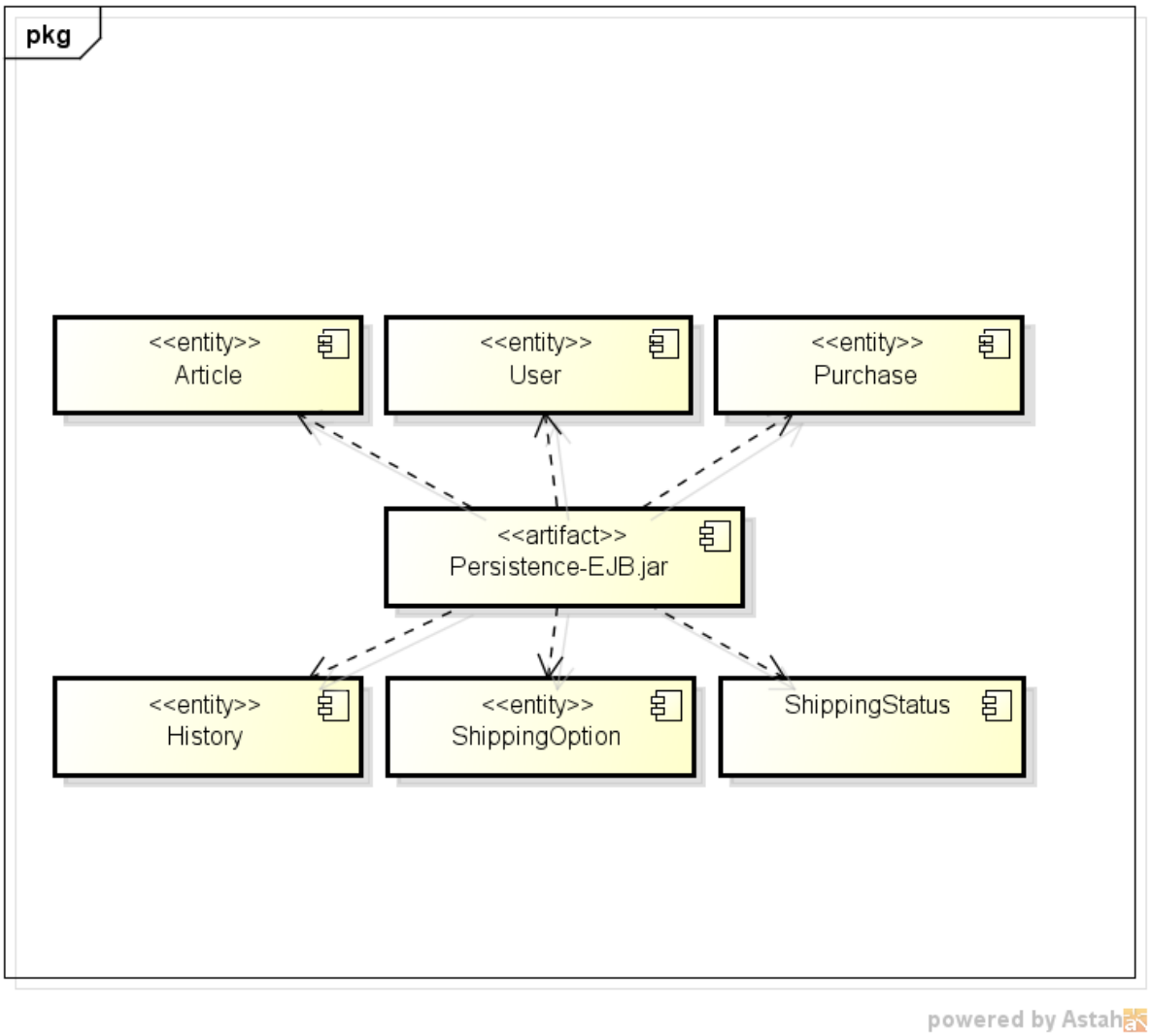
Esta sección describe la estructura de los artefactos de instalación que se muestran en la vista de despliegue.

3.3.2.1 Representación primaria



Aclaración: En el diagrama anterior no se muestran las Entity Classes que corresponden al módulo de Persistence-EJB porque hacía que el diagrama fuera ilegible. A continuación se realiza otro diagrama de instalación para mostrar únicamente las entidades de Persistence-EJB

3.3.2.2 Representación de vista de instalación Persistence-EJB



3.3.2.3 Decisiones de diseño

Patrón Multi-tier

En la vista de despliegue podemos ver que aplicamos el patrón de arquitectura Multi-tier. Esto sucede, en primer lugar, porque la plataforma de Java EE está pensada para realizar aplicaciones distribuidas multi-tier. La idea del patrón es poder dividir el sistema en diversos subsistemas que se ejecutan de forma independiente, incluso en nodos independientes.
Particularmente, en esta entrega tenemos, en primer lugar, un tier correspondiente a los RESTful Web Services, luego un tier correspondiente a la lógica de negocio y finalmente un tier que se encarga del manejo de datos.
La ventaja de la aplicación de este patrón es, en primer lugar, aumentar la modificabilidad del sistema, permitiendo trabajar en un tier sin afectar al resto. En segundo lugar, el uso del patrón favorece la performance, ya que podemos distribuir la aplicación en nodos independientes.

3.4 *Decisiones de diseño del sistema*

REST Web Service vs. SOAP

Decidimos utilizar servicios Web REST por encima de los clásicos servicios SOAP por diferentes motivos. En primer lugar, fueron los trabajados durante el curso por lo que teníamos mayor familiaridad con los mismos. En segundo lugar, los servicios REST son más simples de implementar. Además, los servicios REST tienen una mayor performance, ya que tienen una estructura de almacenamiento en cache. Permiten también mayor escalabilidad, ya que se pueden integrar a otros sistemas de la web y son más modificables que los servicios SOAP.

Uso del formato JSON

En los métodos GET de los Web Services se devuelven objetos (DTOs) en formato JSON. Tiene diversas ventajas frente a otras implementaciones. En primer lugar, el formato es más ligero que un archivo XML, favoreciendo la eficiencia del sistema. En segundo lugar, son fáciles de interpretar y armar, lo que favorece la lectura cuando los recibimos y la creación de nuevos objetos a través de los web services.

3.5 *Errores conocidos y futuras modificaciones*

Log no genera un archivo de texto

El log del sistema muestra por consola el registro de todas las operaciones, pero no genera un archivo con todas las entradas. Desconocemos la fuente del error, aunque de acuerdo a lo visto con el docente de práctico se sabe que hay que cumplir con ciertas configuraciones para que Log4J funcione de forma correcta con el servidor de aplicaciones Glassfish. A pesar de que la letra no aclara que se debe generar un archivo de texto con el log completo, entendemos que sería una buena característica a implementar en un futuro.

Medición de eficiencia en consultas

En un principio quisimos medir la eficiencia de las consultas a la base de datos. De acuerdo a lo visto con el docente de práctico, durante un ejercicio de clase comparamos los tiempos de las consultas de diferentes tipos, en JPQL, named query y query nativa. Si hubiésemos comparado los tiempos de las consultas entre sí podríamos haber argumentado que se perseguía la eficiencia como un atributo de calidad del sistema. Lamentablemente por razones de tiempo no fuimos capaces de realizar la medición por lo que las consultas del sistema fueron escritas de acuerdo al contexto de la misma.