

## SECCIÓN 3

### DISEÑO DE LA ARQUITECTURA

En esta sección se describe la manera en que se realiza el diseño de la arquitectura del sistema cuyos *drivers* fueron listados previamente. Para este caso de estudio, el diseño se lleva a cabo de acuerdo al método ADD (ver sección 3.6.1 del capítulo 3) y por ello se presenta el diseño por medio del resumen de varias iteraciones. Se asume que el paso 1 del ADD (Confirmar que hay suficiente información acerca de los *drivers* o requerimientos) ya ha sido completado (es la información de la sección anterior), y el diseño comienza en el paso 2 en cada una de las iteraciones.

### 3.1 Primera iteración: estructuración general del sistema

En la primera iteración, y como parte del paso 2 del ADD, el arquitecto elige el sistema como elemento a descomponer, dado que se está creando un sistema partiendo de cero (del único punto de partida arquitectónico que se dispone es del diagrama de contexto en el documento de visión y alcance).

En el paso 3 identifica los *drivers* que va a tratar durante la iteración. Dado que se trata de una iteración inicial, el enfoque está en la estructuración general del sistema con el fin de soportar los distintos *drivers* y apegarse a las restricciones.

Una vez que el arquitecto tiene claros los *drivers* procede, en el paso 4 del ADD, a elegir conceptos de diseño para descomponer el elemento elegido anteriormente. En este caso, estos conceptos son patrones de diseño estructurales y de implantación: capas y *N*-tercios.

Ya hecha la elección, en el paso 5 se generan elementos nuevos mediante la instanciación de los patrones y se les asignan responsabilidades. Ello da como resultado estructuras que en este caso son tanto físicas como lógicas.

En el paso 6 se definen interfaces para los elementos instanciados. Sin embargo, dado que se trata de una descomposición estructural de nivel alto, estas no se detallan todavía.

Resumen de iteración		
<b>Elemento a descomponer</b>	El sistema (iteración inicial).	
<b>Drivers elegidos para la iteración</b>	<p>Dado que es la iteración inicial, el arquitecto debe proponer una estructuración general del sistema. Para hacerlo toma en cuenta el conjunto de casos de uso primarios y escenarios de atributos de calidad, así como las siguientes restricciones (véase sección 5.3 del documento de visión):</p> <ul style="list-style-type: none"> <li>• Uso desde navegadores <i>web</i> y dispositivos móviles.</li> <li>• Integración de sistemas externos por medio de servicios <i>web</i>.</li> <li>• Integración de servidor de BD legado.</li> </ul>	
Conceptos de diseño	Concepto	Justificación
	Estilo o patrón arquitectónico de capas ( <i>layers</i> , 182) <sup>1</sup>	Las capas permiten aislar de forma lógica distintas responsabilidades del sistema: los aspectos relacionados con la interacción con el usuario (capa de presentación), con la conexión de sistemas externos (capa de integración), el manejo de la lógica de negocio (capa de negocio) y la persistencia de los datos (capa de datos). Esto permitirá hacer cambios en la interfaz de usuario y asignar componentes a los ingenieros de manera más simple.
	Estilo o patrón arquitectónico de <i>N</i> -tercios.	Con la elección de este estilo se estructura el sistema de un punto de vista de implantación pues las capas de la aplicación se ubican en distintos "tercios" (o nodos físicos). Esto permite aumentar la seguridad y facilita escalar el sistema y su mantenimiento.
	<i>Frameworks</i> JSF <sup>2</sup> , <i>Spring</i> <sup>3</sup> e <i>Hibernate</i> <sup>4</sup> .	Estos se apegan a las capas definidas y son aquellos con los que el equipo de desarrollo está familiarizado.

Continúa...

<sup>1</sup> Hemos optado por dejar el nombre en inglés y la página del libro correspondiente (Buschmann, Henney y Schmidt, 2007).

<sup>2</sup> <http://www.oracle.com/technetwork/java/javase/javaserverfaces-139869.html>

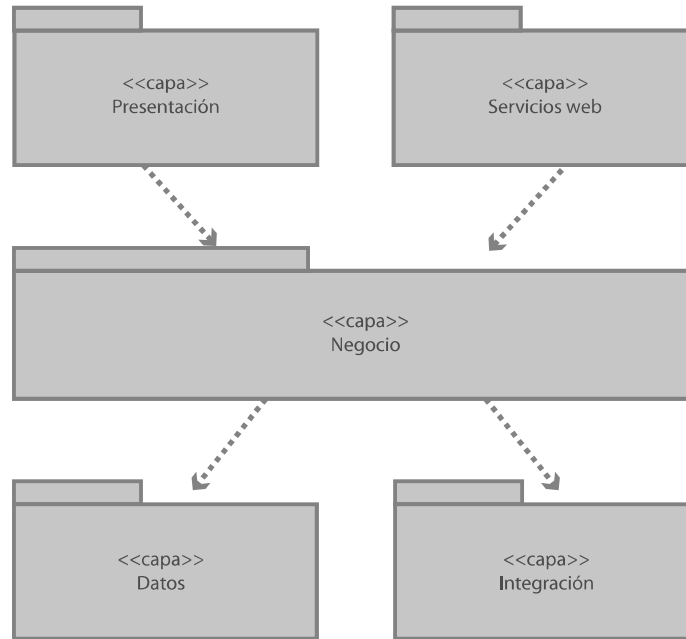
<sup>3</sup> <http://projects.spring.io/spring-framework/>

<sup>4</sup> <http://hibernate.org/>

Continuación...

### Resumen de iteración

Perspectiva lógica: (simbología: UML)



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.

Estructuras resultantes y responsabilidades de los elementos

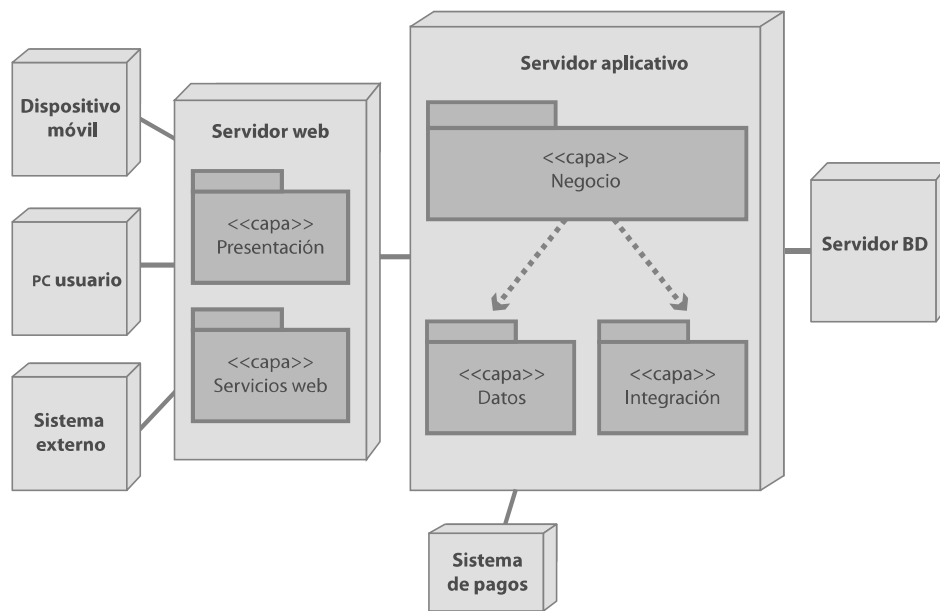
Elemento	Responsabilidad	Propiedades
<b>Capa de presentación</b>	Engloba todos los componentes encargados de recibir interacción por parte del usuario y de mostrarle resultados.	Lenguaje = Java Framework = JSF
<b>Capa de servicios web</b>	Expone servicios <i>web</i> para soportar la integración de sistemas externos.	Lenguaje = Java Framework = Spring-ws
<b>Capa de negocio</b>	Engloba componentes de coordinación que se encargan de llevar a cabo la ejecución de los casos de uso del sistema.	Lenguaje = Java Framework = Spring
<b>Capa de datos</b>	Engloba componentes encargados de almacenar objetos en una base de datos, ocultando del resto de la aplicación el hecho de que se trata de una base de datos relacional.	Lenguaje = Java Framework = Hibernate
<b>Capa de integración</b>	Permite la integración con sistemas externos (por ejemplo, sistema de cobros).	Protocolo = SOAP

Continúa...

Continuación...

## Resumen de iteración

Perspectiva física: (simbología: UML)



© Humberto Cervantes Maceda / Perla Velasco Elzondo / Luis Castro Careaga.

Estructuras resultantes y responsabilidades de los elementos

Elemento	Responsabilidad	Propiedades
PC del usuario	Computadora personal mediante la cual acceden los usuarios y administradores del sistema.	Navegador = Internet Explorer 10+, Firefox 10+, Google Chrome 17+
Dispositivo móvil	Dispositivos por medio de los cuales se accederá en el futuro a la aplicación.	OS = iPhone/iPod Touch, IOS 6+ Android 4+
Servidor web	El servidor web alberga las capas de presentación y servicios web.	Tipo = Tomcat
Servidor Aplicativo	El servidor aplicativo alberga las capas de negocio, datos e integración de la aplicación.	Memoria = Por definir Procesador = Por definir
Servidor BD	Este servidor alberga la base de datos (del otro servidor legado).	RDBMS = Oracle v11.2
Sistema externo	Sistema externo que interactúa con el sistema que se desarrolla mediante servicios web	
Sistema de pagos	Sistema externo a través del cual se realizan los pagos.	

Continúa...

Continuación...

## Resumen de iteración

## Interfaces de los elementos

Interfaces lógicas entre:

- Capas de presentación y negocio.
- Capas de servicios *web* y negocio.
- Capa de negocio y datos.
- Capa de negocio e integración.

Interfaces físicas entre:

- PC de usuario, dispositivo móvil, sistemas externos y sistema de pagos.
- Servidores aplicativos y BD

Al final de esta iteración se revisa si ya se han tomado decisiones de diseño para satisfacer los *drivers* de la arquitectura. Se han hecho muchas, pero no suficientemente detalladas, por ello es necesario realizar iteraciones adicionales.

### 3.2 Segunda iteración: integración de la funcionalidad a las capas

En la segunda iteración de diseño, el arquitecto se enfoca en identificar la manera en que se soportará la funcionalidad primaria del sistema. Para ello, toma las capas de este como elementos a descomponer (paso 2 del ADD). Los *drivers* para esta iteración (elegidos en el paso 3) son los casos de uso primarios, CU-01 y CU-02. Respecto de la elección de los conceptos de diseño (paso 4), se comienza por seleccionar el patrón *Domain Object*, que tiene como propósito encapsular funcionalidades distintas del sistema en bloques autocontenidos llamados objetos de dominio.

Se selecciona también el patrón *Data Mapper*, que es un tipo particular de *Domain Object* con el cual se encapsulan los aspectos de acceso a la base de datos relacional. A efecto de permitir la realización de pruebas, los objetos de dominio refinan mediante los patrones *Explicit Interface* y *Encapsulated Implementation* para separar claramente la interfaz y la implementación.

A continuación se instancian elementos a partir de los patrones (paso 5). El resultado es que en cada una de las capas se crean componentes enfocados en soportar cada uno de los casos de uso. Una vez identificados los componentes dentro de las capas, se procede a definir interfaces para ellos (paso 6). Por esta razón se realiza un análisis dinámico de las interacciones entre los componentes para soportar los flujos asociados a los casos de uso. Para este análisis se usan diagramas de secuencia de UML, y como resultado se identifican los métodos asociados a las interfaces de los componentes.

Al final de esta iteración ya se tiene más claro cuáles serán los componentes necesarios para soportar la funcionalidad primaria del sistema, y esto sirve de marco de referencia para identificar aquellos que implementan el resto de la funcionalidad. Sin embargo, no se han tomado decisiones más específicas para satisfacer los escenarios de atributos de calidad, por lo que la iteración siguiente se enfoca en este aspecto.

## Resumen de iteración

## Elemento a descomponer

Las distintas capas del sistema.

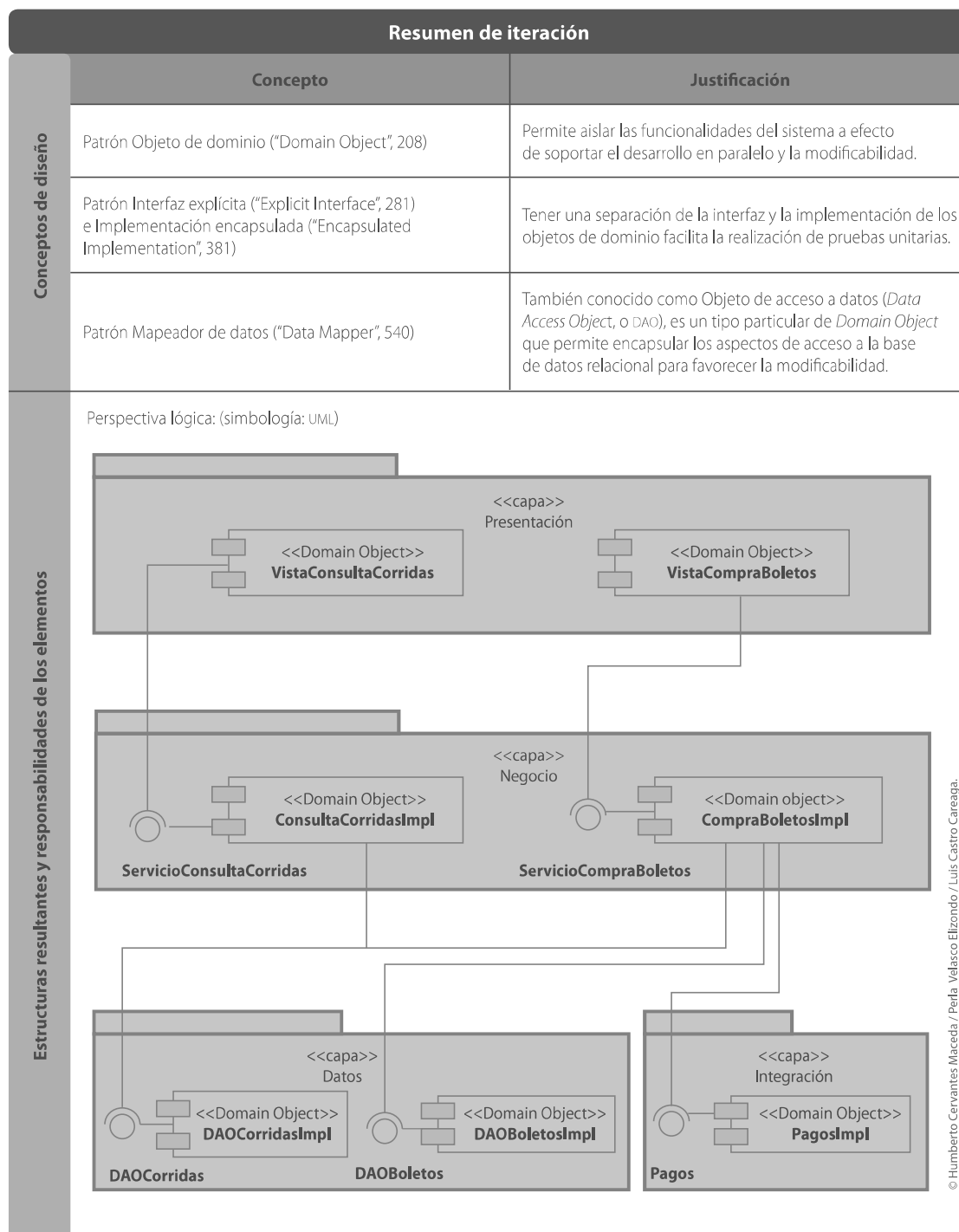
## Drivers elegidos para la iteración

Casos de uso primarios:

- CU-01: consultar corridas.
- CU-02: comprar boleto.

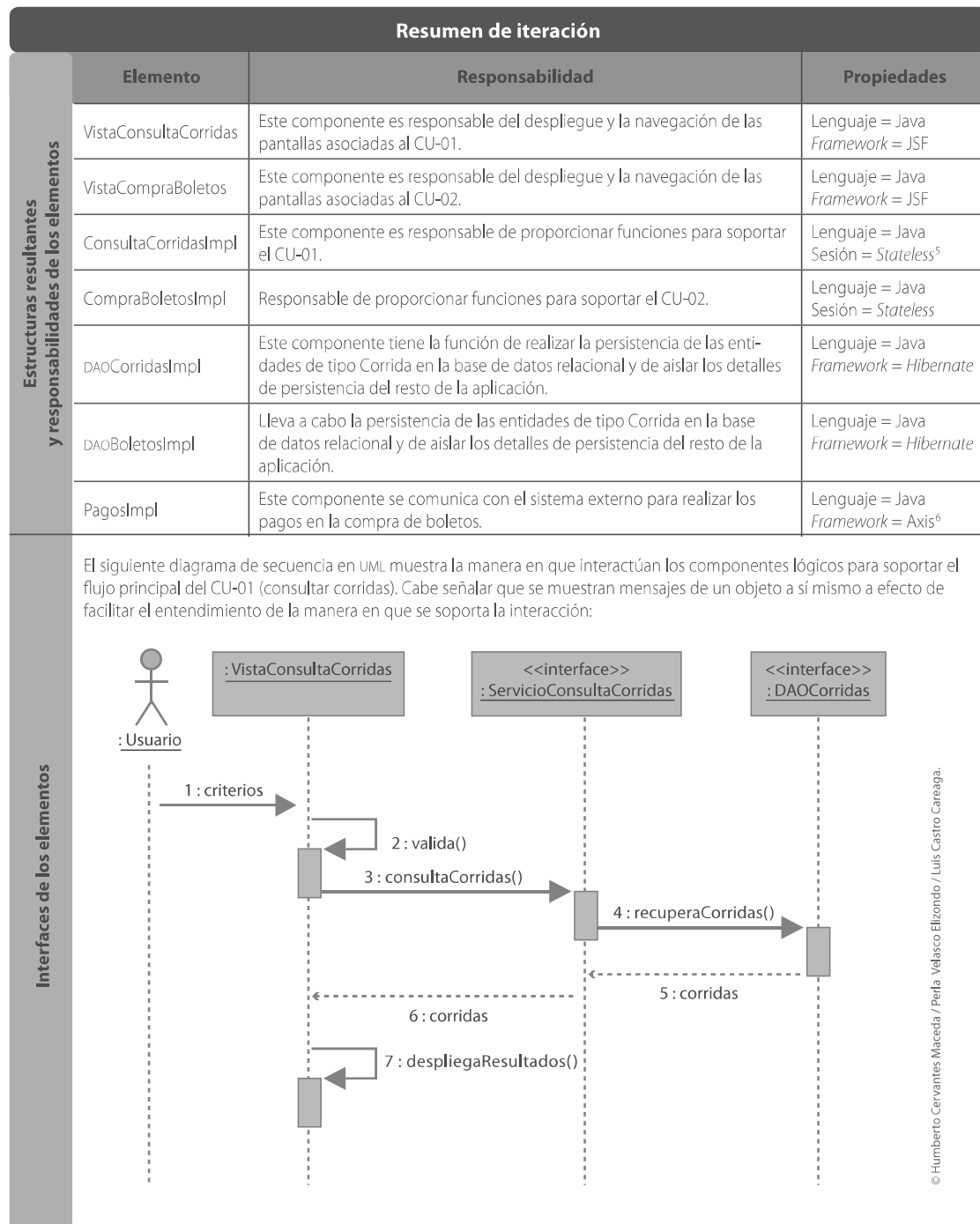
Continúa...

Continuación...



Continúa...

Continuación...

<sup>5</sup> Los componentes *stateless* carecen de estado asociado permitiendo, por ejemplo, que se les acceda de forma concurrente.<sup>6</sup> <http://axis.apache.org/axis/java/>

Continúa...

Continuación...

Resumen de iteración		
Interfaces de los elementos	Los métodos asociados a las interfaces de los componentes se detallan a continuación:	
	Interfaz	Detalles/Descripción
	ServicioConsultaCorridas	Corridas [ ] consultaCorridas (Criterios): recupera corridas de acuerdo con los criterios (invocación síncrona).
	DAOCorridas	boolean creaCorrida (Corrida) Corridas [ ] recuperaCorridas (String criterio) boolean actualizaCorrida (Corrida) boolean borraCorrida (Corrida)  Nota: en general, los DAO proporcionan métodos de creación, recuperación, actualización y borrado (CRUD <sup>1</sup> ), por lo que estos últimos se identifican aun si no son usados en el análisis dinámico.

### 3.3 Tercera iteración: desempeño en capa de datos

En la tercera iteración, el arquitecto decide llevar a cabo decisiones de diseño más detalladas que le permitan satisfacer el escenario de atributo de calidad de desempeño (EAC-01) que resultó prioritario. Recordemos este escenario:

*Un usuario selecciona la opción “buscar” una vez que ha elegido las ciudades de salida y destino, así como las fechas en la pantalla de consulta (CU-01) en un momento normal de operación y hay hasta cien usuarios conectados al sistema. Este procesa la petición y muestra la lista de corridas en un tiempo no mayor a dos segundos.*

Donde:

- Momento normal de operación = 100 usuarios.
- Momento de sobrecarga = 500 usuarios simultáneos.
- Dos segundos incluye únicamente tiempo de procesamiento y no tránsito en red.

En esta iteración, el arquitecto decide enfocarse en la capa de datos, que será entonces el elemento a descomponer (paso 2 del ADD). El *driver* primario seleccionado para esta iteración es el escenario EAC-01 (paso 3 del ADD), y respecto de los conceptos de diseño elegidos en el paso 4 del ADD, estos incluyen los patrones *Lazy Acquisition* e *Eager Acquisition*, así como las tácticas de Mantener copias múltiples y Manejo de recursos.

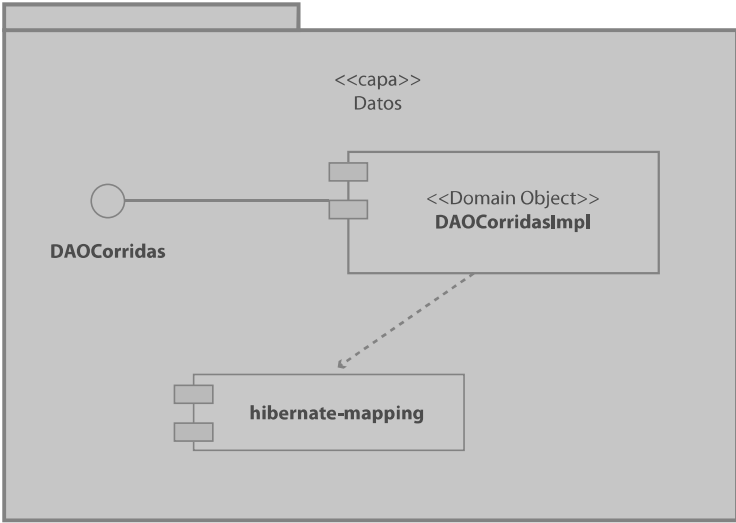
Aunque esos patrones y tácticas son los conceptos de diseño elegidos, forman parte de las opciones de configuración del *framework* usado en la capa de persistencia, por lo que la instanciación de estos conceptos (paso 5 del ADD) requiere solo la configuración del *framework*, que en este caso se lleva al cabo mediante un archivo en formato XML propio al *framework*.

Para el paso 6 del ADD se realiza de nuevo un análisis dinámico que aquí resulta en un refinamiento de las interfaces identificadas en la iteración anterior. Al final de esta iteración, el arquitecto concluye que en el nivel de la capa de datos ha tomado suficientes decisiones de diseño para satisfacer el atributo de calidad de desempeño que fue el *driver* de la iteración. Sin embargo, estas decisiones no son suficientes para satisfacer el escenario pues el desempeño requiere de otras decisiones en las demás capas del sistema, las cuales deberán hacerse en iteraciones subsecuentes.

<sup>1</sup> CRUD: Create, Read, Update and Delete.

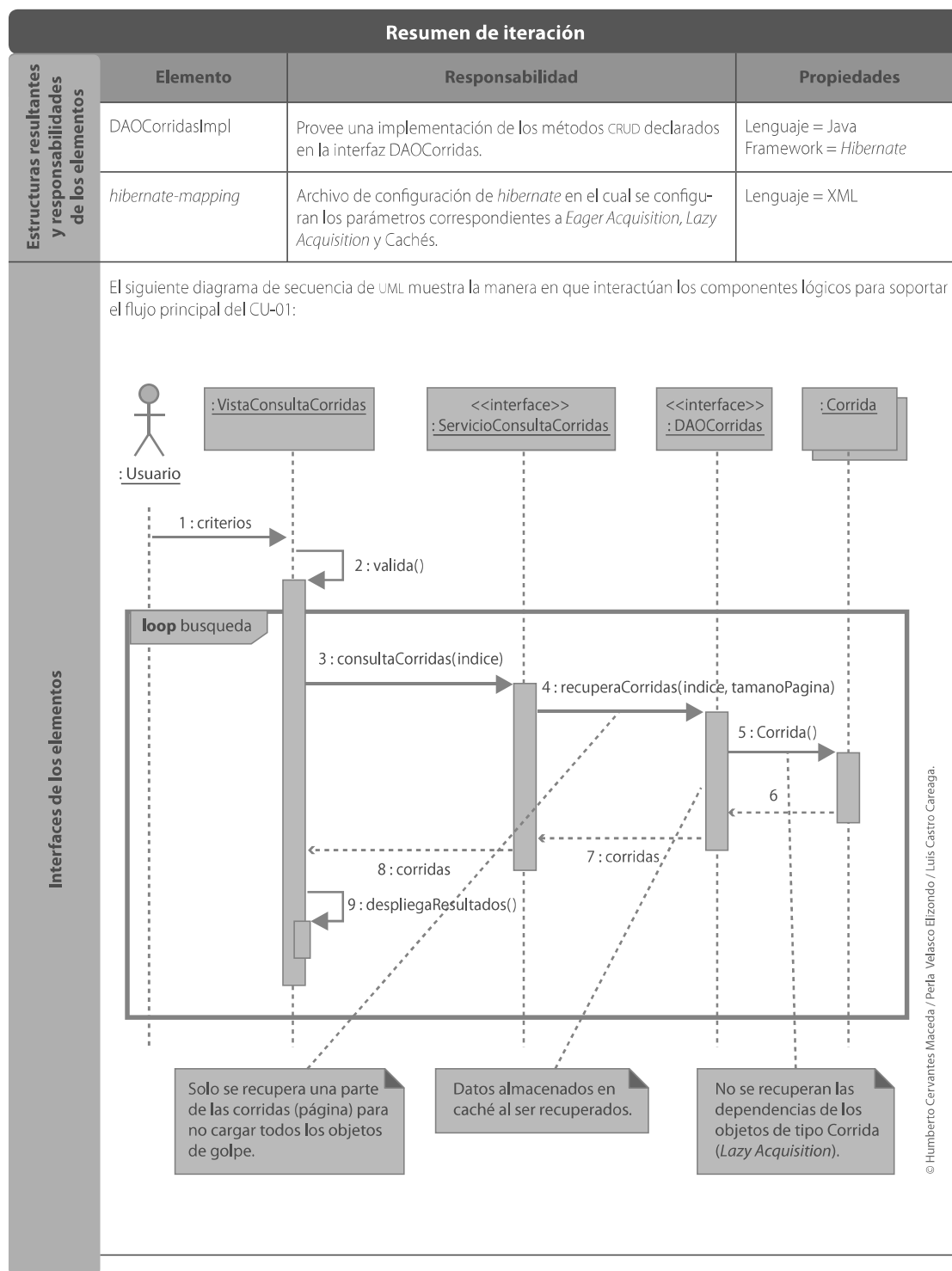


## Resumen de iteración

<b>Elemento a descomponer</b>	Capa de datos	
<b>Drivers elegidos para la iteración</b>	EAC-01	
<b>Conceptos de diseño</b>	<b>Concepto</b>	<b>Justificación</b>
	Patrones Adquisición temprana ( <i>Eager Acquisition</i> , 509) y Adquisición tardía ( <i>Lazy Acquisition</i> , 507)	El uso de <i>Eager Acquisition</i> en determinadas situaciones mejora desempeño pues se evita que <i>Hibernate</i> haga varios <b>SELECT</b> para objetos dependientes que deben ser cargados de forma simultánea.  El uso de <i>Lazy Acquisition</i> en determinadas situaciones mejora desempeño al no cargar de forma inmediata propiedades o dependencias que podrían no usarse (por ejemplo, datos que no se muestran en una pantalla).
	Táctica Mantener copias múltiples  Cachés	Uso de <i>process-level cache</i> (caché a nivel de procesos) para objetos que cambian con poca frecuencia.  Uso de <i>query cache</i> (caché de consultas) para almacenar resultados de las consultas.
	Táctica Manejo de recursos Paginación	Las consultas no regresan todos los datos de la base sino un subconjunto.
<b>Estructuras resultantes y responsabilidades de los elementos</b>	<p>Perspectiva lógica: (simbología: UML)</p>  <p>© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.</p>	

Continúa...

Continuación...



Continúa...

Continuación...

### Resumen de iteración

Interfaces de los elementos

Los métodos asociados a las interfaces de los componentes se detallan a continuación (las llamadas a sí mismo de cada objeto no se muestran pues no son públicas).

Interfaz	Detalles/Descripción
<b>ServicioConsultaCorridas</b>	Corrida [] consultaCorridas (índice) recupera corridas a partir del índice.
<b>DAOCorridas</b>	Corrida [] recuperaCorridas (índice, tamanoPagina) recupera un bloque de corridas con base en el índice y tamaño de página.

Como parte del diseño sería necesario hacer más iteraciones para definir estructuras con las cuales satisfacer los demás *drivers* de la arquitectura. Por falta de espacio, estas no se incluyen.