**DESCRIPCION DE ARQUITECTURA (Ejemplo)**

***Adventure Builder Reference Application***

***Mayo 2010***

**Índice**

[**1.**](#_heading=h.gjdgxs) **Introducción 3**

[1.1](#_heading=h.30j0zll) Propósito 3

[**2.**](#_heading=h.1fob9te) **Antecedentes 3**

[2.1](#_heading=h.3znysh7) Propósito del sistema 3

[2.2](#_heading=h.2et92p0) Objetivos de Arquitectura 3

[*2.2.1*](#_heading=h.tyjcwt) *Resumen de Requerimientos Funcionales 3*

[*2.2.2*](#_heading=h.3dy6vkm) *Resumen de Requerimientos NoFuncionales 4*

[**3.**](#_heading=h.1t3h5sf) **Documentación de la arquitectura 6**

[3.1](#_heading=h.4d34og8) Punto de vista lógico 6

[*3.1.1*](#_heading=h.2s8eyo1) *Vista Descomposición y Usos del sistema AdventureBuilder 6*

[3.1.1.1](#_heading=h.17dp8vu) Representación primaria 6

[3.1.1.2](#_heading=h.3rdcrjn) Catalogo de elementos 6

[3.1.1.3](#_heading=h.26in1rg) Interfaces 7

[3.1.1.4](#_heading=h.lnxbz9) Comportamiento 7

[3.1.1.5](#_heading=h.35nkun2) Decisiones de diseño 7

[*3.1.2*](#_heading=h.1ksv4uv) *Vista Descomposición del OPC (OpcApp) 9*

[3.1.2.1](#_heading=h.44sinio) Representación primaria 9

[3.1.2.2](#_heading=h.2jxsxqh) Catalogo de elementos 9

[3.1.2.3](#_heading=h.z337ya) Interfaces 11

[3.1.2.4](#_heading=h.3j2qqm3) Comportamiento 11

[3.1.2.5](#_heading=h.1y810tw) Decisiones de diseño 11

[*3.1.3*](#_heading=h.4i7ojhp) *Vista de Usos del OPC (OpcApp) 12*

[3.1.3.1](#_heading=h.2xcytpi) Representación primaria 12

[3.1.3.2](#_heading=h.1ci93xb) Catalogo de elementos 12

[3.1.3.3](#_heading=h.3whwml4) Interfaces 12

[3.1.3.4](#_heading=h.2bn6wsx) Comportamiento 14

[3.1.3.5](#_heading=h.qsh70q) Decisiones de diseño 14

[3.2](#_heading=h.3as4poj) Punto de vista de Componentes y conectores 15

[*3.2.1*](#_heading=h.1pxezwc) *Vista de Componentes y Conectores de OPC 15*

[3.2.1.1](#_heading=h.49x2ik5) Representación primaria 15

[3.2.1.2](#_heading=h.2p2csry) Catalogo de elementos 15

[3.2.1.3](#_heading=h.147n2zr) Interfaces 17

[3.2.1.4](#_heading=h.3o7alnk) Comportamiento 17

[3.2.1.5](#_heading=h.23ckvvd) Relación con elementos lógicos 18

[3.2.1.6](#_heading=h.32hioqz) Decisiones de diseño 18

[*3.2.2*](#_heading=h.1hmsyys) *Vista de Capas Físicas del sistema AdventureBuilder. 19*

[3.2.2.1](#_heading=h.41mghml) Representación primaria 19

[3.2.2.2](#_heading=h.2grqrue) Catalogo de elementos 19

[3.2.2.3](#_heading=h.vx1227) Interfaces 21

[3.2.2.4](#_heading=h.3fwokq0) Comportamiento 21

[3.2.2.5](#_heading=h.1v1yuxt) Relación con elementos lógicos 22

[3.2.2.6](#_heading=h.2u6wntf) Decisiones de diseño 22

[3.3](#_heading=h.3tbugp1) Punto de vista Físico. 23

[*3.3.1*](#_heading=h.28h4qwu) *Vista de Despliegue 23*

[3.3.1.1](#_heading=h.nmf14n) Representación primaria 23

[3.3.1.2](#_heading=h.37m2jsg) Catalogo de elementos 24

[3.3.1.3](#_heading=h.1mrcu09) Relación con componentes 24

[3.3.1.4](#_heading=h.46r0co2) Decisiones de diseño 24

# Introducción

*[Describir la visión general del Documento de Descripción de arquitectura (DA), la estructura del documento y la forma en**que los usuarios del mismo]*

*Ref: Documenting Software Architectures: Views and Beyond (Paul Clements et al.) – Sección 10.3.*

* 1. **Propósito**

El propósito del presente documento es proveer una especificación, a modo de ejemplo de la arquitectura de la aplicación “Adventure Builder Reference Application”.

# Antecedentes

* 1. **Propósito del sistema**

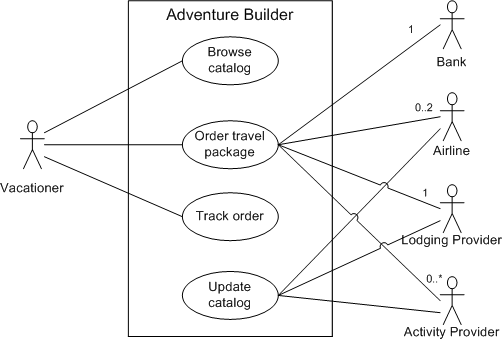
El sistema utilizado como ejemplo en este documento de arquitectura representa una versión adaptada del “Adventure Builder Reference Application” desarrollado en el contexto del programa Java BluePrints de Sun Microsystems.

Adventure Builder es una compañía ficticia que vende paquetes turísticos en Internet. A continuación se describen los principales requerimientos del sistema.

* 1. **Objetivos de Arquitectura**

### Resumen de Requerimientos Funcionales

|  |  |  |
| --- | --- | --- |
| **ID Requerimiento** | **Descripción** | **Actor** |
| UC1 | El usuario puede visitar el sitio Web de AdventureBuilder y navegar por el catálogo de paquetes de viajes, que incluyen vuelos a destinos específicos, opciones de alojamiento, y las actividades que se pueden comprar por adelantado. Las actividades incluyen paseos en bicicleta de montaña, pesca, surf clases, excursiones globo de aire caliente, y el buceo. El usuario puede seleccionar el transporte, alojamiento y actividades diversas para construir su propio viaje de aventura. | Usuario |
| UC2 | El usuario puede realizar una orden de compra por un paquete de vacaciones. Para procesar esta solicitud, el sistema tiene que interactuar con varias entidades externas. Un banco aprobará el pago de los clientes, compañías aéreas proporcionaran los vuelos, proveedores de alojamiento reservarán las habitaciones en los hoteles, y las empresas que ofrecen actividades turísticas programarán las actividades seleccionadas por el cliente. | Usuario |
| UC3 | Después de un pedido, el usuario puede volver a comprobar el estado de su solicitud. Esto es necesario porque algunas interacciones con entidades externas se procesan en segundo plano y puede tardar horas o días en completarse. | Usuario |
| UC4 | El sistema periódicamente interactúa con sus socios comerciales (transporte, alojamiento, y los proveedores de actividades turísticas) para actualizar el catálogo con las ofertas más recientes. | Sistema |



### Resumen de Requerimientos NoFuncionales

|  |  |  |
| --- | --- | --- |
| **ID Requerimiento** | **Atributo de Calidad** | **Descripción** |
| QAS1 | Modificabilidad | Un nuevo socio comercial (aéreos, hospedaje, o proveedor de actividades) que utiliza su propia interfaz de servicios web puede ser agregado al sistema en un máximo de 10 días-persona de esfuerzo para la aplicación. El objetivo de negocio es una fácil integración con nuevos socios comerciales. |
| QAS2 | Eficiencia | Un usuario hace un pedido de un paquete de viajes de aventura en la página Web de la aplicación. El usuario es notificado en pantalla de que el pedido ha sido enviado y es procesado en menos de cinco segundos. |
| QAS3 | Eficiencia | Hasta 500 usuarios hacen clic para ver el catálogo de paquetes de aventura a raíz de una distribución aleatoria de más de 1 minuto, el sistema está en condiciones de funcionamiento normal, la latencia máxima para servir a la primera página de contenido es menor a 5 segundos; el promedio de latencia para la misma es inferior a 2 segundos. |
| QAS4 | Confiabilidad | El sitio Web del consumidor envió una solicitud de orden de compra al centro de procesamiento de pedidos (OPC). La OPC procesa esta petición pero no responde a los consumidores del sitio Web dentro de los cinco segundos, por lo que el sitio Web del consumidor vuelve a enviar la solicitud a la OPC. La OPC recibe la solicitud duplicada, pero solamente se factura al consumidor una única vez, los datos se mantienen en un estado consistente, y el sitio Web del consumidor es notificado que la petición original fue exitosa en un cien por ciento de los casos. |
| QAS5 | Seguridad | La aprobación de crédito y procesamiento de pagos se solicitan para una nueva orden. En el cien por ciento de los casos, la transacción se completa de forma segura y no puede ser repudiada por ninguna de las partes. Uno de los objetivos de negocio es proporcionar a los clientes y socios de negocio la confianza en la seguridad para cumplir obligaciones contractuales, legales y reglamentarias para la seguridad de las transacciones de crédito. |
| QAS6 | Seguridad | El OPC experimenta un excesivo número de peticiones a través del “Web Service Broker” que no corresponden a ninguno los pedidos en curso. En el cien por ciento de los casos, el sistema detecta el nivel anormal de actividad, notifica al administrador del sistema, y continúa atendiendo solicitudes en un modo degradado. |
| QAS7 | Disponibilidad | El sitio Web del consumidor está a disposición del usuario 24x7. Si una instancia de la aplicación OPC falla, se detecta la falla y el administrador del sistema es notificado dentro de los 30 segundos, el sistema continúa procesando otras peticiones; una nueva instancia de OPC se crea, y los datos se mantienen en estado consistente. |

# Documentación de la arquitectura

* 1. **Punto de vista lógico**

### Vista Descomposición y Usos del sistema AdventureBuilder

#### Representación primaria

#### Image:HighLevelModuleUsesView_PP.png

#### Catalogo de elementos

|  |  |
| --- | --- |
| **Elemento** | **Responsabilidades** |
| **Consumer Website** | La interfaz web de usuario del AdventureBuilder es implementada en este módulo. La interfaz de usuario permite al usuario navegar el catálogo de paquetes de viajes, realizar una nueva orden de compra y seguir el estado de los pedidos existentes. Este módulo crea las órdenes de compra a partir de los inputs del usuario enviando los mismos al modulo OpcApp para su procesamiento. Utiliza una implementación del patrón Model-View-Controller llamada Web Application Framework (WAF). El modelo se implementa utilizando Entity Beans, el controlador se implementa utilizando Servlets, y la vista es una colección de páginas JSP y páginas HTML estáticas. Parte del código de cara al cliente se realiza utilizando el framework GWT (Google Web Toolkit). |
| **OpcApp** | OpcApp significa Aplicación Centro de Procesamiento de Orden. La lógica de negocio del AdventureBuilder se lleva a cabo en este módulo. El mismo ofrece las siguientes funcionalidades:   * Acepta solicitudes de órdenes de compra desde el Consumer Website para su procesamiento a partir servicio web “Purchasing Order”. * Proporciona un mecanismo para realizar consultas del estado actual de una orden desde el Consumer Website a partir del servicio web “Order Tracking”. * Se comunica con los proveedores externos para procesar y mantener el estado de una orden de compra. * Luego de finalizado el procesamiento de una orden de compra, envía un correo electrónico al cliente informado el éxito o fracaso de dicho proceso.   En las secciones subsiguientes se presentan las vistas refinadas de descomposición y uso de este modulo. |
| **util** | Este modulo contiene utilidades usadas por el sistema AdventureBuilder. |
| **dao** | Este modulo contiene utilidades para los Objetos de Acceso a Datos (DAO), tales como un “DAO factory”. Aquí **no** se encuentran las clases DAO que acceden a la base de datos. Estas clases se encuentran en el modulo Consumer Website. |
| **logger** | Este modulo contiene utilidades para llevar a cabo tareas de “tracing” y “debugging”. El nombre original era “tracer”. |
| **servicelocator** | Este modulo representa una implementación del patrón de diseño “Service Locator”. |
| **gwt** | “Google Web Toolkit” es un framework open-source para el desarrollo de aplicaciones RIA basadas en Ajax. |
| **waf** | Waf significa “Web Application Framework”. Es un framework MVC similar a Struts. Permite especificar mediante archivos de configuración las pantallas web y acciones asociadas a los inputs (clicks) de usuario en determinados elementos visuales. El archivo de configuración contiene el mapeo de las pantallas y las acciones a clases Java. |
| **Bank credit card service** | Este modulo representa un servicio externo provisto por una institución bancaria para validar las transacciones con tarjeta de crédito. |
| **Airline booking service** | Este modulo representa un servicio externo provisto por una aerolínea para la reserva de vuelos. |
| **Lodging booking service** | Este modulo representa un servicio externo provisto por una compañía de alojamiento para reservar habitaciones de hotel. |
| **Activity booking service** | Este modulo representa un servicio externo provisto por una agencia de turismo para reservar diferentes actividades turísticas. |

#### Interfaces

En esta vista no se describe ningún detalle referente a Interfaces.

#### Comportamiento

Detalles de comportamiento son provistos en las secciones [referencia comportamiento].

#### Decisiones de diseño

**WAF**

El framework WAF fue elegido porque facilita la implementación del Consumer Website al ofrecer clases “template” para el uso del patrón MVC. Para una determinada funcionalidad u operación del usuario, el desarrollador implementa un “action class” (controlador) y las páginas JSP que corresponden a las pantallas de usuario (vista). El desarrollador también utiliza los archivos de configuración para proporcionar un mapeo configurable entre las acciones, los “action class”, eventos y pantallas. La infraestructura WAF puede tomar automáticamente peticiones HTTP e invocar los “action class” y las pantallas JSP.

WAF también proporciona soporte para la comunicación basada en eventos y la internacionalización.

Una alternativa a WAF era utilizar el framework Spring. Spring era una solución más robusta y rico desde el punto de vista técnico, pero fue rechazada debido a que el equipo de desarrollo no está familiarizado con Spring y muy familiarizado con WAF.

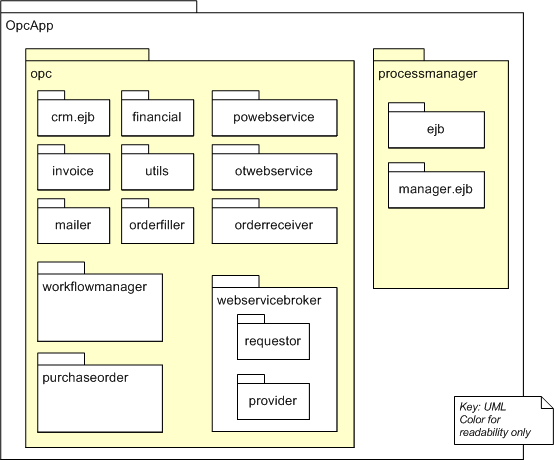
**GWT**

El framework GWT fue escogido por las siguientes razones:

* Es open-source. Lo que permite solucionar problemas y adaptarlo en base a las necesidades.
* Provee un rico entorno de desarrollo con un poderoso soporte para tracing/debugging, integración con el IDE y gestión de “builds”.
* Se integra bien con cualquier otra tecnología de “front-end” ya que no utiliza ningún estándar propietario.
* Ofrece capacidades poderosas para la construcción de componentes reutilizables (widgets).
* Código es escrito en Java (y luego convertido a JavaScript en tiempo de compilación), esto es, un lenguaje OO que le es familiar a todo el equipo de desarrollo.
* Código es compilado/generado en JavaScript, el cual está disponible en casi el 100% de los navegadores web.
* Gran soporte por la comunidad de desarrolladores y desarrollado por un gran proveedor de la industria como Google.

### Vista Descomposición del OPC (OpcApp)

#### Representación primaria



#### Catalogo de elementos

|  |  |
| --- | --- |
| **Elemento** | **Responsabilidades** |
| **OpcApp** | OpcApp significa Aplicación Centro de Procesamiento de Orden. La lógica de negocio del AdventureBuilder se lleva a cabo en este módulo. El mismo ofrece las siguientes funcionalidades:   * Acepta solicitudes de órdenes de compra desde el Consumer Website para su procesamiento a partir servicio web “Purchasing Order”. * Proporciona un mecanismo para realizar consultas del estado actual de una orden desde el Consumer Website a partir del servicio web “Order Tracking”. * Se comunica con los proveedores externos para procesar y mantener el estado de una orden de compra. * Luego de finalizado el procesamiento de una orden de compra, envía un correo electrónico al cliente informado el éxito o fracaso de dicho proceso.   En las secciones subsiguientes se presentan las vistas refinadas de descomposición y uso de este modulo. |
| **opc** | Este paquete contiene toda la lógica de procesamiento de órdenes de compra, incluyendo el workflow, las colas internas para la comunicación entre los elementos y la interacción con los servicios web externos. |
| **crm.ejb** | Este es el modulo de gestión de relacionamiento con clientes (CRM). El propósito de este modulo es enviar un correo electrónico luego de que una orden es procesada satisfactoriamente. Se implementa como un message-driven bean. |
| **invoice** | Este modulo contiene una estructura de datos que encapsula la información que utiliza el OPC para la comunicación con proveedores externos. Esta estructura además mantiene el estado de la orden de compra en la factura (más información sobre el ciclo de vida de las órdenes de compra en la vista de componentes y conectores). |
| **mailer** | Representa un modulo utilitario cuya principal responsabilidad es el envió de correo electrónico utilizando el API de Java Mail. En el futuro este modulo será encapsulado en el modulo Utils fuera del OPC. |
| **financial** | Este modulo es responsable de verificar y realizar el cobro a las tarjetas de crédito de los clientes. Para este propósito, invoca el servicio web externo provisto por la institución bancaria. La verificación de la tarjeta de crédito se ejecuta de forma síncrona. Si la institución bancaria no responde de forma positiva, la aplicación OPC cancela el procesamiento de la orden. |
| **utils** | Encapsula clases utilitarias para la aplicación OPC, tales como la gestión de colas JMS entre otras cosas. En el futuro este modulo será encapsulado en el modulo Utils fuera del OPC. |
| **orderfiller** | Este modulo lee las peticiones de órdenes de compra desde una cola interna. Cuando una orden es recibida, la descompone en múltiples peticiones para los diferentes proveedores involucrados. Dichas peticiones son enviadas utilizando mensajes XML. |
| **powebservice** | Este modulo provee un servicio web que es consumido por el modulo Consumer Website para comunicar los detalles de las órdenes de compra a ser procesados por el modulo OPC. La interfaz de este servicio web se detalla en la sección 3.1.3.3. |
| **otwebservice** | Este modulo provee un servicio web que es consumido por el modulo Consumer Website para consultar el estado de las órdenes de compra a partir de su identificador. La interfaz de este servicio web se detalla en la sección 3.1.3.3. |
| **orderreceiver** | Encapsula la lógica de persistencia para las órdenes de compra. |
| **workflowmanager** | … |
| **purchaseorder** | … |
| **webservicebroker** | … |
| **requestor** | … |
| **provider** | … |
| **processmanager** | … |
| **Ejb** | … |
| **manager.ejb** | … |

#### Interfaces

Los detalles correspondientes a las interfaces de los servicios web *powebservice* y *otwebservice* son documentados en la sección 3.1.3.3.

#### Comportamiento

Detalles de comportamiento son provistos en las secciones [referencia comportamiento].

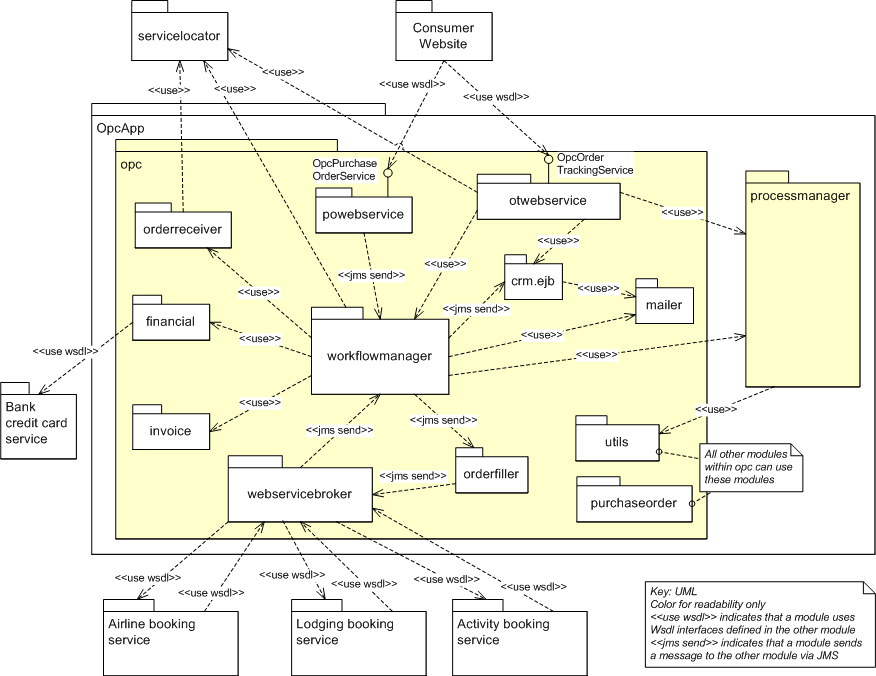
#### Decisiones de diseño

La elección de EJBs en la implementación, incluyendo session-beans, message-driven beans y entity-beans está basada en:

* Desarrolladores familiarizados con el desarrollo de EJBs y desarrollo orientado a componentes.
* Componentes EJB altamente modulares promueven el reúso.

### Vista de Usos del OPC (OpcApp)

#### Representación primaria



#### Catalogo de elementos

Los elementos presentados en esta vista se describen en la sección 3.1.2.2.

#### Interfaces

|  |  |
| --- | --- |
| **Nombre** | **Detalles** |
| **OpcPurchaseOrderService** | **SubmitPurchaseOrder**  *Sintaxis:* String submitPurchaseOrder(PurchaseOrder poObject)  *Semántica:* Envía una orden de compra para ser procesada. Esta operación retorna el identificador de la orden como un objeto del tipo String. La operación verifica si la orden de compra es válida y en caso de serlo, solamente la persiste en la base de datos de la aplicación y se retorna el identificador. El procesamiento real de la orden se lleva a cabo en un proceso por lotes independiente. Cualquier error en dicho procesamiento se manejara de forma independiente y no es responsabilidad de esta interfaz controlar tales errores.  *Pre-Condiciones:* El objeto *poObject* que representa la orden de compra debe ser **no** nulo. Del mismo modo, las propiedades UserId, EmailId, Locale, OrderDate, ShippingInformation, BillingInformation, TotalPrice y CreditCardInformation de dicho objeto tampoco admiten valores nulos.  *Post-Condiciones:* Una invocación exitosa a esta interfaz retornara un identificador único de la orden enviada.  *Restricciones de uso:*   * Autorización: Esta operación puede ser invocada únicamente por usuarios previamente identificados en el sistema. * Acceso concurrente: No existen limitaciones en cuanto al acceso concurrente. El servicio se implementa utilizando EJBs de modo que el acceso concurrente estará gestionado por el contenedor de EJB correspondiente.   *Manejo de errores:*   * InvalidPOException: El servicio retorna esta excepción en caso de que la orden enviada sea nula o en caso de que alguna de sus propiedades requeridas sean nulas. * ProcessingException: Cualquier falla en el servicio, luego del proceso de validación provocara una excepción de este tipo. * RemoteException: Las fallas relacionadas con los mecanismos de comunicación remota serán encapsuladas por este tipo de excepción.   *Atributos de Calidad:*   * Modificabilidad: El servicio se implementa como un componente asíncrono lo que resulta en un bajo acoplamiento entre el elemento Consumer Website y el modulo OPC. Adicionalmente, la interfaz SOAP definida en el WSDL puede ser versionada, soportando múltiples versiones de la misma. |
| **OpcOrderTrackingService** | **GetOrderDetails**  *Sintaxis:* OrderDetails getOrderDetails(String orderId)  *Semántica:* Esta operación permite consultar el estado de las órdenes de compra.  *Pre-Condiciones:* El parámetro *orderId* debe ser no nulo y debe representar un identificador valido de una orden de compra existente.  *Post-Condiciones:* Retorna los detalles de una orden de compra, que incluye el estado de las peticiones a cada una de las aerolíneas involucradas, el estado de las reservas de hotel y de los proveedores de actividades turísticas.  *Restricciones de uso:*   * Autorización: Esta operación puede ser invocada únicamente por usuarios autenticados. Un cliente puede consultar únicamente las ordenes que el mismo creo/envió. Un representante de ventas puede consultar cualquier orden. * Acceso concurrente: No existen limitaciones en cuanto al acceso concurrente. El servicio se implementa utilizando EJBs de modo que el acceso concurrente estará gestionado por el contenedor de EJB correspondiente.   *Manejo de errores:*   * OrderNotFoundException: El servicio arroja este tipo de excepción en caso de no encontrar el identificador de la orden de compra en el repositorio de la aplicación o en caso de recibir un identificador nulo. * RemoteException: Las fallas relacionadas con los mecanismos de comunicación remota serán encapsuladas por este tipo de excepción.   *Atributos de calidad:*   * Eficiencia: Esta operación debe retornar los resultados con una latencia máxima de 1 segundo. |

#### Comportamiento

Detalles de comportamiento son provistos en las secciones [referencia comportamiento].

#### Decisiones de diseño

**Granularidad del servicio web OpcPurchaseOrderService**

Actualmente, existe una única interfaz y servicio web que expone toda la funcionalidad para la creación de las órdenes de compra, incluyendo las órdenes para los vuelos, alojamiento y actividades turísticas. No existen interfaces individuales para las reservas de tickets aéreos, hoteles y actividades.

Por un lado, esto representa limitaciones en cuanto a la flexibilidad para componer las órdenes para los diferentes paquetes turísticos. Por otro lado; sin embargo, esto beneficia la eficiencia de la aplicación al enviar toda la orden de compra completa a partir de una única invocación remota.

**Uso de Java-To-WSDL para el diseño de las interfaces**

Los componentes correspondientes a los módulos “Consumer Website” y “Order Processing Center” residen en la misma organización. Por este motivo, se decidió aprovechar las ventajas en cuanto a la facilidad de desarrollo que provee el compilador JavaToWsdl frente a la estabilidad que proporcionaría el mantenimiento manual del descriptor WSDL.

**Uso de JAX-RPC para el intercambio de datos en los servicios web**

Por los mismos motivos presentados en el punto anterior, se evita utilizar mecanismos complejos de procesamiento XML y se pasan parámetros como objetos Java (POJO). Esto representa un impacto negativo en cuanto a la interoperabilidad pero simplifica ampliamente la implementación.

**Uso de servicios web y protocolos SOAP y WSDL**

Los servicios web son publicados a partir del estándar WSDL en una ubicación estática predeterminada ya que los mismos no están disponibles para el uso público general. Son utilizados únicamente por el “Consumer Website”. La decisión de utilizar servicios web SOAP en lugar de RMI o EJBs directamente se justifica a partir de la posibilidad de reemplazar la implementación de “Consumer Website” utilizando otra tecnología (como ser Microsoft .NET): SOAP provee niveles de interoperabilidad óptimos en este eventual escenario.

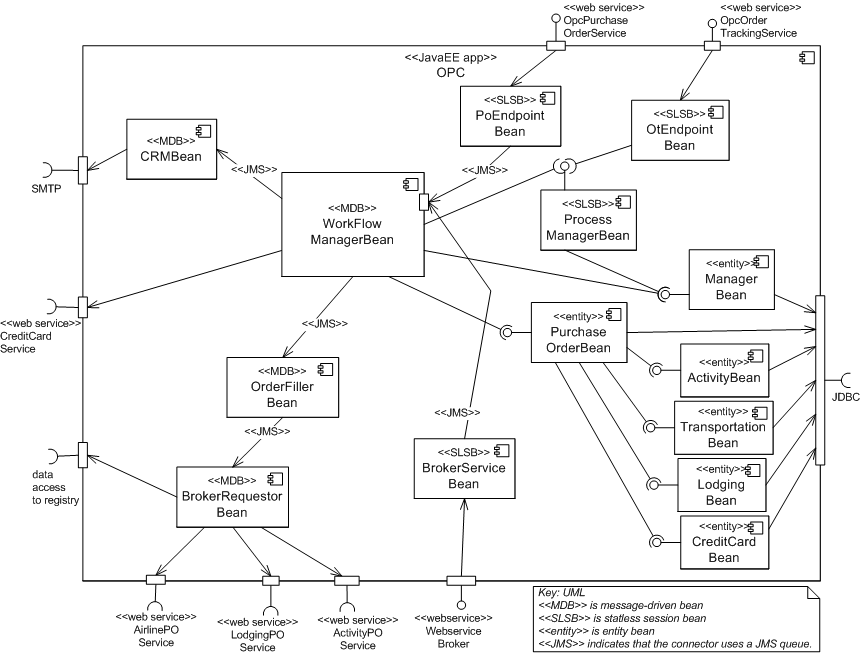
**Uso de EJBs**

Se escogió utilizar componentes EJB debido a que el “Order Processing Center” se implementa a partir de un conjunto de “session beans”.

* 1. **Punto de vista de Componentes y conectores**

### Vista de Componentes y Conectores de OPC

#### Representación primaria

****

#### Catalogo de elementos

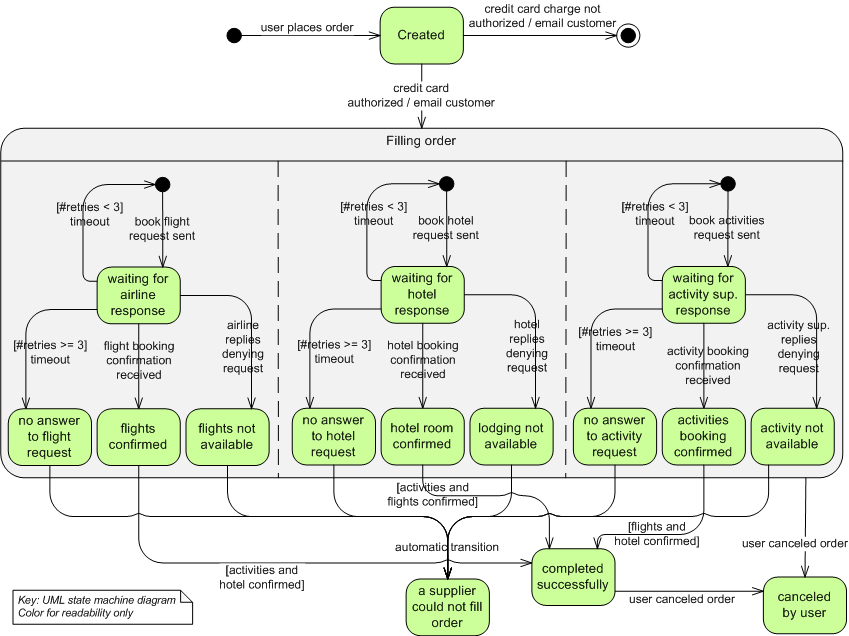
|  |  |  |
| --- | --- | --- |
| **Componente/conector** | **Tipo** | **Descripción** |
| **PoEndPointBean** | Stateless Session Bean | Este “stateless session bean” implementa el servicio web de la interfaz SOAP llamada OpcPurchaseOrderService. Cuando una orden de compra arriba al servicio, éste simplemente la valida y en caso positivo la envía al elemento WorkFlowMgrQueue utilizando JMS. |
| **OtEndPointBean** | Stateless Session Bean | Este “stateless session bean” implementa el servicio web de la interfaz SOAP llamada OpcOrderTrackingService. Las peticiones de información sobre una determinada orden son gestionadas al interactuar con el elemento ProcessManagerBean. La información de las ordenes es obtenida utilizando el entity bean PurchaseOrderBean. |
| **WorkFlowManagerBean** | Message-driven bean | Este “message-driven bean” es activado cuando llega un mensaje en la cola WorkFlowMgrQueue. Este componente lleva a cabo el procesamiento de 2 tipos de mensajes:   * Purchase Order Message: Al procesar estos mensajes, este componente interactua con el elemento ProcessManagerBean para crear un registro en la base de datos que representa una orden en estado Pendiente. Luego interactúa de forma síncrona con el proveedor externo de servicios financieros para validar y cargar la tarjeta de crédito del cliente. Si esta validación es satisfactoria, envía un mensaje a la cola JMS OrderFillerQueue para ser procesado por el componente OrderFillerBean. Finalmente, envía otro mensaje a la cola JMS CRMQueue que será procesado por el componente CrmBean para enviar un correo electrónico de notificación al cliente. * Invoice Message: Este es un mensaje que proviene de un proveedor de servicios externo en respuesta a una orden de reserva. Cuando se recibe un mensaje de este tipo, este componente básicamente interactúa con el ProcessManagerBean para actualizar el estado de la orden correspondiente.   Adicionalmente, este componente configura un timer en el contenedor EJB de modo de activarse periódicamente para verificar el estado de todas las ordenes pendientes.  En la sección 3.2.1.4 se puede observar el ciclo de vida de una orden de compra tal como se implementa en este componente. |
| **PurchaseOrderBean** | Entity Bean | Este componente almacena los detalles de una orden de compra. El estado de la orden no es gestionado ni almacenado por este elemento sino que dicha tarea la lleva a cabo el componente ManagerBean. |
| **ActivityBean** | Entity Bean | Este “entity bean” almacena los detalles correspondientes a la reserva de una actividad turística (identificador, ubicación, precio, fecha, etc). |
| **TransportationBean** | Entity Bean | ... |
| **LodgingBean** | Entity Bean | ... |
| **CreditCardBean** | Entity Bean | ... |
| **ProcessManagerBean** | Stateless Session Bean | Este “session bean” provee operaciones para recuperar y actualizar el estado general de una orden de compra y el estado de las ordenes individuales de cada proveedor de servicios. |
| **ManagerBean** | Entity Bean | Este “entity bean” es utilizado por el WorkFlowManagerBean para persistir el estado de una orden de compra y el estado de cada una de las ordenes individuales de los proveedores de servicio. |
| **OrderFillerBean** | Message-driven bean | ... |
| **BrokerRequestorBean** | Message-driven bean | ... |
| **BrokerServiceBean** | Stateless session bean | ... |
| **CRMBean** | Message-driven bean. | ... |

#### Interfaces

Los detalles correspondientes a las interfaces de los servicios web *powebservice* y *otwebservice* son documentados en la sección 3.1.3.3.

#### Comportamiento

El siguiente diagrama describe el ciclo de vida de las órdenes de compra tal como se implementa en el componente WorkFlowManagerBean.



#### Relación con elementos lógicos

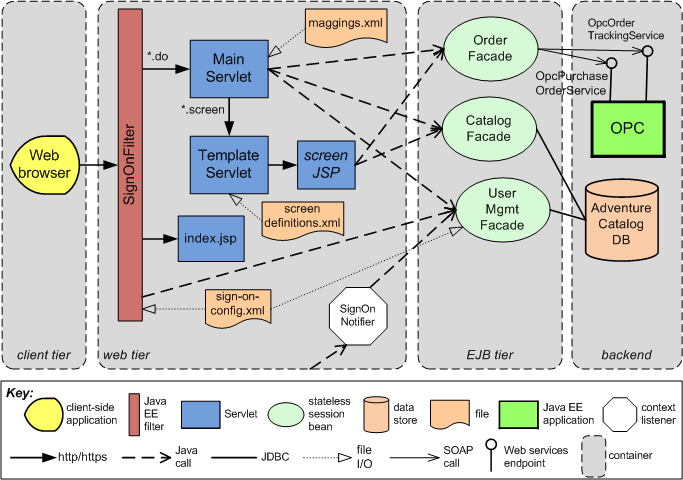
…

#### Decisiones de diseño

…

### Vista de Capas Físicas del sistema AdventureBuilder.

#### Representación primaria

****

#### Catalogo de elementos

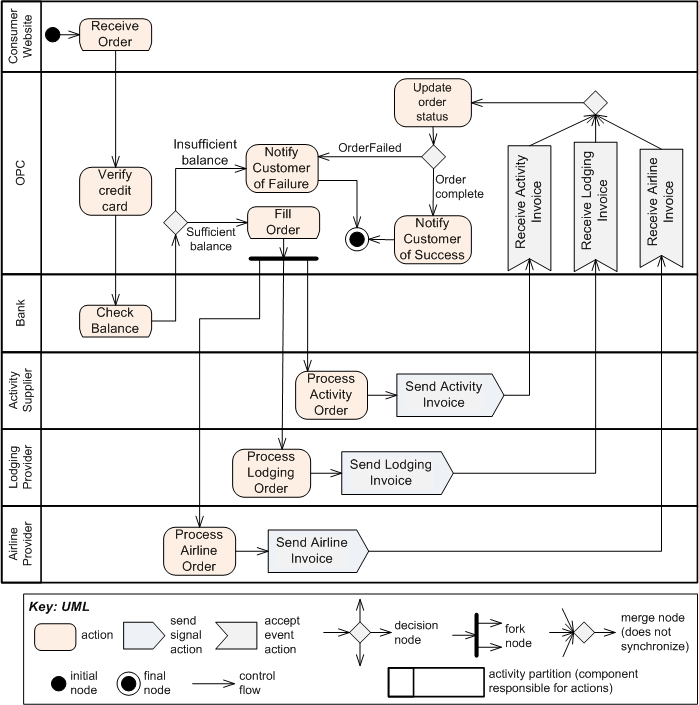
|  |  |  |
| --- | --- | --- |
| **Componente/conector** | **Tipo** | **Descripción** |
| **Web Browser** | Cliente web | Este componente representa la interfaz de usuario de la aplicación ejecutando en un navegador web. Consumer Website es una aplicación Web 2.0 implementada utilizando el framework GWT. A partir de esto, ademas de código HTML, el navegador web ejecuta código JavaScript que utiliza Ajax para la comunicación con el servidor. |
| **SignOnFilter** | Java EE Filter | Este componente es un filtro JEE. Intercepta todas las peticiones HTTP y verifica si el usuario actual está autorizado para realizar la transacción seleccionada (a partir de la URL). Si un usuario anónimo solicita una página que requiere seguridad, este filtro redirecciona al usuario al formulario de autenticación. |
| **sign-on-config.xml** | Archivo de configuración. | Este archivo XML contiene información configurable sobre las restricciones de autenticación y autorización para determinadas URLs del sistema. |
| **MainServlet** | Servlet | Este servlet forma parte del framework WAF. Corresponde al elemento Controlador en la implementación del patrón MVC dentro del framework WAF. Este componente procesa todas las peticiones HTTP a URLs con extensión “.do”. Internamente, este servlet utiliza una tabla configurable que se carga durante su inicialización que mapea cada URL particular a “Action Classes” que procesan las peticiones. Estas “Action Classes” crean instancias de clases POJO e interactúan con los diferentes session beans (fachadas) del sistema. La tabla de mapeos de URLs se lee desde el archivo de configuración mappings.xml. |
| **mappings.xml** | Archivo de configuración. | Archivo de configuración XML que contiene el mapeo entre URLs (con terminación “.do”) y “action classes” (que extienden la clase HTMLActionSupport). |
| **TemplateServlet** | Servlet | Este servlet también forma parte del framework WAF. |
| **screendefinitions.xml** | Archivo de configuracion. | Todas las pantallas del sitio web tienen la misma estructura. Un titulo, una imagen inicial, un menú de acciones, un pie de página y el cuerpo de la página. Este archivo define el mapeo entre los nombres de las pantallas y un conjunto de JSPs que componen la estructura mencionada de cada una de las pantallas. |
| **Screen JSP** | Java Server Pages | Una JSP corresponde una parte de una pantalla. La información a desplegar en cada JSP se configura en la petición HTTP por el servlet MainServlet cuando una acción es ejecutada. |
| **OrderFacade** | Stateless Session Bean | Este “stateless session bean” es una implementación eficiente y ligera del patrón “Session Facade”. Provee operaciones que corresponden a la lógica de negocio de los escenarios relacionados con las órdenes de compra. En particular, este componente provee operaciones para crear órdenes de compra y para el seguimiento posterior del estado de dichas ordenes. Para llevar a cabo estas operaciones, este elemento interactúa con el OPC mediante servicios web SOAP. |
| **CatalogFacade** | Stateless Session Bean | Este “stateless session bean” es una implementación eficiente y ligera del patrón “Session Facade”. Provee operaciones que permiten realizar búsquedas sobre el catalogo de AdventureWorks. Internamente, este elemento interactúa con clases DAO para obtener información de la base de datos del catalogo de AdventureWorks (Adventure Catalog DB). |
| **UserMgmtFacade** | Stateless Session Bean | Este “stateless session bean” es una implementación eficiente y ligera del patrón “Session Facade”. Provee operaciones para realizar consultas sobre los datos de usuarios, actualizar dichos datos, autenticar usuarios y gestionar la seguridad del sistema. |
| **OPC** | Aplicación Java EE | Ver OPC o OpcApp en secciones anteriores. |

#### Interfaces

Los detalles correspondientes a las interfaces de los servicios web *powebservice* y *otwebservice* son documentados en la sección 3.1.3.3.

#### Comportamiento

El siguiente diagrama describe el procesamiento de una orden de compra dentro del sistema AdventureBuilder.



#### Relación con elementos lógicos

…

#### Decisiones de diseño

**WAF**

El framework WAF fue elegido porque facilita la implementación del Consumer Website al ofrecer clases “template” para el uso del patrón MVC. Para una determinada funcionalidad u operación del usuario, el desarrollador implementa un “action class” (controlador) y las páginas JSP que corresponden a las pantallas de usuario (vista). El desarrollador también utiliza los archivos de configuración para proporcionar un mapeo configurable entre las acciones, los “action class”, eventos y pantallas. La infraestructura WAF puede tomar automáticamente peticiones HTTP e invocar los “action class” y las pantallas JSP.

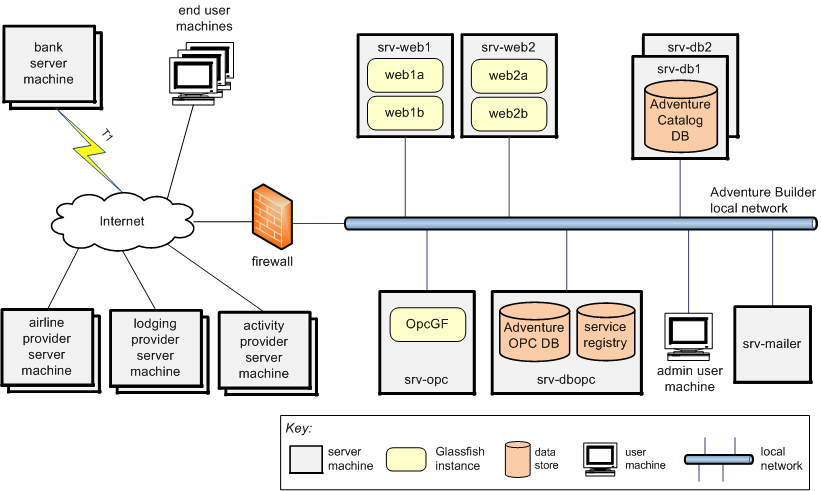
WAF también proporciona soporte para la comunicación basada en eventos y la internacionalización.

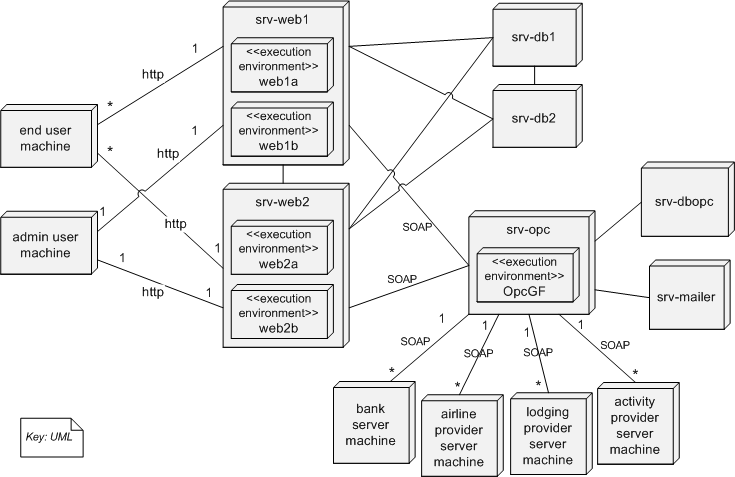
Una alternativa a WAF era utilizar el framework Spring. Spring era una solución más robusta y rico desde el punto de vista técnico, pero fue rechazada debido a que el equipo de desarrollo no está familiarizado con Spring y muy familiarizado con WAF.

* 1. **Punto de vista Físico.**

### Vista de Despliegue

#### Representación primaria

****

****

#### Catalogo de elementos

|  |  |  |
| --- | --- | --- |
| **Nodo** | **Características (velocidad, memoria, etc.)** | **Descripción** |
|  |  |  |
|  |  |  |
|  |  |  |

#### Relación con componentes

|  |  |
| --- | --- |
| **Nodo** | **Componente** |
|  |  |
|  |  |
|  |  |

#### Decisiones de diseño