PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

Guillermo Figueroa - Fernando Florenzano - Matias Jünemann

IIC2143 - Ingeniería de Software
Entrega 3 - Codificiación de Diagrama de Clases UML

1. Versión "Elegida": A

Decidimos quedarnos con nuestra primera versión (A) del diagrama de clases UML de la entrega anterior para modelar ChilExplox. Nuestra principal razón fue el de poder representar casi lo mismo que el segundo diagrama, que era más extenso, pero con menos clases y relaciones. No involucraba tantas clases herederas de buzones e interfaces como proponia el otro, lo que disminuye el acoplamiento al modelo y le aumentaba su cohesión. Aún así, rápidamente nos dimos cuenta de varias cosas que el segundo modelo implementaba que podian favorecer al modelo en A. Ya sea lo mejoraba o solucionaba secciones no modeladas en el primer intento. Cosas como la creacion de clases Dirección, Notificación, etc... Finalmente el modelo al cual llegamos es un buen matrimonio entre los 2 modelos anteriores. A continuación se especifican las clases que resultaron y se codificaron del modelo , como también las clases añadidas y cambios realizados al modelo anterior.

2. Clases y Características

Se llegó a las siguientes clases propias de modelar:

2.1. Backend

- **ChilExplox (ChilExplox.java)**: entidad inicial del software que representa a la empresa misma. Contiene la información de todas las sucursales de la empresa y los usuarios del sistema (funcionarios de la empresa), y a su vez permite a un usuario ingresar al sistema en una determinada sucursal. Se agregá metodo logout para ingresar eventualmente a otra sucursal.
- **Sucursal (Subsidiary.java)**: representan las distintas sucursales a las que se pueden acceder en el sistema y es la clase principal del modelo, de donde se acceden a los métodos de creación y envío pedidos. Implementa todas estas operaciones al nivel de sucursal ya que es donde el usuario debe acceder y trabajar. Es por esto que contiene información local, propia de cada sucursal, como los pedidos realizados en cada una, la flota de camiones (o vehículos) y los camiones (vehículos) que se encuentran en la sucursal para dejar una encomienda. Admite los metodos necesarios para editar la encomienda (no pedido ya que son las encomiendas las con direcciones), también metodos para el envio y recepción de camiones con encomiendas. Las responsabilidades de calculo de precios se delegó a otra clase detallada más adelante: Calculadora de precios.
- **Pedido (Order.java)**: son los conjuntos de encargos que pueden hacer los clientes, se componen de encomiendas, de quienes dependerá el precio del pedido completo y tiene un estado que debe actualizar dependiendo si ya sus partes ya fueron entregadas o no.

- **Encomienda (Parcel.java)**: es la unidad de paquetes de los que están hechos los pedidos. Estos en si solo contienen información y puede actualizar su estado de entre las distintas posiblidades detalladas más adelante.
- **Cliente (Client.java)**: es quien realiza los pedidos y encomiendas y piden su despacho. Actualmente no realiza ninguna operación ya que le sistema solo necesita su información. Es almacenado únicamente dentro de los pedidos ya que por el momento no es necesario almacenar los clientes de la empresa.
- **ITransporte (ITransport.java)**: interfaz que describe el comportamiento de los vehículos que pueden realizar despachos. Actualmente la única clase que la implementa es Camión debido a que es el único medio que utiliza la empresa, por ahora. Estos vehículos deben ser capaces de cargar encomiendas, actualizar su estado de despacho, moverse y realizar los despachos, descargar las encomiendas, devolverse, tiene disponibililidad, patente única, máximo número de encomiendas, sucursal padre, etc... Estos últimos eran atributos especificos para camión pero que pon conveniencia, están mejor como genéricos para todo transporte.
- **Camión (Truck.java)**: único medio de transporte conocido que utiliza la empresa para realizar los despachos de las encomiendas. Implementa ITransporte.
- **Usuario (User.java)**: Representa a los funcionarios de la empresa que pueden acceder al sistema y trabajar en el. Su información está contenida en ChilExplox y acceden a una sucursal mediante ella. También es el usuario quien hace uso del buzón de mensajes la sucursal. Tiene un nombre de usuario, nombre y contraseña que debe usar para entrar a una sucursal.
- **Mensajería (Messaging.java)**: representa sistema de mensajes de la empresa que intercambia mensajes entre sucursales. No interactúa directamente con la sucursal si no que con el buzón de mensajes de cada una. Envia desde una el mensaje y lo hace recibir al otro buzón.
- **Buzón (Mailbox.java)**: es el contenedor de mensajes propio de cada sucursal con el que interactúa la mensajería. Contiene los mensajes que ha intercambiado con cada sucursal y mediante mensajería logra enviar y recibir mensajes.
- **Mensaje (Message.java)**: es la unidad de comunicación entre sucursales, contiene información simple como el remitente, a quien va dirigido y el contenido. Se le agrego un tipo para diferenciar los mensajes regulares de los errores para edicion de encomiendas.
- **Añadido: Dirección (Address.java)**: es una clase que almacena especificaciones de lugares geográficos incluyendo dirección, ciudad, calle, número. Un string era muy básico y límitado para su representación ademas nos otorga libertad sobre los operadores como la comparación.
- **Añadido: Notificación (Notification.java)**: clase utilizada notificar y actuar de recordador que hay pedidos sin enviar aún y debén ser atendidos. Especificado para notificaciones en general, diferenciado de mensajes normales.
- **Añadido: Centro de notificaciones (NotificationCenter.java)**: clase utilizada para controlar la aparición de notificaciones. Esta clase no estaba presente en ninguno de los modelos UML pero consideramos necesaria y propia de cada sucursal, al deber tener un punto donde las notificaciones se almacenen y organicen. Es donde se crean estás, almacenan y eventualmente borran cuando la notificación es resuelta.
- **Añadido: Calculadora de presupuesto (BudgetCalculator.java)**: esta clase estatica que calcula el presupuesto de un pedido. Esta clase no se encontraba en ninguno de los modelos, pero consideramos es necesaria ya que el calculo de presupuestos podría llegar a ser una operación compleja que toma muchos factores distintos.

- **Añadido: Tipo de Mensaje (MessageType.java)**: no una clase propiamente tal, si no una enumeración para reflejar y diferenciar los mensajes regulares de los detección de error. Se creo una enumeración por su fácil expansión a nuevos tipos y por las facilidades que java brinda en su creación.
- **Añadido: Estado (State.java)**: al igual que Tipo de Mensaje, Estado es un enumeración de los 4 distintos posibles estados para un pedido, una orden e incuso se expandió la definición para la disponibilidad de los camiones. Son:"En Origen", "En Transito", "En Destinoz "Entregado". Como las enumeraciones permiten definición de metodos, los metodos de actualización de estados podrian migrar desde pedidos y orden a este archivo.
- **Añadido: Cargador (Loader.java)**:Se encarga de serializar y deserializar los objetos de forma que pueda guardar el estado de la aplicación, en un futuro esto quiere ser remplazado por una base de datos.

2.2. Frontend

- **Aplicación ChilExplox (ChilExploxApp.java)**: versión muy temprana de Frontend del software, usado principalmente para probar el Backend actualmente.

3. Ventajas

3.1. Descentralización de la Información

El modelo propuesto, trata de independizar la información de cada sucursal para un manejo local de esta y así aumentar la velocidad de respuesta, en otras palabras, disminuir el tiempo de las consultas en la sucursal, eliminando información irrelevante a la sucursal y así descentralizándola.

3.2. ITransporte, Enum Estado y Enum Tipo Mensaje

Al implementar la interfaz ITransporte, y las enumeraciones Estado y Tipo de Mensaje, se abren las puertas a nuevos tipos de implementaciones en cualquiera de estas áreas. Ya sea un nuevo transporte que la empresa quisiera implementar en el futuro además de los camiones, nuevos tipos de mensajes que quieran diferenciarse, etc...

3.3. Clientes, Dirección

Implementamos la clase Cliente y dirección en vez de simplemente almacenar su información como atributos por si luego se quisiera guardar su información en el sistema. Pensado como una forma de almacenar los clientes frecuentes en una próxima versión del sistema y poder de auto-completar de datos. Al igual con las direcciones. Pero por ahora es solamente un contenedor de información para el pedido.

3.4. Acoplamiento y Cohesión

El modelo escogido tiene un nivel de acoplamiento bajo y alta cohesión en comparación al otro modelo. Lo cual permite trabajar en las clases sin que tantas uniones sean utilizadas, y las operaciones asignadas a cada clase son más focalizadas. Se incentiva mucho el principio Creador en el modelo, al ser por lo general aquellas clases contenedoras de otras sus creadores únicamente.