8 Modelo de Diseño

El modelo de diseño es un refinamiento y formalización adicional del modelo de análisis donde se toman en cuenta las consecuencias del ambiente de implementación. El resultado del modelo de diseño son especificaciones muy detalladas de todos los objetos, incluyendo sus operaciones y atributos.

Se requiere un modelo de diseño ya que el modelo de análisis no es lo suficientemente formal para poder llegar al código fuente. Por tal motivo se debe refinar los objetos, incluyendo las operaciones que se deben ofrecer, la comunicación entre los diferentes objetos, los eventos que los objetos envían entre si, etc. El sistema real debe adaptarse al ambiente de implementación. En el análisis se asume un mundo ideal para el sistema, en la realidad se debe adaptar el sistema al ambiente de implementación, algo que puede cambiar durante el ciclo de vida del sistema. Se busca además aspectos como, los requisitos de rendimiento, necesidades de tiempo real, concurrencia, el lenguaje de programación, el sistema de manejo de base de datos, etc. Se desea también validar los resultados del análisis. Según el sistema crece y se formaliza, se verá qué tan bien los modelos de requisitos y análisis describen al sistema. Durante el diseño, se puede ver si los resultados del análisis son apropiados para su implementación. Si se descubre aspectos que no están claros en alguno de los modelos anteriores, estos deben ser clarificados, quizás regresando a etapas anteriores.

Aunque esto pudiera verse como deficiencias del resultado de las fases anteriores que deben ser clarificadas aquí, esto sería una visión incorrecta de las diferentes etapas del desarrollo, ya que el propósito de los modelos de requisitos y análisis es comprender el sistema y darle una buena estructura. También es importante comprender que las consideraciones tomadas en cuenta durante el diseño deben influir en la estructura del sistema lo menos posible. Es la propia aplicación la que controla la estructura, no las circunstancias de su implementación.

Se considera el modelo de diseño como una formalización del espacio de análisis, extendiéndolo para incluir una dimensión adicional correspondiente al ambiente de implementación, como se puede ver en el diagrama de la Figura 8.1.

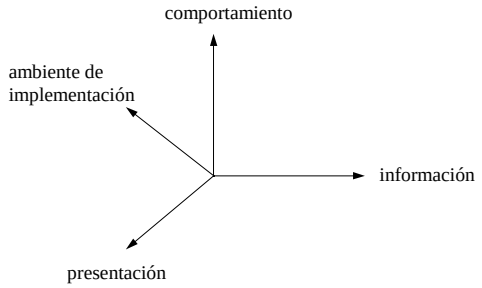

Figura 8.1 El diseño añade el ambiente de implementación como un nuevo eje de desarrollo.

Esta nueva dimensión correspondiente al ambiente de implementación debe considerarse al mismo tiempo que el propio modelo es refinado. La meta es refinarlo hasta que sea fácil escribir código fuente. Como el modelo de análisis define la arquitectura general del sistema, se busca obtener una arquitectura detallada como resultado del modelo de diseño, de manera que haya una continuidad de refinamiento entre los dos modelos, como se puede ver en el diagrama de la Figura 8.2. En particular se puede apreciar el cambio en los modelos a partir de la introducción del ambiente de implementación

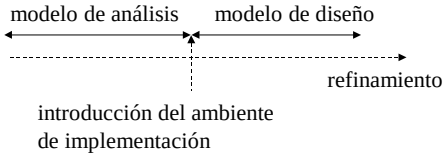

Figura 8.2 El modelo de diseño es una continuación del modelo de análisis.

La transición de análisis a diseño debe decidirse por separado para cada aplicación particular. Aunque es posible continuar trabajando sobre el modelo de análisis, incluso durante la incorporación del ambiente de implementación, esto no es recomendable, ya que aumenta su complejidad. Por lo tanto, es deseable tener un modelo de análisis ideal del sistema durante el ciclo de vida completo del sistema, dado que muchos de los cambios del sistema provienen de cambios en el ambiente de implementación. Tales cambios son entonces fácilmente incorporados, ya que el mismo modelo de análisis sirve de base para el nuevo modelo de diseño. De esta manera se puede ver el modelo de diseño como una especialización del modelo de análisis según un ambiente de implementación específico.

Si un cambio en el modelo de diseño proviene de un cambio en la lógica del sistema, entonces tales cambios deben hacerse en el modelo de análisis. Sin embargo, si el cambio es una consecuencia de la implementación, entonces tales cambios no deben ser incorporados en el modelo de análisis.
Las estructuras con las cuales se trabaja en el modelo de diseño son básicamente las mismas que en el modelo de análisis. Sin embargo, el punto de vista cambia, ya que se toma un paso hacia la implementación. El modelo de análisis debe se visto como un modelo conceptual y lógico del sistema, mientras que el modelo de diseño debe acercarse al código fuente. Esto significa que se cambia el punto del vista del modelo de diseño a una abstracción del código fuente final. Por lo tanto el modelo de diseño debe ser una descripción de cómo el código fuente debe ser estructurado, administrado y escrito.
En cierta manera este enfoque es una extensión del concepto de la separación de la "política" de la implementación, donde la política fue definida durante el modelo de análisis y el diseño tiene la responsabilidad mantener esta separación durante el diseño de métodos, aquellos que sirven para tomar decisiones (control) y aquellos que no (interface y entidad).
En general, cambios en la arquitectura del sistema para mejorar el rendimiento del sistema deben ser pospuestos hasta que el sistema esté (parcialmente) construido. La experiencia muestra que en los sistemas grandes y complejos, uno frecuentemente adivina incorrectamente cuales son los cuellos de botella críticos al rendimiento. Para hacer una evaluación más adecuada es necesario evaluar parte del rendimiento del sistema construido, algo que también se puede ir adelantando a nivel de prototipos.
Para llevar a cabo estos objetivos, se considera por separado los dos aspectos principales del modelo de diseño: el *diseño de objetos* y el *diseño de sistema*:

?? **Diseño de Objetos**. Se refina y formaliza el modelo para generar especificaciones muy detalladas de todos los objetos, incluyendo sus operaciones y atributos. Se describe cómo interaccionan los objetos en cada caso de uso específico, especificando qué debe hacer cada operación en cada objeto. Este paso genera las interfaces de los objetos, las cuales deben ser luego implementadas mediante métodos.

?? **Diseño de Sistema**. Se adapta el modelo al ambiente de implementación. Este paso incluye identificar e investigar las consecuencias del ambiente de implementación sobre el diseño. Aquí deben ser tomadas las decisiones de implementación estratégicas: (i) cómo se incorporará una base de datos en el sistema, (ii) qué bibliotecas de componentes se usarán y cómo, (iii) qué lenguajes de programación se utilizarán, (iv) cómo se manejarán los procesos, incluyendo comunicación y requisitos de rendimiento, (v) cómo se diseñará el manejo de excepciones y recolección de basura, etc. Por ejemplo, como se mencionó en el Capítulo 5, la mayoría de los lenguajes de programación no tienen forma de implementar directamente una *asociación*. Durante el diseño se debe decidir cómo mecanismos abstractos como la asociación, serán implementados. Similarmente, si el lenguaje de programación no ofrece ninguna técnica para apoyar herencia, se debe especificar cómo ésta será implementada. En resumen, se debe especificar cómo las circunstancias del ambiente de implementación deben ser manejadas en el sistema.

En general, si el ambiente de implementación tiene pocas consecuencias en el sistema, el diseño se basará casi exclusivamente en el diseño de objetos, o sea, en una extensión directa y detallada del modelo de análisis describiendo los atributos y operaciones del sistema. Por el contrario, si el ambiente de implementación afecta de manera importante al sistema, el diseño se basará en una combinación de diseño de objetos y diseño de sistema, o sea, en un modelo de análisis que dirije el resultado final del sistema aunque este será adaptado de manera importante al ambiente de implementación. En nuestro caso minimizaremos el efecto del ambiente de implementación sobre el sistema, razón por la cual comenzaremos con la descripción del diseño de objetos. Posteriormente describiremos los aspectos particulares relacionados con el diseño de sistema.
A continuación describimos algunas estrategias generales de diseño antes de proseguir con los aspectos específicos del diseño.

8.1 Estrategias de Diseño

Antes de poder resolver el diseño es necesario tomar decisiones generales sobre las estrategias de diseño a seguir. Algunas de las decisiones a tomar se presentan a continuación y se relacionan con aspectos que incluyen la *arquitectura*, *robustez*, *reuso* y *extensibilidad* del sistema.

Arquitectura

El término arquitectura se refiere, en nuestro caso, a la organización de las clases dentro del sistema. Durante el modelo de análisis se generó una arquitectura de clases para el sistema y se definió la funcionalidad "conceptual" ofrecida por las distintas clases dentro de la arquitectura. Durante el diseño esta arquitectura debe detallarse,

pudiéndose cambiar los aspectos considerados inicialmente, como fue la funcionalidad inicialmente asignada a cada clase, e incluso las propias clases, como hemos mencionado al inicio del capítulo.
El conocimiento y funcionalidad asignada a cada clase puede ser vista como la "inteligencia" de cada clase dentro del sistema. En otras palabras, algunas clases pueden ser vistas como más inteligentes que otras según el conocimiento y control que tengan sobre las demás clases. Por ejemplo, *colecciones* de objetos tales como *listas* o *arreglos*, no se consideran como particularmente inteligentes ya que pueden manipular y obtener información sobre las clases que almacenan, pero tienen relativamente poco impacto sobre estas u otras clases dentro del sistema. Por otro lado, un manejador de interface de usuario requiere mayor inteligencia, ya que debe poder administrar la interacción con el usuario, incluyendo manejo de eventos y manipulaciones sobre las pantallas. Una clase aún más inteligente es el controlador o manejador de la lógica completa de la aplicación, ya que es responsables de administrar a los propios manejadores de interface de usuario y relacionar su funcionalidad con el resto del sistema.
Como parte de la arquitectura de diseño se debe decidir cómo distribuir la inteligencia entre las clases y qué aspectos de la inteligencia total del sistema debe ser asignada a cada una de ellas. Para esto existen tres alternativas principales:

?? Un enfoque es minimizar el número de clases inteligentes. En el caso más extremo, sólo un objeto tendría conocimiento sobre todo el sistema. Todos los demás objetos tendrán un mínimo de inteligencia y el objeto inteligente servirá como controlador del resto. Una ventaja de este enfoque es que sólo se requeriría comprender el flujo de control dentro del objeto principal para comprender toda de la aplicación. Sin embargo, se vuelve más compleja la extensibilidad del sistema, ya que cualquier cambio en el flujo de control se llevaría a cabo en un mismo objeto afectando potencialmente la lógica de toda la aplicación. En cierta manera esto puede considerarse como la "estructuración" del programa, en otras palabras, transformando la orientación a objetos a programación estructurada, donde toda la aplicación consta de un solo "objeto".
?? Otro enfoque opuesto es distribuir la inteligencia del sistema lo más homogéneamente posible, diseñando todas las clases con inteligencia similar. Este enfoque va más con el espíritu de la orientación a objetos. Sin embargo, una distribución perfectamente homogénea es una tarea casi imposible, ya que los objetos varían en sus responsabilidades dependiendo de su razón de ser en la aplicación. Por otro lado, distribuyendo la inteligencia del sistema de manera homogénea entre los objetos permite que cada objeto sepa relativamente menos cosas. Esto produce objetos más pequeños y más fáciles de comprender. La desventaja es que la inteligencia del sistema va de la mano con la especialización de las clases. Si todas las clases son "inteligentes", esto significará que ellas serán muy especializadas, dificultando la extensibilidad del sistema que requiere mayor generalización en las clases.
?? El tercer enfoque es encontrar un balance entre los dos primeros. La idea es homogenizar la inteligencia del sistema sólo entre ciertas clases, tales como las de control. El resto de las clases serán "tontas" o genéricas, cómo las clases entidad e interface, permitiendo un buen porcentaje de extensibilidad en el sistema. Esto sigue la lógica introducida durante el modelo de requisitos y posteriormente análisis, donde se distingue entre las diversas razones de ser de las clases (comportamiento, presentación y dominio) para lograr una mayor robustez del sistema.

Robustez

La robustez de un sistema debe ser uno de los objetivos principales del diseño. Jamás debe agregarse funcionalidad o simplificar código a expensas de la robustez. El sistema debe estar protegido contra errores y debe al menos ofrecer diagnósticos para las fallas que aún pudiesen ocurrir, en particular aquellas que son fatales. Durante el desarrollo es a veces bueno insertar instrucciones internas en el código para descubrir fallas, aunque luego sean removidas durante la producción. En general se debe escoger lenguajes de programación que apoyen estos aspectos, como son el manejo de excepciones. Las principales consideraciones relacionadas con la robustez de un sistema son las siguientes:

?? El sistema debe estar protegido contra parámetros incorrectos proporcionados por el usuario. Cualquier método que acepte parámetros del usuario debe validar la entrada para evitar problemas. El diseñador de métodos debe considerar dos tipos de condiciones de error: (i) errores lógicos que son identificados durante el análisis y (ii) errores de implementación, incluyendo errores del sistema operativo, tales como los errores de asignación de memoria, o errores de archivos de entrada y salida, etc.
?? El sistema no debe optimizarse hasta que este funcione de manera correcta. A menudo los programadores le dedican demasiado esfuerzo a mejorar partes del código que se ejecutan poco frecuente. Optimizar requiere primero medir el rendimiento del sistema. Se debe estudiar las alternativas, como aspectos de memoria,

velocidad, y simplicidad de implementación. No se debe optimizar más de lo necesario, ya que la optimización compromete la extensibilidad, reuso y comprensión del sistema.

?? El sistema debe incluir estructuras de datos que no tengan límites predefinidos. Durante el diseño es difícil predecir la capacidad máxima esperada para la estructura de datos en la aplicación. Por lo tanto, se debe escoger estructuras de datos como las listas, a diferencia de los arreglos.

?? El sistema debe instrumentar un monitoreo de rendimiento y búsqueda de errores. El esfuerzo para llevarlo a cabo depende del ambiente de programación. Si el lenguaje de implementación no proporciona ningún apoyo, se pueden añadir métodos de impresión para cada clase. También se pueden añadir mensajes de entrada y salida a los métodos, imprimiendo selectivamente estos valores.

?? El encapsulamiento juega un papel fundamental para la robustez del sistema. Ocultar la información interna, atributos e implementación de métodos, a una clase permite que ésta pueda ser cambiada sin afectar al resto del sistema. Únicamente la interface delos métodos afecta a las demás clases.

Reuso

El reuso es un aspecto fundamental del diseño. Cuanto más se pueda reutilizar el código mejor será la robustez del sistema. Las siguientes son algunas estrategias para mejorar las posibilidades de reuso del diseño:

?? A través de la herencia se puede incrementar el reuso de código. Se toman los aspectos comunes a clases similares utilizando superclases comunes. Este enfoque es efectivo cuando las diferencias entre las clases son pequeñas y las similitudes son grandes. Es importante considerar la naturaleza de cada herencia para asegurar que no se está llegando a extremos donde la aplicación de la herencia sea inadecuada.

?? El uso impropio de herencia puede hace que los programas sean difíciles de mantener y extender. Como alternativa, la delegación provee un mecanismo para lograr el reuso de código pero sin utilizar herencia. Esto se basa en el uso de agregación a través de clases intermediarias que ocultan la funcionalidad de las clases a las cuales se delega.

?? El encapsulamiento es muy efectivo para lograr el reuso, pudiéndose aplicar tanto al nivel de los objetos como de componentes desarrollados en otras aplicaciones. Estos componentes pueden ser reutilizables como fueron diseñados a simplemente agregando nuevas interfaces.

Extensibilidad

La mayoría de los sistemas son extendidos en manera no prevista por el diseño original. Por lo tanto, los componentes reutilizables mejorarán también la extensibilidad. Las siguientes son algunas de las perspectivas de extensibilidad:

?? Nuevamente, se debe encapsular clases, ocultando su estructura interna a las otras clases. Sólo los métodos de la clase deben accesar sus atributos.

?? No se debe exportar estructuras de datos desde un método. Las estructuras de datos internas son específicas al algoritmo del método. Si se exporta las estructuras se limita la flexibilidad para poder cambiar el algoritmo más tarde.

?? Una clase debe tener un conocimiento limitado de la arquitectura de clases del sistema. Este conocimiento debe abarcar únicamente las asociaciones entre ella y sus vecinos directos. Para interactuar con un vecino indirecto, se debe llamar una operación del objeto vecino para atravesar la siguiente relación. Si la red de asociaciones cambia, el método de la clase puede ser modificado sin cambiar la llamada.

?? Se debe evitar expresiones de casos (*case*) sobre tipos de objetos. Para ello, se debe usar métodos (*polimorfismo*) para seleccionar el comportamiento a ejecutarse basado en el tipo del objeto en lugar de expresiones de *casos*. El polimorfismo evita muchas de estas comparaciones de tipos.

?? Se debe distinguir entre operaciones privadas y públicas. Cuando una operación pública es usada por otras clases, se vuelve costoso cambiar la interface, por lo cual las operaciones públicas deben ser definidas con cuidado. Las operaciones privadas son internas a la clase y sirven únicamente de ayudan para implementar operaciones públicas. Las operaciones privadas pueden ser removidas o su interface cambiada para modificar la implementación de la clase, teniendo un impacto limitado en los demás métodos de la clase.

8.2 Diseño de Objetos

El diseño de objetos es un proceso de añadir detalles al análisis y tomar decisiones junto con diseño de sistema, o sea al ambiente de implementación, de manera que podamos lograr una especificación detallada antes de comenzar la implementación final. Algunos de los aspectos a ser resueltos diseño de objetos son determinar cómo las clases, atributos y asociaciones del modelo de análisis deben implementarse en estructuras de datos especificas. También es necesario determinar si se requiere introducir nuevas clases en el modelo de diseño para los cuales no se tiene

ninguna representación en el modelo de análisis o si se requiere modificar o eliminar clases identificadas durante el modelo de análisis. Se debe agregar herencia para incrementar el reuso del sistema. También es necesario determinar los algoritmos para implementar las operaciones, así como todos los aspectos de optimizaciones.
Para el diseño de objeto se seguirá el *diseño por responsabilidades* (*RDD - Responsibility-Driven Design*). Este diseño está basado en un modelo *cliente-servidor* donde las clases son vistas como clientes cuando generan alguna petición hacia otra clase y como servidores cuando reciben peticiones de alguna otra clase. De tal manera, una misma clase puede ser vista en distintos momentos como cliente o servidor.
La funcionalidad ofrecida por las clases servidores se define en término de sus *responsabilidades*, las cuales deben ser satisfechas por lograr sus *servicios* con las demás clases. Los servicios y responsabilidades corresponderán finalmente a los métodos de la clase. A su vez, las clases servidores a su vez pueden tener *colaboraciones* con otras clases para lograr la satisfacción de responsabilidades que por si solas no pueden lograr. Como consecuencia de esto, se integran las responsabilidades y colaboraciones entre clases para definir *contratos* los cuales definen la naturaleza y alcance de las interacciones cliente-servidor. Los contratos y colaboraciones representan los requisitos de servicios entre los objetos. En resumen, se buscará identificar los contratos entre los objetos cliente y servidor diseñando su distribución entre los distintos objetos del sistema.
En la siguientes secciones describiremos estos conceptos y todo lo relacionado con el diseño por responsabilidades.
En las siguientes secciones especificaremos los aspectos mencionados a continuación con mayor detalle:

?? Especificación de las *tarjetas de clases.*
?? Identificación de las *responsabilidades* del sistema.
?? Identificación de las *colaboraciones* del sistema.
?? Identificación de las *jerarquías* de herencia del sistema.
?? Identificación de los *contratos* de servicio del sistema.
?? Identificación de los *subsistemas* del sistema.
?? Identificación de los *protocolos* del sistema.
?? Identificación de los *atributos* del sistema.
?? Especificación de los *algoritmos* del sistema.

Tarjetas de Clase

Las *tarjetas de clases* (también conocidas como tarjetas *CRC: Clase-Responsabilidad-Colaboración*) permiten al diseñador visualizar las diferentes clases de manera independiente y detallada. El esquema para una tarjeta de clase se muestra en la Tabla 8.1.

Clase:	
Descripción:	
Módulo:	
Estereotipo:	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	

Tabla 8.1. Diagrama para la tarjeta de clase.

La tarjeta se divide en tres secciones:

?? Encabezado consistiendo del nombre de la *clase*, una *descripción* de la clase (similar a la descrita en el diccionario de clases de análisis), el *módulo* al que pertenece la clase, el *estereotipo* de la clase (entidad, interface o control), las *propiedades* de la clase (abstracta o concreta), una lista de *superclases*, una lista de *subclases* y una lista de *atributos.*
?? Dos columnas debajo del encabezado, correspondientes a las *responsabilidades* (a la izquierda) y *colaboraciones* (a la derecha) de la clase. Eventualmente en la columna izquierda se incluirá información sobre los *contratos*. El número de filas en estas dos columnas es extensible y no está limitado al número que aparece en el diagrama de la Tabla 8.1.

Originalmente, detrás de cada tarjeta se agregaba una descripción corta del propósito de cada clase, algo que haremos a través del diccionario de clases. Además, incluimos algunos datos adicionales al esquema original *CRC*. Como ejemplo consideremos la clase *InterfaceUsuario*, una de las clases identificadas durante el modelo de análisis para el ejemplo del sistema de reservaciones de vuelo. La tarjeta de clase sería la que se muestra en la Tabla 8.2.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	

Tabla 8.2. Diagrama para la tarjeta de clase de *InterfaceUsuario*

De tal manera se debe especificar una tarjeta para cada clase en el sistema, donde inicialmente las tarjetas incluirán únicamente entradas para el nombre de la clase, módulo al que pertenecen y estereotipo correspondiente. Dado que aún no se ha identificado herencia, no habrán entradas para propiedades, superclase o subclase. Como se puede apreciar, inicialmente las tarjetas de clase tendrán información muy limitada, algo que a lo largo del diseño será extendido hasta alcanzar los métodos y atributos detallados. Una vez creadas estas tarjetas procedemos a identificar las responsabilidades de cada clase como veremos a continuación.

Responsabilidades

Uno de los esfuerzos más grandes del desarrollo y que involucra mayor complejidad es la especificación del comportamiento de cada una de las clases del sistema. A diferencia de la estructura interna de una clase, representada mediante sus atributos, los comportamientos corresponden a las operaciones y métodos de las clases. Dado que por lo general el número resultante de métodos en un sistema es mucho mayor que el de clases, el proceso de diseño involucra mayor complejidad que el de análisis. Si consideramos también la existencia de los atributos y las propias implementaciones de los métodos dentro de cada clase, la complejidad aumenta drásticamente. Sin embargo, esta complejidad no radica exclusivamente en el número de métodos, sino en el hecho que potencialmente todos los métodos de un objeto pueden ser llamados por todos los objetos del sistema. Esta es la verdadera fuente de la complejidad en la arquitectura del sistema, ya que cualquier cambio en uno de estos métodos afectará potencialmente a todo el resto del sistema. Por lo tanto, es necesario llevar a cabo un proceso muy cuidadoso para la identificación del comportamiento de las diversas clases.

En general, el comportamiento de las clases no debe descomponerse directamente en operaciones, sino seguir un procedimiento más natural comenzando con una descripción verbal de las *responsabilidades* o roles que tiene cada clase dentro del sistema. Obviamente, cada objeto instanciado de una misma clase tendrá responsabilidades similares a las descritas por la clase. Las responsabilidades ofrecidas por un objeto corresponden a los *servicios* apoyados por éste. A partir de estas responsabilidades se podrá eventualmente determinar las operaciones que cada objeto debe tener e incluso el conocimiento que el objeto posee.

Las responsabilidades se identifican a partir de los casos de uso generados durante el modelo de análisis. Para ello, se revisa el modelo de casos de uso, haciendo una lista o subrayando todos las frases descritas para determinar cuales de éstas representan acciones que algún objeto dentro del sistema deba ejecutar. Una de las decisiones más importantes durante la identificación de responsabilidades es a qué clase o clases se les debe asignar. Para ello se debe examinar el contexto en el cual se identificaron las responsabilidades.

Se debe asignar responsabilidades a las clases que almacenen la información más relacionada con la responsabilidad. En otras palabras, si un objeto requiere cierta información, es lógico asignarle la responsabilidad de mantener esa información. Si la información cambia, no será necesario enviar mensajes de actualización a otros objetos. Esto también significa que la responsabilidad de mantener la información no debe se compartida. Compartir información implica una duplicación que puede dar lugar a inconsistencias. Si más de un objeto tiene responsabilidad sobre la misma información se debería reasignar la responsabilidad a un sólo objeto. También puede ocurrir que cierta responsabilidad agrupe varias responsabilidades juntas. En tal caso se puede dividir o compartir la responsabilidad entre dos o más objetos. Si se vuelve difícil decidir a quien se debe asignar una responsabilidad, se

puede experimentar con diversas alternativas, con la ventaja de que durante las primeras etapas del diseño, este proceso no es muy costoso. Será mucho más difícil y tomará más tiempo hacerlo después de haber implementado el sistema completo.
En cuanto a la especificación de las responsabilidades, se debe describir las responsabilidades en términos generales, para poder luego encontrar más fácilmente responsabilidades compartidas entre clases. Por lo tanto, no es necesario considerar aspectos específicos de las responsabilidades, como nombres particulares o incluso parámetros. Otra consideración es asegurarse que los ciclos de responsabilidades y colaboraciones entre clases no sean interrumpidos y mantengan siempre una correspondencia con la arquitectura establecida durante el análisis.
Consideremos el sistema de reservaciones de vuelos. Dado que la complejidad de un sistema se incrementa radicalmente durante el diseño en relación a la arquitectura desarrollada durante el análisis, procederemos con la descripción de solamente una parte del sistema, mientras que el resto del diseño se mostrará en los apéndices. Para ello nos concentraremos inicialmente en los casos de uso relacionados con registro: *Registrar Usuario, Validar Usuario* y *Registrar Tarjeta*.

Validar Usuario

A continuación mostramos el flujo principal del caso de uso *Validar Usuario* como se presentó al final del capítulo de análisis.

Flujo Principal	El ManejadorPrincipal solicita *desplegarPantallaPrincipal* a la InterfaceUsuario. La InterfaceUsuario *despliega* la PantallaPrincipal. La PantallaPrincipal se *despliega*. El Usuario puede seleccionar entre las siguientes opciones: "Registrarse por Primera Vez", "OK" y "Salir". Si la actividad seleccionada es "Registrarse por Primera Vez", la PantallaPrincipal envía el evento "Registrarse por Primera Vez" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Registrarse por Primera Vez" al ManejadorPrincipal. El ManejadorPrincipal solicita *crearRegistroUsuario* al ManejadorRegistroUsuario. Se ejecuta el caso de uso *Registrar Usuario*, subflujo *Crear Registro Usuario* (*S*-1). Si la actividad seleccionada es "OK", se valida el registro de usuario mediante un *login* y un *password* insertados por el Usuario en la PantallaPrincipal. La PantallaPrincipal envía el evento "OK" a la InterfaceUsuario. La InterfaceUsuario envía el evento "OK" al ManejadorPrincipal. El ManejadorPrincipal solicita *validarRegistroUsuario* al ManejadorRegistroUsuario. El ManejadorRegistroUsuario solicita *validarRegistroUsuario* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro solicita *validarRegistroUsuario* a la Base de Datos Registro. La Base de Datos Registro valida al usuario y devuelve el *OK* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroUsuario. El ManejadorRegistroUsuario devuelve el *OK* al ManejadorPrincipal. Una vez validado el usuario (*E*-1), el ManejadorPrincipal solicita *ofrecerServicio* al ManejadorServicio. Se continúa con el caso de uso *Ofrecer Servicios*. Si la actividad seleccionada es "Salir", la PantallaPrincipal envía el evento "Salir" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Salir" al ManejadorPrincipal. El ManejadorPrincipal *sale* del sistema.

Tomamos cada una de las frases que describen el flujo y las analizaremos para decidir que responsabilidad representan y a que clase se le asignan:

1. **El ManejadorPrincipal solicita *desplegarPantallaPrincipal* a la InterfaceUsuario.** La primera pregunta es "cuál es la responsabilidad". La responsabilidad que podemos identificar aquí es "solicita *desplegarPantallaPrincipal*". La siguiente pregunta es "a quién se le asigna la responsabilidad". Existen dos opciones, asignar "solicita *desplegarPantallaPrincipal*" al ManejadorPrincipal o asignar "*desplegarPantallaPrincipal*" a la InterfaceUsuario. Para simplificar nuestra decisión y asegurarnos al menos en esta etapa de no perder ninguna información, tomaremos una de cisión "salomónica", asignaremos dos responsabilidades, una a cada clase. Esto es muy importante, ya que nos aseguramos de comenzar con un buen número de responsabilidades que podremos luego reorganizar durante el transcurso del diseño. Es importante resaltar que dado que muchas responsabilidades pudieran darse de manera duplicada, trataremos de evitar esto revisando las frases que pudiesen generar tales duplicaciones. Por lo tanto, asignaremos la responsabilidad inicial de "solicta *desplegarPantallaPrincipal*" al ManejadorPrincipal, definiendo la responsabilidad de manera completa como "solicta *desplegarPantallaPrincipal* a la InterfaceUsuario". Como veremos con la segunda frase, la responsabilidad de *desplegarPantallaPrincipal* será asignada de manera adecuada a la InterfaceUsuario por lo cual no será necesario hacerlo en este momento. Sin embargo, siempre revisamos que exista la

responsabilidad complementaria para la segunda clase en la frase. El objetivo esencial es asegurarse de asignar responsabilidades a las diferentes clases de manera que no se rompa con el flujo de colaboraciones, algo que debe tratarse con sumo cuidado para resolver cuanto antes el problema general de asignación de servicios. Se debe también recordar que habrán varias etapas dentro de la actividad de diseño donde se podrá afinar la asignación de responsabilidades.

2. **La InterfaceUsuario *despliega* la PantallaPrincipal**. De manera análoga a la oración anterior, la responsabilidad que identificamos es "*despliega* la PantallaPrincipal" y la asignamos a InterfaceUsuario utilizando la misma lógica anterior. Nótese que esta responsabilidad corresponde al servicio solicitado por ManejadorPrincipal en la oración anterior. Esto resalta el hecho de que las responsabilidades se están asignando de manera correcta y sin romper el flujo de colaboraciones.
3. **La PantallaPrincipal se *despliega***. Esta responsabilidad se asigna a la única clase involucrada que es PantallaPrincipal. Nótese también, que esta responsabilidad, "despliega", corresponde al servicio de despliegue de la PantallaPrincipal referido en la oración anterior.
4. **El Usuario puede seleccionar entre las siguientes opciones: "Registrarse por Primera Vez", "OK" y "Salir"**. En general los actores no son parte de la arquitectura del sistema por lo cual no se les asigna ninguna responsabilidad. Además, ¡los actores son bastante *irresponsables* (en particular los usuarios)!
5. **Si la actividad seleccionada es "Registrarse por Primera Vez", la PantallaPrincipal envía el evento "Registrarse por Primera Vez" a la InterfaceUsuario**. De manera análoga a las asignaciones anteriores, se asigna la responsabilidad "envía el evento "Registrarse por Primera Vez" a la InterfaceUsuario" a la PantallaPrincipal. Dado que en la siguiente frase la InterfaceUsuario envía a su vez el evento a otra clase, no agregamos ninguna responsabilidad adicional a esta última clase.
6. **La InterfaceUsuario envía el evento "Registrarse por Primera Vez" al ManejadorPrincipal**. De manera similar, se asigna la responsabilidad "envía el evento "Registrarse por Primera Vez" al ManejadorPrincipal" a la InterfaceUsuario. Regresando a la discusión original de la asignación de responsabilidades a ambas clase, podemos notar que la siguiente frase no asigna una responsabilidad de "recibir el evento" a la clase ManejadorPrincipal. Por tal motivo, haremos una segunda asignación de responsabilidad, la cual llamaremos "maneja el evento "Registrarse por Primera Vez"" y la asignaremos al ManejadorPrincipal.
7. **El ManejadorPrincipal solicita *crearRegistroUsuario* al ManejadorRegistroUsuario**. Se asigna la responsabilidad "solicita *crearRegistroUsuario* al ManejadorRegistroUsuario" al ManejadorPrincipal. Adicionalmente, asignamos la responsabilidad *crearRegistroUsuario* al ManejadorRegistroUsuario, ya que si nos fijamos más adelante, en el caso de uso *Registrar Usuario*, subflujo *Crear Registro Usuario* (*S*-1), no se continua con una frase que defina la misma responsabilidad complementaria.
8. **Se ejecuta el caso de uso *Registrar Usuario*, subflujo *Crear Registro Usuario* (*S*-1)**. Esta frase no genera ninguna responsabilidad, sólo una ramificación en el flujo del caso de uso.
9. **Si la actividad seleccionada es "OK", se valida el registro de usuario mediante un *login* y un *password* insertados por el Usuario en la PantallaPrincipal**. Oraciones que describen responsabilidades para actores no son incluidas ya que no agregan responsabilidades.
10. **La PantallaPrincipal envía el evento "OK" a la InterfaceUsuario**. La responsabilidad es "envía el evento "OK" a la InterfaceUsuario" y se asigna a PantallaPrincipal.
11. **La InterfaceUsuario envía el evento "OK" al ManejadorPrincipal**. La responsabilidad es "envía el evento "OK" al ManejadorPrincipal" y se asigna a InterfaceUsuario. Adicionalmente, se asigna la responsabilidad "maneja el evento "OK"" y se asigna al ManejadorPrincipal.
12. **El ManejadorPrincipal solicita *validarRegistroUsuario* al ManejadorRegistroUsuario**. La responsabilidad es "solicita *validarRegistroUsuario* al ManejadorRegistroUsuario" y se asigna a ManejadorPrincipal.
13. **El ManejadorRegistroUsuario solicita *validarRegistroUsuario* a la InterfaceBaseDatosRegistro**. La responsabilidad es "solicita *validarRegistroUsuario* a la InterfaceBaseDatosRegistro" y se asigna a ManejadorRegistroUsuario.
14. **La InterfaceBaseDatosRegistro solicita *validarRegistroUsuario* a la Base de Datos Registro**. La responsabilidad es "solicita *validarRegistroUsuario* a la Base de Datos Registro" y se asigna a InterfaceBaseDatosRegistro.
15. **La Base de Datos Registro valida al usuario y devuelve el *OK* a la InterfaceBaseDatosRegistro**. Esta frase no agrega responsabilidades ya que involucra un actor externo al sistema junto con un evento de devolución.
16. **La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroUsuario**. Esta frase no agrega responsabilidades ya que describe un evento de devolución.
17. **El ManejadorRegistroUsuario devuelve el *OK* al ManejadorPrincipal**. Nuevamente, esta frase no agrega responsabilidades ya que describe un evento de devolución.

18. **Una vez validado el usuario (*E*-1), el ManejadorPrincipal solicita *ofrecerServicio* al ManejadorServicio**. La responsabilidad es "solicita *ofrecerServicio* al ManejadorServicio" la cual se asigna a ManejadorPrincipal. Adicionalmente asignamos la responsabilidad "*ofrecerServicio*" a la clase ManejadorServicio para asegurarse que exista una responsabilidad complementaria en esta última clase.
19. **Se continúa con el caso de uso *Ofrecer Servicios***. Esta es una frase que describe continuación entre casos de uso y no agrega ninguna responsabilidad.
20. **Si la actividad seleccionada es "Salir", la PantallaPrincipal envía el evento "Salir" a la InterfaceUsuario**. Se asigna la responsabilidad "envía el evento "Salir" a la InterfaceUsuario" a la PantallaPrincipal.
21. **La InterfaceUsuario envía el evento "Salir" al ManejadorPrincipal**. Se asigna la responsabilidad "envía el evento "Salir" al ManejadorPrincipal" a la InterfaceUsuario. Se asigna adicionalmente la responsabilidad "maneja el evento "Salir"" al ManejadorPrincipal.
22. **El ManejadorPrincipal *sale* del sistema.** Se asigna la responsabilidad "*sale* del sistema" al ManejadorPrincipal.

A partir de estas frases obtenemos nuestras primeras responsabilidades y las insertamos en las tarjetas de clase correspondientes, logrando una versión preliminar de las tarjetas de clase. En general, las tarjetas tienen la ventaja de forzar a uno a ser breve, de tal manera que las responsabilidades se listan lo más compacto posible.
En la Tabla 8.3 se muestra las responsabilidades identificadas hasta el momento para la clase *ManejadorPrincipal*. Nótese, que agregamos entre paréntesis el índice de la frase de la cual provienen en la descripción del caso de uso.

Clase: ManejadorPrincipal	
Descripción: El manejador principal es el encargado de desplegar la pantalla principal de interacción con el usuario, y luego delegar las diferentes funciones a los manejadores especializados apropiados.	
Módulo: Principal	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *desplegarPantallaPrincipal* a la InterfaceUsuario (1)	
maneja el evento "Registrarse por Primera Vez" (6)	
solicita *crearRegistroUsuario* al ManejadorRegistroUsuario (7)	
maneja el evento "OK" (11)	
solicita *validarRegistroUsuario* al ManejadorRegistroUsuario (12)	
solicita *ofrecerServicio* al ManejadorServicio (18)	
maneja el evento "Salir" (21)	
sale del sistema (22)	

Tabla 8.3. Tarjeta para la clase *ManejadorPrincipal* con responsabilidades identificadas hasta el momento.

La Tabla 8.5 muestra las responsabilidades identificadas hasta el momento para la clase *PantallaPrincipal*.

Clase: PantallaPrincipal	
Descripción: Pantalla principal (*P*-1).	
Módulo: Principal	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega (3)	
envía el evento "Registrarse por Primera Vez" a la InterfaceUsuario (5)	
envía el evento "OK" a la InterfaceUsuario (10)	
envía el evento "Salir" a la InterfaceUsuario (20)	

Tabla 8.5. Tarjeta para la clase *PantallaPrincipal* con responsabilidades identificadas hasta el momento.

La Tabla 8.6 muestra las responsabilidades identificadas hasta el momento para la clase *ManejadorRegistroUsuario*.

Clase: ManejadorRegistroUsuario
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para

poder utilizar el sistema.	
Módulo: Registro.Usuario	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
crearRegistroUsuario (7)	
solicita *validarRegistroUsuario* a la InterfaceBaseDatosRegistro (13)	

Tabla 8.6. Tarjeta para la clase *ManejadorRegistroUsuario* con responsabilidades identificadas hasta el momento.

La Tabla 8.7 muestra las responsabilidades identificadas hasta el momento para la clase *InterfaceBaseDatosRegistro*.

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceBD	
Estereotipo: Interface	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *validarRegistroUsuario* a la BaseDatosRegistro (14)	

Tabla 8.7. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades identificadas hasta el momento.

La Tabla 8.8 muestra las responsabilidades identificadas hasta el momento para la clase *ManejadorServicio*.

Clase: ManejadorServicio	
Descripción: El manejador de servicios se encarga de enviar las peticiones particulares de servicios a los manejadores espacializados para consulta, reserva y compra.	
Módulo: Servicios	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
ofrecerServicio (18)	

Tabla 8.8. Tarjeta para la clase *ManejadorServicio* con responsabilidades identificadas hasta el momento.

Es posible también obtener responsabilidades a partir de las frase descritas en el manejo de excepciones, como se muestra a continuación.

Excepciones	*E-1 no hubo validación*: El *login*/*password* no se validó correctamente. Se le pide al usuario que vuelva a intentar hasta tres veces después de lo cual se saldrá del sistema.

Sin embargo, nos concentraremos únicamente en los flujos básicos y no los alternos. En general, los flujos alternos, como los de excepción, son los menos comunes y son importantes de diseñar pero no en una primera etapa.

Ofrecer Servicios

A continuación mostramos el flujo principal del caso de uso *Validar Usuario* como se presentó al final del capítulo de análisis.

Flujo Principal	El ManejadorServicio solicita *desplegarPantallaServicio* a la InterfaceUsuario. La InterfaceUsuario *despliega* la PantallaServicio. La PantallaServicio se *despliega*. El Usuario puede seleccionar entre las siguientes actividades: "Consultar Información", "Hacer Reservación", "Obtener Registro" y "Salir". Si la actividad seleccionada es "Consultar Información", la PantallaServicio envía el evento "Consultar Información" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Consultar Información" al ManejadorServicio. El ManejadorServicio solicita *consultar* al ManejadorConsultas. Se continúa con el caso de uso *Consultar Información*, subflujo *Consultar* (*S*-1). Si la actividad seleccionada es "Hacer Reservación", la PantallaServicio envía el evento "Hacer Reservación" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Hacer Reservación" al ManejadorServicio. El ManejadorServicio solicita *reservar* al ManejadorReservas. Se continúa con el caso de uso *Hacer Reservación*, subflujo *Solicitar Clave Reservación* (*S*-1). Si la actividad seleccionada es "Obtener Registro", la PantallaServicio envía el evento "Obtener Registro" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Obtener Registro" al ManejadorServicio. El ManejadorServicio solicita *registrar* al ManejadorRegistroUsuario. Se continúa con el caso de uso *Registrar Usuario*, subflujo *Obtener Registro Usuario* (*S*-2). Si la actividad seleccionada es "Salir", la PantallaServicio envía el evento "Salir" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Salir" al ManejadorServicio. El ManejadorServicio *sale* del sistema.

Nuevamente, tomamos cada una de las frases que describen el flujo y las analizaremos para decidir que responsabilidad representan y a que clase se le asignan:

23. **El ManejadorServicio solicita *desplegarPantallaServicio* a la InterfaceUsuario**. La responsabilidad es "solicita *desplegarPantallaServicio* a la InterfaceUsuario" y se asigna a ManejadorServicio.
24. **La InterfaceUsuario *despliega* la PantallaServicio**. La responsabilidad "*despliega* la PantallaServicio" se asigna a InterfaceUsuario.
25. **La PantallaServicio se despliega**. La responsabilidad es "*despliega*" y se asigna a PantallaServicio.
26. **El Usuario puede seleccionar entre las siguientes actividades: "Consultar Información", "Hacer Reservación", "Obtener Registro" y "Salir"**. Esta frase es informativa describiendo opciones del Usuario, por lo cual no se agregan responsabilidades adicionales.

Las siguientes frases permiten identificar responsabilidades relaciondas con los casos de uso *Consultar Información* y *Hacer Reservación*. Sin embargo, no desarrollaremos por el momento el diseño relacionado a estos casos de usos. Esto afecta a las frase 27-34.

27. **Si la actividad seleccionada es "Consultar Información", la PantallaServicio envía el evento "Consultar Información" a la InterfaceUsuario**.
28. **La InterfaceUsuario envía el evento "Consultar Información" al ManejadorServicio**.
29. **El ManejadorServicio solicita *consultar* al ManejadorConsultas**.
30. **Se continúa con el caso de uso *Consultar Información*, subflujo *Consultar* (*S*-1)**.
31. **Si la actividad seleccionada es "Hacer Reservación", la PantallaServicio envía el evento "Hacer Reservación" a la InterfaceUsuario**.
32. **La InterfaceUsuario envía el evento "Hacer Reservación" al ManejadorServicio**.
33. **El ManejadorServicio solicita *reservar* al ManejadorReservas**.
34. **Se continúa con el caso de uso *Hacer Reservación*, subflujo *Solicitar Clave Reservación* (*S*-1)**.

Retomamos el proceso de identificación de responsabilidades a partir de la siguiente frase, correspondiente a lógica relacionada con el caso de uso de *Registrar Usuario*.

35. **Si la actividad seleccionada es "Obtener Registro", la PantallaServicio envía el evento "Obtener Registro" a la InterfaceUsuario**. La responsabilidad es "envía el evento "Obtener Registro" a la InterfaceUsuario" y se asigna a PantallaServicio.
36. **La InterfaceUsuario envía el evento "Obtener Registro" al ManejadorServicio**. La responsabilidad es "envía el evento "Obtener Registro" al ManejadorServicio" y se asigna a la InterfaceUsuario. Asignamos de manera similar, la responsabilidad "maneja el evento "Obtener Registro"" al ManejadorServicio.
37. **El ManejadorServicio solicita *registrar* al ManejadorRegistroUsuario**. La responsabilidad es "solicita *registrar* al ManejadorRegistroUsuario" y se asigna al ManejadorServicio. Aunque deberíamos asignar al responsabilidad complementaria al ManejadorRegistroUsuario, si continuamos con el sublujo correspondiente podremos observar que la responsabilidad *obtenerRegistroUsuario* se asigna más adelante.

38. **Se continúa con el caso de uso *Registrar Usuario*, en el subflujo *Obtener Registro Usuario* (S-2)**. Esta oración no involucra ninguna responsabilidad.
39. **Si la actividad seleccionada es "Salir", la PantallaServicio envía el evento "Salir" a la InterfaceUsuario**. Se asigna la responsabilidad "envía el evento "Salir" a la InterfaceUsuario" a la clase PantallaServicio.
40. **La InterfaceUsuario envía el evento "Salir" al ManejadorServicio**. Se asigna la responsabilidad "envía el evento "Salir" a la ManejadorServicio" a la clase InterfaceUsuario. Adicionalmanete, se asigna "maneja el evento "Salir"" al ManejadorServicio.
41. **El ManejadorServicio *sale* del sistema.** Se asigna la responsabilidad "*sale* del sistema" al ManejadorServicio.

A partir de estas frases obtenemos nuevas responsabilidades y las insertamos en las tarjetas de clase correspondientes. Dado que no todas las clases agregan nuevas responsabilidades, iremos mostrando sólo aquellas que sí lo hacen. En particular, las responsabilidades identificadas para las clases *ManejadorPrincipal* y *PantallaPrincipal* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.3 y en la Tabla 8.5, respectivamente. De manera similar, las responsabilidades identificadas para las clases *ManejadorRegistroUsuario* y *InterfaceBaseDatosRegistro* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.6 y en la Tabla 8.7, respectivamente.

La Tabla 8.9 muestra las responsabilidades para la clase *InterfaceUsuario* anteriormente identificadas en la Tabla 8.4 junto con las nuevas responsabilidades.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega la PantallaPrincipal (2)	
envía el evento "Registrarse por Primera Vez" al ManejadorPrincipal (6)	
envía el evento "OK" al ManejadorPrincipal (11)	
envía el evento "Salir" al ManejadorPrincipal (21)	
despliega la PantallaServicio (24)	
envía el evento "Obtener Registro" al ManejadorServicio (36)	
envía el evento "Salir" al ManejadorPrincipal (40)	

Tabla 8.9. Tarjeta para la clase *InterfaceUsuario* con responsabilidades identificadas hasta el momento.

La Tabla 8.10 muestra las responsabilidades para la clase *ManejadorServicio* anteriormente identificadas en la Tabla 8.8 junto con las nuevas responsabilidades.

Clase: ManejadorServicio	
Descripción: El manejador de servicios se encarga de enviar las peticiones particulares de servicios a los manejadores espacializados para consulta, reserva y compra.	
Módulo: Servicios	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
ofrecerServicio (18)	
solicita *desplegarPantallaServicio* a la InterfaceUsuario (23)	
maneja el evento "Obtener Registro" (36)	
solicita *registrar* al ManejadorRegistroUsuario (37)	
maneja el evento "Salir" (40)	
sale del sistema (41)	

Tabla 8.10. Tarjeta para la clase *ManejadorServicio* con responsabilidades identificadas hasta el momento.

La Tabla 8.11 muestra las responsabilidades identificadas hasta el momento para la clase *PantallaServicio*.

Clase: PantallaServicio

Descripción: Pantalla de servicios (*P*-2).	
Módulo: Servicios	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega (25)	
envía el evento "Obtener Registro" a la InterfaceUsuario (35)	
envía el evento "Salir" a la InterfaceUsuario (39)	

Tabla 8.11. Tarjeta para la clase *PantallaServicio* con responsabilidades identificadas hasta el momento.

Este caso de uso no escpecifica responsabilidades adicionales a las mencionadas anteriormente, incluyendo responsabilidades por razones de manejo de excepciones.

Registrar Usuario

Continuamos con el flujo principal del caso de uso *Registrar Usuario* como se muestra a continuación,

Flujo Principal	Se ejecuta el caso de uso *Validar Usuario*. Dependiendo de las opciones seleccionadas por el Usuario, se continuará con los diversos subflujos de este caso de uso.

Podemos observar, que el flujo principal no agrega responsabilidades.

Continuamos con el subflujo *Crear Registro Usuario* (*S*-1) del caso de uso *Registrar Usuario* como se muestra a continuación,

Subflujos	*S*-1 *Crear Registro Usuario* El ManejadorRegistroUsuario solicita *desplegarPantallaCrearRegUsuario* a la InterfaceUsuario. La InterfaceUsuario *despliega* la PantallaCrearRegUsuario. La PantallaCrearRegUsuario se *despliega*. Esta pantalla contiene información de registro que debe ser llenada por el Usuario, lo cual incluye nombre, apellido, calle, colonia, ciudad, país, código postal, teléfonos de la casa y oficina, número de fax, *login*, email, *password* y una entrada adicional de repetir *password* para asegurarse de su corrección. El *login* y la *password* serán utilizados por el sistema para validar al usuario. El Usuario puede seleccionar entre las siguientes actividades: "Registrar" y "Salir". Si el Usuario selecciona "Registrar", la PantallaCrearRegUsuario envía el evento "Registrar" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Registrar" al ManejadorRegistroUsuario. El ManejadorRegistroUsuario solicita *crearRegistroUsuario* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro solicita *crearRegistroUsuario* a la Base de Datos Registro (*E*-1, *E*-2, *E*-3, *E*-4). La Base de Datos Registro devuelve el *OK* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroUsuario. Se continúa con el subflujo *Administrar Registro Usuario* (*S*-3). Si la actividad seleccionada es "Salir", la PantallaCrearRegUsuario envía el evento "Salir" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Salir" al ManejadorRegistroUsuario. El ManejadorRegistroUsuario *sale* del sistema. (Si aún no se ha presionado "Registrar", la información será perdida).

Nuevamente, tomamos cada una de las frases y las analizamos para identificar nuevas responsabilidades.

42. **El ManejadorRegistroUsuario solicita *desplegarPantallaCrearRegUsuario* a la InterfaceUsuario**. La responsabilidad es "solicita *desplegarPantallaCrearRegUsuario* a la InterfaceUsuario" y se asigna a ManejadorRegistroUsuario.
43. **El InterfaceUsuario** despliega **la PantallaCrearRegUsuario**. La responsabilidad es "*despliega* la PantallaCrearRegUsuario" y se asigna a InterfaceUsuario.
44. **El PantallaCrearRegUsuario se *despliega***. La responsabilidad es "*despliega*" y se asigna a PantallaCrearRegUsuario.
45. **Esta pantalla contiene información de registro que debe ser llenada por el Usuario, lo cual incluye nombre, apellido, calle, colonia, ciudad, país, código postal, teléfonos de la casa y oficina, número de fax, *login*, email, *password* y una entrada adicional de repetir *password* para asegurarse de su corrección. El *login* y la *password* serán utilizados por el sistema para validar al usuario**. Estas frases son informativas y no desriben ninguna responsabilidad particular de interés en este momento.

46. **El Usuario puede seleccionar entre las siguientes actividades: "Registrar" y "Salir"**. Esta es nuevamente una frase informativa sobre las opciones del Usuario y no agrega responsabilidades.
47. **Si el Usuario selecciona "Registrar", la PantallaCrearRegUsuario envía el evento "Registrar" a la InterfaceUsuario**. Se identifica la responsabilidad "envía el evento "Registrar" a la InterfaceUsuario" y se asigna a la PantallaCrearRegUsuario.
48. **La InterfaceUsuario envía el evento "Registrar" al ManejadorRegistroUsuario**. Se identifica la responsabilidad "envía el evento "Registrar" al ManejadorRegistroUsuario" y se asigna a la InterfaceUsuario. Adicionalmente, asignamos la responsabilidad "maneja el evento "Registrar"" al ManejadorRegistroUsuario.
49. **El ManejadorRegistroUsuario solicita *crearRegistroUsuario* a la InterfaceBaseDatosRegistro**. Se identifica la responsabilidad "solicita *crearRegistroUsuario* a la InterfaceBaseDatosRegistro" y se asigna al ManejadorRegistroUsuario.
50. **La InterfaceBaseDatosRegistro solicita *crearRegistroUsuario* a la Base de Datos Registro (*E*-1, *E*-2, *E*-3, *E*-4)**. Se identifica la responsabilidad "solicita *crearRegistroUsuario* a la Base de Datos Registro" y se asigna a la InterfaceBaseDatosRegistro. Obviamente, no tiene mucho sentido asignar responsabilidades a la Base de Datos Registro dado que los actores son externos al sistema.
51. **La Base de Datos Registro devuelve el *OK* a la InterfaceBaseDatosRegistro**. Nuevamente, no se asigna ninguna responsabilidad a la Base de Datos Registro por ser externa al sistema. Por otro lado, y vale la pena resaltar, las frases de tipo "devolución de información" no resultan en la asignación de responsabilidades ya que son únicamente respuestas a solicitudes anteriores. Sólo las propias solicitudes son registradas ya que corresponden a servicios del sistema.
52. **La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroUsuario**. No se asigna responsabilidades dado que la frase describe una devolución de información.
53. **Se continúa con el subflujo *Administrar Registro Usuario* (*S*-3)**. Nuevamente, esta es una frase informativa de la continuación interna del caso de uso por lo cual no se asignan responsabilidades.
54. **Si la actividad seleccionada es "Salir", la PantallaCrearRegUsuario envía el evento "Salir" a la InterfaceUsuario**. Se asigna la responsabilidad "envía el evento "Salir" a la InterfaceUsuario" a la PantallaCrearRegUsuario.
55. **La InterfaceUsuario envía el evento "Salir" al ManejadorRegistroUsuario**. Se asigna la responsabilidad "envía el evento "Salir" al ManejadorRegistroUsuario" a la InterfaceUsuario. Adicionalmente, asignamos la responsabilidad "maneja el evento "Salir"" al ManejadorRegistroUsuario.
56. **El ManejadorRegistroUsuario *sale* del sistema**. Se asigna la responsabilidad "*sale* del sistema" al ManejadorRegistroUsuario.
57. **(Si aún no se ha presionado "Registrar", la información será perdida)**. Nuevamente, esta es una frase informativa por lo cual no se asignan nuevas responsabilidades.

A partir de estas frases obtenemos nuevas responsabilidades y las insertamos en las tarjetas de clase correspondientes. Dado que no todas las clases agregan nuevas responsabilidades, iremos mostrando sólo aquellas que sí lo hacen. En particular, las responsabilidades identificadas para las clases *ManejadorPrincipal* y *PantallaPrincipal* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.3 y en la Tabla 8.5, respectivamente. De manera similar, las responsabilidades identificadas para las clases *ManejadorServicio* y *PantallaServicio* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.10 y en la Tabla 8.11, respectivamente.

En la Tabla 8.12 se muestra las responsabilidades para la clase *InterfaceUsuario* anteriormente identificadas en la Tabla 8.9 junto con las nuevas responsabilidades.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega la PantallaPrincipal (2)	
envía el evento "Registrarse por Primera Vez" al ManejadorPrincipal (6)	
envía el evento "OK" al ManejadorPrincipal (11)	
envía el evento "Salir" al ManejadorPrincipal (21)	

despliega la PantallaServicio (24)	
envía el evento "Obtener Registro" al ManejadorServicio (36)	
envía el evento "Salir" al ManejadorPrincipal (40)	
despliega la PantallaCrearRegUsuario (43)	
envía el evento "Registrar" al ManejadorRegistroUsuario (48)	
envía el evento "Salir" al ManejadorRegistroUsuario (55)	

Tabla 8.12. Tarjeta para la clase *InterfaceUsuario* con responsabilidades identificadas hasta el momento.

En la Tabla 8.13 se muestra las responsabilidades para la clase *ManejadorRegistroUsuario* anteriormente identificadas en la Tabla 8.6 junto con las nuevas responsabilidades.

Clase: ManejadorRegistroUsuario	
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para poder utilizar el sistema.	
Módulo: Registro.Usuario	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
crearRegistroUsuario (7)	
solicita *validarRegistroUsuario* a la InterfaceBaseDatosRegistro (13)	
solicita *desplegarPantallaCrearRegUsuario* a la InterfaceUsuario (42)	
maneja el evento "Registrar" (48)	
solicita *crearRegistroUsuario* a la InterfaceBaseDatosRegistro (49)	
maneja el evento "Salir" (55)	
sale del sistema (56)	

Tabla 8.13. Tarjeta para la clase *ManejadorRegistroUsuario* con responsabilidades identificadas hasta el momento.

Agregamos una nueva tarjeta de clase describiendo las responsabilidades para la clase *PantallaCrearRegUsuario*, como se muestra en la Tabla 8.14.

Clase: PantallaCrearRegUsuario	
Descripción: Pantalla de solicitud de registro de usuario (*P*-3).	
Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega (44)	
envía el evento "Registrar" a la InterfaceUsuario (47)	
envía el evento "Salir" a la InterfaceUsuario (54)	

Tabla 8.14. Tarjeta para la clase *PantallaCrearRegUsuario* con nuevas responsabilidades identificadas hasta el momento.

En la Tabla 8.15 se muestra las responsabilidades para la clase *InterfaceBaseDatosRegistro* anteriormente identificadas en la Tabla 8.7 junto con las nuevas responsabilidades.

Clase: InterfaceBaseDatosRegistro
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.
Módulo: Registro.InterfaceBD
Estereotipo: Interface
Propiedades:
Superclases:
Subclases:

Atributos:	
solicita *validarRegistroUsuario* a la BaseDatosRegistro (14)	
solicita *crearRegistroUsuario* a la BaseDatosRegistro (50)	

Tabla 8.15. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades identificadas hasta el momento.

Continuamos con el subflujo *Obtener Registro Usuario* (*S*-2) del caso de uso *Registrar Usuario* como se muestra a continuación,

Subflujos	*S*-2 *Obtener Registro Usuario* El ManejadorRegistroUsuario solicita *obtenerRegistroUsuario* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro solicita *obtenerRegistroUsuario* a la Base de Datos Registro. La Base de Datos Registro devuelve el *OK* y el RegistroUsuario a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro devuelve el *OK* y el RegistroUsuario al ManejadorRegistroUsuario. Se continúa con el subflujo *Administrar Registro Usuario* (*S*-3).

Nuevamente, tomamos cada una de las frases y las analizamos para identificar nuevas responsabilidades.

58. **El ManejadorRegistroUsuario solicita *obtenerRegistroUsuario* a la InterfaceBaseDatosRegistro**. La responsabilidad es "solicita *obtenerRegistroUsuario* a la InterfaceBaseDatosRegistro" y se asigna a ManejadorRegistroUsuario.
59. **La InterfaceBaseDatosRegistro solicita *obtenerRegistroUsuario* a la Base de Datos Registro**. La responsabilidad es "solicita *obtenerRegistroUsuario* a la Base de Datos Registro" y se asigna a InterfaceBaseDatosRegistro.
60. **La Base de Datos Registro devuelve el *OK* y el RegistroUsuario a la InterfaceBaseDatosRegistro**. Nuevamente, eventos de devolución de información no significan responsabilidades adicionales, por lo cual esta frase no agrega ninguna.
61. **La InterfaceBaseDatosRegistro devuelve el *OK* y el RegistroUsuario al ManejadorRegistroUsuario**. Esta frase tampoco agrega ninguna responsabilidad.
62. **Se continúa con el subflujo *Administrar Registro Usuario* (*S*-3)**. Esta es una frase exclusivamente informativa de la continuación interna del caso de uso y no agrega nuevas responsabilidades.

En total se agregaron dos nuevas responsabilidades. La primera responsabilidad (58) se agrega a la clase *ManejadorRegistroUsuario* como se muestra en la Tabla 8.16 la cual incluye las responsabilidades anteriormente identificadas en la Tabla 8.13.

Clase: ManejadorRegistroUsuario	
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para poder utilizar el sistema.	
Módulo: Registro.Usuario	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
crearRegistroUsuario (7)	
solicita *validarRegistroUsuario* a la InterfaceBaseDatosRegistro (13)	
solicita *desplegarPantallaCrearRegUsuario* a la InterfaceUsuario (42)	
maneja el evento "Registrar" (48)	
solicita *crearRegistroUsuario* a la InterfaceBaseDatosRegistro (49)	
maneja el evento "Salir" (55)	
sale del sistema (56)	
solicita *obtenerRegistroUsuario* a la InterfaceBaseDatosRegistro (58)	

Tabla 8.16. Tarjeta para la clase *ManejadorRegistroUsuario* con responsabilidades identificadas hasta el momento.

La segunda responsabilidad (59) se agrega a la clase *InterfaceBaseDatosRegistro* como se muestra en la Tabla 8.17 la cual incluye las responsabilidades anteriormente identificadas en la Tabla 8.13.

Clase: InterfaceBaseDatosRegistro
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de

guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceBD	
Estereotipo: Interface	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *validarRegistroUsuario* a la BaseDatosRegistro (14)	
solicita *crearRegistroUsuario* a la BaseDatosRegistro (50)	
solicita *obtenerRegistroUsuario* a la BaseDatosRegistro (59)	

Tabla 8.14. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades identificadas hasta el momento.

Continuamos con el subflujo *Administrar Registro Usuario* (*S*-3) del caso de uso *Registrar Usuario* como se muestra a continuación,

Subflujos	*S*-3 *Administrar Registro Usuario* El ManejadorRegistroUsuario solicita *desplegarPantallaObtenerRegUsuario* a la InterfaceUsuario. La InterfaceUsuario *despliega* la PantallaObtenerRegUsuario. La PantallaObtenerRegUsuario se *despliega*. El Usuario puede seleccionar entra las siguientes actividades: "Eliminar", "Actualizar", "Registrar Tarjeta", "Servicios" y "Salir". Si el usuario presiona "Actualizar" se ejecuta el subflujo *Actualizar Registro Usuario* (*S*-4). Si el usuario selecciona "Eliminar" se ejecuta el subflujo *Eliminar Registro Usuario* (*S*-5). Si el usuario presiona "Registrar Tarjeta", la PantallaObtenerRegUsuario envía el evento "Registrar Tarjeta" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Registrar Tarjeta" al ManejadorRegistroUsuario. El ManejadorRegistroUsuario solicita *registrarTarjeta* al ManejadorRegistroTarjeta, se continúa con el caso de uso *Registrar Tarjeta*. Si la actividad seleccionada es "Servicios", la PantallaObtenerRegUsuario envía el evento "Servicios" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Servicios" al ManejadorRegistroUsuario. El ManejadorRegistroUsuario solicita *ofrecerServicio* al ManejadorServicio, se continúa con el caso de uso *Ofrecer Servicios*. Si la actividad seleccionada es "Salir", la PantallaObtenerRegUsuario envía el evento "Salir" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Salir" al ManejadorRegistroUsuario. El ManejadorRegistroUsuario *sale* del sistema. (Si aún no se ha presionado "Actualizar", la nueva información será perdida).

Nuevamente, tomamos cada una de las frase y las analizamos para identificar nuevas responsabilidades.

63. **El ManejadorRegistroUsuario solicita *desplegarPantallaObtenerRegUsuario* a la InterfaceUsuario**. La responsabilidad es "solicita *desplegarPantallaObtenerRegUsuario* a la InterfaceUsuario" y se asigna a la ManejadorRegistroUsuario.
64. **La InterfaceUsuario *despliega* la PantallaObtenerRegUsuario**. La responsabilidad es "*despliega* la PantallaObtenerRegUsuario" y se asigna a la InterfaceUsuario.
65. **La PantallaObtenerRegUsuario se *despliega***. La responsabilidad es "*despliega*" y se asigna a la PantallaObtenerRegUsuario.
66. **El Usuario puede seleccionar entra las siguientes actividades: "Eliminar", "Actualizar", "Registrar Tarjeta", "Servicios" y "Salir"**. Esta frase es informativa describiendo las diversas opciones del Usuario y no agrega responsabilidades.
67. **Si el usuario presiona "Actualizar" se ejecuta el subflujo *Actualizar Registro Usuario* (*S*-4)**. Esta frase es informativa describiendo una continuación interna del flujo del caso de uso y no agrega responsabilidades.
68. **Si el usuario selecciona "Eliminar" se ejecuta el subflujo *Eliminar Registro Usuario* (*S*-5)**. De manera similar a la frase anterior, esta frase es informativa describiendo una continuación interna del flujo del caso de uso y no agrega responsabilidades.
69. **Si el usuario presiona "Registrar Tarjeta", la PantallaObtenerRegUsuario envía el evento "Registrar Tarjeta" a la InterfaceUsuario**. La responsabilidad es "envía el evento "Registrar Tarjeta" a la InterfaceUsuario" y se asigna a la PantallaObtenerRegUsuario.
70. **La InterfaceUsuario envía el evento "Registrar Tarjeta" al ManejadorRegistroUsuario**. La responsabilidad es "envía el evento "Registrar Tarjeta" al ManejadorRegistroUsuario" y se asigna a la

InterfaceUsuario. De manera adicional, se asigna la nueva responsabilidad "maneja el evento "Registrar Tarjeta"" al ManejadorRegistroUsuario.

71. **El ManejadorRegistroUsuario solicita *registrarTarjeta* al ManejadorRegistroTarjeta, se continúa con el caso de uso *Registrar Tarjeta***. La responsabilidad es "solicita *registrarTarjeta* al ManejadorRegistroTarjeta" y se asigna al ManejadorRegistroUsuario. Adicionalmente, asignamos la responsabilidad "*registrarTarjeta*" al ManejadorRegistroTarjeta. La última sección de la frase describe una continuación de lógica entre casos de uso y no agrega responsabilidades.
72. **Si la actividad seleccionada es "Servicios", la PantallaObtenerRegUsuario envía el evento "Servicios" a la InterfaceUsuario**. La responsabilidad es "envía el evento "Servicios" a la InterfaceUsuario" y se asigna a PantallaObtenerRegUsuario.
73. **La InterfaceUsuario envía el evento "Servicios" al ManejadorRegistroUsuario**. La responsabilidad es "envía el evento "Servicios" al ManejadorRegistroUsuario" y se asigna a InterfaceUsuario. Adicionalmente se agrega la responsabilidad "maneja el evento "Servicios"" al ManejadorRegistroUsuario.
74. **El ManejadorRegistroUsuario solicita *ofrecerServicio* al ManejadorServicio, se continúa con el caso de uso *Ofrecer Servicios***. La responsabilidad es "solicita *ofrecerServicio* al ManejadorServicio" y se asigna al ManejadorRegistroUsuario. No es necesario volver a asignar la responsabilidad "*ofrecerServicio*" a ManejadorServicio, ya que esto ha sido hecho anteriormente. La última sección de la frase describe continuación del flujo interno del caso de uso y no agrega nuevas responsabilidades.
75. **Si la actividad seleccionada es "Salir", la PantallaObtenerRegUsuario envía el evento "Salir" a la InterfaceUsuario**. La responsabilidad es "envía el evento "Salir" a la InterfaceUsuario" y se asigna a PantallaObtenerRegUsuario.
76. **La InterfaceUsuario envía el evento "Salir" al ManejadorRegistroUsuario**. La responsabilidad es "envía el evento "Salir" al ManejadorRegistroUsuario" y se asigna a InterfaceUsuario. Adicionalmente se debe asignar la responsabilidad "maneja el evento "Salir"" a ManejadorRegistroUsuario. Sin embargo, esta responsabilidad ya ha sido asignada anteriormente, por lo cual no es necesario duplicarla.
77. **El ManejadorRegistroUsuario *sale* del sistema**. La responsabilidad es "*sale* del sistema" y se debe asignar a ManejadorRegistroUsuario. Sin embargo, esta es una responsabilidad duplicada por lo cual no la volvemos a asignar.
78. (**Si aún no se ha presionado "Actualizar", la nueva información será perdida)**. Esta frase es informativa y no agrega responsabilidades adicionales.

A partir de estas frases obtenemos responsabilidades adicionales y las insertamos en las tarjetas de clase correspondientes. Nuevamente, no todas las clases agregan nuevas responsabilidades. En particular, las responsabilidades identificadas para las clases *ManejadorPrincipal* y *PantallaPrincipal* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.3 y en la Tabla 8.5, respectivamente. De manera similar, las responsabilidades identificadas para las clases *ManejadorServicio* y *PantallaServicio* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.10 y en la Tabla 8.11, respectivamente. Adicionalmente, se mantienen iguales las clases *PantallaCrearRegUsuario* correspondiente a la Tabla 8.14, junto con la *InterfaceBaseDatosRegistro* correspondiente a la Tabla 8.17.

En la Tabla 8.18 se muestra las responsabilidades para la clase *InterfaceUsuario* anteriormente identificadas en la Tabla 8.12 junto con sus nuevas responsabilidades.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega la PantallaPrincipal (2)	
envía el evento "Registrarse por Primera Vez" al ManejadorPrincipal (6)	
envía el evento "OK" al ManejadorPrincipal (11)	
envía el evento "Salir" al ManejadorPrincipal (21)	
despliega la PantallaServicio (24)	
envía el evento "Obtener Registro" al ManejadorServicio (36)	
envía el evento "Salir" al ManejadorPrincipal (40)	

despliega la PantallaCrearRegUsuario (43)	
envía el evento "Registrar" al ManejadorRegistroUsuario (48)	
envía el evento "Salir" al ManejadorRegistroUsuario (55)	
despliega la PantallaObtenerRegistroUsuario (64)	
envía el evento "Registrar Tarjeta" al ManejadorRegistroUsuario (70)	
envía el evento "Servicios" al ManejadorRegistroUsuario (73)	

Tabla 8.18. Tarjeta para la clase *InterfaceUsuario* con responsabilidades identificadas hasta el momento.

En la Tabla 8.19 se muestra las responsabilidades para la clase *ManejadorRegistroUsuario* anteriormente identificadas en la Tabla 8.13 junto con las nuevas responsabilidades.

Clase: ManejadorRegistroUsuario	
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para poder utilizar el sistema.	
Módulo: Registro.Usuario	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
crearRegistroUsuario (7)	
solicita *validarRegistroUsuario* a la InterfaceBaseDatosRegistro (13)	
solicita *desplegarPantallaCrearRegUsuario* a la InterfaceUsuario (42)	
maneja el evento "Registrar" (48)	
solicita *crearRegistroUsuario* a la InterfaceBaseDatosRegistro (49)	
maneja el evento "Salir" (55)	
sale del sistema (56)	
solicita *obtenerRegistroUsuario* a la InterfaceBaseDatosRegistro (58)	
solicita *desplegarPantallaObtenerRegistroUsuario* a la InterfaceUsuario (63)	
maneja el evento "Registrar Tarjeta" (70)	
solicita *registrarTarjeta* al ManejadorRegistroTarjeta (71)	
maneja el evento "Servicios" (73)	
solicita *ofrecerServicios* al ManejadorServicio (74)	

Tabla 8.19. Tarjeta para la clase *ManejadorRegistroUsuario* con responsabilidades identificadas hasta el momento.

Agregamos una nueva tarjeta de clase describiendo las responsabilidades para la clase *PantallaObtenerRegUsuario*, como se muestra en la Tabla 8.20.

Clase: PantallaObtenerRegUsuario	
Descripción: Pantalla de devolución con información de registro de usuario (*P*-4).	
Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega (65)	
envía el evento "Registrar Tarjeta" a la InterfaceUsuario (69)	
envía el evento "Servicios" a la InterfaceUsuario (72)	
envía el evento "Salir" a la InterfaceUsuario (75)	

Tabla 8.20. Tarjeta para la clase *PantallaObtenerRegUsuario* con nuevas responsabilidades identificadas hasta el momento.

La Tabla 8.21 muestra las responsabilidades identificadas hasta el momento para la clase *ManejadorRegistroTarjeta*.

Clase: ManejadorRegistroTarjeta

Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.	
Módulo: Registro.Tarjeta	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
registrarTarjeta (71)	

Tabla 8.21. Tarjeta para la clase *ManejadorRegistroTarjeta* con responsabilidades identificadas hasta el momento.

Continuamos con el subflujo *Actualizar Registro Usuario* (*S*-4) del caso de uso *Registrar Usuario* como se muestra a continuación,

Subflujos	*S*-4 *Actualizar Registro Usuario* La PantallaObtenerRegUsuario envía el evento "Actualizar" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Actualizar" al ManejadorRegistroUsuario. El ManejadorRegistroUsuario solicita *actualizarRegistroUsuario* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro solicita *actualizarRegistroUsuario* a la Base de Datos Registro. La Base de Datos Registro actualiza el RegistroUsuario (*E*-1, *E*-2, *E*-4) y devuelve el *OK* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroUsuario. Se continua con el subflujo *Administrar Registro Usuario* (*S*-3).

Nuevamente, tomamos cada una de las frases y las analizamos para identificar nuevas responsabilidades.

79. **La PantallaObtenerRegUsuario envía el evento "Actualizar" a la InterfaceUsuario**. La responsabilidad es "envía el evento "Actualizar" a la InterfaceUsuario" y se asigna a PantallaObtenerRegUsuario.
80. **La InterfaceUsuario envía el evento "Actualizar" al ManejadorRegistroUsuario**. La responsabilidad es "envía el evento "Actualizar" al ManejadorRegistroUsuario" y se asigna a InterfaceUsuario. Adicionalmente, se agrega la responsabilidad "maneja el evento "Actualizar"" y se asigna al ManejadorRegistroUsuario.
81. **El ManejadorRegistroUsuario solicita *actualizarRegistroUsuario* a la InterfaceBaseDatosRegistro**. La responsabilidad es "solicita *actualizarRegistroUsuario* a la InterfaceBaseDatosRegistro" y se asigna al ManejadorRegistroUsuario.
82. **La InterfaceBaseDatosRegistro solicita *actualizarRegistroUsuario* a la Base de Datos Registro**. La responsabilidad es "solicita *actualizarRegistroUsuario* a la Base de Datos Registro" y se asigna a la InterfaceBaseDatosRegistro.
83. **La Base de Datos Registro actualiza el RegistroUsuario (*E*-1, *E*-2, *E*-4) y devuelve el *OK* a la InterfaceBaseDatosRegistro**. Esta frase se refiere a Base de Datos Registro, un actor externo al sistema, por lo cual no se agregan nuevas responsabilidades. La segunda parte de la frase describe una devolución por lo cual tampoco se agrega ninguna responsabilidad.
84. **La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroUsuario**. Nuevamente, se describe una devolución por lo cual no se agrega ninguna responsabilidad.
85. **Se continua con el subflujo *Administrar Registro Usuario* (*S*-3)**. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.

A partir de estas frases obtenemos nuevas responsabilidades y las insertamos en las tarjetas de clase correspondientes. Dado que no todas las clases agregan nuevas responsabilidades, iremos mostrando sólo aquellas que sí lo hacen. En particular, las responsabilidades identificadas para las clases *ManejadorPrincipal* y *PantallaPrincipal* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.3 y en la Tabla 8.5, respectivamente. De manera similar, las responsabilidades identificadas para las clases *ManejadorServicio* y *PantallaServicio* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.10 y en la Tabla 8.11, respectivamente. Adicionalmente, se mantiene igual la clase *PantallaCrearRegUsuario* correspondiente a la Tabla 8.14.

En la Tabla 8.22 se muestra las responsabilidades para la clase *InterfaceUsuario* anteriormente identificadas en la Tabla 8.18 junto con sus nuevas responsabilidades.

Clase: InterfaceUsuario
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.
Módulo: InterfaceUsuario

Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega la PantallaPrincipal (2)	
envía el evento "Registrarse por Primera Vez" al ManejadorPrincipal (6)	
envía el evento "OK" al ManejadorPrincipal (11)	
envía el evento "Salir" al ManejadorPrincipal (21)	
despliega la PantallaServicio (24)	
envía el evento "Obtener Registro" al ManejadorServicio (36)	
envía el evento "Salir" al ManejadorPrincipal (40)	
despliega la PantallaCrearRegUsuario (43)	
envía el evento "Registrar" al ManejadorRegistroUsuario (48)	
envía el evento "Salir" al ManejadorRegistroUsuario (55)	
despliega la PantallaObtenerRegistroUsuario (64)	
envía el evento "Registrar Tarjeta" al ManejadorRegistroUsuario (70)	
envía el evento "Servicios" al ManejadorRegistroUsuario (73)	
envía el evento "Actualizar" al ManejadorRegistroUsuario (80)	

Tabla 8.22. Tarjeta para la clase *InterfaceUsuario* con responsabilidades identificadas hasta el momento.

En la Tabla 8.23 se muestra las responsabilidades para la clase *ManejadorRegistroUsuario* anteriormente identificadas en la Tabla 8.19 junto con las nuevas responsabilidades.

Clase: ManejadorRegistroUsuario	
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para poder utilizar el sistema.	
Módulo: Registro.Usuario	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
crearRegistroUsuario (7)	
solicita *validarRegistroUsuario* a la InterfaceBaseDatosRegistro (13)	
solicita *desplegarPantallaCrearRegUsuario* a la InterfaceUsuario (42)	
maneja el evento "Registrar" (48)	
solicita *crearRegistroUsuario* a la InterfaceBaseDatosRegistro (49)	
maneja el evento "Salir" (55)	
sale del sistema (56)	
solicita *obtenerRegistroUsuario* a la InterfaceBaseDatosRegistro (58)	
solicita *desplegarPantallaObtenerRegistroUsuario* a la InterfaceUsuario (63)	
maneja el evento "Registrar Tarjeta" (70)	
solicita *registrarTarjeta* al ManejadorRegistroTarjeta (71)	
maneja el evento "Servicios" (73)	
solicita *ofrecerServicios* al ManejadorServicio (74)	
maneja el evento "Actualizar" (80)	
solicita *actualizarRegistroUsuario* a la InterfaceBaseDatosRegistro (81)	

Tabla 8.23. Tarjeta para la clase *ManejadorRegistroUsuario* con responsabilidades identificadas hasta el momento.

En la Tabla 8.24 se muestra las responsabilidades para la clase *PantallaObtenerRegUsuario* anteriormente identificadas en la Tabla 8.20 junto con las nuevas responsabilidades.

Clase: PantallaObtenerRegUsuario
Descripción: Pantalla de devolución con información de registro de usuario (*P*-4).

Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega (65)	
envía el evento "Registrar Tarjeta" a la InterfaceUsuario (69)	
envía el evento "Servicios" a la InterfaceUsuario (72)	
envía el evento "Salir" a la InterfaceUsuario (75)	
envía el evento "Actualizar" a la InterfaceUsuario (79)	

Tabla 8.24. Tarjeta para la clase *PantallaObtenerRegUsuario* con nuevas responsabilidades identificadas hasta el momento.

En la Tabla 8.25 se muestra las responsabilidades para la clase *InterfaceBaseDatosRegistro* anteriormente identificadas en la Tabla 8.14 junto con las nuevas responsabilidades.

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceBD	
Estereotipo: Interface	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *validarRegistroUsuario* a la BaseDatosRegistro (14)	
solicita *crearRegistroUsuario* a la BaseDatosRegistro (50)	
solicita *obtenerRegistroUsuario* a la BaseDatosRegistro (59)	
solicita *actualizarRegistroUsuario* a la BaseDatosRegistro (82)	

Tabla 8.25. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades identificadas hasta el momento.

Continuamos con el subflujo *Eliminar Registro Usuario* (*S*-5) del caso de uso *Registrar Usuario* como se muestra a continuación,

Subflujos	*S*-5 *Eliminar Registro Usuario* La PantallaObtenerRegUsuario envía el evento "Eliminar" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Eliminar" al ManejadorRegistroUsuario. El ManejadorRegistroUsuario solicita *eliminarRegistroUsuario* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro envía el evento *eliminarRegistroUsuario* a la Base de Datos Registro. La Base de Datos Registro elimina el RegistroUsuario y devuelve el *OK* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroUsuario. Se continúa con el subflujo *Crear Registro Usuario* (*S*-1).

Nuevamente, tomamos cada una de las frases y las analizamos para identificar nuevas responsabilidades.

86. **La PantallaObtenerRegUsuario envía el evento "Eliminar" a la InterfaceUsuario**. La responsabilidad es "envía el evento "Eliminar" a la InterfaceUsuario" y se asigna a PantallaObtenerRegUsuario.
87. **La InterfaceUsuario envía el evento "Eliminar" al ManejadorRegistroUsuario**. La responsabilidad es "envía el evento "Eliminar" al ManejadorRegistroUsuario" y se asigna a InterfaceUsuario. Adicionalmente, se agrega la responsabilidad "maneja el evento "Eliminar"" y se asigna al ManejadorRegistroUsuario.
88. **El ManejadorRegistroUsuario solicita *eliminarRegistroUsuario* a la InterfaceBaseDatosRegistro**. La responsabilidad es "solicita *eliminarRegistroUsuario* a la InterfaceBaseDatosRegistro" y se asigna al ManejadorRegistroUsuario.
89. **La InterfaceBaseDatosRegistro envía el evento *eliminarRegistroUsuario* a la Base de Datos Registro**. La responsabilidad es "solicita *eliminarRegistroUsuario* a la Base de Datos Registro" y se asigna a la InterfaceBaseDatosRegistro.

90. **La Base de Datos Registro elimina el RegistroUsuario y devuelve el *OK* a la InterfaceBaseDatosRegistro**. Esta frase se refiere a Base de Datos Registro, un actor externo al sistema, por lo cual no se agregan nuevas responsabilidades. La segunda parte de la frase describe una devolución por lo cual tampoco se agrega ninguna responsabilidad.
91. **La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroUsuario**. Nuevamente, se describe una devolución por lo cual no se agrega ninguna responsabilidad.
92. **Se continúa con el subflujo *Crear Registro Usuario* (*S*-1)**. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.

A partir de estas frases obtenemos nuevas responsabilidades y las insertamos en las tarjetas de clase correspondientes. Dado que no todas las clases agregan nuevas responsabilidades, iremos mostrando sólo aquellas que sí lo hacen. En particular, las responsabilidades identificadas para las clases *ManejadorPrincipal* y *PantallaPrincipal* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.3 y en la Tabla 8.5, respectivamente. De manera similar, las responsabilidades identificadas para las clases *ManejadorServicio* y *PantallaServicio* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.10 y en la Tabla 8.11, respectivamente. Adicionalmente, se mantiene igual la clase *PantallaCrearRegUsuario* correspondiente a la Tabla 8.14.

En la Tabla 8.26 se muestra las responsabilidades para la clase *InterfaceUsuario* anteriormente identificadas en la Tabla 8.22 junto con sus nuevas responsabilidades.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega la PantallaPrincipal (2)	
envía el evento "Registrarse por Primera Vez" al ManejadorPrincipal (6)	
envía el evento "OK" al ManejadorPrincipal (11)	
envía el evento "Salir" al ManejadorPrincipal (21)	
despliega la PantallaServicio (24)	
envía el evento "Obtener Registro" al ManejadorServicio (36)	
envía el evento "Salir" al ManejadorPrincipal (40)	
despliega la PantallaCrearRegUsuario (43)	
envía el evento "Registrar" al ManejadorRegistroUsuario (48)	
envía el evento "Salir" al ManejadorRegistroUsuario (55)	
despliega la PantallaObtenerRegistroUsuario (64)	
envía el evento "Registrar Tarjeta" al ManejadorRegistroUsuario (70)	
envía el evento "Servicios" al ManejadorRegistroUsuario (73)	
envía el evento "Actualizar" al ManejadorRegistroUsuario (80)	
envía el evento "Eliminar" al ManejadorRegistroUsuario (87)	

Tabla 8.26. Tarjeta para la clase *InterfaceUsuario* con responsabilidades identificadas hasta el momento.

En la Tabla 8.27 se muestra las responsabilidades para la clase *ManejadorRegistroUsuario* anteriormente identificadas en la Tabla 8.23 junto con las nuevas responsabilidades.

Clase: ManejadorRegistroUsuario
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para poder utilizar el sistema.
Módulo: Registro.Usuario
Estereotipo: Control
Propiedades:
Superclases:
Subclases:
Atributos:

crearRegistroUsuario (7)	
solicita *validarRegistroUsuario* a la InterfaceBaseDatosRegistro (13)	
solicita *desplegarPantallaCrearRegUsuario* a la InterfaceUsuario (42)	
maneja el evento "Registrar" (48)	
solicita *crearRegistroUsuario* a la InterfaceBaseDatosRegistro (49)	
maneja el evento "Salir" (55)	
sale del sistema (56)	
solicita *obtenerRegistroUsuario* a la InterfaceBaseDatosRegistro (58)	
solicita *desplegarPantallaObtenerRegistroUsuario* a la InterfaceUsuario (63)	
maneja el evento "Registrar Tarjeta" (70)	
solicita *registrarTarjeta* al ManejadorRegistroTarjeta (71)	
maneja el evento "Servicios" (73)	
solicita *ofrecerServicios* al ManejadorServicio (74)	
maneja el evento "Actualizar" (80)	
solicita *actualizarRegistroUsuario* a la InterfaceBaseDatosRegistro (81)	
maneja el evento "Eliminar" (87)	
solicita *eliminarRegistroUsuario* a la InterfaceBaseDatosRegistro (88)	

Tabla 8.27. Tarjeta para la clase *ManejadorRegistroUsuario* con responsabilidades identificadas hasta el momento.

En la Tabla 8.28 se muestra las responsabilidades para la clase *PantallaObtenerRegUsuario* anteriormente identificadas en la Tabla 8.24 junto con las nuevas responsabilidades.

Clase: PantallaObtenerRegUsuario	
Descripción: Pantalla de devolución con información de registro de usuario (*P*-4).	
Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega (65)	
envía el evento "Registrar Tarjeta" a la InterfaceUsuario (69)	
envía el evento "Servicios" a la InterfaceUsuario (72)	
envía el evento "Salir" a la InterfaceUsuario (75)	
envía el evento "Actualizar" a la InterfaceUsuario (79)	
envía el evento "Eliminar" a la InterfaceUsuario (86)	

Tabla 8.28. Tarjeta para la clase *PantallaObtenerRegUsuario* con nuevas responsabilidades identificadas hasta el momento.

En la Tabla 8.29 se muestra las responsabilidades para la clase *InterfaceBaseDatosRegistro* anteriormente identificadas en la Tabla 8.25 junto con las nuevas responsabilidades.

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceBD	
Estereotipo: Interface	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *validarRegistroUsuario* a la BaseDatosRegistro (14)	
solicita *crearRegistroUsuario* a la BaseDatosRegistro (50)	
solicita *obtenerRegistroUsuario* a la BaseDatosRegistro (59)	

solicita *actualizarRegistroUsuario* a la BaseDatosRegistro (82)	
solicita *eliminarRegistroUsuario* a la BaseDatosRegistro (89)	

Tabla 8.29. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades identificadas hasta el momento.

De manera general, se puede también obtener responsabilidades a partir del manejajo de excepciones, como se muestra a continuación,

Excepciones	*E-1 información incompleta*: Falta llenar información en el registro de usuario. Se le vuelve a pedir al usuario que complete el registro. *E-2 registro ya existe*: Si ya existe un registro bajo ese *login*, se le pedirá al usuario que lo cambie o que termine el caso de uso. *E-3 login incorrecto*: El *login* no es válido. Se le vuelve a pedir al usuario que complete el registro. *E-4 contraseña incorrecta*: La contraseña escogida es muy sencilla o no se validó correctamente. Se le vuelve a pedir al usuario que complete el registro.

Sin embargo, nos concentraremos únicamente en los flujos básicos y no los alternos. En general, los alternos que son los menos comunes, son importantes de diseñar pero no en una primera etapa.

Registrar Tarjeta

A continuación mostramos el flujo principal del caso de uso *Registrar Tarjeta* como se presentó al final del capítulo de análisis.

Flujo Principal	Se continúa con el subflujo *Obtener Registro Tarjeta* (*S*-2). Si no existe un RegistroTarjeta válido se continúa con el subflujo *Crear Registro Tarjeta* (*S*-1). De lo contrario, si ya existe un RegistroTarjeta válido, se continúa con el subflujo *Administrar Registro Tarjeta* (*S*-3).

Tomamos cada una de las frases que describen el flujo y las analizaremos para decidir que responsabilidad representan y a que clase se le asignan:

93. **Se continúa con el subflujo *Obtener Registro Tarjeta* (*S*-2)**. Esta frase describe flujo interno del caso de uso y no agrega responsabilidades.
94. **Si no existe un RegistroTarjeta válido se continúa con el subflujo *Crear Registro Tarjeta* (*S*-1)**. Esta frase es informativa y no agrega responsabilidades a las clases.
95. **De lo contrario, si ya existe un RegistroTarjeta válido, se continúa con el subflujo *Administrar Registro Tarjeta* (*S*-3)**. Nuevamente, esta frase es informativa y no agrega responsabilidades a las clases.

Como podemos apreciar, el flujo anterior no agrega responsabilidades adicionales.

Continuamos con el subflujo *Crear Registro Tarjeta* (*S*-1) del caso de uso *Registrar Tarjeta* como se muestra a continuación,

Subflujos	*S*-1 *Crear Registro Tarjeta* El ManejadorRegistroTarjeta solicita *desplegarPantallaCrearRegTarjeta* a la InterfaceUsuario. La InterfaceUsuario *despliega* a la PantallaCrearRegTarjeta. La PantallaCrearRegTarjeta se *despliega*. Esta pantalla contiene información que debe ser llenada por el Usuario, lo cual incluye el nombre como aparece el la tarjeta, número de tarjeta, el tipo de tarjeta, y la fecha de vencimiento. El Usuario puede seleccionar entre las siguientes actividades: "Registrar", "Servicios" y "Salir". Si el Usuario selecciona “Registrar”, la PantallaCrearRegTarjeta envía el evento “Registrar” a la InterfaceUsuario. La InterfaceUsuario envía el evento “Registrar” al ManejadorRegistroTarjeta. El ManejadorRegistroTarjeta solicita *crearRegistroTarjeta* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro solicita *crearRegistroTarjeta* a la Base de Datos Registro (*E*-1). La Base de Datos Registro devuelve el *OK* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroTarjeta. Se continúa con el subflujo *Administrar Registro Tarjeta* (*S*-3). Si la actividad seleccionada es "Servicios", la PantallaCrearRegTarjeta envía el evento “Servicios” a la InterfaceUsuario. La InterfaceUsuario envía el evento “Servicios” al ManejadorRegistroTarjeta. El ManejadorRegistroTarjeta solicita *ofrecerServicio* al ManejadorServicio, se continúa con el caso de uso *Ofrecer Servicios*. Si la actividad seleccionada es "Salir", la PantallaCrearRegTarjeta envía el evento “Salir” a la InterfaceUsuario. La InterfaceUsuario envía el evento “Salir” al ManejadorRegistroTarjeta. El ManejadorRegistroTarjeta *sale* del sistema. (Si aún no se ha presionado "Registrar", la información ésta será perdida).

Nuevamente, tomamos cada una de las frases y las analizamos para identificar nuevas responsabilidades.

96. **El ManejadorRegistroTarjeta solicita *desplegarPantallaCrearRegTarjeta* a la InterfaceUsuario**. La responsabilidad es "solicita *desplegarPantallaCrearRegTarjeta* a la InterfaceUsuario" y se asigna a ManejadorRegistroTarjeta.
97. **La InterfaceUsuario *despliega* a la PantallaCrearRegTarjeta**. La responsabilidad es "*despliega* la PantallaCrearRegTarjeta" y se asigna a InterfaceUsuario.
98. **La PantallaCrearRegTarjeta se *despliega***. La responsabilidad es "*despliega*" y se asigna a PantallaCrearRegTarjeta.
99. **Esta pantalla contiene información que debe ser llenada por el Usuario, lo cual incluye el nombre como aparece el la tarjeta, número de tarjeta, el tipo de tarjeta, y la fecha de vencimiento**. Esta es una frase totalmente informativa por lo cual no agrega responsabilidades.
100. **El Usuario puede seleccionar entre las siguientes actividades: "Registrar", "Servicios" y "Salir"**. Esta es nuevamente una frase informativa sobre las opciones del Usuario y no agrega responsabilidades.
101. **Si el Usuario selecciona "Registrar", la PantallaCrearRegTarjeta envía el evento "Registrar" a la InterfaceUsuario**. Se identifica la responsabilidad "envía el evento "Registrar" a la InterfaceUsuario" y se asigna a la PantallaCrearRegTarjeta.
102. **La InterfaceUsuario envía el evento "Registrar" al ManejadorRegistroTarjeta**. Se identifica la responsabilidad "envía el evento "Registrar" al ManejadorRegistroTarjeta" y se asigna a la InterfaceUsuario. Adicionalmente, asignamos la responsabilidad "maneja el evento "Registrar"" al ManejadorRegistroTarjeta.
103. **El ManejadorRegistroTarjeta solicita *crearRegistroTarjeta* a la InterfaceBaseDatosRegistro**. Se identifica la responsabilidad "solicita *crearRegistroTarjeta* a la InterfaceBaseDatosRegistro" y se asigna al ManejadorRegistroTarjeta.
104. **La InterfaceBaseDatosRegistro solicita *crearRegistroTarjeta* a la Base de Datos Registro (*E*-1)**. Se identifica la responsabilidad "solicita *crearRegistroTarjeta* a la Base de Datos Registro" y se asigna a la InterfaceBaseDatosRegistro. Nuevamente, no tiene mucho sentido asignar responsabilidades a la Base de Datos Registro dado que los actores son externos al sistema.
105. **La Base de Datos Registro devuelve el *OK* a la InterfaceBaseDatosRegistro**. No se asigna ninguna responsabilidad a la Base de Datos Registro por ser externa al sistema. Tampoco se agregan responsabilidades por razones de devolución de información.
106. **La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroTarjeta**. No se asigna responsabilidades dado que la frase describe una devolución de información.
107. **Se continúa con el subflujo *Administrar Registro Tarjeta* (*S*-3)**. Esta es una frase informativa de la continuación interna del caso de uso por lo cual no se asignan responsabilidades.
108. **Si la actividad seleccionada es "Servicios", la PantallaCrearRegTarjeta envía el evento "Servicios" a la InterfaceUsuario**. Se asigna la responsabilidad "envía el evento "Servicios" a la InterfaceUsuario" a la PantallaCrearRegTarjeta.
109. **La InterfaceUsuario envía el evento "Servicios" al ManejadorRegistroTarjeta**. Se asigna la responsabilidad "envía el evento "Servicios"" a la InterfaceUsuario. De manera adicional se asigna la rsponsabilidad "maneja el evento "Servicios"" al ManejadorRegistroTarjeta.
110. **El ManejadorRegistroTarjeta solicita *ofrecerServicio* al ManejadorServicio, se continúa con el caso de uso *Ofrecer Servicios***. Se identifica la responsabilidad "solicita *ofrecerServicio* al ManejadorServicio" y se asigna al ManejadorRegistroTarjeta. La última sección no agrega responsabilidades por describir un flujo entre casos de uso.
111. **Si la actividad seleccionada es "Salir", la PantallaCrearRegTarjeta envía el evento "Salir" a la InterfaceUsuario**. Se asigna la responsabilidad "envía el evento "Salir" a la InterfaceUsuario" a la PantallaCrearRegTarjeta.
112. **La InterfaceUsuario envía el evento "Salir" al ManejadorRegistroTarjeta**. Se asigna la responsabilidad "envía el evento "Salir" al ManejadorRegistroTarjeta" a la InterfaceUsuario. Adicionalmente, asignamos la responsabilidad "maneja el evento "Salir"" al ManejadorRegistroTarjeta.
113. **El ManejadorRegistroTarjeta *sale* del sistema**. Se asigna la responsabilidad "*sale* del sistema" al ManejadorRegistroTarjeta.
114. **(Si aún no se ha presionado "Registrar", la información ésta será perdida)**. Nuevamente, esta es una frase informativa por lo cual no se asignan nuevas responsabilidades.

A partir de estas frases obtenemos nuevas responsabilidades y las insertamos en las tarjetas de clase correspondientes. Dado que no todas las clases agregan nuevas responsabilidades, iremos mostrando sólo aquellas que sí lo hacen. En particular, las responsabilidades identificadas para las clases *ManejadorPrincipal* y

PantallaPrincipal no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.3 y en la Tabla 8.5, respectivamente. Las responsabilidades identificadas para las clases *ManejadorServicio* y *PantallaServicio* tampoco han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.10 y en la Tabla 8.11, respectivamente. Adicionalmente, las responsabilidades identificadas para las clases *ManejadorRegistroUsuario*, *PantallaCrearRegUsuario* y *PantallaObtenerRegUsuario* tampoco han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.27, Tabla 8.14 y Tabla 8.28, respectivamente.
En la Tabla 8.30 se muestra las responsabilidades para la clase *InterfaceUsuario* anteriormente identificadas en la Tabla 8.26 junto con sus nuevas responsabilidades.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega la PantallaPrincipal (2)	
envía el evento "Registrarse por Primera Vez" al ManejadorPrincipal (6)	
envía el evento "OK" al ManejadorPrincipal (11)	
envía el evento "Salir" al ManejadorPrincipal (21)	
despliega la PantallaServicio (24)	
envía el evento "Obtener Registro" al ManejadorServicio (36)	
envía el evento "Salir" al ManejadorPrincipal (40)	
despliega la PantallaCrearRegUsuario (43)	
envía el evento "Registrar" al ManejadorRegistroUsuario (48)	
envía el evento "Salir" al ManejadorRegistroUsuario (55)	
despliega la PantallaObtenerRegistroUsuario (64)	
envía el evento "Registrar Tarjeta" al ManejadorRegistroUsuario (70)	
envía el evento "Servicios" al ManejadorRegistroUsuario (73)	
envía el evento "Actualizar" al ManejadorRegistroUsuario (80)	
envía el evento "Eliminar" al ManejadorRegistroUsuario (87)	
despliega la PantallaCrearRegTarjeta (97)	
envía el evento "Registrar" al ManejadorRegistroTarjeta (102)	
envía el evento "Servicios" al ManejadorRegistroTarjeta (109)	
envía el evento "Salir" al ManejadorRegistroTarjeta (112)	

Tabla 8.30. Tarjeta para la clase *InterfaceUsuario* con responsabilidades identificadas hasta el momento.

En la Tabla 8.31 se muestra las responsabilidades para la clase *ManejadorRegistroTarjeta* anteriormente identificadas en la Tabla 8.21 junto con las nuevas responsabilidades.

Clase: ManejadorRegistroTarjeta	
Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.	
Módulo: Registro.Tarjeta	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
registrarTarjeta (71)	
solicita *desplegarPantallaCrearRegTarjeta* a la InterfaceUsuario (96)	
maneja el evento "Registrar" (102)	
solicita *crearRegistroTarjeta* a la InterfaceBaseDatosRegistro (103)	
maneja el evento "Servicios" (109)	
solicita *ofrecerServicio* al ManejadorServicio (110)	

maneja el evento “Salir” (112)	
sale del sistema (113)	

Tabla 8.31. Tarjeta para la clase *ManejadorRegistroTarjeta* con responsabilidades identificadas hasta el momento.

Agregamos una nueva tarjeta de clase describiendo las responsabilidades para la clase *PantallaCrearRegTarjeta*, como se muestra en la Tabla 8.32.

Clase: PantallaCrearRegTarjeta	
Descripción: Pantalla de solicitud de registro de tarjeta (*P*-5).	
Módulo: Registro.Tarjeta	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega (98)	
envía el evento “Registrar” a la InterfaceUsuario (101)	
envía el evento “Servicios” a la InterfaceUsuario (108)	
envía el evento “Salir” a la InterfaceUsuario (111)	

Tabla 8.32. Tarjeta para la clase *PantallaCrearRegTarjeta* con nuevas responsabilidades identificadas hasta el momento.

En la Tabla 8.33 se muestra las responsabilidades para la clase *InterfaceBaseDatosRegistro* anteriormente identificadas en la Tabla 8.29 junto con las nuevas responsabilidades.

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceBD	
Estereotipo: Interface	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *validarRegistroUsuario* a la BaseDatosRegistro (14)	
solicita *crearRegistroUsuario* a la BaseDatosRegistro (50)	
solicita *obtenerRegistroUsuario* a la BaseDatosRegistro (59)	
solicita *actualizarRegistroUsuario* a la BaseDatosRegistro (82)	
solicita *eliminarRegistroUsuario* a la BaseDatosRegistro (89)	
solicita *crearRegistroTarjeta* a la BaseDatosRegistro (104)	

Tabla 8.33. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades identificadas hasta el momento.

Continuamos con el subflujo *Obtener Registro Tarjeta* (*S*-2) del caso de uso *Registrar Tarjeta* como se muestra a continuación,

Subflujos	*S*-2 *Obtener Registro Tarjeta* El ManejadorRegistroTarjeta solicita *obtenerRegistroTarjeta* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro solicita *obtenerRegistroTarjeta* a la Base de Datos Registro. La Base de Datos Registro devuelve el *OK* y el RegistroTarjeta a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro devuelve el *OK* y el RegistroTarjeta al ManejadorRegistroTarjeta. Se regresa al flujo anterior.

Nuevamente, tomamos cada una de las frases y las analizamos para identificar nuevas responsabilidades.

115. **El ManejadorRegistroTarjeta solicita *obtenerRegistroTarjeta* a la InterfaceBaseDatosRegistro**. La responsabilidad es “solicita *obtenerRegistroTarjeta* a la InterfaceBaseDatosRegistro” y se asigna a ManejadorRegistroTarjeta.

116. **La InterfaceBaseDatosRegistro solicita *obtenerRegistroTarjeta* a la Base de Datos Registro**. La responsabilidad es "solicita *obtenerRegistroTarjeta* a la Base de Datos Registro" y se asigna a InterfaceBaseDatosRegistro.
117. **La Base de Datos Registro devuelve el *OK* y el RegistroTarjeta a la InterfaceBaseDatosRegistro**. Nuevamente, eventos de devolución de información no significan responsabilidades adicionales, por lo cual esta frase no agrega ninguna.
118. **La InterfaceBaseDatosRegistro devuelve el *OK* y el RegistroTarjeta al ManejadorRegistroTarjeta**. Esta frase tampoco agrega ninguna responsabilidad.
119. **Se regresa al flujo anterior**. Esta es una frase exclusivamente informativa de la continuación del caso de uso y no agrega nuevas responsabilidades.

En total se agregaron dos nuevas responsabilidades. La primera responsabilidad (115) se agrega a la clase *ManejadorRegistroTarjeta* como se muestra en la Tabla 8.34 la cual incluye las responsabilidades anteriormente identificadas en la Tabla 8.31.

Clase: ManejadorRegistroTarjeta	
Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.	
Módulo: Registro.Tarjeta	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
registrarTarjeta (71)	
solicita *desplegarPantallaCrearRegTarjeta* a la InterfaceUsuario (96)	
maneja el evento "Registrar" (102)	
solicita *crearRegistroTarjeta* a la InterfaceBaseDatosRegistro (103)	
maneja el evento "Servicios" (109)	
solicita *ofrecerServicio* al ManejadorServicio (110)	
maneja el evento "Salir" (112)	
sale del sistema (113)	
solicita *obtenerRegistroTarjeta* a la InterfaceBaseDatosRegistro (115)	

Tabla 8.34. Tarjeta para la clase *ManejadorRegistroTarjeta* con responsabilidades identificadas hasta el momento.

La segunda responsabilidad se agrega a la clase *InterfaceBaseDatosRegistro* como se muestra en la Tabla 8.35 la cual incluye las responsabilidades anteriormente identificadas en la Tabla 8.33.

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceBD	
Estereotipo: Interface	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *validarRegistroUsuario* a la BaseDatosRegistro (14)	
solicita *crearRegistroUsuario* a la BaseDatosRegistro (50)	
solicita *obtenerRegistroUsuario* a la BaseDatosRegistro (59)	
solicita *actualizarRegistroUsuario* a la BaseDatosRegistro (82)	
solicita *eliminarRegistroUsuario* a la BaseDatosRegistro (89)	
solicita *crearRegistroTarjeta* a la BaseDatosRegistro (104)	
solicita *obtenerRegistroTarjeta* a la BaseDatosRegistro (116)	

Tabla 8.35. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades identificadas hasta el momento.

Continuamos con el subflujo *Administrar Registro Tarjeta* (*S*-3) del caso de uso *Registrar Tarjeta* como se muestra a continuación,

Subflujos	*S*-3 *Administrar Registro Tarjeta* El ManejadorRegistroTarjeta solicita *desplegarPantallaObtenerRegTarjeta* a la InterfaceUsuario. La InterfaceUsuario *despliega* la PantallaObtenerRegTarjeta. La PantallaObtenerRegTarjeta se *despliega*. El Usuario puede seleccionar entre las siguientes actividades: "Actualizar", "Eliminar", "Servicios" y "Salir". Si el usuario presiona "Actualizar" se ejecuta el subflujo *Actualizar Registro Tarjeta* (*S*-4). Si el usuario presiona "Eliminar" se ejecuta el subflujo *Eliminar Registro Tarjeta* (*S*-5). Si la actividad seleccionada es "Servicios", la PantallaObtenerRegTarjeta envía el evento "Servicios" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Servicios" al ManejadorRegistroTarjeta. El ManejadorRegistroTarjeta solicita *ofrecerServicio* al ManejadorServicio, se continúa con el caso de uso *Ofrecer Servicios*. Si la actividad seleccionada es "Salir", la PantallaObtenerRegTarjeta envía el evento "Salir" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Salir" al ManejadorRegistroTarjeta. El ManejadorRegistroTarjeta *sale* del sistema. (Si aún no se ha presionado "Actualizar", la nueva información ésta será perdida).

Nuevamente, tomamos cada una de las frases y las analizamos para identificar nuevas responsabilidades.

120. **El ManejadorRegistroTarjeta solicita *desplegarPantallaObtenerRegTarjeta* a la InterfaceUsuario**. La responsabilidad es "solicita *desplegarPantallaObtenerRegTarjeta* a la InterfaceUsuario" y se asigna a la ManejadorRegistroTarjeta.
121. **La InterfaceUsuario *despliega* la PantallaObtenerRegTarjeta**. La responsabilidad es "*despliega* la PantallaObtenerRegTarjeta" y se asigna a la InterfaceUsuario.
122. **La PantallaObtenerRegTarjeta se *despliega***. La responsabilidad es "*despliega*" y se asigna a la PantallaObtenerRegTarjeta.
123. **El Usuario puede seleccionar entre las siguientes actividades: "Actualizar", "Eliminar", "Servicios" y "Salir"**. Esta frase es informativa describiendo las diversas opciones del Usuario y no agrega responsabilidades.
124. **Si el usuario presiona "Actualizar" se ejecuta el subflujo *Actualizar Registro Tarjeta* (*S*-4)**. Esta frase es informativa describiendo una continuación interna del flujo del caso de uso y no agrega responsabilidades.
125. **Si el usuario presiona "Eliminar" se ejecuta el subflujo *Eliminar Registro Tarjeta* (*S*-5)**. De manera similar a la frase anterior, esta frase es informativa describiendo una continuación interna del flujo del caso de uso y no agrega responsabilidades.
126. **Si la actividad seleccionada es "Servicios", la PantallaObtenerRegTarjeta envía el evento "Servicios" a la InterfaceUsuario**. La responsabilidad es "envía el evento "Servicios" a la InterfaceUsuario" y se asigna a PantallaObtenerRegTarjeta.
127. **La InterfaceUsuario envía el evento "Servicios" al ManejadorRegistroTarjeta**. La responsabilidad es "envía el evento "Servicios" al ManejadorRegistroTarjeta" y se debe asignar a InterfaceUsuario. Esta responsabilidad está duplicada y no es necesario volverla a agregar. Adicionalmente, tampoco es necesario agregar la responsabilidad "maneja el evento "Servicios"" al ManejadorRegistroTarjeta, ya que también está duplicada.
128. **El ManejadorRegistroTarjeta solicita *ofrecerServicio* al ManejadorServicio, se continúa con el caso de uso *Ofrecer Servicios***. La responsabilidad es "solicita *ofrecerServicio* al ManejadorServicio" y se debe asignar al ManejadorRegistroTarjeta. Nuevamente, esta es una responsabilidad duplicada. La última sección de la frase describe continuación del flujo interno del caso de uso y no agrega nuevas responsabilidades.
129. **Si la actividad seleccionada es "Salir", la PantallaObtenerRegTarjeta envía el evento "Salir" a la InterfaceUsuario**. La responsabilidad es "envía el evento "Salir" a la InterfaceUsuario" y se asigna a PantallaObtenerRegTarjeta.
130. **La InterfaceUsuario envía el evento "Salir" al ManejadorRegistroTarjeta**. La responsabilidad es "envía el evento "Salir" al ManejadorRegistroTarjeta" y se asigna a InterfaceUsuario. Adicionalmente se debe asignar la responsabilidad "maneja el evento "Salir"" a ManejadorRegistroTarjeta. Sin embargo, esta responsabilidad ya ha sido asignada anteriormente, por lo cual no es necesario duplicarla.

131. **El ManejadorRegistroTarjeta *sale* del sistema**. La responsabilidad es "*sale* del sistema" y se debe asignar a ManejadorRegistroTarjeta. Nuevamente, esta es una responsabilidad duplicada por lo cual no la volvemos a asignar.
132. **(Si aún no se ha presionado "Actualizar", la nueva información ésta será perdida)**. Esta frase es informativa y no agrega responsabilidades adicionales.

A partir de estas frases obtenemos responsabilidades adicionales y las insertamos en las tarjetas de clase correspondientes. Nuevamente, no todas las clases agregan nuevas responsabilidades. En particular, las responsabilidades identificadas para las clases *ManejadorPrincipal* y *PantallaPrincipal* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.3 y en la Tabla 8.5, respectivamente. Las responsabilidades identificadas para las clases *ManejadorServicio* y *PantallaServicio* tampoco han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.10 y en la Tabla 8.11, respectivamente. Adicionalmente, las responsabilidades identificadas para las clases *ManejadorRegistroUsuario*, *PantallaCrearRegUsuario* y *PantallaObtenerRegUsuario* tampoco han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.27, Tabla 8.14 y Tabla 8.28, respectivamente. Tampoco se modifican las responsabilidades de *PantallaCrearRegTarjeta* manteniéndose iguales a las descritas en la Tabla 8.32.

En la Tabla 8.36 se muestra las responsabilidades para la clase *InterfaceUsuario* anteriormente identificadas en la Tabla 8.26 junto con sus nuevas responsabilidades.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega la PantallaPrincipal (2)	
envía el evento "Registrarse por Primera Vez" al ManejadorPrincipal (6)	
envía el evento "OK" al ManejadorPrincipal (11)	
envía el evento "Salir" al ManejadorPrincipal (21)	
despliega la PantallaServicio (24)	
envía el evento "Obtener Registro" al ManejadorServicio (36)	
envía el evento "Salir" al ManejadorPrincipal (40)	
despliega la PantallaCrearRegUsuario (43)	
envía el evento "Registrar" al ManejadorRegistroUsuario (48)	
envía el evento "Salir" al ManejadorRegistroUsuario (55)	
despliega la PantallaObtenerRegistroUsuario (64)	
envía el evento "Registrar Tarjeta" al ManejadorRegistroUsuario (70)	
envía el evento "Servicios" al ManejadorRegistroUsuario (73)	
envía el evento "Actualizar" al ManejadorRegistroUsuario (80)	
envía el evento "Eliminar" al ManejadorRegistroUsuario (87)	
despliega la PantallaCrearRegTarjeta (97)	
envía el evento "Registrar" al ManejadorRegistroTarjeta (102)	
envía el evento "Servicios" al ManejadorRegistroTarjeta (109)	
envía el evento "Salir" al ManejadorRegistroTarjeta (112)	
despliega la PantallaObtenerRegTarjeta (121)	

Tabla 8.36. Tarjeta para la clase *InterfaceUsuario* con responsabilidades identificadas hasta el momento.

En la Tabla 8.37 se muestra las responsabilidades para la clase *ManejadorRegistroTarjeta* anteriormente identificadas en la Tabla 8.31 junto con las nuevas responsabilidades.

Clase: ManejadorRegistroTarjeta
Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.
Módulo: Registro.Tarjeta
Estereotipo: Control

Propiedades:	
Superclases:	
Subclases:	
Atributos:	
registrarTarjeta (71)	
solicita *desplegarPantallaCrearRegTarjeta* a la InterfaceUsuario (96)	
maneja el evento "Registrar" (102)	
solicita *crearRegistroTarjeta* a la InterfaceBaseDatosRegistro (103)	
maneja el evento "Servicios" (109)	
solicita *ofrecerServicio* al ManejadorServicio (110)	
maneja el evento "Salir" (112)	
sale del sistema (113)	
solicita *obtenerRegistroTarjeta* a la InterfaceBaseDatosRegistro (115)	
solicita *desplegarPantallaObtenerRegTarjeta* a la InterfaceUsuario (120)	

Tabla 8.37. Tarjeta para la clase *ManejadorRegistroTarjeta* con responsabilidades identificadas hasta el momento.

Agregamos una nueva tarjeta de clase describiendo las responsabilidades para la clase *PantallaObtenerRegTarjeta*, como se muestra en la Tabla 8.38.

Clase: PantallaObtenerRegTarjeta	
Descripción: Pantalla de devolución con información de registro de tarjeta (*P*-6).	
Módulo: Registro.Tarjeta	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega (122)	
envía el evento "Servicios" a la InterfaceUsuario (126)	
envía el evento "Salir" a la InterfaceUsuario (129)	

Tabla 8.38. Tarjeta para la clase *PantallaObtenerRegTarjeta* con nuevas responsabilidades identificadas hasta el momento.

Continuamos con el subflujo *Actualizar Registro Tarjeta* (*S*-4) del caso de uso *Registrar Tarjeta* como se muestra a continuación,

Subflujos	*S*-4 *Actualizar Registro Tarjeta* La PantallaObtenerRegTarjeta envía el evento "Actualizar" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Actualizar" al ManejadorRegistroTarjeta. El ManejadorRegistroTarjeta solicita *actualizarRegistroTarjeta* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro solicita *actualizarRegistroTarjeta* a la Base de Datos Registro (*E*-1). La Base de Datos Registro actualiza el RegistroTarjeta y devuelve el evento *OK* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroTarjeta. Se continua con el subflujo *Administrar Registro Tarjeta* (*S*-3).

Nuevamente, tomamos cada una de las frases y las analizamos para identificar nuevas responsabilidades.

133. **La PantallaObtenerRegTarjeta envía el evento "Actualizar" a la InterfaceUsuario**. La responsabilidad es "envía el evento "Actualizar" a la InterfaceUsuario" y se asigna a PantallaObtenerRegTarjeta.
134. **La InterfaceUsuario envía el evento "Actualizar" al ManejadorRegistroTarjeta**. La responsabilidad es "envía el evento "Actualizar" al ManejadorRegistroTarjeta" y se asigna a InterfaceUsuario. Adicionalmente, se agrega la responsabilidad "maneja el evento "Actualizar"" y se asigna al ManejadorRegistroTarjeta.
135. **El ManejadorRegistroTarjeta solicita *actualizarRegistroTarjeta* a la InterfaceBaseDatosRegistro**. La responsabilidad es "solicita *actualizarRegistroTarjeta* a la InterfaceBaseDatosRegistro" y se asigna al ManejadorRegistroTarjeta.
136. **La InterfaceBaseDatosRegistro solicita *actualizarRegistroTarjeta* a la Base de Datos Registro (*E*-1)**. La responsabilidad es "solicita *actualizarRegistroTarjeta* a la Base de Datos Registro" y se asigna a la InterfaceBaseDatosRegistro.

137. **La Base de Datos Registro actualiza el RegistroTarjeta y devuelve el evento *OK* a la InterfaceBaseDatosRegistro**. Esta frase se refiere a Base de Datos Registro, un actor externo al sistema, por lo cual no se agregan nuevas responsabilidades. La segunda parte de la frase describe una devolución por lo cual tampoco se agrega ninguna responsabilidad.
138. **La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroTarjeta**. Nuevamente, se describe una devolución por lo cual no se agrega ninguna responsabilidad.
139. **Se continua con el subflujo *Administrar Registro Tarjeta* (*S*-3)**. Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.

A partir de estas frases obtenemos responsabilidades adicionales y las insertamos en las tarjetas de clase correspondientes. Nuevamente, no todas las clases agregan nuevas responsabilidades. En particular, las responsabilidades identificadas para las clases *ManejadorPrincipal* y *PantallaPrincipal* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.3 y en la Tabla 8.5, respectivamente. Las responsabilidades identificadas para las clases *ManejadorServicio* y *PantallaServicio* tampoco han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.10 y en la Tabla 8.11, respectivamente. Adicionalmente, las responsabilidades identificadas para las clases *ManejadorRegistroUsuario*, *PantallaCrearRegUsuario* y *PantallaObtenerRegUsuario* tampoco han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.27, Tabla 8.14 y Tabla 8.28, respectivamente. Tampoco se modifican las responsabilidades de *PantallaCrearRegTarjeta* manteniéndose iguales a las descritas en la Tabla 8.32.

En la Tabla 8.39 se muestra las responsabilidades para la clase *InterfaceUsuario* anteriormente identificadas en la Tabla 8.36 junto con sus nuevas responsabilidades.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega la PantallaPrincipal (2)	
envía el evento "Registrarse por Primera Vez" al ManejadorPrincipal (6)	
envía el evento "OK" al ManejadorPrincipal (11)	
envía el evento "Salir" al ManejadorPrincipal (21)	
despliega la PantallaServicio (24)	
envía el evento "Obtener Registro" al ManejadorServicio (36)	
envía el evento "Salir" al ManejadorPrincipal (40)	
despliega la PantallaCrearRegUsuario (43)	
envía el evento "Registrar" al ManejadorRegistroUsuario (48)	
envía el evento "Salir" al ManejadorRegistroUsuario (55)	
despliega la PantallaObtenerRegistroUsuario (64)	
envía el evento "Registrar Tarjeta" al ManejadorRegistroUsuario (70)	
envía el evento "Servicios" al ManejadorRegistroUsuario (73)	
envía el evento "Actualizar" al ManejadorRegistroUsuario (80)	
envía el evento "Eliminar" al ManejadorRegistroUsuario (87)	
despliega la PantallaCrearRegTarjeta (97)	
envía el evento "Registrar" al ManejadorRegistroTarjeta (102)	
envía el evento "Servicios" al ManejadorRegistroTarjeta (109)	
envía el evento "Salir" al ManejadorRegistroTarjeta (112)	
despliega la PantallaObtenerRegTarjeta (121)	
envía el evento "Actualizar" al ManejadorRegistroTarjeta (134)	

Tabla 8.39. Tarjeta para la clase *InterfaceUsuario* con responsabilidades identificadas hasta el momento.

En la Tabla 8.40 se muestra las responsabilidades para la clase *ManejadorRegistroTarjeta* anteriormente identificadas en la Tabla 8.37 junto con las nuevas responsabilidades.

Clase: ManejadorRegistroTarjeta

Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.	
Módulo: Registro.Tarjeta	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
registrarTarjeta (71)	
solicita *desplegarPantallaCrearRegTarjeta* a la InterfaceUsuario (96)	
maneja el evento "Registrar" (102)	
solicita *crearRegistroTarjeta* a la InterfaceBaseDatosRegistro (103)	
maneja el evento "Servicios" (109)	
solicita *ofrecerServicio* al ManejadorServicio (110)	
maneja el evento "Salir" (112)	
sale del sistema (113)	
solicita *obtenerRegistroTarjeta* a la InterfaceBaseDatosRegistro (115)	
solicita *desplegarPantallaObtenerRegTarjeta* a la InterfaceUsuario (120)	
maneja el evento "Actualizar" (134)	
solicita *actualizarRegistroTarjeta* a la InterfaceBaseDatosRegistro (135)	

Tabla 8.40. Tarjeta para la clase *ManejadorRegistroTarjeta* con responsabilidades identificadas hasta el momento.

En la Tabla 8.41 se muestra las responsabilidades para la clase *PantallaObtenerRegTarjeta* anteriormente identificadas en la Tabla 8.38 junto con sus nuevas responsabilidades.

Clase: PantallaObtenerRegTarjeta	
Descripción: Pantalla de devolución con información de registro de tarjeta (*P*-6).	
Módulo: Registro.Tarjeta	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega (122)	
envía el evento "Servicios" a la InterfaceUsuario (126)	
envía el evento "Salir" a la InterfaceUsuario (129)	
envía el evento "Actualizar" a la InterfaceUsuario (133)	

Tabla 8.41. Tarjeta para la clase *PantallaObtenerRegTarjeta* con nuevas responsabilidades identificadas hasta el momento.

En la Tabla 8.42 se muestra las responsabilidades para la clase *InterfaceBaseDatosRegistro* anteriormente identificadas en la Tabla 8.35 junto con sus nuevas responsabilidades.

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceBD	
Estereotipo: Interface	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *validarRegistroUsuario* a la BaseDatosRegistro (14)	
solicita *crearRegistroUsuario* a la BaseDatosRegistro (50)	
solicita *obtenerRegistroUsuario* a la BaseDatosRegistro (59)	

solicita *actualizarRegistroUsuario* a la BaseDatosRegistro (82)	
solicita *eliminarRegistroUsuario* a la BaseDatosRegistro (89)	
solicita *crearRegistroTarjeta* a la BaseDatosRegistro (104)	
solicita *obtenerRegistroTarjeta* a la BaseDatosRegistro (116)	
solicita *actualizarRegistroTarjeta* a la BaseDatosRegistro (136)	

Tabla 8.42. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades identificadas hasta el momento.

Continuamos con el subflujo *Eliminar Registro Tarjeta* (*S*-5) del caso de uso *Registrar Tarjeta* como se muestra a continuación,

Subflujos	*S*-5 *Eliminar Registro Tarjeta* La PantallaObtenerRegTarjeta envía el evento "Eliminar" a la InterfaceUsuario. La InterfaceUsuario envía el evento "Eliminar" al ManejadorRegistroTarjeta. El ManejadorRegistroTarjeta solicita *eliminarRegistroTarjeta* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro solicita *eliminarRegistroTarjeta* a la Base de Datos Registro. La Base de Datos Registro elimina el RegistroTarjeta (*E*-1) y devuelve el *OK* a la InterfaceBaseDatosRegistro. La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroTarjeta. Se continua con el subflujo *Crear Registro Tarjeta* (*S*-1).

Nuevamente, tomamos cada una de las frases y las analizamos para identificar nuevas responsabilidades.

140. **La PantallaObtenerRegTarjeta envía el evento "Eliminar" a la InterfaceUsuario**. La responsabilidad es "envía el evento "Eliminar" a la InterfaceUsuario" y se asigna a PantallaObtenerRegTarjeta.
141. **La InterfaceUsuario envía el evento "Eliminar" al ManejadorRegistroTarjeta**. La responsabilidad es "envía el evento "Eliminar" al ManejadorRegistroTarjeta" y se asigna a InterfaceUsuario. Adicionalmente, se agrega la responsabilidad "maneja el evento "Eliminar"" y se asigna al ManejadorRegistroTarjeta.
142. **El ManejadorRegistroTarjeta solicita *eliminarRegistroTarjeta* a la InterfaceBaseDatosRegistro**. La responsabilidad es "solicita *eliminarRegistroTarjeta* a la InterfaceBaseDatosRegistro" y se asigna al ManejadorRegistroTarjeta.
143. **La InterfaceBaseDatosRegistro solicita *eliminarRegistroTarjeta* a la Base de Datos Registro**. La responsabilidad es "solicita *eliminarRegistroTarjeta* a la Base de Datos Registro" y se asigna a la InterfaceBaseDatosRegistro.
144. **La Base de Datos Registro elimina el RegistroTarjeta (*E*-1) y devuelve el *OK* a la InterfaceBaseDatosRegistro**. Esta frase se refiere a Base de Datos Registro, un actor externo al sistema, por lo cual no se agregan nuevas responsabilidades. La segunda parte de la frase describe una devolución por lo cual tampoco se agrega ninguna responsabilidad.
145. **La InterfaceBaseDatosRegistro devuelve el *OK* al ManejadorRegistroTarjeta**. Nuevamente, se describe una devolución por lo cual no se agrega ninguna responsabilidad.
146. **Se continua con el subflujo *Crear Registro Tarjeta* (*S*-1).** Esta es una frase que describe flujo interno de continuación del caso de uso, por lo cual no agrega responsabilidades.

A partir de estas frases obtenemos responsabilidades adicionales y las insertamos en las tarjetas de clase correspondientes. Nuevamente, no todas las clases agregan nuevas responsabilidades. En particular, las responsabilidades identificadas para las clases *ManejadorPrincipal* y *PantallaPrincipal* no han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.3 y en la Tabla 8.5, respectivamente. Las responsabilidades identificadas para las clases *ManejadorServicio* y *PantallaServicio* tampoco han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.10 y en la Tabla 8.11, respectivamente. Adicionalmente, las responsabilidades identificadas para las clases *ManejadorRegistroUsuario*, *PantallaCrearRegUsuario* y *PantallaObtenerRegUsuario* tampoco han sido modificadas y se mantienen iguales a las descritas en la Tabla 8.27, Tabla 8.14 y Tabla 8.28, respectivamente. Tampoco se modifican las responsabilidades de *PantallaCrearRegTarjeta* manteniéndose iguales a las descritas en la Tabla 8.32.

En la Tabla 8.43 se muestra las responsabilidades para la clase *InterfaceUsuario* anteriormente identificadas en la Tabla 8.39 junto con sus nuevas responsabilidades.

Clase: InterfaceUsuario
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.
Módulo: InterfaceUsuario
Estereotipo: Borde
Propiedades:

Superclases:	
Subclases:	
Atributos:	
despliega la PantallaPrincipal (2)	
envía el evento "Registrarse por Primera Vez" al ManejadorPrincipal (6)	
envía el evento "OK" al ManejadorPrincipal (11)	
envía el evento "Salir" al ManejadorPrincipal (21)	
despliega la PantallaServicio (24)	
envía el evento "Obtener Registro" al ManejadorServicio (36)	
envía el evento "Salir" al ManejadorPrincipal (40)	
despliega la PantallaCrearRegUsuario (43)	
envía el evento "Registrar" al ManejadorRegistroUsuario (48)	
envía el evento "Salir" al ManejadorRegistroUsuario (55)	
despliega la PantallaObtenerRegistroUsuario (64)	
envía el evento "Registrar Tarjeta" al ManejadorRegistroUsuario (70)	
envía el evento "Servicios" al ManejadorRegistroUsuario (73)	
envía el evento "Actualizar" al ManejadorRegistroUsuario (80)	
envía el evento "Eliminar" al ManejadorRegistroUsuario (87)	
despliega la PantallaCrearRegTarjeta (97)	
envía el evento "Registrar" al ManejadorRegistroTarjeta (102)	
envía el evento "Servicios" al ManejadorRegistroTarjeta (109)	
envía el evento "Salir" al ManejadorRegistroTarjeta (112)	
despliega la PantallaObtenerRegTarjeta (121)	
envía el evento "Actualizar" al ManejadorRegistroTarjeta (134)	
envía el evento "Eliminar" al ManejadorRegistroTarjeta (141)	

Tabla 8.43. Tarjeta para la clase *InterfaceUsuario* con responsabilidades identificadas hasta el momento.

En la Tabla 8.44 se muestra las responsabilidades para la clase *ManejadorRegistroTarjeta* anteriormente identificadas en la Tabla 8.40 junto con las nuevas responsabilidades.

Clase: ManejadorRegistroTarjeta	
Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.	
Módulo: Registro.Tarjeta	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
registrarTarjeta (71)	
solicita *desplegarPantallaCrearRegTarjeta* a la InterfaceUsuario (96)	
maneja el evento "Registrar" (102)	
solicita *crearRegistroTarjeta* a la InterfaceBaseDatosRegistro (103)	
maneja el evento "Servicios" (109)	
solicita *ofrecerServicio* al ManejadorServicio (110)	
maneja el evento "Salir" (112)	
sale del sistema (113)	
solicita *obtenerRegistroTarjeta* a la InterfaceBaseDatosRegistro (115)	
solicita *desplegarPantallaObtenerRegTarjeta* a la InterfaceUsuario (120)	
maneja el evento "Actualizar" (134)	
solicita *actualizarRegistroTarjeta* a la InterfaceBaseDatosRegistro (135)	
maneja el evento "Eliminar" (141)	
solicita *eliminarRegistroTarjeta* a la InterfaceBaseDatosRegistro (142)	

Tabla 8.44. Tarjeta para la clase *ManejadorRegistroTarjeta* con responsabilidades identificadas hasta el momento.

En la Tabla 8.45 se muestra las responsabilidades para la clase *PantallaObtenerRegTarjeta* anteriormente identificadas en la Tabla 8.41 junto con sus nuevas responsabilidades.

Clase: PantallaObtenerRegTarjeta	
Descripción: Pantalla de devolución con información de registro de tarjeta (*P*-6).	
Módulo: Registro.Tarjeta	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega (122)	
envía el evento “Servicios” a la InterfaceUsuario (126)	
envía el evento “Salir” a la InterfaceUsuario (129)	
envía el evento “ Actualizar” a la InterfaceUsuario (133)	
envía el evento “Eliminar” a la InterfaceUsuario (140)	

Tabla 8.45. Tarjeta para la clase *PantallaObtenerRegTarjeta* con nuevas responsabilidades identificadas hasta el momento.

En la Tabla 8.46 se muestra las responsabilidades para la clase *InterfaceBaseDatosRegistro* anteriormente identificadas en la Tabla 8.42 junto con sus nuevas responsabilidades.

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceBD	
Estereotipo: Interface	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *validarRegistroUsuario* a la BaseDatosRegistro (14)	
solicita *crearRegistroUsuario* a la BaseDatosRegistro (50)	
solicita *obtenerRegistroUsuario* a la BaseDatosRegistro (59)	
solicita *actualizarRegistroUsuario* a la BaseDatosRegistro (82)	
solicita *eliminarRegistroUsuario* a la BaseDatosRegistro (89)	
solicita *crearRegistroTarjeta* a la BaseDatosRegistro (104)	
solicita *obtenerRegistroTarjeta* a la BaseDatosRegistro (116)	
solicita *actualizarRegistroTarjeta* a la BaseDatosRegistro (136)	
solicita *eliminarRegistroTarjeta* a la BaseDatosRegistro (143)	

Tabla 8.46. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades identificadas hasta el momento.

Se puede también obtener responsabilidades a partir del manejajo de excepciones, como se muestra a continuación.

Excepciones	*E-1 información incompleta*: Falta llenar información indispensable para completar el registro de tarjeta. Se le vuelve a pedir al usuario que complete el registro de tarjeta.

Nuevamente, dejaremos esto por el momento siguiendo las mismas consideraciones anteriores.

De manera similar a los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*, se tendrá que definir las responsabilidades por clase para el sistema completo, algo que presentaremos en los apéndices y bajo un alcance limitado. En las siguientes secciones mostramos las tarjetas clases relevantes hasta el momento junto con sus responsabilidades aignadas. Las clases completas se pueden ver en el apéndice.

InterfaceUsuario

La clase *InterfaceUsuario* recibe y envía eventos a un gran número de clases, algo que es resaltado por la lista extensa de responsabilidades identificadas, como se muestra en la Tabla 8.9.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega la PantallaPrincipal	
envía el evento “OK” al ManejadorPrincipal	
envía el evento “Salir” al ManejadorPrincipal	
envía el evento “Registrarse por Primera Vez” al ManejadorPrincipal	
despliega la PantallaServicio	
envía el evento “Obtener Registro” al ManejadorServicio	
despliega la PantallaCrearRegUsuario	
envía el evento “Registrar” al ManejadorRegistroUsuario	
envía el evento “Salir” al ManejadorRegistroUsuario	
despliega la PantallaObtenerRegUsuario	
envía el evento “Actualizar" al ManejadorRegistroUsuario	
envía el evento “Eliminar” al ManejadorRegistroUsuario	
envía el evento “Registrar Tarjeta” al ManejadorRegistroUsuario	
envía el evento “Servicios” al ManejadorRegistroUsuario	
despliega la PantallaCrearRegTarjeta	
despliega la PantallaObtenerRegTarjeta	
envía el evento “Registrar” al ManejadorRegistroTarjeta	
envía el evento “Servicios” al ManejadorRegistroTarjeta	
envía el evento “Salir” al ManejadorRegistroTarjeta	
envía el evento “Actualizar” al ManejadorRegistroTarjeta	
envía el evento “Eliminar” al ManejadorRegistroTarjeta	

Tabla 8.9. Tarjeta para la clase *InterfaceUusario* con responsabilidades identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Principal

Esta sección incluye las clases “principales” de la arquitectura que son el *ManejadorPrincipal* y la *PntallaPrincipal*. La clase *ManejadorPrincipal* recibe y envía eventos entre manejadores y la *InterfaceUsuario*, como se muestra en la Tabla 8.10.

Clase: ManejadorPrincipal	
Descripción: El manejador principal es el encargado de desplegar la pantalla principal de interacción con el usuario, y luego delegar las diferentes funciones a los manejadores especializados apropiados.	
Módulo: Principal	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *desplegarPantallaPrincipal* a la InterfaceUsuario	
maneja el evento “Registrarse por Primera Vez”	
solicita *crearRegistroUsuario* al ManejadorRegistroUsuario	
maneja el evento “OK”	
solicita *ofrecerServicio* al ManejadorServicio	
solicita *validarRegistroUsuario* al ManejadorRegistroUsuario	
maneja el evento “Salir”	
sale del sistema	

Tabla 8.10. Tarjeta para la clase *ManejadorPrincipal* con responsabilidades identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

La clase *PantallaPrincipal* es la encarga de presentar las opciones de inicio del sistema, como se muestra en la Tabla 8.11.

Clase: PantallaPrincipal	
Descripción: Pantalla principal (*P*-1).	
Módulo: Principal	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega	
envía el evento "Registrarse por Primera Vez" a la InterfaceUsuario	
envía el evento "OK" a la InterfaceUsuario	
envía el evento "Salir" a la InterfaceUsuario	

Tabla 8.11. Tarjeta para la clase *PantallaPrincipal* con responsabilidades de interface identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Registro

Este módulo se compone de los módulos de *Usuario*, *Tarjeta* e *InterfaceBD*.

Usuario

Esta sección involucra las clases de registro de usuario que son *ManejadoRegistroUsuario*, *PantallaCrearRegUsuario*, *PantallaObtenerRegUsuario* y *RegistroUsuario*.

La clase *ManejadoRegistroUsuario* administra todo lo relacionado con registro de usuario, lo cual incluye recibir y enviar eventos entre el *ManejadorPrincipal*, la *InterfaceUsuario*, y la *InterfaceBaseDatosRegistro*, como se muestra en la Tabla 8.12. Nótese que eliminamos las frases de tipo "devuelve el OK...".

Clase: ManejadorRegistroUsuario	
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para poder utilizar el sistema.	
Módulo: Registro.Usuario	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
crearRegistroUsuario	
solicita *desplegarPantallaCrearRegUsuario* a la InterfaceUsuario	
maneja el evento "Registrar"	
solicita *crearRegistroUsuario* a la InterfaceBaseDatosRegistro	
solicita *validarRegistroUsuario* a la InterfaceBaseDatosRegistro	
solicita *desplegarPantallaObtenerRegUsuario* a la InterfaceUsuario	
solicita *obtenerRegistroUsuario* a la InterfaceBaseDatosRegistro	
maneja el evento "Actualizar"	
solicita *actualizarRegistroUsuario* a la InterfaceBaseDatosRegistro	
maneja el evento "Eliminar"	
solicita *eliminarRegistroUsuario* a la InterfaceBaseDatosRegistro	
maneja el evento "Registrar Tarjeta"	
solicita *registrarTarjeta* al ManejadorRegistroTarjeta	
maneja el evento "Servicios"	
solicita *ofrecerServicio* al ManejadorServicio	
maneja el evento "Salir"	
sale del sistema	

Tabla 8.12. Tarjeta para la clase *ManejadoRegistroUsuario* con responsabilidades identificadas de los casos de uso *RegistrarUsuario* y *ValidarUsuario*.

La clase *PantallaCrearRegUsuario* administra la pantalla relacionada con la creación de registro de usuario, como se muestra en la Tabla 8.13.

Clase: PantallaCrearRegUsuario	
Descripción: Pantalla de solicitud de registro de usuario (*P*-3).	
Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega	
envía el evento "Registrar" a la InterfaceUsuario	
envía el evento "Salir" a la InterfaceUsuario	

Tabla 8.13. Tarjeta para la clase *PantallaCrearRegUsuario* con responsabilidades de interface identificadas del caso de uso *RegistrarUsuario*.

La clase *PantallaObtenerRegUsuario* administra la pantalla relacionada con la modificación y eliminación de registro de usuario, como se muestra en la Tabla 8.14.

Clase: PantallaObtenerRegUsuario	
Descripción: Pantalla de devolución con información de registro de usuario (*P*-4).	
Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega	
envía el evento "Actualizar" al InterfaceUsuario	
envía el evento "Eliminar" a la InterfaceUsuario	
envía el evento "Registrar Tarjeta" a la InterfaceUsuario	
envía el evento "Servicios" a la InterfaceUsuario	
envía el evento "Salir" a la InterfaceUsuario	

Tabla 8.14. Tarjeta para la clase *PantallaObtenerRegUsuario* con responsabilidades de interface identificadas del caso de uso *RegistrarUsuario*.

La clase *RegistroUsuario* guarda la información de registro de usuario, como se muestra en la Tabla 8.15. Nótese que hasta el momento no se han asignado responsabilidades a esta clase. Esto no debe preocuparnos por el momento ya que por ser clases entidad sus responsabilidades son bastante limitadas y localizadas. En otras palabras, cambios en ellas no afectarán en mayor manera la lógica de la aplicación.

Clase: RegistroUsuario	
Descripción: Para poder utilizar el sistema de reservaciones, el usuario debe estar registrado con el sistema. El registro contiene información acerca del usuario que incluye nombre, dirección, colonia, ciudad, país, código postal, teléfono de casa, teléfono de oficina, fax, email, login y password.	
Módulo: Registro.Usuario	
Estereotipo: Entidad	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	

Tabla 8.15. Tarjeta para la clase *RegistroUsuario* sin responsabilidades iniciales de registro para el caso de uso *RegistrarUsuario*.

Tarjeta

Esta sección involucra las clases de registro tarjeta que son *ManejadoRegistroTarjeta*, *PantallaCrearRegTarjeta*, *PantallaObtenerRegTarjeta* y *RegistroTarjeta*.

La clase *ManejadoRegistroTarjeta* administra todo lo relacionado con registro de tarjeta, lo cual incluye recibir y enviar eventos entre el *ManejadorRegistroUsuario*, la *InterfaceUsuario*, y la *InterfaceBaseDatosRegistro*, como se muestra en la Tabla 8.16.

Clase: ManejadorRegistroTarjeta	
Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.	
Módulo: Registro.Tarjeta	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
registrarTarjeta	
crearRegistroTarjeta	
solicita *desplegarPantallaCrearRegTarjeta* a la InterfaceUsuario	
solicita *obtenerRegistroTarjeta* a la InterfaceBaseDatosRegistro	
solicita *desplegarPantallaObtenerRegTarjeta* a la InterfaceUsuario	
manejar el evento "Registrar"	
solicita *crearRegistroTarjeta* a la InterfaceBaseDatosRegistro	
manejar el evento "Actualizar"	
solicita *actualizarRegistroTarjeta* a la InterfaceBaseDatosRegistro	
manejar el evento "Eliminar"	
solicita *eliminarRegistroTarjeta* a la InterfaceBaseDatosRegistro	
manejar el evento "Servicios"	
solicita *ofrecerServicio* al ManejadorServicio	
manejar el evento "Salir"	
sale del sistema	

Tabla 8.16. Tarjeta para la clase *ManejadoRegistroTarjeta* con responsabilidades identificadas del caso de uso *RegistrarTarjeta*.

La clase *PantallaCrearRegTarjeta* administra la pantalla relacionada con la creación de registro de tarjeta, como se muestra en la Tabla 8.17.

Clase: PantallaCrearRegTarjeta	
Descripción: Pantalla de solicitud de registro de tarjeta (*P*-5).	
Módulo: Registro.Tarjeta	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega	
envía el evento "Registrar" a la InterfaceUsuario	
envía el evento "Servicios" a la InterfaceUsuario	
envía el evento "Salir" a la InterfaceUsuario	

Tabla 8.17. Tarjeta para la clase *PantallaCrearRegTarjeta* con responsabilidades de interface identificadas del caso de uso *RegistrarTarjeta*.

La clase *PantallaObtenerRegTarjeta* administra la pantalla relacionada con la modificación y eliminación de registro de tarjeta, como se muestra en la Tabla 8.18.

Clase: PantallaObtenerRegTarjeta
Descripción: Pantalla de devolución con información de registro de tarjeta (*P*-6).
Módulo: Registro.Tarjeta

Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega	
envía el evento “ Actualizar” a la InterfaceUsuario	
envía el evento “Eliminar” a la InterfaceUsuario	
envía el evento “Servicios” a la InterfaceUsuario	
envía el evento “Salir” a la InterfaceUsuario	

Tabla 8.18. Tarjeta para la clase *PantallaObtenerRegTarjeta* con responsabilidades de interface identificadas del caso de uso *RegistrarTarjeta*.

La clase *RegistroTarjeta* guarda la información de registro de tarjeta, como se muestra en la Tabla 8.19. Nuevamente, aún no se incluyen responsabilidades por ser una clase entidad.

Clase: RegistroTarjeta	
Descripción: Para poder hacer un pago con una tarjeta de crédito, se debe tener un registro de tarjeta. El registro contiene información acerca de la tarjeta incluyendo nombre, número, expedidor y vencimiento. La tarjeta está ligada a un registro de usuario.	
Módulo: Registro.Tarjeta	
Estereotipo: Entidad	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	

Tabla 8.19. Tarjeta para la clase *RegistroTarjeta* sin responsabilidades de registro para el caso de uso *RegistrarTarjeta*.

Interface Base de Datos

La clase *InterfaceBaseDatosRegistro* es la encargada de interactuar con el actor *BaseDatosRegistro* para escribir y leer la información allí guardada, tanto de registro de usuario como de registro de tarjeta, como se muestra en la Tabla 8.20.

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceBD	
Estereotipo: Interface	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *crearRegistroUsuario* a la BaseDatosRegistro	
solicita *validarRegistroUsuario* a la BaseDatosRegistro	
solicita *obtenerRegistroUsuario* a la BaseDatosRegistro	
solicita *actualizarRegistroUsuario* a la BaseDatosRegistro	
solicita *eliminarRegistroUsuario* a la BaseDatosRegistro	
solicita *crearRegistroTarjeta* a la BaseDatosRegistro	
solicita *obtenerRegistroTarjeta* a la BaseDatosRegistro	
solicita *actualizarRegistroTarjeta* a la BaseDatosRegistro	
solicita *eliminarRegistroTarjeta* a la BaseDatosRegistro	

Tabla 8.20. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades de escribir y leer información de registro de usuario y registro de tarjeta para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Servicios

La clase *ManejadorServicio* es la encargada de todo lo relacionado con consultas, reservaciones y pagos. Además de esto, es responsable de permitir al usuario accesar la información de registro, como se muestra en la Tabla 8.21.

Clase: ManejadorServicio	
Descripción: El manejador de servicios se encarga de enviar las peticiones particulares de servicios a los manejadores espacializados para consulta, reserva y compra.	
Módulo: Servicios	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
ofrecerServicio	
solicita *desplegarPantallaServicio* a la InterfaceUsuario	
maneja el evento "Obtener Registro"	
solicita *registrar* al ManejadorRegistroUsuario	
maneja el evento "Salir"	
sale del sistema	

Tabla 8.21. Tarjeta para la clase *ManejadorServicio* con responsabilidades a partir de los casos de uso *RegistrarUsuario* y *RegistrarTarjeta*.

La clase *PantallaServicio* es la encarga de presentar las opciones de servicio del sistema, como se muestra en la Tabla 8.22.

Clase: PantallaServicio	
Descripción: Pantalla de servicios (*P*-2).	
Módulo: Servicios	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega	
envía el evento "Obtener Registro" a la InterfaceUsuario	

Tabla 8.22. Tarjeta para la clase *PantallaServicio* con responsabilidades a partir de los casos de uso *RegistrarUsuario* y *RegistrarTarjeta*.

Colaboraciones

Es indispensable que los objetos dentro de un programa colaboren entre si o de lo contrario el programa consistirá de múltiples "mini-programas" independientes. Las colaboraciones entre objetos se dan en base a las relaciones entre las responsabilidades de las distintas clases. Dado la infinidad de posibles relaciones entre clase, las colaboraciones son la fuente principal de complejidad en el diseño de objetos.

Las colaboraciones representan solicitudes de un objeto *cliente* a un objeto *servidor*. Los objetos pueden jugar el papel de clientes o servidores dependiendo de su actividad en ese momento. Un objeto juega el papel de servidor cuando satisface una solicitud hecha por un objeto cliente. Desde el punto de vista del cliente, cada una de sus solicitudes de servicio o colaboración se asocia con una responsabilidad implementada por algún servidor. A su vez, la clase que ofrece el servicio puede solicitar una colaboración con alguna otra clase servidor. Esta cadena de solicitudes y servicios puede extenderse según sea necesario.

El proceso de identificación de colaboraciones se basa en la funcionalidad de la aplicación y de la arquitectura de clases propuesta, algo que fue hecho durante la etapa de análisis. Para identificar colaboraciones se analiza las responsabilidades de cada clase, decidiendo si cada una de estas clases es capaz de satisfacer sus responsabilidades por si misma o si requiere de otra clase para lograrlo. Se analiza qué tanto sabe cada clase y qué otras clases

necesitan su conocimiento o funcionalidad. El patrón de colaboraciones revela el flujo de control e información dentro del sistema. Analizando los patrones de comunicación entre objetos puede revelar cuando una responsabilidad se ha asignado de forma incorrecta., incluso puede revelar responsabilidades ausentes. Se analizan las colaboraciones estudiando qué objetos dentro del sistema tienen el conocimiento que se busca. Se busca aquellas clases que colaboran y las que no colaboran con las demás, qué grupos de clases parecen trabajar juntas y donde existen cuellos de botella. A través de las colaboraciones se analizan las dependencias entre clases.
Si una clase no tiene colaboraciones con otras clases, en otras palabras, nadie colabora con ella y ella no colabora con nadie, esta clase debe descartarse. Sin embargo, antes de hacerlo es bueno revisar el diseño completo verificando todas las clases y colaboraciones. Es importante recordar que las colaboraciones son en una sola dirección, y que ciertos tipos de clases son típicamente clientes o servidores, ubicándose las clase en uno u otro extremo de la colaboración. En la mayoría de los casos, las clases correspondientes al estereotipo de *borde* y *entidad* no son clientes de otros objetos en el sistema, si no que existen para ser servidores, donde otras clases colaboran con ellas, típicamente las clases de *control*.
Durante el proceso de identificación de las colaboraciones se toma la tarjeta de clase que juega el papel de cliente, escribiendo en la columna derecha el nombre de la clase que juega el papel de servidor o sea la clase que colabora para la responsabilidad particular. Se escribe ese nombre directamente a la derecha de la responsabilidad a la cual la colaboración ayuda a satisfacer. Aunque una responsabilidad pudiera requerir de varias colaboraciones, se escribe el nombre de la clase principal como servidor de la colaboración. Las demás clases típicamente pasan a ser parámetros dentro de las llamadas de métodos al momento de implementarse, por lo cual se pueden mantener como parte de la responsabilidad general. Por otro lado, si varias responsabilidades requieren una misma clase para colaborar, se graban varias colaboraciones, una para cada responsabilidad. Se debe asegurar que la responsabilidad correspondiente exista para cada colaboración que se grabe. Se debe grabar la colaboración aunque ésta corresponda a otras instancias de la misma clase. Se debe omitir si la colaboración corresponde al mismo objeto ya que se busca identificar relaciones entre objetos o clases pero no llamadas dentro del mismo objeto.
En esta sección se describen las colaboraciones para el Sistema de Reservaciones en base a los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*. Nótese que se omiten actores primarios, como *Usuario*, en las colaboraciones, aunque se incluyen los actores secundarios como *BaseDatosRegistro*.

InterfaceUsuario

A partir de la Tabla 8.9 se generan las colaboraciones para la clase *InterfaceUsuario*, como se muestra en la Tabla 8.23. Nótese que simplemente se pasa la clase subrayada en la columna izquierda a la columna derecha. Este procedimiento se mantiene lo más mecánico posible durante esta etapa.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega	PantallaPrincipal
envía el evento “OK”	ManejadorPrincipal
envía el evento “Salir”	ManejadorPrincipal
envía el evento “Registrarse por Primera Vez”	ManejadorPrincipal
despliega	PantallaServicio
envía el evento “Obtener Registro”	ManejadorServicio
despliega	PantallaCrearRegUsuario
envía el evento “Registrar”	ManejadorRegistroUsuario
envía el evento “Salir”	ManejadorRegistroUsuario
despliega	PantallaObtenerRegUsuario
envía el evento “Actualizar"	ManejadorRegistroUsuario
envía el evento “Eliminar”	ManejadorRegistroUsuario
envía el evento “Registrar Tarjeta”	ManejadorRegistroUsuario
envía el evento “Servicios”	ManejadorRegistroUsuario
despliega	PantallaCrearRegTarjeta

despliega	PantallaObtenerRegTarjeta
envía el evento "Registrar"	ManejadorRegistroTarjeta
envía el evento "Servicios"	ManejadorRegistroTarjeta
envía el evento "Salir"	ManejadorRegistroTarjeta
envía el evento "Actualizar"	ManejadorRegistroTarjeta
envía el evento "Eliminar"	ManejadorRegistroTarjeta

Tabla 8.23. Tarjeta para la clase *InterfaceUusario* con responsabilidades y colaboraciones identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Principal

A partir de la Tabla 8.10 se generan las colaboraciones para la clase *ManejadorPrincipal*, como se muestra en la Tabla 8.24.

Clase: ManejadorPrincipal	
Descripción: El manejador principal es el encargado de desplegar la pantalla principal de interacción con el usuario, y luego delegar las diferentes funciones a los manejadores especializados apropiados.	
Módulo: Principal	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *desplegarPantallaPrincipal*	InterfaceUsuario
maneja el evento "Registrarse por Primera Vez"	
solicita *crearRegistroUsuario*	ManejadorRegistroUsuario
maneja el evento "OK"	
solicita *ofrecerServicio*	ManejadorServicio
solicita *validarRegistroUsuario*	ManejadorRegistroUsuario
maneja el evento "Salir"	
sale del sistema	

Tabla 8.24. Tarjeta para la clase *ManejadorPrincipal* con responsabilidades y colaboraciones identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

A partir de la Tabla 8.11 se generan las colaboraciones para la clase *PantallaPrincipal*, como se muestra en la Tabla 8.25.

Clase: PantallaPrincipal	
Descripción: Pantalla principal (*P*-1).	
Módulo: Principal	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega	
envía el evento "Registrarse por Primera Vez"	InterfaceUsuario
envía el evento "OK"	InterfaceUsuario
envía el evento "Salir"	InterfaceUsuario

Tabla 8.25. Tarjeta para la clase *PantallaPrincipal* con responsabilidades y colaboraciones identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Registro

Este módulo se compone de los módulos de *Usuario*, *Tarjeta* e *InterfaceBD*.

Usuario

Esta sección involucra las clases de registro de usuario que son *ManejadoRegistroUsuario*, *PantallaCrearRegUsuario*, *PantallaObtenerRegUsuario* y *RegistroUsuario*.

A partir de la Tabla 8.12 se generan las colaboraciones para la clase *ManejadoRegistroUsuario*, como se muestra en la Tabla 8.26.

Clase: ManejadorRegistroUsuario	
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para poder utilizar el sistema.	
Módulo: Registro.Usuario	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
crearRegistroUsuario	
solicita *desplegarPantallaCrearRegUsuario*	InterfaceUsuario
manejar el evento "Registrar"	
solicita *crearRegistroUsuario*	InterfaceBaseDatosRegistro
solicita *validarRegistroUsuario*	InterfaceBaseDatosRegistro
solicita *obtenerRegistroUsuario*	InterfaceBaseDatosRegistro
solicita *desplegarPantallaObtenerRegUsuario*	InterfaceUsuario
manejar el evento "Actualizar"	
solicita *actualizarRegistroUsuario*	InterfaceBaseDatosRegistro
manejar el evento "Eliminar"	
solicita *eliminarRegistroUsuario*	InterfaceBaseDatosRegistro
manejar el evento "Registrar Tarjeta"	
solicita *registrarTarjeta*	ManejadorRegistroTarjeta
manejar el evento "Servicios"	
solicita *ofrecerServicio*	ManejadorServicio
manejar el evento "Salir"	
sale del sistema	

Tabla 8.26. Tarjeta para la clase *ManejadoRegistroUsuario* con responsabilidades y colaboraciones identificadas de los casos de uso *RegistrarUsuario* y *ValidarUsuario*.

A partir de la Tabla 8.12 se generan las colaboraciones para la clase *PantallaCrearRegUsuario*, como se muestra en la Tabla 8.27.

Clase: PantallaCrearRegUsuario	
Descripción: Pantalla de solicitud de registro de usuario (*P*-3).	
Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega	
envía el evento "Registrar"	InterfaceUsuario
envía el evento "Salir"	InterfaceUsuario

Tabla 8.27. Tarjeta para la clase *PantallaCrearRegUsuario* con responsabilidades y colaboraciones identificadas del caso de uso *RegistrarUsuario*.

A partir de la Tabla 8.13 se generan las colaboraciones para la clase *PantallaObtenerRegUsuario*, como se muestra en la Tabla 8.28.

Clase: PantallaObtenerRegUsuario
Descripción: Pantalla de devolución con información de registro de usuario (*P*-4).
Módulo: Registro.Usuario
Estereotipo: Borde
Propiedades:
Superclases:

Subclases:	
Atributos:	
despliega	
envía el evento “Actualizar"	InterfaceUsuario
envía el evento “Eliminar”	InterfaceUsuario
envía el evento “Registrar Tarjeta”	InterfaceUsuario
envía el evento “Servicios”	InterfaceUsuario
envía el evento “Salir”	InterfaceUsuario

Tabla 8.28. Tarjeta para la clase *PantallaCrearRegUsuario* con responsabilidades y colaboraciones identificadas del caso de uso *RegistrarUsuario*.

A partir de la Tabla 8.14 se generan las colaboraciones para la clase *RegistroUsuario*, como se muestra en la Tabla 8.29. Dado que *RegistroUsuario* es una clase entidad, sus responsabilidades por lo general no involucran colaboraciones. En este caso aún no se han asignado dichas responsabilidades..

Clase: RegistroUsuario	
Descripción: Para poder utilizar el sistema de reservaciones, el usuario debe estar registrado con el sistema. El registro contiene información acerca del usuario que incluye nombre, dirección, colonia, ciudad, país, código postal, teléfono de casa, teléfono de oficina, fax, email, login y password.	
Módulo: Registro.Usuario	
Estereotipo: Entidad	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	

Tabla 8.29. Tarjeta para la clase *RegistroUsuario* sin responsabilidades ni colaboraciones de registro para el caso de uso *RegistrarUsuario*.

Tarjeta

Esta sección involucra las clases de registro de tarjeta que son *ManejadoRegistroTarjeta*, *PantallaCrearRegTarjeta*, *PantallaObtenerRegTarjeta* y *RegistroTarjeta*.

A partir de la Tabla 8.15 se generan las colaboraciones para la clase *ManejadoRegistroTarjeta*, como se muestra en la Tabla 8.30. Revisando las diferentes asignaciones de responsabilidades podemos observar que la colaboración de “solicita *registrarTarjeta*” descrita en la Tabla 8.26 y en colaboración con el *ManejadorRegistroTarjeta* no está definida en este último. Por lo tanto podemos agregar ya en esta etapa una responsabilidad “registrarTarjeta” a la propia clase *ManejadorRegistroTarjeta*.

Clase: ManejadorRegistroTarjeta	
Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.	
Módulo: Registro.Tarjeta	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
registrarTarjeta	
crearRegistroTarjeta	
solicita *desplegarPantallaCrearRegTarjeta*	InterfaceUsuario
solicita *obtenerRegistroTarjeta*	InterfaceBaseDatosRegistro
solicita *desplegarPantallaObtenerRegTarjeta*	InterfaceUsuario
manejar el evento “Registrar”	
solicita *crearRegistroTarjeta*	InterfaceBaseDatosRegistro
manejar el evento “Actualizar”	

solicita *actualizarRegistroTarjeta*	InterfaceBaseDatosRegistro
manejar el evento "Eliminar"	
solicita *eliminarRegistroTarjeta*	InterfaceBaseDatosRegistro
manejar el evento "Servicios"	
solicita *ofrecerServicio*	ManejadorServicio
manejar el evento "Salir"	
sale del sistema	

Tabla 8.30. Tarjeta para la clase *ManejadoRegistroTarjeta* con responsabilidades y colaboraciones identificadas del caso de uso *RegistrarTarjeta*.

A partir de la Tabla 8.16 se generan las colaboraciones para la clase *PantallaCrearRegTarjeta*, como se muestra en la Tabla 8.31.

Clase: PantallaCrearRegTarjeta	
Descripción: Pantalla de solicitud de registro de tarjeta (*P*-5).	
Módulo: Registro.Tarjeta	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega	
envía el evento "Registrar"	InterfaceUsuario
envía el evento "Servicios"	InterfaceUsuario
envía el evento "Salir"	InterfaceUsuario

Tabla 8.31. Tarjeta para la clase *PantallaCrearRegTarjeta* con responsabilidades y colaboraciones identificadas del caso de uso *RegistrarTarjeta*.

A partir de la Tabla 8.17 se generan las colaboraciones para la clase *PantallaObtenerRegTarjeta*, como se muestra en la Tabla 8.32.

Clase: PantallaObtenerRegTarjeta	
Descripción: Pantalla de devolución con información de registro de tarjeta (*P*-6).	
Módulo: Registro.Tarjeta	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega	
envía el evento "Actualizar"	InterfaceUsuario
envía el evento "Eliminar"	InterfaceUsuario
envía el evento "Servicios"	InterfaceUsuario
envía el evento "Salir"	InterfaceUsuario

Tabla 8.32. Tarjeta para la clase *PantallaCrearRegTarjeta* con responsabilidades y colaboraciones identificadas del caso de uso *RegistrarTarjeta*.

A partir de la Tabla 8.18 se generan las colaboraciones para la clase *RegistroTarjeta*, como se muestra en la Tabla 8.33. Dado que *RegistroTarjeta* es una clase entidad, como en el caso de *RegistroUsuario*, aún no se agregan responsabilidades por lo cual no existen colaboraciones.

Clase: RegistroTarjeta
Descripción: Para poder hacer un pago con una tarjeta de crédito, se debe tener un registro de tarjeta. El registro contiene información acerca de la tarjeta incluyendo nombre, número, expedidor y vencimiento. La tarjeta está ligada a un registro de usuario.
Módulo: Registro.Tarjeta
Estereotipo: Entidad
Propiedades:
Superclases:

Subclases:	
Atributos:	

Tabla 8.33. Tarjeta para la clase *RegistroTarjeta* sin responsabilidades ni colaboraciones de registro para el caso de uso *RegistrarTarjeta*.

Interface Base Datos

A partir de la Tabla 8.19 se generan las colaboraciones para la clase *InterfaceBaseDatosRegistro*, como se muestra en la Tabla 8.34.

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceBD	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
solicita *crearRegistroUsuario*	BaseDatosRegistro
solicita *validarRegistroUsuario*	BaseDatosRegistro
solicita *obtenerRegistroUsuario*	BaseDatosRegistro
solicita *actualizar RegistroUsuario*	BaseDatosRegistro
solicita *eliminarRegistroUsuario*	BaseDatosRegistro
solicita *crearRegistroTarjeta*	BaseDatosRegistro
solicita *obtenerRegistroTarjeta*	BaseDatosRegistro
solicita *actualizarRegistroTarjeta*	BaseDatosRegistro
solicita *eliminarRegistroTarjeta*	BaseDatosRegistro

Tabla 8.34. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades y colaboraciones de escribir y leer información de registro de usuario y registro de tarjeta para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Servicios

A partir de la Tabla 8.20 se generan las colaboraciones para la clase *ManejadorServicio*, como se muestra en la Tabla 8.35.

Clase: ManejadorServicio	
Descripción: El manejador de servicios se encarga de enviar las peticiones particulares de servicios a los manejadores espacializados para consulta, reserva y compra.	
Módulo: Servicio	
Estereotipo: Control	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
ofrecerServicio	
solicita *desplegarPantallaServicio*	InterfaceUsuario
maneja el evento “Obtener Registro”	
solicita *registrar*	ManejadorRegistroUsuario
maneja el evento “Salir”	
sale del sistema	

Tabla 8.35. Tarjeta para la clase *ManejadorServicio* con responsabilidades y colaboraciones a partir de los casos de uso *RegistrarUsuario* y *RegistrarTarjeta*.

A partir de la Tabla 8.21 se generan las colaboraciones para la clase *PantallaServicio*, como se muestra en la Tabla 8.36.

Clase: PantallaServicio	
Descripción: Pantalla de servicios (*P*-2).	
Módulo: Servicio	
Estereotipo: Borde	
Propiedades:	
Superclases:	
Subclases:	
Atributos:	
despliega	
envía el evento "Obtener Registro"	InterfaceUsuario

Tabla 8.36. Tarjeta para la clase *PantallaServicio* con responsabilidades y colaboraciones a partir de los casos de uso *RegistrarUsuario* y *RegistrarTarjeta*.

Jerarquías

El diseño de las jerarquías de herencia es uno de los aspectos de programación más importantes de la orientación a objetos. Mediante la herencia se puede lograr una buena reutilización del código del sistema, logrando arquitecturas de clases más compactas lo cual puede reducir radicalmente el tamaño del sistema final.
La herencia se identifica a partir de las responsabilidades y colaboraciones obtenidas anteriormente. De manera general, existen dos formas de aprovechar la herencia. La forma más común es la creación de superclases que guarden responsabilidades comunes a múltiples clases. La forma adicional y más avanzada está relacionada con el *polimorfismo* y busca aprovechar no sólo responsabilidades comunes sino también colaboraciones comunes entre clases. Estos dos enfoques también sirven de base para la extensibilidad de la arquitectura de clases. Por ejemplo, si varias clases definen una responsabilidad similar, se puede introducir una superclase de la cual estas clases hereden la responsabilidad común. Las nuevas superclases típicamente son clases abstractas. Cuanto mayor sea el número de clases concretas que puedan extenderse a partir de una funcionalidad abstracta, mayor será la probabilidad de que la abstracción sobreviva las pruebas del tiempo y mejoras del software. Se necesita solo una responsabilidad para definir una superclase abstracta, pero se necesita por lo menos dos subclases antes de esperar diseñar una abstracción general útil. Si no se tiene o no se prevé por lo menos dos casos, no se debe perder tiempo en construir la abstracción. Probablemente no se pueda diseñar funcionalidad apropiada general sin varios ejemplos concretos que sirvan de guía.
En este etapa se debe revisar las tarjetas de clases y clasificar cada clase según si es concreta o abstracta. Se debe además agregar tarjetas para las superclases (abstractas o concretas) según sea necesario y reasignar responsabilidades para corresponder al nuevo diseño. Mediante la reasignación de responsabilidades se busca producir jerarquías de clases que puedan reutilizarse y extenderse mas fácilmente. Aunque llevaremos a cabo una sola etapa de diseño de jerarquías, la herencia debe aplicarse continuamente a lo largo del diseño de objetos. Se listan en la tarjeta de clase las responsabilidades propias no heredadas y aquellas responsabilidades que se sobrescriben. No es necesario volver a escribir las responsabilidades que se heredan de las superclases. Nótese en las siguientes secciones que estaremos agregando nuevas superclases, para lo cual agregaremos las correspondientes tarjetas de clases, y estaremos agregando información en las secciones de subclases y superclases, además de la especificación de si la clase es concreta o abstracta.
A continuación se describen las jerarquías de herencia para el Sistema de Reservaciones en base a los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

InterfaceUsuario

Para comenzar analizamos las diversas responsabilidades y colaboraciones asignadas a la clase *InterfaceUsuario* en la Tabla 8.23 las cuales se muestran nuevamente en la tabla 8.37.

despliega	PantallaPrincipal
envía el evento "OK"	ManejadorPrincipal
envía el evento "Salir"	ManejadorPrincipal
envía el evento "Registrarse por Primera Vez"	ManejadorPrincipal
despliega	PantallaServicio
envía el evento "Obtener Registro"	ManejadorServicio
despliega	PantallaCrearRegUsuario

envía el evento “Registrar”	ManejadorRegistroUsuario
envía el evento “Salir”	ManejadorRegistroUsuario
despliega	PantallaObtenerRegUsuario
envía el evento “Actualizar"	ManejadorRegistroUsuario
envía el evento “Eliminar”	ManejadorRegistroUsuario
envía el evento “Registrar Tarjeta”	ManejadorRegistroUsuario
envía el evento “Servicios”	ManejadorRegistroUsuario
despliega	PantallaCrearRegTarjeta
despliega	PantallaObtenerRegTarjeta
envía el evento “Registrar”	ManejadorRegistroTarjeta
envía el evento “Servicios”	ManejadorRegistroTarjeta
envía el evento “Salir”	ManejadorRegistroTarjeta
envía el evento “Actualizar”	ManejadorRegistroTarjeta
envía el evento “Eliminar”	ManejadorRegistroTarjeta

Tabla 8.37. Responsabilidades y colaboraciones para la clase *InterfaceUusario*.

Si analizamos estas responsabilidades y colaboraciones con más detalle podemos apreciar que hay dos grupos de responsabilidades, aquellos correspondientes a “despliega” y las correspondientes a “envía el evento...”, como se muestra de manera condensada en la Tabla 8.38. Es importante apreciar que estamos *generalizando* las diversas responsabilidades “envía el evento ...” en una común en donde el evento particular “OK”, “Registrar”, etc., son abstraidos de la responsabilidad.

despliega	PantallaPrincipal, PantallaServicio, PantallaCrearRegUsuario, PantallaObtenerRegUsuario, PantallaCrearRegTarjeta, PantallaObtenerRegTarjeta
envía el evento ...	ManejadorPrincipal, ManejadorServicio, ManejadorRegistroUsuario, ManejadorRegistroTarjeta

Tabla 8.38. Grupos de responsabilidades y colaboraciones para la clase *InterfaceUusario*.

En el caso de la responsabilidad “despliega” la responsabilidad se llama también “despliega” para todas las clases colaboradoras *PantallaPrincipal, PantallaServicio, PantallaCrearRegUsuario, PantallaObtenerRegUsuario, PantallaCrearRegTarjeta* y *PantallaObtenerRegTarjeta*, como se muestra en la Tabla 8.39. Esto es sumamente importante si deseamos aprovechar el polimorfismo.

despliega	

Tabla 8.39. Grupos de responsabilidades “despliega” para las distintas pantallas, correspondientes a las clases PantallaPrincipal, PantallaServicio, PantallaCrearRegUsuario, PantallaObtenerRegUsuario, PantallaCrearRegTarjeta y PantallaObtenerRegTarjeta.

En el caso de los manejadores vemos que *ManejadorPrincipal*, *ManejadorServicio*, *ManejadorRegistroUsuario* y *ManjeadorRegistroTarjeta* todos tienen una resposabilidad comun “manejarEvento” correspondiente a la responsabilidad inicial “envía el evento ...”, como se muestra en la Tabla 8.40.

manejarEvento	

Tabla 8.40. Grupos de responsabilidades “manejarEvento” para los distintos manejadores, correspondientes a las clases *ManejadorPrincipal, MnajeadorServicio, ManejadorRegistroUsuario* y *ManjeadorRegistroTarjeta*.

Estos dos grupos de responsabilidades se muestran en la Figura 8.3. Nótese que la dirección de la flecha representa la dirección de la llamada de servicio, o sea en dirección de la clase colaboradora. Para el nombre de la asociación utilizamos la responsabilidad definida en la clase colaboradora correspondiente.

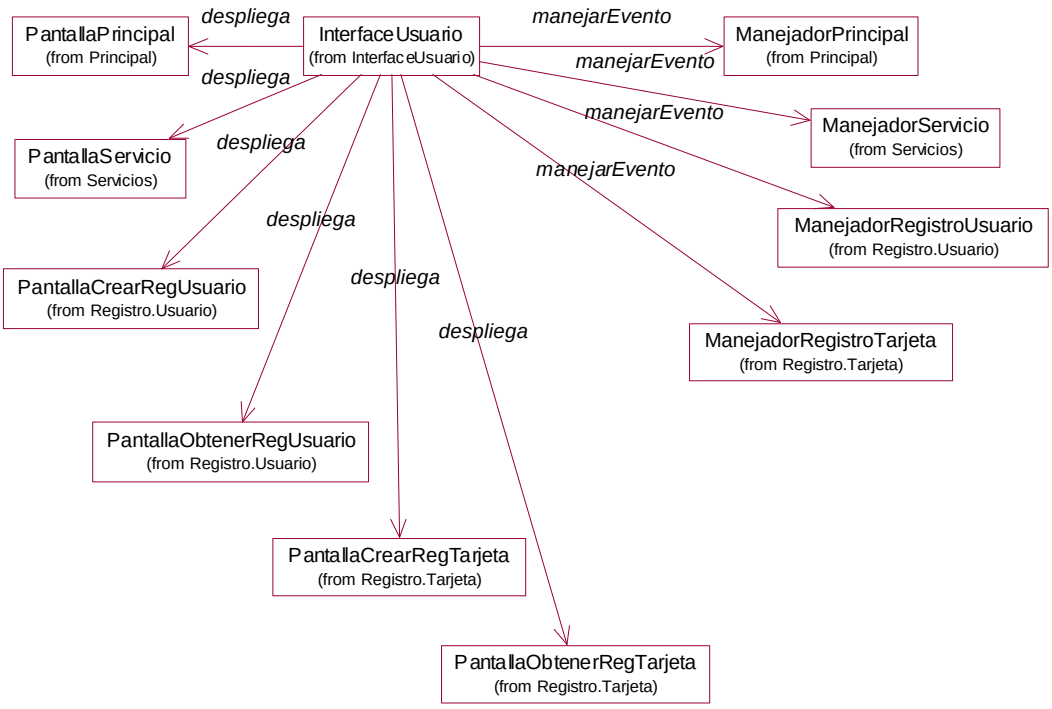

Figura 8.3 El diagrama muestra las colaboraciones descritas hasta el momento para la clase *InterfaceUsuario*.

En este momento nos preguntamos cómo podemos simplificar estas responsabilidades y colaboraciones. Analicemos las dos situaciones. En el caso de "despliega", las colaboraciones están relacionadas con diferentes pantallas, mientras que en el caso de "envía el evento ...", la colaboraciones están relacionadas con diferentes manejadores. Esta es una típica situación de polimorfismo, donde una misma responsabilidad es aprovechada por diversas clases colaboradoras que funcionalmente son similares. La solución es crear una nueva superclase *Pantalla* y otra *Manejador* de manera que las relaciones anteriores descritas en la Tablas 8.38 se conviertan según se describen en la Tabla 8.41. Nótese que aunque la relación de colaboración se simplifica, aún incluimos la lista de clases colaboradoras para no perder esta información a nivel descriptiva. Esto se denota utilizando la notación de "*superclase*" seguida por ":" y finalmente por la lista de "*clases colaboradoras*". Sin embargo, el objetivo de diseño es que la *InterfaceUsuario* deja de conocer explícitamente a todas las clases colaboradoras y que únicamente conozca a la clase general que luego será sobrecargada.

despliega	Pantalla : PantallaPrincipal, PantallaServicio, PantallaCrearRegUsuario, PantallaObtenerRegUsuario, PantallaCrearRegTarjeta, PantallaObtenerRegTarjeta
envía el evento ...	Manejador : ManejadorPrincipal, ManejadorServicio, ManejadorRegistroUsuario, ManejadorRegistroTarjeta

Tabla 8.41. Grupos de responsabilidades y colaboraciones para la clase *InterfaceUusario* revisados según la creación de dos nuevas superclases: *Pantalla* y *Manejador*.

En la Figura 8.4 se muestra la nueva jerarquía de herencia.

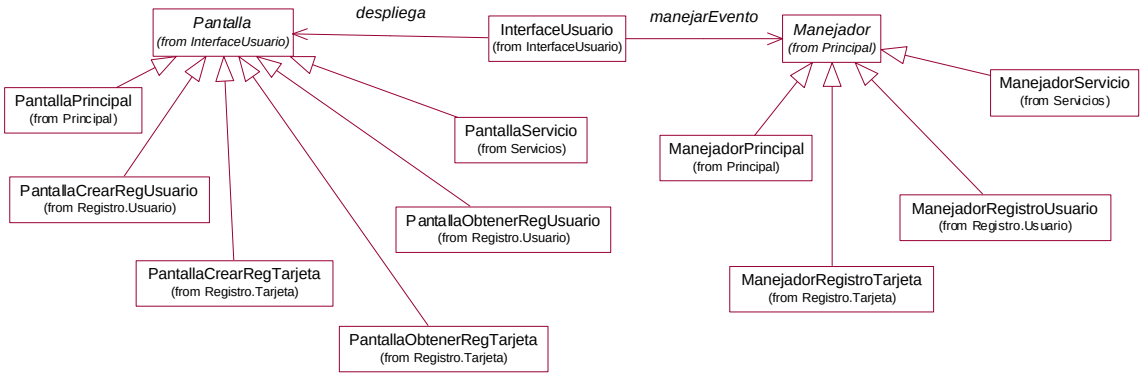

Figura 8.4 El diagrama muestra las colaboraciones descritas hasta el momento para la clase *InterfaceUsuario* luego de la introducción de las superclases *Pantalla* y *Manejador*.

Es importante resaltar que se tendrá que agregar dos nuevas tarjetas de clases correspondientes a las nuevas clases *Pantalla* y *Manejador* recién introducidas. Dichas tarjetas deberán incluir responsabilidades correspondientes a "despliega" y "manejarEvento" que serán sobrescritas por las diversas pantallas y manejadores en la jerarquía de herencia.

La tarjeta de clase modificada para la clase *InterfaceUsuario* se muestra en la Tabla 8.42. Como parte del proceso de afinación de nombres cambiamos "despliega" por "desplegarPantalla", el cual es más descriptivo, y "envía el evento ..." por "enviarEvento" lo cual es más compacto. De tal manera, se reducen de manera radical el número de responsabilidades y colaboraciones de la clase *InterfaceUsuario*. Nótese como los diversos "envía el evento ..." son abstraídos o generalizados por una sola responsabilidad más genérica llamada *enviarEvento*. Vale la pena resaltar la gran reducción en el número de responsabilidades y colaboraciones definidas para la clase *InterfaceUsuario*.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos:	
desplegarPantalla	Pantalla : PantallaPrincipal, PantallaServicio, PantallaCrearRegUsuario, PantallaObtenerRegUsuario, PantallaCrearRegTarjeta, PantallaObtenerRegTarjeta
enviarEvento	Manejador : ManejadorPrincipal, ManejadorServicio, ManejadorRegistroUsuario, ManejadorRegistroTarjeta

Tabla 8.42 Tarjeta para la clase *InterfaceUusario* con responsabilidades, colaboraciones y jerarquías identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Antes de especificar la tarjeta de clase para la nueva superclase *Pantalla* veamos otra fuente de generalización a partir de las diversas pantallas. En la Tabla 8.43 se muestra de manera compacta las responsabilidades y colaboraciones para las diversas pantallas descritas en la sección de colaboraciones. Las diversas pantallas contienen una responsabilidad "despliega" y otra "envía el evento ...", la cual colabora con *InterfaceUsuario*. Nuevamente abstraemos las diversas responsabilidades "enviar el evento ..." en una sola.

despliega	
envía el evento ...	InterfaceUsuario

Tabla 8.43. Grupos de responsabilidades y colaboraciones para las diversas pantallas.

Estas responsabilidades con sus colaboraciones se muestran en la Figura 8.5. Nótese que la dirección de la flecha va ahora en dirección de la *InterfaceUsuario*. Para el nombre de la asociación utilizamos la responsabilidad reescrita "enviarEvento" definida en la clase *InterfaceUsuario*, la clase colaboradora.

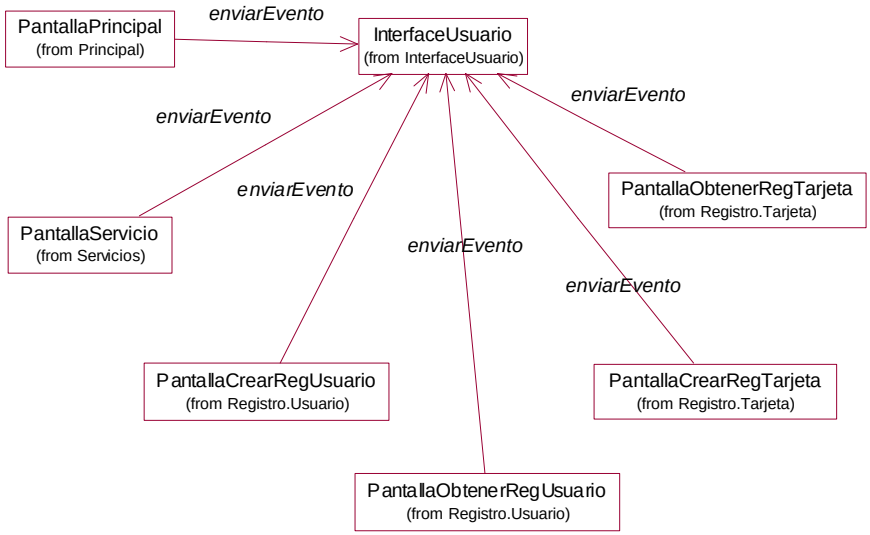

Figura 8.5 El diagrama muestra las colaboraciones descritas a partir de las diversas pantallas y en dirección a la *InterfaceUsuario*.

Esta es nuevamente una fuente de polimorfismo donde podemos aprovechar la clase *Pantalla* antes agregada. En la Figura 8.6 se muestra el diagrama de herencia correspondiente a la Figura 8.5 incluyendo la nueva superclase *Pantalla* en base al polimorfismo "enviarEvento" a partir de las diversas pantallas.

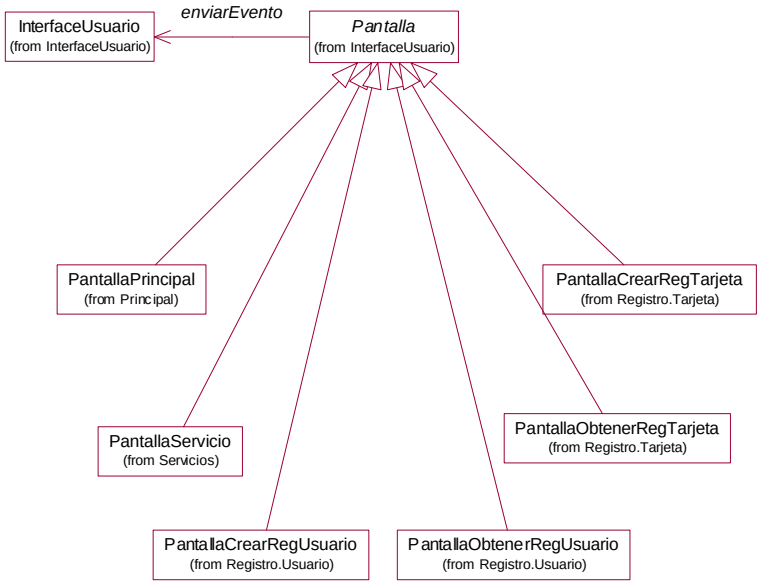

Figura 8.6 El diagrama muestra las colaboraciones descritas a partir de las diversas pantallas conteniendo la superclase *Pantalla* y en dirección a la *InterfaceUsuario*.

La responsabilidad "envía el evento ..." es rescrita como "enviarEvento" y junto con "desplegarPantalla" definen las responsabilidades para la clase *Pantalla*, como se muestra en la Tabla 8.44. La clase *Pantalla* es definida como *Abstracta* ya que es una *superclase*. Adicionalmente, en la sección de subclases se describen las diversas clases que heredan de ella.

Clase: Pantalla
Descripción: Pantalla heredada por las demás clases de tipo pantalla.
Módulo: InterfaceUsuario

Estereotipo: Borde	
Propiedades: Abstracta	
Superclases:	
Subclases: PantallaPrincipal, PantallaServicio, PantallaCrearRegUsuario, PantallaObtenerRegUsuario, PantallaCrearRegTarjeta, PantallaObtenerRegTarjeta	
Atributos:	
desplegarPantalla	
enviarEvento	InterfaceUsuario

Tabla 8.44. Tarjeta para la superclase clase *Pantalla* con responsabilidades, colaboraciones y jerarquías identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Antes de continuar con la superclase *Manejador* aprovecharemos para agregar dos nuevas superclases de tipo pantalla agregadas exclusivamente por razones de herencia y no polimorfismo. Estas clases son *PantallaRegUsuario* la cual define elementos comunes a las pantallas *PantallaCrearRegUsuario* y *PantallaObtenerRegUsuario*, y *PantallaRegTarjeta* la cual define elementos comunes a las pantallas *PantallaCrearRegTarjeta* y *PantallaObtenerRegTarjeta*. Los elementos comunes para estas pantallas son básicamente todos los campos de textos que se repiten entre ellas, difiriendo únicamente en los botones.

En la Figura 8.7 se muestra la jerarquía de herencia a partir de la clase *Pantalla* y conteniendo las clases *PantallaRegUsuario* y *PantallaRegTarjeta*.

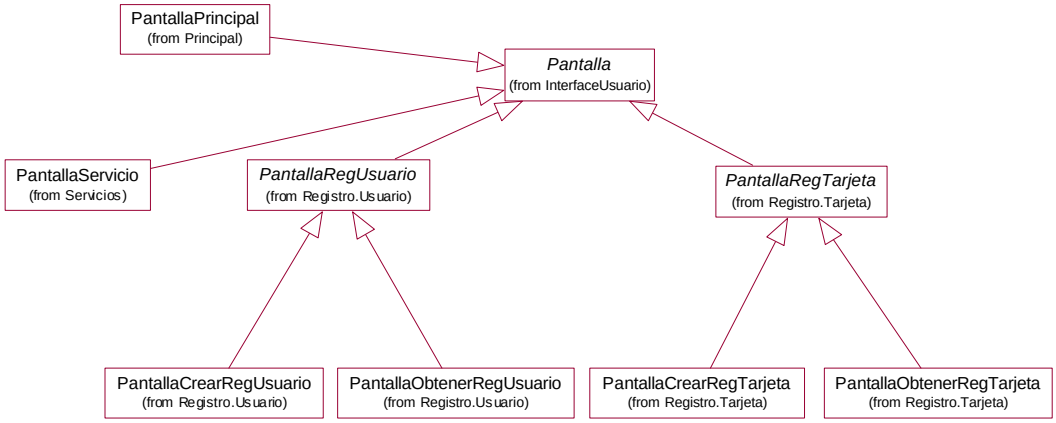

Figura 8.7 El diagrama muestra la jerarquía de clases para el módulo de registro a partir de la clase *Pantalla* incluyendo las clases *PantallaRegUsuario* y *PantallaRegTarjeta*.

En la Tabla 8.45 se describe la clase *Pantalla* redefinida de acuerdo a las modificaciones con las clases *PantallaRegUsuario* y *PantallaRegTarjeta* correspondiente a la Figura 8.7. El cambio se da únicamente en la sección de subclases.

Clase: Pantalla	
Descripción: Pantalla heredada por las demás clases de tipo pantalla.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Abstracta	
Superclases:	
Subclases: PantallaPrincipal, PantallaServicio, PantallaRegUsuario, PantallaRegTarjeta	
Atributos:	
desplegarPantalla	
enviarEvento	InterfaceUsuario

Tabla 8.45. Tarjeta para la superclase clase *Pantalla* con responsabilidades, colaboraciones y jerarquías revisadas.

Principal

A continuación llevaremos un proceso de generalización para la clase *Manejador* similar al proceso llevado a cabo para la clase *Pantalla*. En la Tabla 8.46 se muestra de manera compacta las responsabilidades y colaboraciones

comunes para los diversos manejadores descritos en la sección de colaboraciones. Los diversos manejadores contienen una responsabilidad "manejarEvento" que es sobrescrita por cada uno de ellos, una responsabilidad "solicita desplegarPantalla..." en colaboración con la *InterfaceUsuario* y que puede ser generalizada de manera similar a "envía el evento ...", una responsabilidad "solicita ofrecerServicio" en colaboración con ManejadorServicio que es común a los diversos manejadores y otra responsabilidad común "salir". Los diversos manejadores contienen otras responsabilidades pero estas ya no son comunes entre ellos.

manejarEvento	
solicita desplegarPantalla...	InterfaceUsuario
solicita ofrecerServicio	ManejadorServicio
salir	

Tabla 8.46. Grupos de responsabilidades y colaboraciones para los diversos manejadores.

La responsabilidad "solicita desplegarPantalla..." puede ser reescrita simplemente como "desplegarPantalla" mientras que "solicita ofrecerServicio" puede ser reescrita como "ofrecerServicio". Estas responsabilidades junto con sus colaboraciones correspondientes se muestran en la Figura 8.8. Nuevamente, la responsabilidad "desplegarPantalla" debe existir en la clase *InterfaceUsuario* y "ofrecerServicio" en la clase *ManejadorServicio* para que el diagrama sea correcto.

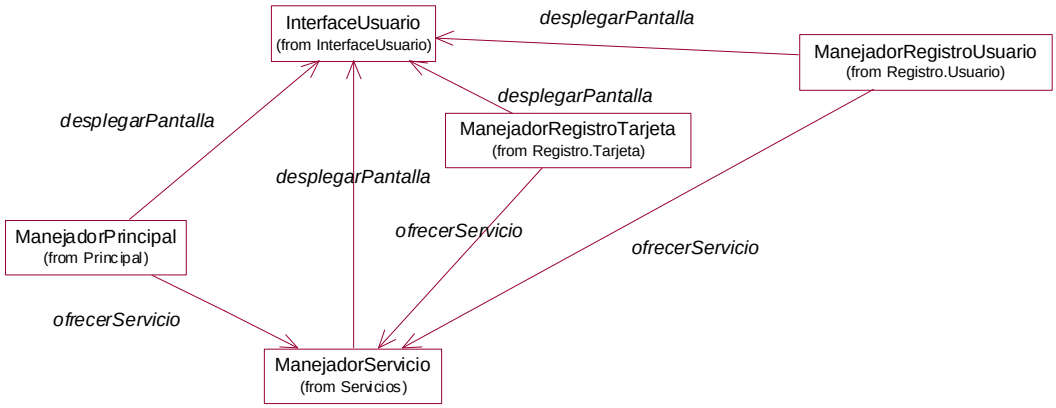

Figura 8.8 El diagrama muestra las colaboraciones descritas a partir de los diversos manejadores y en dirección a la *InterfaceUsuario* y *ManejadorServicio*.

Con la introducción de la clase *Manejador* se puede generalizar estas relaciones y aprovechar el polimorfismo correspondiente. En la Figura 8.9 se muestra el diagrama de herencia correspondiente a la Figura 8.8 con la inclusión de la nueva superclase *Manejador*.

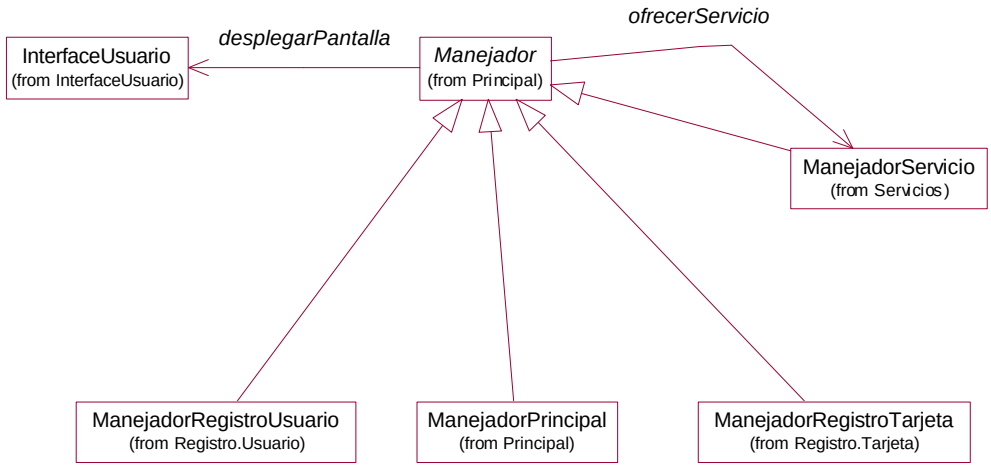

Figura 8.9 El diagrama muestra las colaboraciones descritas a partir de los diversos manejadores conteniendo la superclase *Manejador* y en dirección a la *InterfaceUsuario* y *ManejadorServicio*.

Las responsabilidades anteriores para los diversos manejadores son ahora descritos en la superclase *Manejador*, como se muestra en la Tabla 8.47. La clase *Manejador* es definida como *Abstracta* ya que es una *superclase*. En la sección de subclases se agregan las diversas clases que heredan de ella.

Clase: Manejador	
Descripción: Superclase heredada por todos los manejadores del sistema.	
Módulo: Principal	
Estereotipo: Control	
Propiedades: Abstracta	
Superclases:	
Subclases: ManejadorPrincipal, ManejadorServicio, ManejadorRegistroUsuario, ManejadorRegistroTarjeta	
Atributos:	
manejarEvento	
desplegarPantalla	InterfaceUsuario
ofrecerServicio	ManejadorServicio
salir	

Tabla 8.47. Tarjeta para la clase *Manejador* con responsabilidades, colaboraciones y jerarquías identificadas de los diversos manejadores para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

En la Figura 8.10 se muestra la jerarquía de herencia a partir de la superclase *Manejador*.

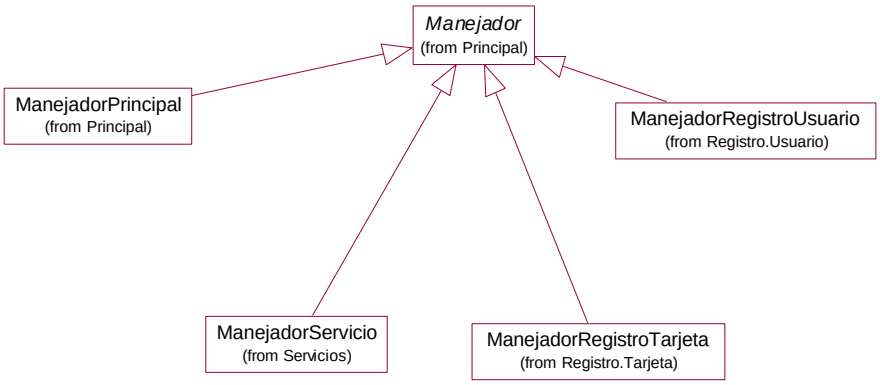

Figura 8.10 El diagrama muestra la jerarquía de clases a partir de la clase *Manejador*.

Ahora veamos como se describen las demás clases a partir de estas modificaciones de herencia. A partir de la Tabla 8.24 se generan las jerarquías para la clase *ManejadorPrincipal*, como se muestra en la Tabla 8.48. La clase se

define como concreta y se especifica su superclase. La única responsabilidad sobrescrita de las definidas en la superclase *Manejador* es "manejarEvento" ya que el polimorfismo va en dirección de la clase *InterfaceUsuario* a los diversos manejadores. Las demás responsabilidades descritas en *Manejador* son únicamente heredadas por los diversos manejadores. Adicionalmente la clase *ManejadorPrincipal* definía las responsabilidades "solicita *crearRegistroUsuario*" y "solicita *validarRegistroUsuario*" que son reescritas como "crearRegistroUsuario" y "validarRegistroUsuario", respectivamente.

Clase: ManejadorPrincipal	
Descripción: El manejador principal es el encargado de desplegar la pantalla principal de interacción con el usuario, y luego delegar las diferentes funciones a los manejadores especializados apropiados.	
Módulo: Principal	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
manejarEvento	
crearRegistroUsuario	ManejadorRegistroUsuario
validarRegistroUsuario	ManejadorRegistroUsuario

Tabla 8.48. Tarjeta para la clase *ManejadorPrincipal* con responsabilidades, colaboraciones y jerarquías identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

En el caso de las pantallas, todas las responsabilidades se especifican en la superclase *Pantalla* por lo cual ya no hay necesidad de incluir las responsabilidades descritas anteriormente. Por lo tanto, la clase *PantallaPrincipal* descrita anteriormente en la Tabla 8.25 se describirá de acuerdo a las modificaciones en las jerarquías de herencia, como se muestra en la Tabla 8.49. Se eliminan las responsabilidades y se especifica la clase como concreta y se agrega su superclase.

Clase: PantallaPrincipal	
Descripción: Pantalla principal (*P*-1).	
Módulo: Principal	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases: Pantalla	
Subclases:	
Atributos:	

Tabla 8.49. Tarjeta para la clase *PantallaPrincipal* con responsabilidades, colaboraciones y jerarquías identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Dominio

Si consideramos que hemos agregado una nueva superclase para las diversas pantallas correspondientes a las clases *borde* al igual para los diversos manejadores correspondientes a las clases de *control*, resulta que sería también una buena idea agregar una superclase para las diversas clases *entidad*. Estos no es obvio en este momento ya que nuestras clases entidad, *RegistroUsuario* y *RegistroTarjeta*, no tienen responsabilidades asignadas aún. Sin embargo, agregar una superclase a grupos de clases con estereotipo común es siempre una buena idea. En la Figura 8.11 se muestra la jerarquía de herencia a partir de una superclase general llamada *Datos*.

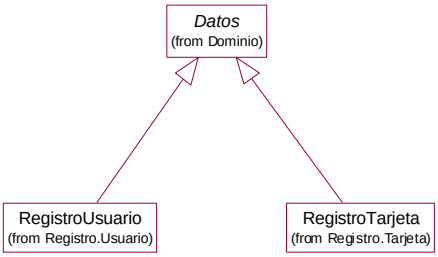

Figura 8.11 El diagrama muestra la jerarquía de clases para el módulo de registro a partir de la clase *Datos*.

La superclase *Datos* que generaliza a las diferentes clases entidad, en este caso *RegistroUsuario* y *RegistroTarjeta*, se muestra en la Tabla 8.50.

Clase: Datos	
Descripción: Superclase para todas las clases entidad.	
Módulo: Dominio	
Estereotipo: Entidad	
Propiedades: Abstracta	
Superclases:	
Subclases: RegistroUsuario, RegistroTarjeta	
Atributos:	

Tabla 8.50. Tarjeta para la clase *Datos* con responsabilidades, colaboraciones y jerarquías identificadas de las diversas clases entidad para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Registro

Este módulo se compone de los módulos de *Usuario*, *Tarjeta* e *InterfaceBD*.

Usuario

Esta sección involucra las clases de registro de usuario que son *ManejadoRegistroUsuario*, *PantallaCrearRegUsuario*, *PantallaObtenerRegUsuario* y *RegistroUsuario*.

Correspondiente a la Tabla 8.26 se describe la clase *ManejadoRegistroUsuario*, como se muestra en la Tabla 8.51. Con excepción de "manejarEvento" sobrescrita por todos los manejadores, las responsabilidades "desplegarPantalla", "ofrecerServicio" y "salir" son descritas únicamente en la superclase *Manejador*. Las demás responsabilidades descritas en la Tabla 8.26 son reescritas únicamente eliminando la palabra "solicita" para volver las responsabilidades más compactas.

Clase: ManejadorRegistroUsuario	
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para poder utilizar el sistema.	
Módulo: Registro.Usuario	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
manejarEvento	
validarRegistroUsuario	InterfaceBaseDatosRegistro
crearRegistroUsuario	InterfaceBaseDatosRegistro
obtenerRegistroUsuario	InterfaceBaseDatosRegistro
actualizarRegistroUsuario	InterfaceBaseDatosRegistro
eliminarRegistroUsuario	InterfaceBaseDatosRegistro
registrarTarjeta	ManejadorRegistroTarjeta

Tabla 8.51. Tarjeta para la clase *ManejadoRegistroUsuario* con responsabilidades, colaboraciones y jerarquías identificadas de los casos de uso *RegistrarUsuario* y *ValidarUsuario*.

Se agrega la superclase *PantallaRegUsuario* antes mencionada, la cual se muestra en la Tabla 8.52.

Clase: PantallaRegUsuario	
Descripción: Superclase con diseño gráfico común para las pantallas de registro de usuario.	
Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades: Abstracta	
Superclases: Pantalla	
Subclases: PantallaCrearRegUsuario, PantallaObtenerRegUsuario	
Atributos:	

Tabla 8.52. Tarjeta para la clase *PantallaRegUsuario* con la jerarquía identificadas de las pantallas de registro de usuario para el caso de uso *RegistrarUsuario*.

A partir de la Tabla 8.27 se generan las jerarquías para la clase *PantallaCrearRegUsuario*, como se muestra en la Tabla 8.53. De manera similar a la clase *PantallaPrincipal*, se eliminaron las responsabilidades en esta clase al ser descritas en la superclase *Pantalla*.

Clase: PantallaCrearRegUsuario	
Descripción: Pantalla de solicitud de registro de usuario (*P*-3).	
Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases: PantallaRegUsuario	
Subclases:	
Atributos:	

Tabla 8.53. Tarjeta para la clase *PantallaCrearRegUsuario* con responsabilidades, colaboraciones y jerarquías identificadas del caso de uso *RegistrarUsuario*.

A partir de la Tabla 8.28 se generan las jerarquías para la clase *PantallaObtenerRegUsuario*, como se muestra en la Tabla 8.54.

Clase: PantallaObtenerRegUsuario	
Descripción: Pantalla de devolución con información de registro de usuario (*P*-4).	
Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases: PantallaRegUsuario	
Subclases:	
Atributos:	

Tabla 8.54. Tarjeta para la clase *PantallaCrearRegUsuario* con responsabilidades, colaboraciones y jerarquías identificadas del caso de uso *RegistrarUsuario*.

A partir de la Tabla 8.29 se generan las jerarquías para la clase *RegistroUsuario*, como se muestra en la Tabla 8.55. Nótese que *RegistroUsuario* es una subclase de la recién introducida clase *Datos*.

Clase: RegistroUsuario
Descripción: Para poder utilizar el sistema de reservaciones, el usuario debe estar registrado con el sistema. El registro contiene información acerca del usuario que incluye nombre, dirección, colonia, ciudad, país, código postal, teléfono de casa, teléfono de oficina, fax, email, login y password.
Módulo: Registro.Usuario
Estereotipo: Entidad
Propiedades: Concreta
Superclases: Datos
Subclases:

Atributos:	

Tabla 8.55. Tarjeta para la clase *RegistroUsuario* con responsabilidades, colaboraciones y jerarquías de actualizar y consultar información de registro para el caso de uso *RegistrarUsuario*.

Tarjeta

Esta sección involucra las clases de registro de tarjeta que son *ManejadoRegistroTarjeta*, *PantallaCrearRegTarjeta*, *PantallaObtenerRegTarjeta* y *RegistroTarjeta*.
De manera similar a los manejadores anteriores, se describe a partir de la Tabla 8.30 la clase *ManejadoRegistroTarjeta*, como se muestra en la Tabla 8.56. Se sobrescribe la responsabilidad "manejarEvento" y se eliminan de esta tarjeta las responsabilidades descritas en la superclase *Pantalla*, o sea, "desplegarPantalla", "ofrecerServicio" y "Salir". Se mantienen las demás responsabilidades reescribiéndolas sin la palabra "solicita".

Clase: ManejadorRegistroTarjeta	
Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.	
Módulo: Registro.Tarjeta	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
manejarEvento	
registrarTarjeta	
crearRegistroTarjeta	InterfaceBaseDatosRegistro
obtenerRegistroTarjeta	InterfaceBaseDatosRegistro
actualizarRegistroTarjeta	InterfaceBaseDatosRegistro
eliminarRegistroTarjeta	InterfaceBaseDatosRegistro

Tabla 8.56. Tarjeta para la clase *ManejadoRegistroTarjeta* con responsabilidades, colaboraciones y jerarquías identificadas del caso de uso *RegistrarTarjeta*.

Se agrega la superclase *PantallaRegTarjeta* antes mencionada, la cual se muestra en la Tabla 8.57.

Clase: PantallaRegTarjeta	
Descripción: Superclase con diseño gráfico común para las pantallas de registro de tarjeta.	
Módulo: Registro. Tarjeta	
Estereotipo: Borde	
Propiedades: Abstracta	
Superclases: Pantalla	
Subclases: PantallaCrearRegTarjeta, PantallaObtenerRegTarjeta	
Atributos:	

Tabla 8.57. Tarjeta para la clase *PantallaRegTarjeta* con la jerarquía identificadas de las pantallas de registro de tarjeta para el caso de uso *RegistrarTarjeta*.

A partir de la Tabla 8.30 se generan las jerarquías para la clase *PantallaCrearRegTarjeta*, como se muestra en la Tabla 8.58.

Clase: PantallaCrearRegTarjeta
Descripción: Pantalla de solicitud de registro de tarjeta (*P*-5).
Módulo: Registro.Tarjeta
Estereotipo: Borde
Propiedades: Concreta
Superclases: PantallaRegTarjeta
Subclases:
Atributos:

Tabla 8.58. Tarjeta para la clase *PantallaCrearRegTarjeta* con responsabilidades, colaboraciones y jerarquías identificadas del caso de uso *RegistrarTarjeta*.

A partir de la Tabla 8.32 se generan las jerarquías para la clase *PantallaObtenerRegTarjeta*, como se muestra en la Tabla 8.59.

Clase: PantallaObtenerRegTarjeta	
Descripción: Pantalla de devolución con información de registro de tarjeta (*P*-6).	
Módulo: Registro.Tarjeta	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases: PantallaRegTarjeta	
Subclases:	
Atributos:	

Tabla 8.59. Tarjeta para la clase *PantallaCrearRegTarjeta* con responsabilidades, colaboraciones y jerarquías identificadas del caso de uso *RegistrarTarjeta*.

A partir de la Tabla 8.33 se generan las jerarquías para la clase *RegistroTarjeta*, como se muestra en la Tabla 8.60. Nótese que *RegistroTarjeta* es una subclase de la recién introducida clase *Datos*.

Clase: RegistroTarjeta	
Descripción: Para poder hacer un pago con una tarjeta de crédito, se debe tener un registro de tarjeta. El registro contiene información acerca de la tarjeta incluyendo nombre, número, expedidor y vencimiento. La tarjeta está ligada a un registro de usuario.	
Módulo: Registro.Tarjeta	
Propiedades: Concreta	
Estereotipo: Entidad	
Superclases: Datos	
Subclases:	
Atributos:	

Tabla 8.60. Tarjeta para la clase *RegistroTarjeta* con responsabilidades, colaboraciones y jerarquías de actualizar y consultar información de registro para el caso de uso *RegistrarTarjeta*.

Interface Base Datos

A partir de la Tabla 8.34 se generan las jerarquías para la clase *InterfaceBaseDatosRegistro*, como se muestra en la Tabla 8.61. Se reescriben las responsabilidades eliminando la palabra “solicita”.

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceDB	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos:	
validarRegistroUsuario	BaseDatosRegistro
obtenerRegistroUsuario	BaseDatosRegistro
crearRegistroUsuario	BaseDatosRegistro
actualizarRegistroUsuario	BaseDatosRegistro
eliminarRegistroUsuario	BaseDatosRegistro

obtenerRegistroTarjeta	BaseDatosRegistro
crearRegistroTarjeta	BaseDatosRegistro
actualizarRegistroTarjeta	BaseDatosRegistro
eliminarRegistroTarjeta	BaseDatosRegistro

Tabla 8.61. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades, colaboraciones y jerarquías de escribir y leer información de registro de usuario y registro de tarjeta para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Servicios

A partir de la Tabla 8.35 se generan las jerarquías para la clase *ManejadorServicio*, como se muestra en la Tabla 8.62. Los cambios son similares a los demás manejadores.

Clase: ManejadorServicio	
Descripción: El manejador de servicios se encarga de enviar las peticiones particulares de servicios a los manejadores espacializados para consulta, reserva y compra.	
Módulo: Servicios	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
manejarEvento	
ofrecerServicio	
registrar	ManejadorRegistroUsuario

Tabla 8.62. Tarjeta para la clase *ManejadorServicio* con responsabilidades, colaboraciones y jerarquías a partir de los casos de uso *RegistrarUsuario* y *RegistrarTarjeta*.

A partir de la Tabla 8.36 se generan las jerarquías para la clase *PantallaServicio*, como se muestra en la Tabla 8.63.

Clase: PantallaServicio	
Descripción: Pantalla de servicios (*P*-2).	
Módulo: Servicios	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases: Pantalla	
Subclases:	
Atributos:	

Tabla 8.63. Tarjeta para la clase *PantallaServicio* con responsabilidades, colaboraciones y jerarquías a partir de los casos de uso *RegistrarUsuario* y *RegistrarTarjeta*.

Contratos

Un *contrato* es una mecanismo de diseño para agrupar las distintas responsabilidades de una clase que están relacionadas lógicamente entre si. Los contratos sirven como indicadores de los diversos servicios provistos por cada clase. El objetivo final del contrato es ser un elemento de abstracción adicional en el manejo de la complejidad del sistema, dado que la funcionalidad completa del sistema dada a bajo nivel por las responsabilidades, puede ser vista a alto nivel como un grupo de servicios o contratos.

El contrato no es simplemente otro nombre para la responsabilidad, ya que la responsabilidad corresponde a una acción específica, mientras que un contrato define un conjunto de responsabilidades cercanas una de la otra.

Cada responsabilidad puede ser parte de un sólo contrato, aunque no tiene que ser necesariamente parte de algún contrato. Esto ocurre cuando las responsabilidades representan comportamiento que una clase debe tener, pero que son privadas a los propios objetos. En general, una clase puede apoyar uno o más contratos, aunque a menudo una clase con varias responsabilidades apoya un sólo contrato.

Un contrato entre dos clases representa una lista de servicios que una instancia de una clase puede solicitar de una instancia de otra clase. Todos los servicios especificados en un contrato particular son la responsabilidad del servidor para ese contrato. Las responsabilidades correspondientes al contrato deben ser ofrecidas públicamente. Por

lo tanto, para un mejor manejo de la complejidad del sistema, se debe posponer la definición de los aspecto privados de una clase y concentrarse inicialmente en las responsabilidades públicas.
En general, un contrato entre un cliente y un servidor no especifica cómo se hacen las cosas, solo qué se hace. Los contratos especifican quien colabora con quien, y qué se espera de la colaboración. El contrato cliente-servidor divide los objetos en dos categoría aquellos que proveen servicios (servidores), y aquellos que piden servicios (clientes). De tal manera, un contrato es una lista de pedidos que le hace un cliente a un servidor. Ambos deben satisfacer el contrato, el cliente hace la solicitud y el servidor respondiendo de manera correspondiente.
Como se mencionó anteriormente, cliente y servidor son roles que juegan los objetos en cierto momento y no características inherentes de los propios objetos. Un objeto puede tomar el rol de cliente o servidor en relación a otros objetos. Por ejemplo, los *componentes* casi siempre juegan el rol de servidores, ya que satisfacen una responsabilidad específica. Raramente los componentes necesitaran pedir servicios del propio sistema para poder satisfacer esas responsabilidades. Por el contrario, los *marcos* (*frameworks*) generalmente toman ambos roles, ya que se integran a las aplicaciones para solicitar y proveer funcionalidad.
Se puede determinar qué responsabilidades pertenecen a cuales contratos siguiendo ciertos criterios. Una forma de encontrar responsabilidades relacionadas es buscar responsabilidades que serán usadas por el mismo cliente. Aunque es posible que algunas clases den servicio a diferentes clientes mediante diversas responsabilidades, es más significativo cuando dos o más responsabilidades de una clase dan servicio a los mismos clientes. En tal caso sería innecesario definir distintos contratos. En su lugar, se puede abstraer de las responsabilidades específicas un sólo contrato, satisfaciendo la responsabilidad general.
Si una clase define un contrato que tiene relativamente poco en común con el resto de los contratos definidos por esa clase, este contrato debe ser movido a una clase diferente, usualmente una superclase o subclase. De tal manera, se puede refinar las jerarquías de clases maximizando la cohesión de contratos para cada clase, lo cual minimizará el número de contratos apoyados por cada clase. Esto es algo deseable, ya que cuanto menos contratos existan, más fácil será comprender el sistema. En general, la mejor manera de reducir el número de contratos es buscar responsabilidades similares que puedan generalizarse. Esto también resultará en jerarquías de clases más extensibles.
Un buen diseño hace un balance entre clases pequeñas con pocos contratos, fáciles de comprender y reutilizar, junto a un número reducido de clases más complejas cuyas relaciones entre ellas se puede comprender mas fácilmente.
Una técnica para definir contratos es comenzar definiendo primero los contratos de las clases más arriba en las jerarquías. Posteriormente, se definen nuevos contratos para las subclases que agregan nueva funcionalidad. Se debe examinar las responsabilidades para cada subclase y determinar si estas representan nueva funcionalidad o si son simplemente formas específicas de expresar responsabilidades heredadas, en cuyo caso serían parte del contrato heredado.
En cada tarjeta de clase se divide la sección de responsabilidades en dos. En la parte superior se especifica una sección para los contratos, mientras que en la parte inferior se especifica una sección para las responsabilidades privadas. Cada contrato debe incluir un nombre y un número de contrato. Se debe listar cada contrato para el cual esta clase es un servidor. También se debe listar contratos heredados, e indicar la clase de la cual se heredan, aunque no hay necesidad de repetir los detalles del contrato. Para cada contrato, se debe listar las responsabilidades de la clase en la cual se basan, asignando cada responsabilidad al contrato correspondiente.
Si la responsabilidad requiere colaborar con una clase que define varios contratos, se debe indicar el número del contrato correspondiente entre paréntesis en la columna de la colaboración, para así simplificar el seguimiento de qué responsabilidad se relaciona con qué contrato.
En esta sección se describen los contratos para el Sistema de Reservaciones, en base a los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

InterfaceUsuario

Consideremos las dos responsabilidades “desplegarPantalla” y “enviarEvento” asignadas a la clase *InterfaceUsuario* en la sección anterior de jerarquías las cuales se muestran en la Tabla 8.64.

desplegarPantalla	Pantalla : PantallaPrincipal, PantallaServicio, PantallaCrearRegUsuario, PantallaObtenerRegUsuario, PantallaCrearRegTarjeta, PantallaObtenerRegTarjeta
enviarEvento	Manejador : ManejadorPrincipal, ManejadorServicio, ManejadorRegistroUsuario, ManejadorRegistroTarjeta

Tabla 8.64 Responsabilidades asignadas a la clase *InterfaceUusario* luego de la etapa de jerarquías.

Si investigamos con mayor detalle estas responsabilidades podemos ver que “desplegarPantalla” es llamada por los diversos manejadores mientras que “enviarEvento” es llamada por las diversas pantallas, algo que se muestra en la Figura 8.12. Nótese que las colaboraciones que aparecen en la Tabla 8.64 se dan más adelante en la colaboración,

mientras que aquí mostramos las clases que solicitan servicio a las responsabilidades descritas en la clase *InterfaceUsuario*.

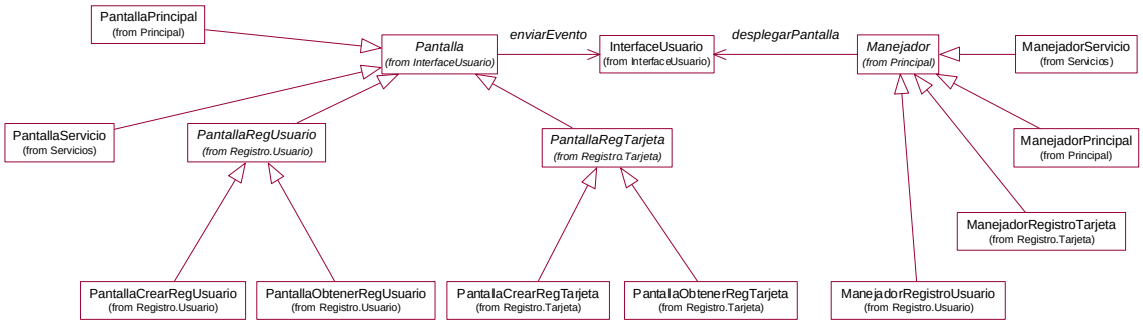

Figura 8.12 El diagrama muestra a las diversas clases manejadores y pantallas solicitando servicios de la clase *InterfaceUsuario* a través de "desplegarPantalla" y "enviarEvento", respectivamente..

Estos dos grupos de relaciones realmente están definiendo dos contratos teniendo como servidor a la clase *InterfaceUsuario*. El primer contrato lo llamaremos "desplegarPantalla", al igual que la responsabilidad correspondiente, teniendo como cliente a los diversos manejadores y como servidor a la *InterfaceUsuario*. Lo identificaremos como el contrato número "1" de la clase *InterfaceUsuario*. El segundo contrato lo llamaremos "enviarEvento", al igual que la responsabilidad correspondiente, teniendo como cliente a las diversas pantallas y como servidor a la clase *InterfaceUsuario*. Lo identificaremos como el contrato número "2" de la clase *InterfaceUsuario*. Históricamente se describían tarjetas de contratos describiendo cada uno de los contratos en término de los clientes y servidor involucrado. Nosotros omitiremos esta descripción ya que la información está implícita aunque distribuida en la diversas tarjetas. En general, toda documentación adicional tiene un precio no sólo en su generación sino más importante en su mantenimiento. En este caso decidimos reducir esta documentación adicional. En otras situaciones donde lo documentación aún no exista, será necesario agregarla. De tal manera y a partir de la Tabla 8.42 se generan los contratos para la clase *InterfaceUsuario*, como se muestra en la Tabla 8.65. Se asignan nuevas entradas por contratos mientras que las responsabilidades se mantienen igual únicamente asignándolas a los diferentes contratos identificados. Del lado derecho se mantienen las mismas colaboraciones originales para cada contrato. Los nombres de los contratos pueden ser distintos al de las responsabilidades y en general deben ser más descriptivos. Para resaltar el hecho de que son descripciones generales les asignaremos el nombre "Desplegar Pantalla" al primer contrato y "Enviar Evento" al segundo.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos:	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla	Pantalla : PantallaPrincipal, PantallaServicio, PantallaCrearRegUsuario, PantallaObtenerRegUsuario, PantallaCrearRegTarjeta, PantallaObtenerRegTarjeta
2. Envíar Evento	
envíarEvento	Manejador : ManejadorPrincipal, ManejadorServicio, ManejadorRegistroUsuario, ManejadorRegistroTarjeta

Tabla 8.65. Tarjeta para la clase *InterfaceUusario* con responsabilidades, colaboraciones, jerarquías y contratos identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

De manera similar podemos identificar dos grupos de responsabilidades lógicamente separadas para las diversas pantallas, "desplegarPantalla" y "enviarEvento", ambas definidas en la superclase *Pantalla*, como se muestra en la Tabla 8.66 a partir de las responsabilidades definidas para la clase *Pantalla* en la Tabla 8.44. Dio la casualidad que

dichas responsabilidades corresponden a los mismos nombres en la clase *InterfaceUsuario*, sin embargo representan responsabilidades separados.

desplegarPantalla	
enviarEvento	InterfaceUsuario

Tabla 8.66. Responsabilidades para la superclase clase *Pantalla* definidas en la Tabla 8.44 en la sección de jerarquías.

Si analizamos estas responsabilidades podemos observar que “desplegarPantalla” tiene como cliente a la *InterfaceUsuario*. Sin embargo “enviarEvento” no tiene ningún cliente identificado por lo cual la convertiremos en una responsabilidad privada. Nótese que esta responsabilidad colabora con los diversos manejadores a través de la superclase *Manejador*. En general, tanto las responsabilidades privadas como las públicas, a través de sus contratos, pueden tener ambas colaboradores. En la Figura 8.13. se muestra la clase *InterfaceUsuario* que solicita servicio a las responsabilidades de las diversas pantallas a través de las responsabilidades descritas en la clase *Pantalla*.

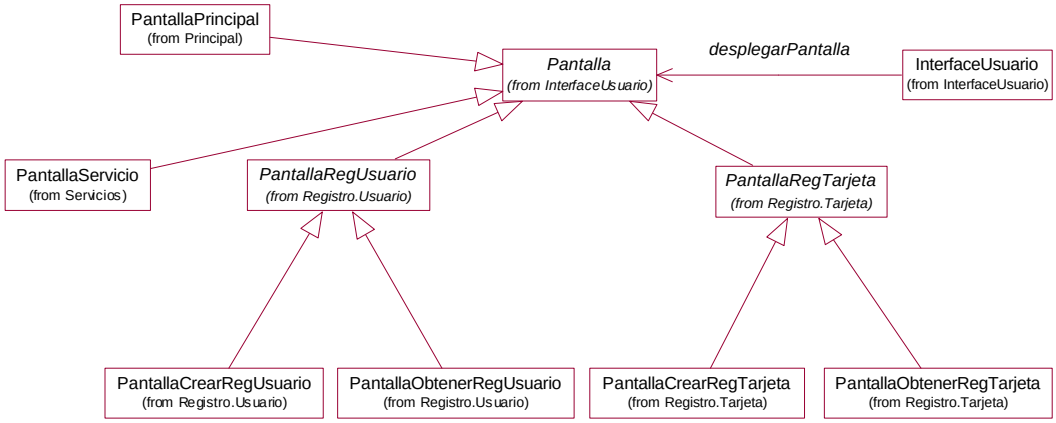

Figura 8.13 El diagrama muestra la clase *InterfaceUsuario* como cliente de las diversas pantallas a través de la responsabilidad “desplegarPantalla” de la clase *Pantalla*.

Asignaremos como contrato número “1” a “desplegarPantalla”, mientras que “enviarEvento” será una responsabilidad privada. En base a estas consideraciones y a partir de la Tabla 8.44 se genera la descripción para la clase *Pantalla*, como se muestra en la Tabla 8.67. Nótese en la columna derecha como *InterfaceUsuario* aparece como colaborador de la responsabilidad “enviarEvento” a pesar de que ésta es privada. Adicionalmente, se agregó un “2” entre paréntesis a la derecha de la clase colaboradora *InterfaceUsuario*. Este “2” corresponde al contrato número “2” definido en la clase *InterfaceUsuario*, en otras palabras el contrato “Enviar Evento”. Agregando este número tenemos información adicional sobre la colaboración entre clases pero a nivel de contratos.

Clase: Pantalla	
Descripción: Pantalla heredada por las demás clases de tipo pantalla.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Abstracta	
Superclases:	
Subclases: PantallaPrincipal, PantallaServicio, PantallaRegUsuario, PantallaRegTarjeta	
Atributos:	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla	
Responsabilidades Privadas	
enviarEvento	InterfaceUsuario (2)

Tabla 8.67. Tarjeta para la superclase clase *Pantalla* con responsabilidades, colaboraciones, jerarquías y contratos identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Principal

De manera similar a *Pantalla* podemos identificar cuatro grupos de responsabilidades lógicamente separadas para los diversos manejadores, "manejarEvento", "desplegarPantalla", "ofrecerServicio" y "salir", todos definidos en la superclase *Manejador*, como se muestra en la Tabla 8.68 a partir de las responsabilidades definidas para la clase *Manejador* en la Tabla 8.47.

manejarEvento	
desplegarPantalla	InterfaceUsuario
ofrecerServicio	ManejadorServicio
salir	

Tabla 8.68. Responsabilidades para la superclase clase *Manejador* definidas en la Tabla 8.45 en la sección de jerarquías.

Si analizamos estas responsabilidades podemos observar que "manejarEvento" tiene como cliente a la *InterfaceUsuario*. Sin embargo, las otras tres responsabilidades, "desplegarPantalla", "ofrecerServicio" y "salir", no tiene ningún cliente externo a la clase por lo cual la convertiremos en responsabilidades privadas. Nótese nuevamente que dos de estas responsabilidades colaboran con otras clases a pesar de ser privadas. En la Figura 8.15. se muestra la clase *InterfaceUsuario* que solicita servicio a las responsabilidades de los diversos manejadores a través de la responsabilidad "manejarEvento" descrita en la clase *Manejador*.

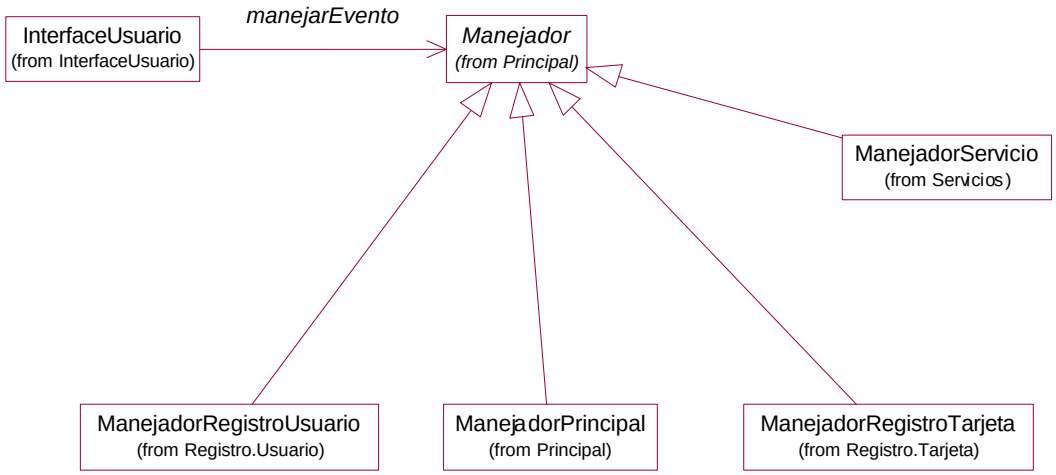

Figura 8.15 El diagrama muestra la clase *InterfaceUsuario* como cliente de los diversos manejadores a través de la responsabilidad "manejarEvento" de la clase *Manejador*.

Asignaremos como contrato número "1" a "manejarEvento", mientras que "desplegarPantalla", "ofrecerServicio" y "salir", serán responsabilidades privadas. En base a estas consideraciones y a partir de la Tabla 8.47 se genera la descripción para la clase *Manejador*, como se muestra en la Tabla 8.69. Nótese en la columna derecha como *InterfaceUsuario* aparece como colaborador de la responsabilidad "desplegarPantalla" a pesar de que ésta es privada. Adicionalmente, se agregó un "1" entre paréntesis a la derecha de la clase colaboradora *InterfaceUsuario*. Este "1" corresponde al contrato número "1" definido en la clase *InterfaceUsuario*, en otras palabras el contrato "Desplegar Pantalla". Adicionalmente se agregó el número "2" entre paréntesis a la derecha de la clase colaboradora *ManjeadorServicio* para la responsabilidad "ofrecerServicio", correspondiente al contrato "2" (aún no definido) para la clase *ManejadorServicio*. Los contratos para la clase *Manejador* se muestran en la Tabla 8.69.

Clase: Manejador	
Descripción: Superclase heredada por todos los manejadores del sistema.	
Módulo: Principal	
Estereotipo: Control	
Propiedades: Abstracta	
Superclases:	
Subclases: ManejadorPrincipal, ManejadorServicio, ManejadorRegistroUsuario, ManejadorRegistroTarjeta	
Atributos:	
Contratos	

1. Manejar Evento	
manejarEvento	
Responsablidades Privadas	
desplegarPantalla	InterfaceUsuario (1)
ofrecerServicio	ManejadorServicio (2)
salir	

Tabla 8.69. Tarjeta para la clase *Manejador* con responsabilidades, colaboraciones, jerarquías y contratos identificadas de los diversos manejadores para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Dado que ya hemos explicado los números entre paréntesis agregados a las clases colaboradoras en la columna derecha, volveremos a definir la tarjeta para la clase *InterfaceUsuario* correspondiente a la Tabla 8.65. Esto se muestra en la Tabla 8.70 agregando la colaboración con el contrato "1", "Desplegar Pantalla", definido en la clase *Pantalla* y el contrato "2", "Manejar Evento", definido en la clase *Manejador*.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos:	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla	Pantalla (1) : PantallaPrincipal (1), PantallaServicio (1), PantallaCrearRegUsuario (1), PantallaObtenerRegUsuario (1), PantallaCrearRegTarjeta (1), PantallaObtenerRegTarjeta (1)
2. Enviar Evento	
enviarEvento	Manejador (1) : ManejadorPrincipal (1), ManejadorServicio (1), ManejadorRegistroUsuario (1), ManejadorRegistroTarjeta (1)

Tabla 8.70. Tarjeta para la clase *InterfaceUsuario* revisada con números de contratos para las colaboraciones.

A continuación consideramos las responsabilidades definidas para la clase *ManejadorPrincipal* descritas en la Tabla 8.48. La responsabilidad "manejarEvento" sobreescribe la responsabilidad con el mismo nombre en la clase *Manejador* por lo cual generaremos un contrato "1" que sobrescribe al contrato general definida en la superclase. Las responsabilidades "crearRegsitroUsuario" y "validarRegistroUsuario" son privadas ya que son llamadas por la propia clase, en realidad como consecuencia del contrato "Manejar Evento". La clase *ManejadorPrincipal* con los contratos respectivos, se muestra en la Tabla 8.71. Nótese que se agrega el contrato "2" a la clase colaboradora *ManejadorRegistroUsuario*, algo que aún no se ha definido, pero lo haremos más adelante. Básicamente los contratos "1" corresponden en todos los manejadores al contrato "Manejar Evento" mientras que los contratos a partir del número "2" representan los contratos adicionales de los manejadores y se van dando según una numeración local incremental.

Clase: ManejadorPrincipal	
Descripción: El manejador principal es el encargado de desplegar la pantalla principal de interacción con el usuario, y luego delegar las diferentes funciones a los manejadores especializados apropiados.	
Módulo: Principal	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
Contratos	

1. Manejar Evento	
manejarEvento	
Responsablidades Privadas	
crearRegistroUsuario	ManejadorRegistroUsuario (2)
validarRegistroUsuario	ManejadorRegistroUsuario (2)

Tabla 8.71. Tarjeta para la clase *ManejadorPrincipal* con responsabilidades, colaboraciones, jerarquías y contratos identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

A partir de la Tabla 8.49 se generan los contratos para la clase *PantallaPrincipal*, como se muestra en la Tabla 8.72. Dado que los contratos como las responsabilidades fueron todas definidas en la superclase *Pantalla*, esta clase se mantiene igual en su descripción.

Clase: PantallaPrincipal	
Descripción: Pantalla principal (*P*-1).	
Módulo: Principal	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases: Pantalla	
Subclases:	
Atributos:	

Tabla 8.72. Tarjeta para la clase *PantallaPrincipal* con responsabilidades, colaboraciones, jerarquías y contratos identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Dominio

A partir de la Tabla 8.50 se generan los contratos para la clase *Datos*, como se muestra en la Tabla 8.73. Dado que aún no se han definido responsabilidades para las clases entidad, esta clase se mantiene igual a la anterior.

Clase: Datos	
Descripción: Superclase para todas las clases entidad.	
Módulo: Dominio	
Estereotipo: Entidad	
Propiedades: Abstracta	
Superclases:	
Subclases: RegistroUsuario, RegistroTarjeta	
Atributos:	

Tabla 8.73. Tarjeta para la clase *Datos* con responsabilidades, colaboraciones y jerarquías identificadas de las diversas clases entidad para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Registro

Este módulo se compone de los módulos de *Usuario*, *Tarjeta* e *InterfaceBD*.

Usuario

Esta sección involucra las clases de registro de usuario que son *ManejadoRegistroUsuario*, *PantallaCrearRegUsuario*, *PantallaObtenerRegUsuario* y *RegistroUsuario*.

Las responsabilidades de la clase *ManejadoRegistroUsuario* fueron descritas en la Tabla 8.51, las cuales se vuelven a describir en la Tabla 8.74.

manejarEvento	
validarRegistroUsuario	InterfaceBaseDatosRegistro
crearRegistroUsuario	InterfaceBaseDatosRegistro
obtenerRegistroUsuario	InterfaceBaseDatosRegistro
actualizarRegistroUsuario	InterfaceBaseDatosRegistro
eliminarRegistroUsuario	InterfaceBaseDatosRegistro
registrarTarjeta	ManejadorRegistroTarjeta

Tabla 8.74. Responsabilidades definidas para la clase *ManejadoRegistroUsuario* según las Tabla 8.51.
De manera similar a *ManejadorPrincipal*, la responsabilidad "manejarEvento" será asignada al contrato "1", "Manejar Evento", por sobrescribir el contrato con el mismo número en la superclase *Manejador*. Del resto de las responsabildiades sólo tres son accesadas externamente, "crearRegistroUsuario", "validarRegistroUsuario" y "obtenerRegistroUsuario", las dos primeras por *ManejadorPrincipal*, mientras que la tercera por *ManejadorServicio*, como se muestra en la Figura 8.16.

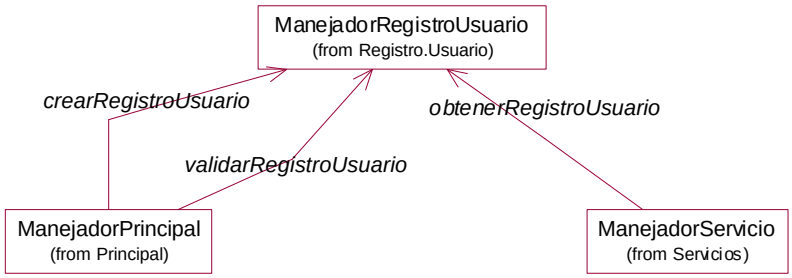

Figura 8.16 El diagrama muestra las clases *ManejadorPrincipal* y *ManejadorServicio* como clientes de las diversas responsabilidades de la clase *ManejadoRegistroUsuario*.

En general los diagramas de colaboración, como se muestran en la Figura 8.16, son extremadamente útiles para comprender las diversas colaboraciones que ocurren dentro del sistema. Sin embargo, mostrar llamadas a nivel de responsabilidades puede resultar en diagramas extremadamente densos, por lo cual se busca más bien describir las colaboraciones a nivel de los contratos los cuales son más reducidos en su número que las llamadas por responsabilidad. El mismo diagrama pero a nivel de contratos se muestra en la Figura 8.17. En este caso introducimos un solo contrato llamado "Registrar Usuario", el contrato número "2", el cual incluye las tres responsabilidades llamadas externamente y que manipulan el registro del usuario, sea a nivel de creación, validación u obtención. Existe siempre la alternativa de definir múltiples contratos, sin embargo, esto únicamente aumentaría la complejidad en este caso ya que la funcionalidad puede ser considerada como lógicamente similar.

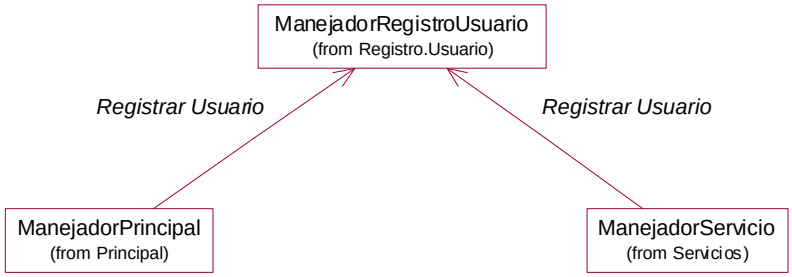

Figura 8.17 El diagrama muestra las clases *ManejadorPrincipal* y *ManejadorServicio* como clientes del contrato "Registrar Usuario", contrato número "2", de la clase *ManejadoRegistroUsuario*.

En la Tabla 8.75 se describe la tarjeta para la clase *ManejadorRegistroUsuario*. La responsabilidad "manejarEvento" se asigna al contrato número "1", "Manejar Evento", mientras que las responsabilidades "crearRegistroUsuario", "validarRegistroUsuario" y "obtenerRegistroUsuario", son asignadas al contrato número "2", "Registrar Usuario". El resto de las responsabilidades, "actualizarRegistroUsuario", "eliminarRegistroUsuario" y "registrarTarjeta", se mantienen como responsabilidades privadas ya que son llamadas localmente dentro de la clase *ManejadorRegistroUsuario*. Nótese que nuevamente se agregaron números de contratos a las diferentes clases colaboradoras. Estos contratos están aún por definirse, sin embargo, mantenemos la numeración lógica anteriormente mencionada.

Clase: ManejadorRegistroUsuario
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para poder utilizar el sistema.
Módulo: Registro.Usuario
Estereotipo: Control
Propiedades: Concreta
Superclases: Manejador

Subclases:	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento	
2. Registrar Usuario	
crearRegistroUsuario	InterfaceBaseDatosRegistro (1)
validarRegistroUsuario	InterfaceBaseDatosRegistro (1)
obtenerRegistroUsuario	InterfaceBaseDatosRegistro (1)
Responsabilidades Privadas	
actualizarRegistroUsuario	InterfaceBaseDatosRegistro (1)
eliminarRegistroUsuario	InterfaceBaseDatosRegistro (1)
registrarTarjeta	ManejadorRegistroTarjeta (2)

Tabla 8.75. Tarjeta para la clase *ManejadoRegistroUsuario* con responsabilidades, colaboraciones, jerarquías y contratos identificadas de los casos de uso *RegistrarUsuario* y *ValidarUsuario*.

De manera similar a las demás pantallas, la clase *PantallaRegUsuario* se describe igual a como se describió originalmente en la Tabla 8.50, como se muestra en la Tabla 8.76.

Clase: PantallaRegUsuario	
Descripción: Superclase con diseño gráfico común para las pantallas de registro de usuario.	
Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades: Abstracta	
Superclases: Pantalla	
Subclases: PantallaCrearRegUsuario, PantallaObtenerRegUsuario	
Atributos:	

Tabla 8.76. Tarjeta para la clase *PantallaRegUsuario* con responsabilidades, colaboraciones, jerarquías y contratos identificadas del caso de uso *RegistrarUsuario*.

A partir de la Tabla 8.53 se genera la clase *PantallaCrearRegUsuario*, como se muestra en la Tabla 8.77.

Clase: PantallaCrearRegUsuario	
Descripción: Pantalla de solicitud de registro de usuario (*P*-3).	
Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases: Pantalla	
Subclases:	
Atributos:	

Tabla 8.77. Tarjeta para la clase *PantallaCrearRegUsuario* con responsabilidades, colaboraciones, jerarquías y contratos identificadas del caso de uso *RegistrarUsuario*.

A partir de la Tabla 8.54 se genera la clase *PantallaObtenerRegUsuario*, como se muestra en la Tabla 8.78.

Clase: PantallaObtenerRegUsuario	
Descripción: Pantalla de devolución con información de registro de usuario (*P*-4).	
Módulo: Registro.Usuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases: Pantalla	
Subclases:	
Atributos:	

Tabla 8.78. Tarjeta para la clase *PantallaCrearRegUsuario* con responsabilidades, colaboraciones, jerarquías y contratos identificadas del caso de uso *RegistrarUsuario*.

La clase *RegistroUsuario* se mantiene igual a como se describió anteriormente en la Tabla 8.55, como se muestra en la Tabla 8.79.

Clase: RegistroUsuario	
Descripción: Para poder utilizar el sistema de reservaciones, el usuario debe estar registrado con el sistema. El registro contiene información acerca del usuario que incluye nombre, dirección, colonia, ciudad, país, código postal, teléfono de casa, teléfono de oficina, fax, email, login y password.	
Módulo: Registro.Usuario	
Estereotipo: Entidad	
Propiedades: Concreta	
Superclases: Datos	
Subclases:	
Atributos:	

Tabla 8.79. Tarjeta para la clase *RegistroUsuario* con responsabilidades, colaboraciones, jerarquías y contratos de actualizar y consultar información de registro para el caso de uso *RegistrarUsuario*.

Tarjeta

Esta sección involucra las clases de registro de tarjeta que son *ManejadoRegistroTarjeta*, *PantallaCrearRegTarjeta*, *PantallaObtenerRegTarjeta* y *RegistroTarjeta*.

Las responsabilidades de la clase *ManejadoRegistroTarjeta* fueron descritas en la Tabla 8.56, las cuales se vuelven a describir en la Tabla 8.80.

manejarEvento	
registrarTarjeta	
crearRegistroTarjeta	InterfaceBaseDatosRegistro
obtenerRegistroTarjeta	InterfaceBaseDatosRegistro
actualizarRegistroTarjeta	InterfaceBaseDatosRegistro
eliminarRegistroTarjeta	InterfaceBaseDatosRegistro

Tabla 8.80. Responsabilidades definidas para la clase *ManejadoRegistroTarjeta* según las Tabla 8.56.

De manera similar a *ManejadorPrincipal* y *ManejadorRegistroUsuario*, la responsabilidad "manejarEvento" será asignada al contrato "1", "Manejar Evento", por sobrescribir el contrato con el mismo número en la superclase *Manejador*. La única otra responsabilidad accesada externamente es "registrarTarjeta", la cual es llamada por el *ManejadorRegistroUsuario*, como se muestra en la Figura 8.18.

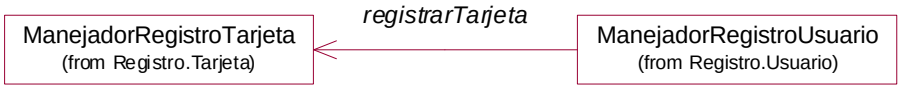

Figura 8.18 El diagrama muestra la clase *ManejadoRegistroUsuario* como cliente de la responsabilidad "registrarTarjeta" de la clase *ManejadoRegistroTarjeta*.

En la Tabla 8.81 se describe la tarjeta para la clase *ManejadorRegistroTarjeta*. La responsabilidad "manejarEvento" se asigna al contrato número "1", "Manejar Evento", mientras que la responsabilidad "registrarTarjeta" es asignada al contrato número "2", "Registrar Tarjeta". El resto de las responsabildiades, "crearRegistroTarjeta", "obtenerRegistroTarjeta", "actualizarRegistroTarjeta" y "eliminarRegistroTarjeta", se mantienen como responsabilidades privadas ya que son llamadas localmente dentro de la clase *ManejadorRegistroTarjeta*. Nótese que nuevamente se agregaron números de contratos a las diferentes clases colaboradoras. Estos contratos están aún por definirse, sin embargo, mantenemos la numeración lógica anteriormente mencionada.

A partir de la Tabla 8.56 se generan los contratos para la clase *ManejadoRegistroTarjeta*, como se muestra en la Tabla 8.81.

Clase: ManejadorRegistroTarjeta
Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.

Propiedades: Concreta	
Módulo: Registro.Tarjeta	
Estereotipo: Control	
Superclases: Manejador	
Subclases:	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento	
2. Registrar Tarjeta	
registrarTarjeta	
Responsabilidades Privadas	
crearRegistroTarjeta	InterfaceBaseDatosRegistro (2)
obtenerRegistroTarjeta	InterfaceBaseDatosRegistro (2)
actualizarRegistroTarjeta	InterfaceBaseDatosRegistro (2)
eliminarRegistroTarjeta	InterfaceBaseDatosRegistro (2)

Tabla 8.81. Tarjeta para la clase *ManejadoRegistroTarjeta* con responsabilidades, colaboraciones, jerarquías y contratos identificadas del caso de uso *RegistrarTarjeta*.

Las pantallas se describen nuevamente de manera similar a las anteriores. A partir de la Tabla 8.57 se vuelve a describir la clase *PantallaRegTarjeta*, como se muestra en la Tabla 8.82.

Clase: PantallaRegTarjeta	
Descripción: Superclase con diseño gráfico común para las pantallas de registro de tarjeta.	
Módulo: Registro. Tarjeta	
Estereotipo: Borde	
Propiedades: Abstracta	
Superclases: Pantalla	
Subclases: PantallaCrearRegTarjeta, PantallaObtenerRegTarjeta	
Atributos:	

Tabla 8.82. Tarjeta para la clase *PantallaRegTarjeta* con responsabilidades, colaboraciones, jerarquías y contratos identificadas del caso de uso *RegistrarTarjeta*.

A partir de la Tabla 8.58 se describe la clase *PantallaCrearRegTarjeta*, como se muestra en la Tabla 8.83.

Clase: PantallaCrearRegTarjeta	
Descripción: Pantalla de solicitud de registro de tarjeta (*P*-5).	
Módulo: Registro.Tarjeta	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases: Pantalla	
Subclases:	
Atributos:	

Tabla 8.83. Tarjeta para la clase *PantallaCrearRegTarjeta* con responsabilidades, colaboraciones, jerarquías y contratos identificadas del caso de uso *RegistrarTarjeta*.

A partir de la Tabla 8.59 se describe la clase *PantallaObtenerRegTarjeta*, como se muestra en la Tabla 8.84.

Clase: PantallaObtenerRegTarjeta
Descripción: Pantalla de devolución con información de registro de tarjeta (*P*-6).
Módulo: Registro.Tarjeta
Estereotipo: Borde
Propiedades: Concreta
Superclases: Pantalla

Subclases:	
Atributos:	

Tabla 8.84. Tarjeta para la clase *PantallaCrearRegTarjeta* con responsabilidades, colaboraciones, jerarquías y contratos identificadas del caso de uso *RegistrarTarjeta*.

El *RegistroTarjeta* se mantiene igual a la descripción anterior, de manera análoga a *RegistroUsuario*. A partir de la Tabla 8.60 se describe la clase *RegistroTarjeta*, como se muestra en la Tabla 8.85.

Clase: RegistroTarjeta	
Descripción: Para poder hacer un pago con una tarjeta de crédito, se debe tener un registro de tarjeta. El registro contiene información acerca de la tarjeta incluyendo nombre, número, expedidor y vencimiento. La tarjeta está ligada a un registro de usuario.	
Módulo: Registro.Tarjeta	
Estereotipo: Entidad	
Propiedades: Concreta	
Superclases: Datos	
Subclases:	
Atributos:	

Tabla 8.85. Tarjeta para la clase *RegistroTarjeta* con responsabilidades, colaboraciones, jerarquías y contratos de actualizar y consultar información de registro para el caso de uso *RegistrarTarjeta*.

Interface Base Datos

Las responsabilidades de la clase *InterfaceBaseDatosRegistro* fueron descritas en la Tabla 8.61, las cuales se vuelven a describir en la Tabla 8.86.

validarRegistroUsuario	BaseDatosRegistro
obtenerRegistroUsuario	BaseDatosRegistro
crearRegistroUsuario	BaseDatosRegistro
actualizarRegistroUsuario	BaseDatosRegistro
eliminarRegistroUsuario	BaseDatosRegistro
obtenerRegistroTarjeta	BaseDatosRegistro
crearRegistroTarjeta	BaseDatosRegistro
actualizarRegistroTarjeta	BaseDatosRegistro
eliminarRegistroTarjeta	BaseDatosRegistro

Tabla 8.86. Responsabilidades definidas para la clase *InterfaceBaseDatosRegistro* según las Tabla 8.61.

Se pueden apreciar dos grupos de responsabilidades para la clase *InterfaceBaseDatosRegistro*, aquellas responsabilidades relacionadas con el registro de usuario y aquellas relacionadas con el registro de tarjeta. En la Figura 8.19 podemos apreciar esto en mayor detalle.

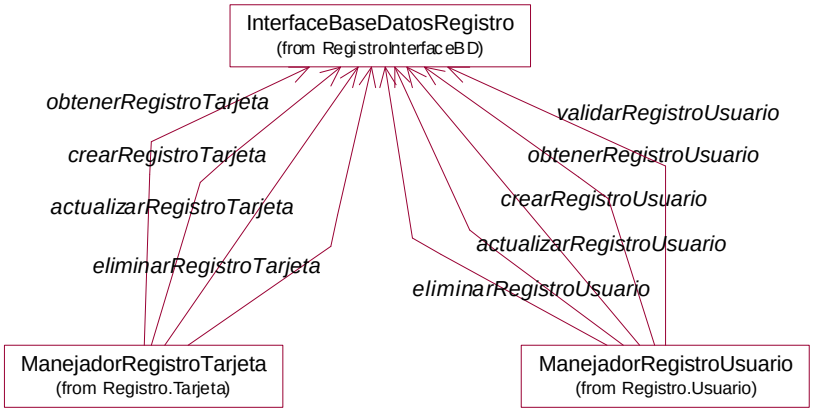

Figura 8.19 El diagrama muestra las clases *ManejadoRegistroUsuario* y *ManejadoRegistroTarjeta* como clientes de la clase *InterfaceBaseDatosRegistro*.

En la Figura 8.19 podemos nuevamente apreciar la complejidad de diagramar las relaciones entre clases a nivel de responsabilidades, sería menos complejo hacerlo a nivel de contratos siempre cuando estos agrupen a las responsabilidades. Si consideramos que existen dos grupos de responsabilidades, organizadas de acuerdo a las dos clases clientes, *ManejadoRegistroUsuario* y *ManejadoRegistroTarjeta*, podemos de manera natural definir dos contratos, el número "1" correspondiente a "Registrar Usuario" y el número "2" correspondiente a "Registrar Tarjeta". Esta agrupación simplifica la visualización de los contratos como podemos apreciar en la Figura 8.20.

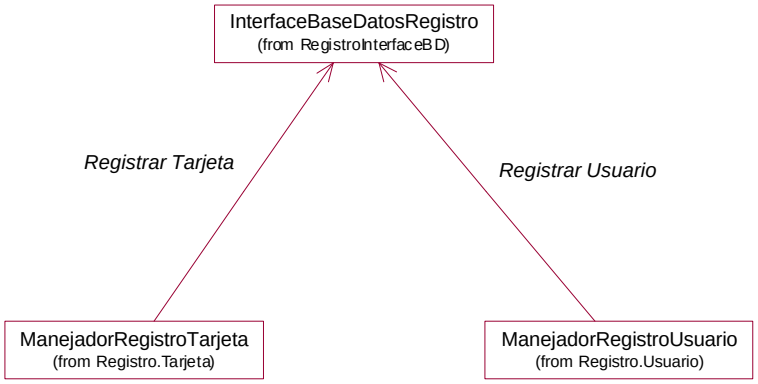

Figura 8.20 El diagrama muestra las clases *ManejadoRegistroUsuario* y *ManejadoRegistroTarjeta* como clientes de la clase *InterfaceBaseDatosRegistro* describiendo las relaciones a nivel de contratos.

En la Tabla 8.87 se describe la tarjeta para la clase *InterfaceBaseDatosRegistro*. Se definen dos contratos correspondientes a "Registrar Usuario" y "Registrar Tarjeta".

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceDB	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos:	
Contratos	
1. Registrar Usuario	
crearRegistroUsuario	BaseDatosRegistro

obtenerRegistroUsuario	BaseDatosRegistro
actualizarRegistroUsuario	BaseDatosRegistro
eliminarRegistroUsuario	BaseDatosRegistro
validarRegistroUsuario	BaseDatosRegistro
2. Registrar Tarjeta	
crearRegistroTarjeta	BaseDatosRegistro
obtenerRegistroTarjeta	BaseDatosRegistro
actualizarRegistroTarjeta	BaseDatosRegistro
eliminarRegistroTarjeta	BaseDatosRegistro

Tabla 8.87. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades, colaboraciones, jerarquías y contratos de escribir y leer información de registro de usuario y registro de tarjeta para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Servicios

La descripción del *ManejadorServicio* es similar a los demás manejadores. Se genera un contrato "Manejar Evento" similar a los demás manejadores y otro propio a esta clase, "Ofrecer Servicio" el cual tiene como cliente a los demás manejadores, como se muestra en la Figura 8.21.

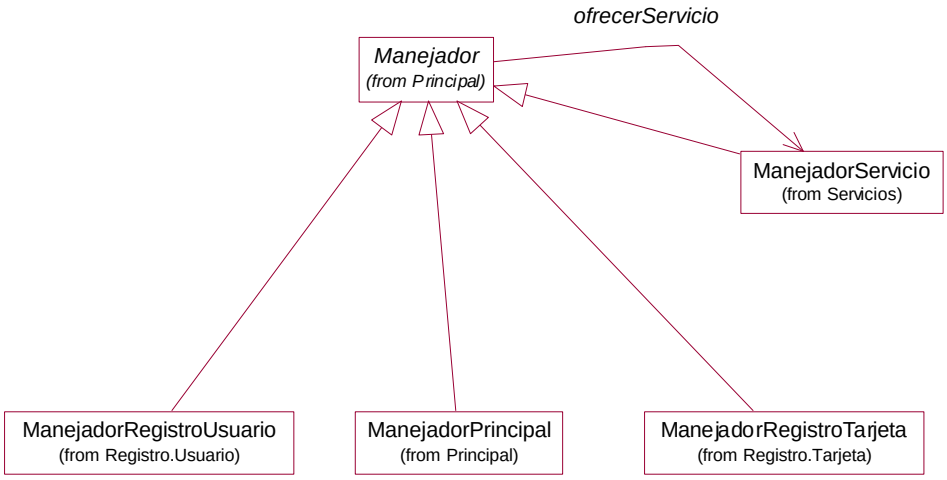

Figura 8.21 El diagrama muestra a las diversas clases de manejadores como clientes de de la clase *ManejadorServicio*.

A partir de la Tabla 8.62 se describe la clase *ManejadorServicio*, como se muestra en la Tabla 8.88. Además de los dos contratos antes mencionados, se agrega una responsabilidad privada llamada "registrar".

Clase: ManejadorServicio	
Descripción: El manejador de servicios se encarga de enviar las peticiones particulares de servicios a los manejadores espacializados para consulta, reserva y compra.	
Módulo: Servicios	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento	
2. Ofrecer Servicio	
ofrecerServicio	
Responsablidades Privadas	
registrar	ManejadorRegistroUsuario (2)

Tabla 8.88. Tarjeta para la clase *ManejadorServicio* con responsabilidades, colaboraciones, jerarquías y contratos a partir de los casos de uso *RegistrarUsuario* y *RegistrarTarjeta*.

De manera similar a las demás pantallas y a partir de la Tabla 8.63 se describe la clase *PantallaServicio*, como se muestra en la Tabla 8.89.

Clase: PantallaServicio	
Descripción: Pantalla de servicios (*P*-2).	
Módulo: Servicios	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases: Pantalla	
Subclases:	
Atributos:	

Tabla 8.89. Tarjeta para la clase *PantallaServicio* con responsabilidades, colaboraciones, jerarquías y contratos a partir de los casos de uso *RegistrarUsuario* y *RegistrarTarjeta*.

Subsistemas

Como se ha podido apreciar hasta el momento, la complejidad del sistema aumenta a medida que se incorporan nuevos detalles en el diseño, algo que por lo general es inevitable. Para lograr un mejor manejo de esta complejidad introducimos el concepto de *subsistemas*, el cual permite dividir el sistema completo en diversas partes, inspirado en la idea de "divide y conquista". Los subsistemas permiten agrupar objetos relacionados para lograr cierta funcionalidad en "mini-sistemas". El sistema completo se compone de estos "mini-sistemas" o subsistemas y cada subsistema puede ser subdividido en subsistemas adicionales o ser definido en términos de los objetos finales. Los subsistemas también permiten trabajar en diferentes partes del sistema en paralelo mediante su asignación a múltiples diseñadores.

La organización en subsistemas se logra a partir de los contratos identificadas anteriormente entre los objetos. Externamente, los subsistemas son mecanismos de encapsulamiento, vistos como "cajas negras", donde sus objetos cooperan para proveer una unidad de funcionalidad claramente delimitada por el subsistema. Internamente, los subsistemas pueden tener estructuras complejas, con clases colaborando entre si para satisfacer sus distintas responsabilidades contribuyendo al objetivo general del subsistema, o sea, satisfacer sus responsabilidades.

Se busca tener un fuerte acoplamiento funcional dentro de cada subsistema y un débil acoplamiento entre subsistemas, en otras palabras, se busca tener la mínima comunicación entre los diferentes subsistemas.

Un subsistema bien diseñado tiene pocas clases o subsistemas que directamente apoyan contratos, y un número mayor de colaboraciones entre clases y subsistemas internos.

Es importante distinguir entre el concepto de *módulo* y *subsistema*. Un módulo agrupa clases, correspondiente a *bibliotecas*, o sea, organizaciones estáticas definidas para almacenar y facilitar el acceso a las clases. Esto es similar al concepto de *directorios* en una computadora. Son estructuras puramente organizacionales sin ningún efecto sobre los propios procesos. Por otro lado, el subsistema agrupa objetos, siendo una abstracción dinámica correspondiente a la funcionalidad del proceso. En general, los objetos pertenecientes a un subsistema son instancias de clases que deben existir forzosamente en algún módulo. Mientras que las clases no deben duplicarse entre módulos, se permite instanciar objetos de la misma clase en múltiples subsistemas, siendo esto parte de la reutilización de código.

Los subsistemas deben incluir completos o no ser incluidos, de manera que un sistema pueda ser ofrecido con o sin ciertos subsistemas. Dada la dependencia entre subsistemas, si se incluye un subsistema, se debe también entregar el subsistema del cual depende.

Típicamente, los objetos de las clases entidad son reutilizados en los diversos subsistemas, mientras que los objetos de control y por lo general las de borde son propias a cierto subsistema.

Los subsistemas son una pieza fundamental en el manejo de la modularidad y extensibilidad del sistema. Por tal motivo, los subsistemas deben mantener una buena correspondencia con los casos de uso, de manera que cualquier cambios causado por modificaciones en la funcionalidad del sistema pueda ser rastreado a los subsistemas afectados. Cabe resaltar que los subsistemas no existen durante la ejecución de la aplicación, son puramente abstracciones del sistema.

Si consideramos que la mayor fuente de complejidad en el sistema radica en las relaciones entre objetos (a través de sus colaboraciones), la meta es reducir o al menos manejar de mejor manera estas relaciones. Un enfoque es definir los subsistemas de manera que, aunque el número total de relaciones sea el mismo, las relaciones estarán

organizadas en grupos (*clusters*). Si se observan que cierto grupo de objetos están muy relacionados entre si pero no tanto con otro grupo de objetos, esto deben asignarse a subsistemas comunes.
Como parte del proceso de identificación de subsistemas, se define un protocolo de comunicación para cada subsistema, el cual especifica las posibles interacciones entre subsistemas. Esto se logra exportando o haciendo públicos ciertos contratos pertenecientes a ciertos objetos del subsistema. La interface de estos objetos, sus servicios, definen la interface del subsistema completo. Para determinar los contratos apoyados por un subsistema, se tiene que encontrar todas los objetos o clases que proveen servicios a clientes fuera del subsistema. Al igual que las clases, se debe describir los contratos ofrecidos por cada subsistema. Esto se hace introduciendo el concepto de tarjeta de subsistemas de manera análoga al de clases. Dado que los subsistemas son solamente entidades conceptuales, estos no pueden directamente satisfacer ninguno de sus contratos. En su lugar, los subsistemas delegan cada contrato a una clase interna que lo satisface.
Los subsistemas se describen en *tarjetas de subsistemas*, un subsistema por tarjeta. Cada tarjeta incluye un nombre en la primera línea y dos columnas, como se puede ver en la Tabla 8.90.

Subsistema: Nombre del Subsistema	
Descripción: Descripción del Subsistema	
Clases: Grupo de clases (instanciadas) que participan en este subsistema.	

Tabla 8.90. Tarjeta para subsistemas.

En la columna izquierda se muestran los contratos ofrecidos de manera externa al subsistema, mientras que en la columna derecha, al lado de cada contrato se escribe la clase interna que ofrece el servicio, en otras palabras, a quien se le delega el contrato. Es importante notar que las clases abstractas no se asignan a ningún subsistema ya que no podrían ser instanciados directamente. En su lugar asignamos únicamente clases concretas. Por ejemplo, en la Tabla 8.91 se muestra un ejemplo para un subsistema *InterfaceUsuario*, donde el contrato externo al subsistema sería "Manejar Evento" mientras que la clase servidora para este contrato sería *InterfaceUsuario* a través de su contrato "1".

Subsistema: SubsistemaInterfaceUsuario	
Descripción: Este subsistema agrupa todos los objetos involucrados con el manejo general de las interfaces de usuario.	
Clases: InterfaceUsuario.	
1. Manejar Evento	InterfaceUsuario (1)

Tabla 8.91. Tarjeta para el subsistema *InterfaceUsuario*.

Antes de definir los diferentes subsistemas para el sistema de reservaciones, describiremos los diagramas de colaboración a nivel de subsistemas. Una de las maneras de hacerlo, y como la haremos aquí, es describir el subsistema como un rectángulo y cada uno de los contratos del subsistema como círculos. Estos contratos son asociados gráficamente con la clase servidor que implementa el contrato real mediante una relación de *realización*, como se muestra en la Figura 8.22.

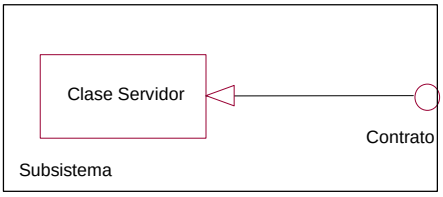

Figura 8.22 Diagrama de subsistema con un contrato asociado con la clase servidor que lo implementa.

Lamentablemente, muchas de las herramientas CASE no apoyan de manera completa la diagramación de los subsistemas. Por ejemplo, el subsistema descrito en la Tabla 8.91 se describiría como se muestra en la Figura 8.23.

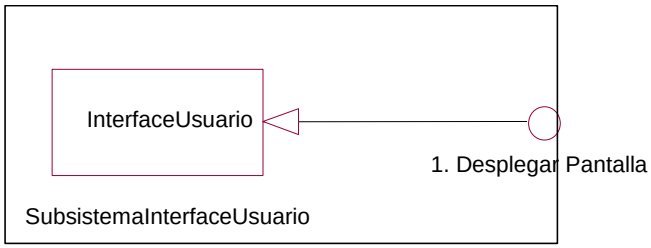

Figura 8.23 Diagrama de subsistema para el ejemplo del subsistema *InterfaceUsuario*.

Dado que los subsistemas son puramente abstracciones, los contratos se dan entre clases clientes solicitando servicios a los subsistemas a través de sus respectivas clases servidores utilizando una flecha del cliente al servidor, como se muestra en la Figura 8.24 (la flecha de la derecha proveniente de *Clase Cliente*).

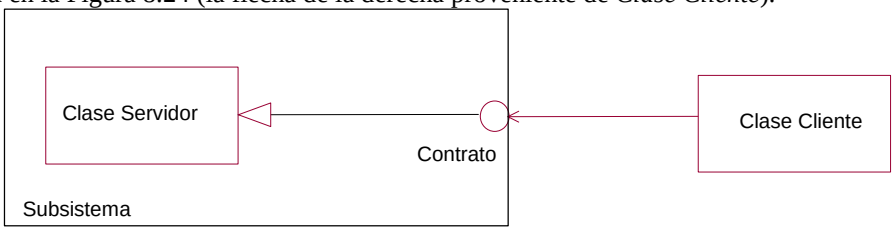

Figura 8.24 Diagrama de colaboración donde *Clase Cliente* llama al *Contrato* del *Subsistema* implementado por la *Clase Servidor*.

Si dos clases hacen solicitudes a un mismo contrato, se dibuja múltiples flechas al mismo círculo. Por ejemplo, en la Figura 8.25 se muestra dos manejadores llamando al contrato "Desplegar Pantalla" del subsistema *InterfaceUsuario*.

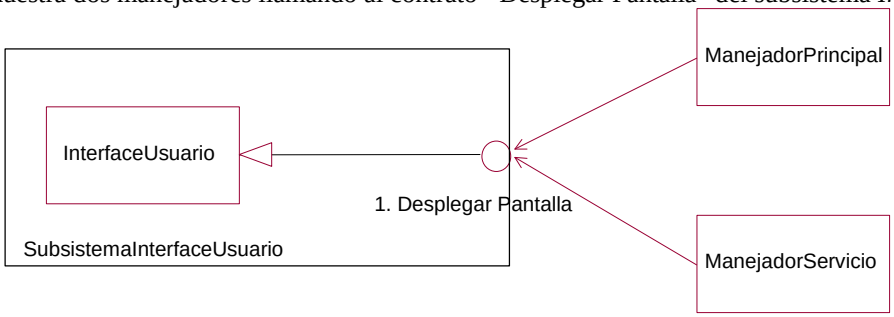

Figura 8.25 Diagrama de subsistema para el ejemplo del subsistema *InterfaceUsuario* con dos clientes para el contrato "Desplegar Pantalla", *ManejadorPrincipal* y *ManejadorServicio*.

El problema de la complejidad es evidente cuando uno se fija en el diagrama de colaboración. El diagrama se vuelve un "espagueti". No se puede entender fácilmente y la aplicación se vuelve imposible de mantener o modificar. Por lo tanto la meta es simplificar los patrones de colaboración, algo que luego se traduce en simplificación en los diagramas de colaboración.

A continuación identificaremos los subsistemas para el Sistema de Reservaciones. Dado que el enfoque del desarrollo del sistema ha sido a través de casos de uso, será natural inicialmente identificar subsistemas a partir de ellos. Sin embargo, el criterio principal para la asignación de clase a los subsistemas es minimizar al interacción entre clases en distintos subsistemas. Por otro lado, se debe mantener cierta relación con la lógica introducida en la arquitectura de clase. No debemos olvidarnos que estamos diseñando. Recordemos también que los subsistemas pueden definirse de manera jerárquica, a diferencia de los casos de uso.

Entonces, comencemos a analizar estas consideraciones en relación a nuestro sistema para poder identificar los subsistemas relevantes. Viendo el sistema a un alto nivel pudiéramos definir dos grupos de clases generales: aquellos relacionados con lo relacionado con registros y todo lo relacionado con consultas, reservaciones y pagos, o sea, los propios servicios del sistema. Por lo tanto, a un primer nivel podemos definir un *SubsistemaRegistro* y otro *SubsistemaServicio*. Estos subsistemas puede dividirse en subsistemas adicionales si así se desearía. Sin embargo, antes de definir subsistema más detallados veamos que ocurre con los niveles superiores. Aunque quedaría bastante clara la asignación de la mayoría de clases a estos dos subsistemas, como en el caso del *ManejadorRegistroUsuario*, *ManejadorReservas*, etc., hay ciertas clases que requieren un análisis más detallado. Por ejemplo, ¿a que subsistema asignamos la *InterfaceUsuario* y el *ManejajadorPrincipal*?

Tanto la *InterfaceUsuario* como el *ManejadorPrincipal* son clases que no pertenecen de manera natural a ninguno de los dos subsistemas anteriores. Pudiéramos definir un nuevo subsistema para ambos o si lógicamente son

independientes, definir dos nuevos subsistemas, uno para cada uno. Esto lo haremos, dado que la *InterfaceUsuario* es genérica en relación a la lógica interna y bastante independiente de la aplicación en si, a diferencia del *ManejadorPrincipal* que tiene cierto conocimiento sobre el registro y servicios de la aplicación.
Otra manera de conceptuar la creación de estos dos nuevos subsistemas es que inicialmente se definió el sistema en base a su funcionalidad en término de los casos de uso, o sea sus servicios externos. Sin embargo, también podemos considerar el sistema en término a sus servicios internos. Por ejemplo, la *InterfaceUsuario* ofrece servicios internos al sistema para que se pueda desplegar las diferentes pantallas y se pueda obtener eventos del *Usuario*. De igual manera, el *ManejadorPrincipal* ofrece funcionalidad para inicializar la aplicación, nuevamente un servicio interno que coordina entre funcionalidades externas. Por lo tanto, pudiéramos definir el sistema completo en término de cuatro subsistemas, *SubsistemaInterfaceUsuario*, *SubsistemaPrincipal*, *SubsistemaRegistro* y *SusbsistemaServicio*, como se muestra en la Figura 8.26.

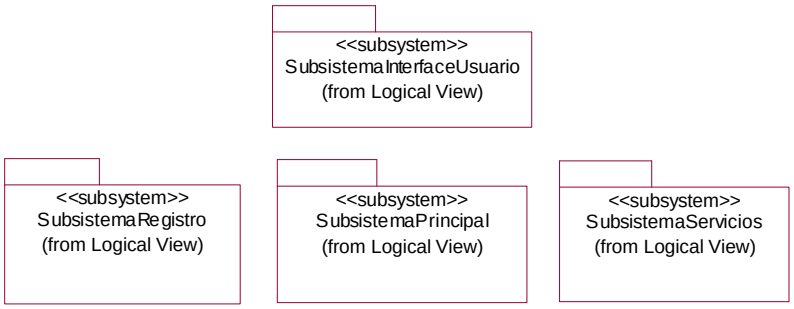

Figura 8.26 Subsistemas de alto nivel para el sistema de reservaciones de vuelos.

Estos subsistemas serán descritos con mayor detalle a continuación, aunque limitándonos a los subsistemas relevantes para la funcionalidad descrita en los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

SubsistemaInterfaceUsuario

El *SubsistemaInterfaceUsuario* fue descrito de manera preliminar en la Tabla 8.91. Este subsistema consiste principalmente de la clase *InterfaceUsuario*. Podemos apreciar que en la Tabla 8.70 se definieron dos contratos para la clase *InterfaceUsuario*, "Desplegar Pantalla" y "Enviar Evento". Estos contratos tienen como clientes a los diversos manejadores y a las diversas pantallas. El contrato "Desplegar Pantalla" se define como externo ya que los manejadores serán parte de otros subsistemas. En el caso de las pantallas debemos preguntarnos si éstas deberán asignarse al *SubsistemaInterfaceUsuario* por ser todas clases *borde* o distribuirse entre los diversos subsistemas según su funcionalidad. La decisión depende de hacer un balance entre la relación de esta clase con las demás clases a nivel de comunicaciones y funcionalmente a cual de los subsistemas debieran pertenecer. A nivel de comunicaciones, las diversas pantallas tienen los contratos "Desplegar Pantalla" definidos en la superclase *Pantalla*, llamado por la *InterfaceUsuario*, y a su vez llaman al contrato "Enviar Evento" definido por la clase *InterfaceUsuario*. En otras palabras, a nivel de comunicaciones, lo lógico sería asignar todas las pantallas a este subsistema. A nivel lógico, las pantallas se deberían asignar a los subsistemas que definan la funcionalidad correspondiente. Sin embargo, nuestro enfoque será el primero, asignar todas las pantallas al *SubsistemaInterfaceUsuario*. Esto nos evita tener que definir todos os contratos "Enviar Evento" de las diversas pantallas como externos a cada uno de los subsistemas. Se pudiera hacer un análisis similar para los manejadores, donde terminaríamos asignando todos ellos a un solo subsistema, lo cual sería contraproducente. La gran diferencia entre las pantallas y los manejadores es que los manejadores representan la funcionalidad del sistema mientras que las pantallas representan únicamente aspectos de presentación los cuales son hasta cierto punto independientes de la funcionalidad. Si la lógica de la aplicación cambia, todo el sistema debe cambiar, pero si la presentación cambia, no necesariamente debe cambiar toda la aplicación. Este es el caso, por ejemplo, de pasar este ejemplo de reservaciones al Internet donde la lógica se mantiene pero todo el diseño de pantallas (no el diseño gráfico) es completamente distinto. Por lo tanto, la decisión de asignar las pantallas a este subsistema es más justificable que el caso de asignar a los manejadores. La tarjeta para el *SubsistemaInterfaceUsuario* se muestra en la Tabla 8.92.

Subsistema: SubsistemaInterfaceUsuario	
Descripción: Este subsistema agrupa todos los objetos involucrados con el manejo general de las interfaces de usuario.	
Clases: InterfaceUsuario, PantallaPrincipal, PantallaServicio, PantallaCrearRegUsuario, PantallaObtenerRegUsuario, PantallaCrearRegTarjeta, PantallaObtenerRegTarjeta	
Contratos	**Servidor**

1. Desplegar Pantalla	InterfaceUsuario (1)

Tabla 8.92. Tarjeta de subsistema para *SubsistemaInterfaceUsuario* mostrando sus contratos y servidores a partir de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

El *SubsistemaInterfaceUsuario* se muestra gráficamente en la Figura 8.27. Se muestra únicamente la clase *InterfaceUsuario* por ser el servidor del subsistema.

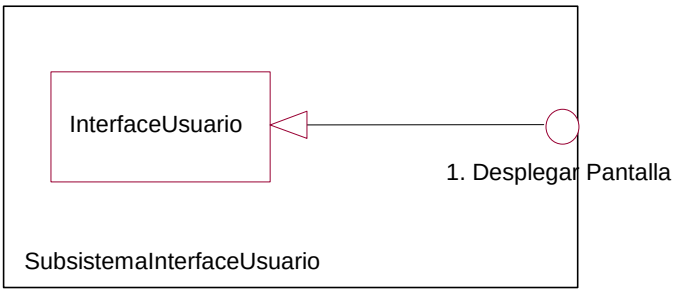

Figura 8.27 Diagrama de subsistemas para el *SubsistemaInterfaceUsuario* mostrando únicamente a *InterfaceUsuario*, la clase servidor.

SubsistemaPrincipal

El *SubsistemaPrincipal* consiste principalmente de la clase *ManejadorPrincipal*. Podemos apreciar que en la Tabla 8.70 se definió un sólo contrato, "Manejar Evento", el cual tiene como cliente a la clase *InterfaceUsuario*. Dado que la clase *InterfaceUsuario* pertence al *SubsistemaInterfaceUsuario*, el contrato "Manejar Evento" debe definirse como externo al subsistema. La tarjeta para el *SubsistemaPrincipal* se muestra en la Tabla 8.93.

Subsistema: SubsistemaPrincipal	
Descripción: Este subsistema agrupa todos los objetos involucrados con el manejo general del sistema.	
Clases: ManejadorPrincipal	
Contratos	**Servidor**
1. Manejar Evento	ManejadorPrincipal (1)

Tabla 8.93. Tarjeta de subsistema para *SubsistemaPrincipal* mostrando sus contratos y servidores a partir de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

El *SubsistemaPrincipal* se muestra gráficamente en la Figura 8.28. No se incluye en el diagrama la clase *PantallaPrincipal* ya que ésta no ofrece servicios externos.

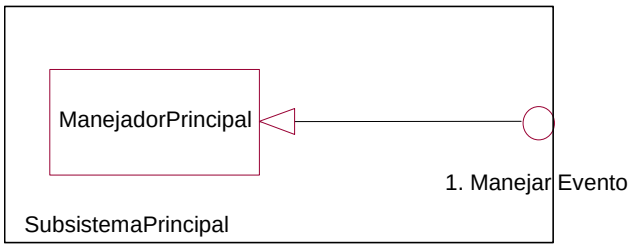

Figura 8.28 Diagrama de subsistemas para el *SubsistemaPrincipal*.

SubsistemaRegistro

El *SubsistemaRegistro* consiste de todas las clases relacionadas con el registro con excepción de las pantallas que fueron asignadas al subsistema *SubsistemaInterfaceUsuario*. Las clases incluidas son *ManejadorRegistroUsuario*, *RegistroUsuario*, *ManejadorRegistroTarjeta*, *RegistroTarjeta* y *InterfaceBaseDatosRegistro*. Se incluyeron las clases entidad junto con los manejadores y la clase de borde con la base de datos. En término de contratos, podemos observar que los dos contratos "Manejar Evento" pertenecientes a los dos manejadores, deben ser externos al subsistema ya que son llamados por la *InterfaceUsuario*. Adicionalmente, existe otro contrato que puede ser llamados externamente, "Registrar Usuario" (el contrato "Registrar Tarjeta" aunque no es llamado por clases pertenecientes a otros subsistemas para los casos de uso analizados hasta el momento, eventualmente deberá ser agregado al subsistema como apoyo al caso de uso de "Pagar Reservación"). Por otro lado las clases entidad, *RegistroUsuario* y *RegistroTarjeta*, aún no definen contratos, mientras que la clase *InterfaceBaseDatosRegistro* contiene contratos únicamente para los manejadores internos de este subsistema. La tarjeta para el *SubsistemaRegistro* se muestra en la Tabla 8.94. Nótese que existen dos servidores para el primer contrato. Esto no es un error, ya que los servidores pueden estar activos en diferentes momentos durante la ejecución del sistema.

Adicionalemente, pudiéramos definir “sub-subsistemas”, o sea subsistemas a un nivel inferior en la jerarquía, sin embargo, consideramos que dado el número limitado de clases en este subsistema, esto no será necesario.

Subsistema: SubsistemaRegistro	
Descripción: Este subsistema agrupa todos los objetos involucrados con el manejo de registro de usuario, incluyendo registro de tarjeta.	
Clases: ManejadorRegistroUsuario, RegistroUsuario, ManejadorRegistroTarjeta, RegistroTarjeta, InterfaceBaseDatosRegistro	
Contratos	**Servidor**
1. Manejar Evento	ManejadorRegistroUsuario (1), ManejadorRegistroTarjeta (1)
2. Registrar Usuario	ManejadorRegistroUsuario (2)

Tabla 8.94. Tarjeta de subsistema para *SubsistemaRegistro* mostrando sus contratos y servidores a partir de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

El *SubsistemaRegistro* se muestra gráficamente en la Figura 8.29. Nótese nuevamente que el subsistema ofrece dos contratos externos ligados a los servidores internos correspondientes.

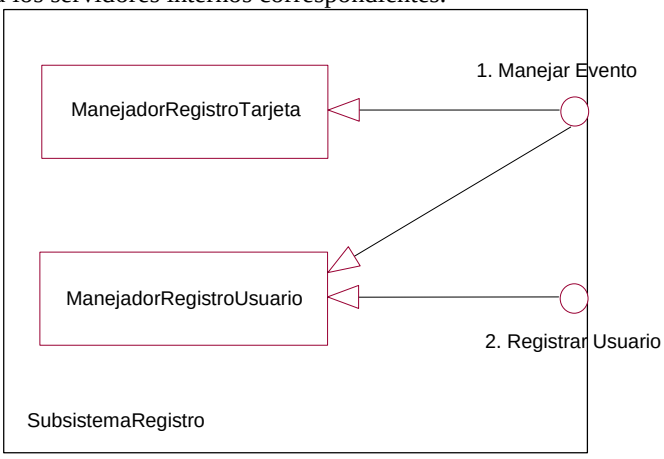

Figura 8.29 Diagrama de subsistemas para el *SubsistemaRegistro*.

SubsistemaServicios

El *SubsistemaServicios* consiste principalmente de la clase *ManejadorServicio*. Eventualemente, este subsistema incluirá clases y subsistemas internos adicionales como parte de la funcionalidad apoyada para los demás casos de uso, algo que se muestra en los apéndices. En la Tabla 8.95 se muestran los dos contratos definidos para este subsistema, “Manejar Evento”, similar a los demás subsistema que contienen manejadores, y “Ofrecer Servicio”, contrato que permite seleccionar la funcionalidad del sistema deseado. Ambos contratos son implementados por la clase *ManejadorServicio*.

Subsistema: SubsistemaServicios	
Descripción: Este subsistema agrupa los objetos generales para el manejo de los servicios de reservaciones.	
Clases: ManejadorServicio	
Contratos	**Servidor**
1. Manejar Evento	ManejadorServicio (1)
2. Ofrecer Servicio	ManejadorServicio (2)

Tabla 8.95. Tarjeta de subsistema para *SubsistemaServicios* mostrando sus contratos y servidores a partir del caso de uso *RegistrarUsuario*.

El *SubsistemaServicio* se muestra gráficamente en la Figura 8.30. Ambos contratos son ofrecidos por el *ManjeadorServicio*.

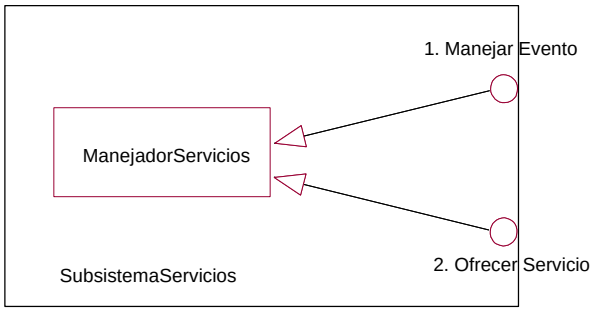

Figura 8.30 Diagrama de subsistemas para el *SubsistemaServicios*.

Sistema

Una de las ventajas de definir subsistemas es la posibilidad de visualizar el sistema completo a partir de estas. En la Figura 8.31 se integran los subsistemas anteriores mostrando los contratos entre subsistemas. Se pueden apreciar seis contratos los cuales son llamados por las clases manejadoras en los subsistemas de *Registro*, *Servicios* y *Principal* y la *InterfaceUsuario* en el subsistema correspondiente.

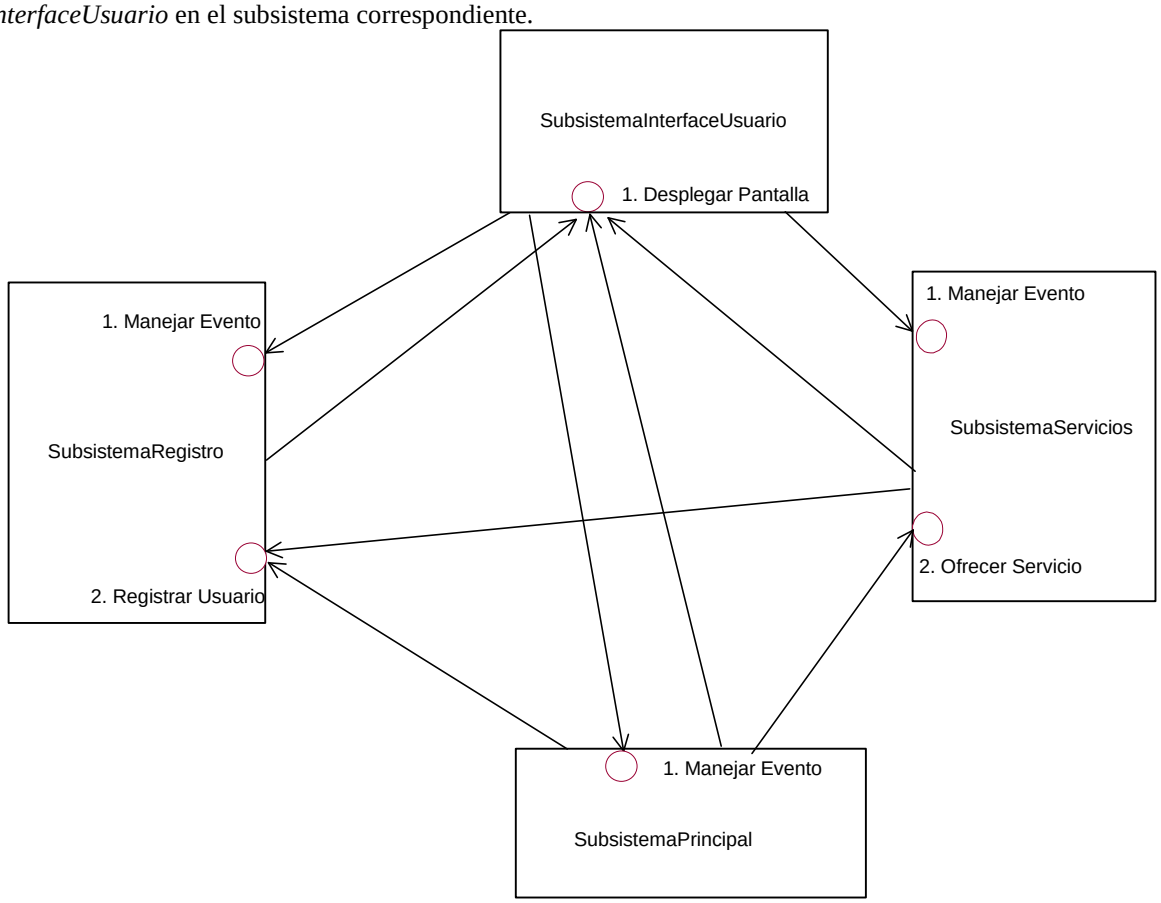

Figura 8.31 Subsistemas del sistema de reservaciones de vuelo.

Al incluir los subsistemas es necesario modificar las tarjetas de clase para hacer referencias a estos subsistemas y no a las clases que éstas encapsulan. Si una clase externa a un subsistema colabora con una clase dentro de un subsistema, se debe cambiar esto a una colaboración con el subsistema y no directamente con la clase. Por lo tanto, no modificaremos las tarjetas de clase con excepción a aquellas donde las colaboraciones sean con clases en otros subsistemas.

Módulo InterfaceUsuario

La clase *InterfaceUsuario*, tal como se describió en la Tabla 8.70, define dos responsabilidades “desplegarPantalla” y “enviarEvento”. Las colaboraciones a partir de “desplegarPantalla” son con pantallas las cuales residen dentro del *SubsistemaInterfaceUsuario*, por lo cual ningún cambio en la descripción de las colaboraciones es necesario. La segunda responsabilidad, “enviarEvento”, requiere colaboraciones con los manejadores a través de los contratos de

"Manejar Evento", los cuales están definidos a partir de la superclase *Manejador* y sobrescritos por los diversos manejadores. Aunque la clase *InterfaceUsuario* colabora con todos sin saber su verdadera identidad , estas clases residen en diversos subsistemas, por lo cual se debe modificar los nombres de clases y en su lugar poner los nombres de los subsistemas correspondientes a los cuales pertenecen las diversas clases manejadores. La tarjeta de clase modificada para la clase *InterfaceUsuario* se muestra en la Tabla 8.96.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos:	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla	Pantalla (1) : PantallaPrincipal (1), PantallaServicio (1), PantallaCrearRegUsuario (1), PantallaObtenerRegUsuario (1), PantallaCrearRegTarjeta (1), PantallaObtenerRegTarjeta (1)
2. Enviar Evento	
enviarEvento	Manejador (1) : SubsistemaPrincipal (1), SubsistemaServicio (1), SubsistemaRegistro (1)

Tabla 8.96. Tarjeta para la clase *InterfaceUsuario* con responsabilidades, colaboraciones, jerarquías, contratos y subsistemas identificados de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

La clase *Pantalla*, como se definió en la Tabla 8.67, y todas sus subclases correspondientes a las distintas pantallas no son modificadas ya que la colaboración de la clase es con la clase *InterfaceUsuario* que pertenece al mismo subsistema.

Módulo Principal

La clase *ManejadorPrincipal* debe modificarse ya que sus colaborador es la clase ManejadorRegistroUsuario como se puede apreciar en la Tabla 8.71. Por lo tanto, se actualizan los contratos para la clase *ManejadorPrincipal*, en este caso l contrato "2" del *SubsistemaRegistro*, como se muestra en la Tabla 8.97.

Clase: ManejadorPrincipal	
Descripción: El manejador principal es el encargado de desplegar la pantalla principal de interacción con el usuario, y luego delegar las diferentes funciones a los manejadores especializados apropiados.	
Módulo: Principal	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento	
Responsablidades Privadas	
crearRegUsuario	SubsistemaRegistro (2)
validarRegUsuario	SubsistemaRegistro (2)

Tabla 8.97. Tarjeta para la clase *ManejadorPrincipal* con responsabilidades, colaboraciones, jerarquías, contratos y subsistemas identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

La clase *Manejador*, como se mostró en la Tabla 8.66 llama a los contratos "1" de la *InterfaceUsuario* y el contrato "2" del *ManejadorServicio*. Por lo tanto se actualizan los contratos al contrato "1" del *SubsistemaInterfaceUsuario* y el contrato "2" del *SubsistemaServicio* para la clase *Manejador*, como se muestra en la Tabla 8.98.

Clase: Manejador

Descripción: Pantalla heredada por las demás clases de tipo pantalla.	
Módulo: Principal	
Estereotipo: Control	
Propiedades: Abstracta	
Superclases:	
Subclases: ManejadorPrincipal, ManejadorServicio, ManejadorRegistroUsuario, ManejadorRegistroTarjeta	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento	
Responsablidades Privadas	
desplegarPantalla	SubsistemaInterfaceUsuario (1)
ofrecerServicio	SubsistemaServicio (2)
salir	

Tabla 8.98. Tarjeta para la clase *Manejador* con responsabilidades, colaboraciones, jerarquías, contratos y subsistemas identificadas de los diversos manejadores para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Módulo Dominio

La descripción de la clase *Datos* se mantiene igual.

Módulo Registro

En el módulo de *Registro*, compuesto de los módulos *Usuario*, *Tarjeta* e *InterfaceBD*, no hay tampoco necesidad de modificar las clases.

Módulo Servicios

La clase *ManejadorServicio*, como se muestra en la Tabla 8.88, hace llamadas al contrato "2" de la clase *ManejadorRegistroUsuario* el cual pertenece al *SubsistemaRegistro*. La colaboración debe por lo tanto modificarse de manera correspondiente. Los cambios para la clase *ManejadorServicio* se muestran en la Tabla 8.99.

Clase: ManejadorServicio	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Servicios	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento	
2. Ofrecer Servicio	
ofrecerServicio	
Responsablidades Privadas	
registrar	SubsistemaRegistro (2)

Tabla 8.99. Tarjeta para la clase *ManejadorServicio* con responsabilidades, colaboraciones, jerarquías, contratos y subsistemas a partir de los casos de uso *RegistrarUsuario* y *RegistrarTarjeta*.

Protocolos

Una vez completadas las etapas anteriores, se debe detallar la especificación de cada clase hasta llegar a métodos y atributos finales. Aunque pudiéramos generar una especificación completa a nivel de "pseudo-código" o incluso código en un lenguaje particular, esto sería sobrespecificar el diseño ya que el diseño debe definir los aspectos más relevantes de la solución pero sin ser el código final. El diseño debe dar cierta libertad al programador o implementador siempre y cuando se mantenga dentro del marco de la arquitectura de objetos generada hasta el

momento. Entonces, ¿hasta donde debe llegar el diseño?. Nuevamente, la respuesta está relacionada con aspectos en el manejo de la complejidad. Si el diseño desarrollado ya resuelve los aspectos más importantes de la arquitectura, en particular todo lo relacionado con la asignación de responsabilidades, entonces ya hemos resuelto gran parte del problema. Sin embargo, existen otros aspectos que son fuente de complejidad más allá de las responsabilidades, estos incluyen los aspectos algorítmicos, estructuras de datos, entre otros junto con la adecuación a los demás aspectos consideramos dentro del "diseño de sistema" como se mencionó al inicio del capítulo. Cuando ya no existen decisiones "importantes" que pudieran afectar de manera importante la arquitectura del sistema entonces ya se puede continuar con la implementación.

Nosotros continuaremos hasta donde se considera que la arquitectura del sistema de reservaciones ya resuelve los aspectos que más afectan a toda la arquitectura. Esto incluye de manera importante la definición precisa de las responsabilidades públicas, ya que cualquier cambio en ellas afectará a todos los respectivos clientes. Para ello se define el concepto de *protocolo*, donde un *protocolo* corresponde al conjunto de *firmas* correspondientes a las distintas responsabilidades de una clase. El objetivo de los protocolos es refinar las responsabilidades hasta llegar a métodos precisos dando especial énfasis a las responsabilidades agrupadas en los contratos ofrecidos por las clases. En general, las responsabilidades privadas representan notas de implementación para un programador y no deben sobre-especificarse. Sin embargo, en algunos casos se debe generar protocolos para responsabilidades privadas. Por ejemplo, si una superclase abstracta será implementada por un programador y sus subclases implementadas por otro, las responsabilidades privadas usadas por sus subclases deben especificarse completamente.

Se busca también incrementar la reutilización en el sistema si logramos que los protocolos contenga sólo uno pocos parámetros, simplificando su comprensión e incrementando la probabilidad de descubrir nuevas bases para la herencia. Se debe escoger los nombres con cuidado, siguiendo un patrón uniforme en el momento de asignar estos nombres.

Durante esta etapa es común descubrir sitios en el diseño donde el modelo fue impreciso, incorrecto, o pobremente comprendido. Por lo tanto, puede que sea necesario tener que repetir etapas anteriores del proceso de diseño antes de poder completar las definiciones de clases y contratos.

A continuación se describen las clases principales incorporando protocolos en el Sistema de Reservaciones y en base a los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

InterfaceUsuario

La clase *InterfaceUsuario* define dos responsabilidades públicas, correspondientes a dos contratos, como se especificó anteriormente en la Tabla 8.96 y como se muestra en la Tabla 8.100.

1. Desplegar Pantalla	
desplegarPantalla	Pantalla (1) : PantallaPrincipal (1), PantallaServicio (1), PantallaCrearRegUsuario (1), PantallaObtenerRegUsuario (1), PantallaCrearRegTarjeta (1), PantallaObtenerRegTarjeta (1)
2. Enviar Evento	
enviarEvento	Manejador (1) : SubsistemaPrincipal (1), SubsistemaServicio (1), SubsistemaRegistro (1)

Tabla 8.100. Responsabilidades públicas para la clase *InterfaceUsuario* especificada en la Tabla 8.93.

El método "desplegarPantalla" es llamado por los diversos manejadores como parte del contrato "Desplegar Pantalla". Dado que esta responsabilidad es cliente del contrato "1", con el mismo nombre, de la clase *Pantalla* y es sobrecargada por las diversas pantallas, entonces es necesario definir como parte del protocolo un parámetro correspondiente a la pantalla a ser desplegada. Este parámetro de tipo *Pantalla* corresponde a un objeto ya instanciado por parte del manejador controlador de dicha *Pantalla*. Tenemos también que definir algún tipo de resultado, de manera sencilla podemos simplemente devolver un tipo nulo ("void"). Todo esto se muestra en la Tabla 8.101. Debe resaltarse que el parámetro corresponde a la superclase en lugar de incluir alguna pantalla particular, siendo esto esencial para lograr el polimorfismo.

1. Desplegar Pantalla	
desplegarPantalla(Pantalla)	Pantalla (1) : PantallaPrincipal (1), PantallaServicio (1), PantallaCrearRegUsuario (1), PantallaObtenerRegUsuario (1), PantallaCrearRegTarjeta (1), PantallaObtenerRegTarjeta (1)

Tabla 8.101. Responsabilidad “desplegarPantalla” con protocolo para la clase *InterfaceUsuario* especificada en la Tabla 8.100.

En el caso de la otra responsabilidad, “enviarEvento”, la situación es la opuesta. En este caso las diversas pantallas solicitan a la *InterfaceUsuario* enviar los eventos generados por el *Usuario*. Posteriormente, la *InterfaceUsuario* llama al contrato “1”, correspondiente a “Manejar Evento”, definido en la clase *Manejador* y sobrecargado por los diferentes manejadores. En principio, deberíamos definir dos parámetros, el evento generado y otro correspondiente a la clase *Manejador* a quien se le enviará posteriormente el evento. El evento puede corresponder a una clase tipo “Evento” que deberá corresponder a algún tipo definido posteriormente y de acuerdo al lenguaje de implementación. El tipo de devolución puede ser nuevamente un “void”. Esto se muestra en la Tabla 8.102.

2. Enviar Evento	
enviarEvento(Evento, Manejador) devuelve void	Manejador (1) : SubsistemaPrincipal (1), SubsistemaServicio (1), SubsistemaRegistro (1)

Tabla 8.102. Responsabilidad “enviarEvento” con protocolo para la clase *InterfaceUsuario* especificada en la Tabla 8.99.

La tarjeta de clase modificada para la clase *InterfaceUsuario* se muestra en la Tabla 8.103.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos:	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla(Pantalla) devuelve void	Pantalla (1) : PantallaPrincipal (1), PantallaServicio (1), PantallaCrearRegUsuario (1), PantallaObtenerRegUsuario (1), PantallaCrearRegTarjeta (1), PantallaObtenerRegTarjeta (1)
2. Enviar Evento	
enviarEvento(Evento, Manejador) devuelve void	Manejador (1) : SubsistemaPrincipal (1), SubsistemaServicio (1), SubsistemaRegistro (1)

Tabla 8.103. Tarjeta para la clase *InterfaceUsuario* con responsabilidades, colaboraciones, jerarquías, contratos, subsistemas y protocolos identificados de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

En la Tabla 8.67 se definieron las responsabilidades, asignadas a contratos y privadas, para la clase *Pantalla*, como se muestra en la Tabla 8.104.

1. Desplegar Pantalla	
desplegarPantalla	
Responsabilidades Privadas	
enviarEvento	InterfaceUsuario (2)

Tabla 8.104. Tarjeta para la superclase clase *Pantalla* con responsabilidades asignadas a contratos y privadas.

El contrato “desplegarPantalla” es llamado por la clase InterfaceUsuario solicitando a las diversas pantallas que sobrescriben la responsabilidad que se desplieguen. Dado que el contenido de la pantalla es conocida por las diversas pantallas, en principio, no es necesario agregar ningún parámetro. Adicionalmente, se puede devolver un tipo “void”. Esto se muestra en la Tabla 8.105.

1. Desplegar Pantalla	
desplegarPantalla() devuelve void	

Tabla 8.105. Tarjeta para la superclase clase *Pantalla* con protocolo para la responsabilidad “desplegarPantalla”.

En el caso de la responsabilidad privada “enviarEvento”, ésta es llamada localmente por lo cual se puede dejar el parámetro sin asignar y se puede devolver un tipo “void”. Esto se muestra en la Tabla 8.106.

Responsabilidades Privadas	
enviarEvento() devuelve void	InterfaceUsuario (2)

Tabla 8.106. Tarjeta para la superclase clase *Pantalla* con protocolo para la responsabilidad "enviarEvento".

La tarjeta de clase modificada para la clase *Pantalla* se muestra en la Tabla 8.107.

Clase: Pantalla	
Descripción: Pantalla heredada por las demás clases de tipo pantalla.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Abstracta	
Superclases:	
Subclases: PantallaPrincipal, PantallaServicio, PantallaRegUsuario, PantallaRegTarjeta	
Atributos:	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla() devuelve void	
Responsabilidades Privadas	
enviarEvento() devuelve void	InterfaceUsuario (2)

Tabla 8.107. Tarjeta para la superclase clase *Pantalla* con responsabilidades, colaboraciones, jerarquías, contratos, subsistemas y protocolos identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Principal

A partir de la Tabla 8.98 se muestran las diversas responsabilidades para la clase *Manejador*, como se muestra en la Tabla 8.108.

1. Manejar Evento	
manejarEvento	
Responsablidades Privadas	
desplegarPantalla	SubsistemaInterfaceUsuario (1)
ofrecerServicio	SubsistemaServicio (2)
salir	

Tabla 8.108. Tarjeta para la clase *Manejador* según se describió en la Tabla 8.95.

El contrato "Manejar Evento" es llamado por la clase *InterfaceUsuario* la cual debe enviar el evento generada por el usuario y enviada luego por las diversas pantallas. Por lo tanto debe incluirse un parámetro de tipo "Evento" y nuevamente podemos asignar un tipo "void" de devolución. Las demás responsabilidades son privadas y podemos dejar de asignar parámetros y definir un tipo "void" de devolución. Esto se muestra en la Tabla 8.109.

Clase: Manejador	
Descripción: Pantalla heredada por las demás clases de tipo pantalla.	
Módulo: Principal	
Estereotipo: Control	
Propiedades: Abstracta	
Superclases:	
Subclases: ManejadorPrincipal, ManejadorServicio, ManejadorRegistroUsuario, ManejadorRegistroTarjeta	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void	
Responsablidades Privadas	
desplegarPantalla() devuelve void	SubsistemaInterfaceUsuario (1)
ofrecerServicio() devuelve void	SubsistemaServicio (2)
salir() devuelve void	

Tabla 8.109. Tarjeta para la clase *Manejador* con responsabilidades, colaboraciones, jerarquías, contratos, subsistemas y protocolos identificadas de los diversos manejadores para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

A partir de la Tabla 8.97 podemos definir los protocolos para la clase *ManejadorPrincipal* a partir del contrato "Manejar Evento" y dos responsabilidades privadas. El contrato "Manejar Evento" contiene la responsabilidad "manejarEvento" la cual se debe definir de manera similar a la responsabilidad que es sobrescrita de la superclase Manjeador. Las dos responsabilidades privadas se pueden dejar sin parámetros y devolviendo un tipo "void", como se muestra en la Tabla 8.110.

Clase: ManejadorPrincipal	
Descripción: El manejador principal es el encargado de desplegar la pantalla principal de interacción con el usuario, y luego delegar las diferentes funciones a los manejadores especializados apropiados.	
Módulo: Principal	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void	
Responsablidades Privadas	
crearRegistroUsuario() devuelve void	SubsistemaRegistro (2)
validarRegistroUsuario() devuelve void	SubsistemaRegistro (2)

Tabla 8.110. Tarjeta para la clase *ManejadorPrincipal* con responsabilidades, colaboraciones, jerarquías, contratos, subsistemas y protocolos identificadas de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

La descripción de la clase *PantallaPrincipal* se mantiene igual ya que no incluye responsabilidades al igual que las demás pantallas.

Dominio

La descripción de la clase *Datos* se mantiene igual, ya que no incluye hasta el momento responsabilidades.

Registro

En el módulo de *Registro*, compuesto de los módulos *Usuario*, *Tarjeta* e *InterfaceBD*, sólo se modifican las clases de control pero no las de interface o entidad, como se verá a continuación.

Usuario

En la Tabla 8.75 se definieron las diferentes responsabilidades para la clase *ManejadoRegistroUsuario*, como se muestra en la Tabla 8.111.

Contratos	
1. Manejar Evento	
manejarEvento	
2. Registrar Usuario	
crearRegistroUsuario	InterfaceBaseDatosRegistro (1)
validarRegistroUsuario	InterfaceBaseDatosRegistro (1)
obtenerRegistroUsuario	InterfaceBaseDatosRegistro (1)
Responsabilidades Privadas	
actualizarRegistroUsuario	InterfaceBaseDatosRegistro (1)
eliminarRegistroUsuario	InterfaceBaseDatosRegistro (1)
registrarTarjeta	ManejadorRegistroTarjeta (2)

Tabla 8.111. Tarjeta para la clase *ManejadoRegistroUsuario* como se definió en la Tabla 8.75.

En el caso del contrato "Manejar Evento" debemos definir un protocolo para la responsabilidad "manejarEvento" similar a la de la clase *Manejador*, como se hizo con el *ManejadorPrincipal*.

En relación al contrato "Registrar Usuario" tenemos tres responsabilidades, "crearRegistroUsuario", "validarRegistroUsuario" y "obtenerRegistroUsuario". Las dos primeras responsabilidades son llamadas por el *ManejadorPrincipal* para iniciar transacciones de registro. En el caso de "crearRegistroUsuario", el *ManejadorPrincipal* solicita al *ManejadoRegistroUsuario* que cree un nuevo registro de usuario, algo que será

controlado completamente por este último. Dado que todo el control de la transacción está a cargo del *ManejadoRegistroUsuario*, no hay necesidad de agregar ningún parámetro y se puede devolver un tipo "void".
En el caso de "validarRegistroUsuario", el *ManejadorPrincipal* solicita al *ManejadoRegistroUsuario* que valide los datos del usuario los cuales son insertados en la *PantallaPrincipal*, la cual está a cargo del *ManejadorPrincipal*. El *ManejadorPrincipal* debe hacerle llegar los datos del usuario al *ManejadoRegistroUsuario*, algo que podemos incluir como dos parámetros de tipo "String" ("cadenas de caracteres"), correspondientes al nombre del usuario y su contraseña. Dado que el *ManejadorPrincipal* debe decidir en base al resultado si proceder con los servicios, también será necesario devolver algún tipo de valor que nos comunique si la solicitud fue aceptada, por lo tanto agregaremos un tipo "bolean" como resultado.
En el caso de la responsabilidad "obtenerRegistroUsuario", el *ManejadorServicio* solicita al *ManejadoRegistroUsuario* que obtenga el registro de usuario y continúe el procesamiento. En tal caso no se envía ningún parámetro y no es necesario esperar ninguna respuesta por lo cual la devolución la podemos dejar de tipo "void".
En el caso de las responsabilidades privadas, las podemos dejar sin ningún parámetro y devolviendo tipos "void".
Todos estos protocolos con la tarjeta de clase completa se muestran en la Tabla 8.112.

Clase: ManejadorRegistroUsuario	
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para poder utilizar el sistema.	
Módulo: Registro.Usuario	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void	
2. Registrar Usuario	
crearRegistroUsuario() devuelve void	InterfaceBaseDatosRegistro (1)
validarRegistroUsuario(String,String) devuelve boolean	InterfaceBaseDatosRegistro (1)
obtenerRegistroUsuario() devuelve void	InterfaceBaseDatosRegistro (1)
Responsabilidades Privadas	
actualizarRegistroUsuario() devuelve void	InterfaceBaseDatosRegistro (1)
eliminarRegistroUsuario() devuelve void	InterfaceBaseDatosRegistro (1)
registrarTarjeta() devuelve void	ManejadorRegistroTarjeta (2)

Tabla 8.112. Tarjeta para la clase *ManejadoRegistroUsuario* con responsabilidades, colaboraciones, jerarquías, contratos, subsistemas y protocolos identificadas de los casos de uso *RegistrarUsuario* y *ValidarUsuario*.

La descripción de las clases *PantallaRegUsuario*, *PantallaCrearRegUsuario*, *PantallaObtenerRegUsuario* y *RegistroUsuario* se mantienen igual.

Tarjeta

En la Tabla 8.80 se definieron las diferentes responsabilidades para la clase *ManejadoRegistroTarjeta*, como se muestra en la Tabla 8.113.

1. Manejar Evento	
manejarEvento	
2. Registrar Tarjeta	
registrarTarjeta	
Responsabilidades Privadas	
crearRegistroTarjeta	InterfaceBaseDatosRegistro (2)
obtenerRegistroTarjeta	InterfaceBaseDatosRegistro (2)
actualizarRegistroTarjeta	InterfaceBaseDatosRegistro (2)
eliminarRegistroTarjeta	InterfaceBaseDatosRegistro (2)

Tabla 8.113. Tarjeta para la clase *ManejadoRegistroTarjeta* como se definió en la Tabla 8.80.

En el caso del contrato "Manejar Evento" debemos definir un protocolo para la responsabilidad "manejarEvento" similar a la de la clase *Manejador*, como se hizo con el *ManejadorPrincipal* y también con el *ManejadorRegistroUsuario*.
En relación al contrato "Registrar Tarjeta", la responsabilidad "registrarTarjeta" es llamada por el *ManejadorRegistroUsuario*. Dado que debe haber alguna manera de relacionar el registro de usuario con el registro de tarjeta, podemos enviar como parámetro algún identificador, por ejemplo de tipo "String". El control del contrato continúa en el *ManejadoRegistroTarjeta* por lo cual el tipo a devolver puede ser "void".
Las responsabilidades privadas nuevamente se definen sin argumentos y con devolución de tipo "void".
Todos estos protocolos con la tarjeta de clase completa se muestran en la Tabla 8.114.

Clase: ManejadorRegistroTarjeta	
Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.	
Módulo: Registro.Tarjeta	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void	
2. Registrar Tarjeta	
registrarTarjeta(String) devuelve void	
Responsabilidades Privadas	
crearRegistroTarjeta() devuelve void	InterfaceBaseDatosRegistro (2)
obtenerRegistroTarjeta() devuelve void	InterfaceBaseDatosRegistro (2)
actualizarRegistroTarjeta() devuelve void	InterfaceBaseDatosRegistro (2)
eliminarRegistroTarjeta() devuelve void	InterfaceBaseDatosRegistro (2)

Tabla 8.114. Tarjeta para la clase *ManejadoRegistroTarjeta* con responsabilidades, colaboraciones, jerarquías, contratos, subsistemas y protocolos identificadas del caso de uso *RegistrarTarjeta*.

La descripción de las clases *PantallaRegTarjeta*, *PantallaCrearRegTarjeta* y *PantallaObtenerRegTarjeta* se actualizan de manera similar a la *PantallaPrincipal*. Por otro lado, *RegistroTarjeta* se actualiza de manera similar a *RegistroUsuario*.

Interface Base Datos Registro

En la Tabla 8.87 se definieron las diferentes responsabilidades para la clase *InterfaceBaseDatosRegistro*, como se muestra en la Tabla 8.115.

1. Registrar Usuario	
crearRegistroUsuario	BaseDatosRegistro
obtenerRegistroUsuario	BaseDatosRegistro
actualizarRegistroUsuario	BaseDatosRegistro
eliminarRegistroUsuario	BaseDatosRegistro
validarRegistroUsuario	BaseDatosRegistro
2. Registrar Tarjeta	
crearRegistroTarjeta	BaseDatosRegistro
obtenerRegistroTarjeta	BaseDatosRegistro
actualizarRegistroTarjeta	BaseDatosRegistro
eliminarRegistroTarjeta	BaseDatosRegistro

Tabla 8.115. Tarjeta para la clase *InterfaceBaseDatosRegistro* como se definió en la Tabla 8.87.

Se definen dos contratos, "Registrar Usuario" y "Registrar Tarjeta", los cuales son encargados de interactuar con al base de datos para el almacenamiento de los registros de usuario y registros de tarjeta, respectivamente. Todas las responsabilidades deben enviar parámetros correspondientes a los registros a ser guardados en la base de datos o incluso a ser devueltos, ya que podemos hacer esto último a través de los parámetros.

En el caso del contrato "Registrar Usuario" tendríamos como parámetro a la clase *RegistroUsuario* para las diversas responsabilidades con excepción de la validación. En el caso de "obtenerRegistroUsuario" es necesario también enviar algún identificador que especifique el registro particular que buscamos. En el caso de "validarRegistroUsuario", se tiene que enviar dos parámetros de tipo "String" correspondientes al nombre del usuario y su contraseña.
En el caso del contrato "Registrar Tarjeta" se haría algo similar con las responsabilidades, con la excepción que el parámetro sería de tipo *RegistroTarjeta*.
En general podemos devolver un tipo "void" con excepción de la validación que debe devolver un "boolean".
Todos estos protocolos con la tarjeta de clase completa se muestran en la Tabla 8.116.

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceDB	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos:	
Contratos	
1. Registrar Usuario	
crearRegistro(RegistroUsuario) devuelve void	BaseDatosRegistro
obtenerRegistro(RegistroUsuario) devuelve void	BaseDatosRegistro
actualizarRegistro(RegistroUsuario) devuelve void	BaseDatosRegistro
eliminarRegistro(RegistroUsuario) devuelve void	BaseDatosRegistro
validarRegistro(String, String) devuelve boolean	BaseDatosRegistro
2. Registrar Tarjeta	
crearRegistro(RegistroTarjeta) devuelve void	BaseDatosRegistro
obtenerRegistro(RegistroTarjeta) devuelve void	BaseDatosRegistro
actualizarRegistro(RegistroTarjeta) devuelve void	BaseDatosRegistro
eliminarRegistro(RegistroTarjeta) devuelve void	BaseDatosRegistro

Tabla 8.116. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades, colaboraciones, jerarquías, contratos y protocolos de escribir y leer información de registro de usuario y registro de tarjeta para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Servicios

En la Tabla 8.99 se definieron las diferentes responsabilidades para la clase *ManejadorServicio*, como se muestra en la Tabla 8.117.

1. Manejar Evento	
manejarEvento	
2. Ofrecer Servicio	
ofrecerServicio	
Responsablidades Privadas	
registrar	SubsistemaRegistro (2)

Tabla 8.117. Tarjeta para la clase *ManejadorServicio* como se definió en la Tabla 8.99.
En el caso del contrato "Manejar Evento" debemos definir un protocolo para la responsabilidad "manejarEvento" similar a la de la clase *Manejador*, como se hizo con los demás manejadores.
En relación al contrato "Ofrecer Servicio", la responsabilidad "ofrecerServicio" es llamada por el los demás manejadores, algo que inicia un servicio controlado totalmente por esta clase, por lo tanto no hay necesidad de incluir ningún parámetro e incluso se puede devolver un tipo "void".
La responsabilidad privada nuevamente se define sin argumentos y con devolución de tipo "void".
Todos estos protocolos con la tarjeta de clase completa se muestran en la Tabla 8.118.

Clase: ManejadorServicio
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa

mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Servicios	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void	
2. Ofrecer Servicio	
ofrecerServicio() devuelve void	
Responsablidades Privadas	
registrar() devuelve void	SubsistemaRegistro (2)

Tabla 8.118. Tarjeta para la clase *ManejadorServicio* con responsabilidades, colaboraciones, jerarquías, contratos, subsistemas y protocolos a partir de los casos de uso *RegistrarUsuario* y *RegistrarTarjeta*.

La descripción de la clase *PantallaServicio* se mantiene de manera similar a las demás pantallas.

Atributos

En las secciones anteriores se ha diseñado las clases hasta llegar a los protocolos de cada una de ellas. Estos protocolos describen las firmas para las responsabilidades, en otras palabras, las interfaces externas de la clase. Es difícil decidir en que momento termina el diseño y comienza la implementación. El objetivo general es lograr un diseño robusto que le de cierta flexibilidad al programador pero sin modificar de manera importante la arquitectura de diseño. Lo que haremos a continuación, es detallar un poco más nuestro diseño especificando *atributos* correspondientes a estructuras de datos y *algoritmos* que especifiquen la menara de implementar las diversas responsabilidades correspondientes a los múltiples contratos de la aplicación.

En general, los *atributos* corresponden a los aspectos estructurales de las clases, como son los valores (números y textos), junto con las referencias a otros objetos. Estos conceptos varía de gran manera entre lenguajes. Por ejemplo, en el caso de Java y C++ la distinción entre valores y referencias es clara. En el caso de lenguajes como Smalltalk, no hay distinción alguna dado que todos los atributos corresponden a referencias entre objetos. Los atributos correspondientes a referencias deben agregarse de acuerdo a las colaboraciones establecidas durante el diseño. Los atributos que guardan valores son lo último que se especifica en el diseño de objetos.

A continuación se describen los atributos esenciales para las clases del Sistema de Reservaciones y en base a los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

InterfaceUsuario

Como se pudo observar anteriormente, la clase *InterfaceUsuario* se relaciona primordialmente y a través de polimorfismo con las diversas pantallas y manejadores. Dado que sólo se desplegará una pantalla a la vez y se comunicará con un solo manejador en un momento específico, es suficiente definir dos tipos de referencias, *Manejador* para poder comunicarse con los diversos manejadores y *Pantalla* para comunicarse con las diversas pantallas. Por lo tanto, modificamos la tarjeta de clase correspondiente a la *InterfaceUsuario* como se muestra en la Tabla 8.119.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos: Manejador, Pantalla	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla(Pantalla) devuelve void	Pantalla (1) : PantallaPrincipal (1), PantallaServicio

	(1), PantallaCrearRegUsuario (1), PantallaObtenerRegUsuario (1), PantallaCrearRegTarjeta (1), PantallaObtenerRegTarjeta (1)
2. Enviar Evento	
enviarEvento(Evento, Manejador) devuelve void	Manejador (1) : SubsistemaPrincipal (1), SubsistemaServicio (1), SubsistemaRegistro (1)

Tabla 8.119. Tarjeta para la clase *InterfaceUsuario* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos y atributos identificados de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

De manera similar a la *InterfaceUsuario*, las diversas pantallas deben tener conocimiento de la *InterfaceUsuario* y de los manejadores que las controlan. Aunque se pudiera especificar el tipo particular de manejador que administra una pantalla en particular, podemos aprovechar y definir una referencia de tipo genérica a *Manejador* ya que cada pantalla es administrada por un solo manejador. Este conocimiento se implementa mediante atributos correspondientes a las clases anteriores, como se muestra en la Tabla 8.120.

Clase: Pantalla	
Descripción: Pantalla heredada por las demás clases de tipo pantalla.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Abstracta	
Superclases:	
Subclases: PantallaPrincipal, PantallaServicio, PantallaRegUsuario, PantallaRegTarjeta	
Atributos: InterfaceUsuario, Manejador	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla() devuelve void	
Responsabilidades Privadas	
enviarEvento() devuelve void	InterfaceUsuario (2)

Tabla 8.120. Tarjeta para la superclase *Pantalla* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos y atributos identificados de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Principal

De manera similar a la *InterfaceUsuario* y las pantallas, los manejadores deben poder accesar a la propia *InterfaceUsuario* y también a sus diferentes pantallas. Por tal motivo definimos dos atributos, uno de tipo *InterfaceUsuario* y otro de tipo *Pantalla* correspondiente a la pantalla actualmente siendo desplegada. Si revisamos un poco más, podemos apreciar que la gran mayoría de los menajadores requieren comuincarse con el *SubsistemaServicio* para ofrecer algún servicio. Por lo tanto, también se requiere un atributo de referencia de tipo *ManejadorServicio* el cual puede ser definido a este nivel en la superclase *Manejador*. Adicionalmente, vamos a agregar una referencia genérica para el manejador *padre* el cual reside a un nivel administrativo superior, en otras palabras, el manejador encargado de instanciar al siguiente nivel de manejadores y así sucesivamente. Dado que el manejador *padre* puede tener cualquier tipo dentro de la jerarquía de manejadores, lo agregaremos bajo el tipo genérico *Manejador*. Esta referencia es simpre buena tener, ya que un manejador particular puede tener acceso a cualquier otro manejador siempre y cuando no pierda la referencia a su superior. La clase *Manejador* se muestra en la Tabla 8.121.

Clase: Manejador	
Descripción: Pantalla heredada por las demás clases de tipo pantalla.	
Módulo: Principal	
Estereotipo: Control	
Propiedades: Abstracta	
Superclases:	
Subclases: ManejadorPrincipal, ManejadorServicio, ManejadorRegistroUsuario, ManejadorRegistroTarjeta	
Atributos: InterfaceUsuario, Pantalla, ManejadorServicio, Manejador	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void	

Responsablidades Privadas	
desplegarPantalla() devuelve void	SubsistemaInterfaceUsuario (1)
ofrecerServicio() devuelve void	SubsistemaServicio (2)
salir() devuelve void	

Tabla 8.121. Tarjeta para la clase *Manejador* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos y atributos identificados de los diversos manejadores para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

En el caso de la clase *ManejadorPrincipal* es necesario agegar una referencia particular a cada pantalla y manejador que pueda ser accesado desde este clase, tomando en cuenta los ya definidos a nivel de la superclase *Manejador*. Por lo tanto, defnimos un atributo de tipo *PantallaPrincipal*, un *ManejadorServicio* y otro *ManejadorRegistroUsuario*, como se muestra en la Tabla 8.122.

Clase: ManejadorPrincipal	
Descripción: El manejador principal es el encargado de desplegar la pantalla principal de interacción con el usuario, y luego delegar las diferentes funciones a los manejadores especializados apropiados.	
Módulo: Principal	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos: PantallaPrincipal, ManejadorServicio, ManejadorRegistroUsuario	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void	
Responsablidades Privadas	
crearRegistroUsuario() devuelve void	SubsistemaRegistro (2)
validarRegistroUsuario() devuelve void	SubsistemaRegistro (2)

Tabla 8.122. Tarjeta para la clase *ManejadorPrincipal* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos y atributos identificados de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

La clase *PantallaPrincipal* no agrega atributos más allá de aquellos definidos en la superclase *Pantalla* por lo cual no volvemos a repetir la descripción.

Dominio

La clase *Datos* es una superclase con estereotipo "entidad". Típicamente, estas clases se mantienen lo más limitadas posible en término de sus relaciones con otras clases por lo cual no agregamos atributos hasta el momento.

Registro

En el módulo de *Registro*, compuesto de los módulos *Usuario*, *Tarjeta* e *InterfaceBD*, sólo se modifican las clases de control pero no las de interface o entidad, como se verá a continuación.

Usuario

En el caso de la clase *ManejadoRegistroUsuario*, definimos referencias a las diversas pantallas bajo control de esta clase, *PantallaCrearRegUsuario* y *PantallaObtenerRegUsuario*, a la clase entidad donde se guardará la información manipulada por esta clase, *RegistroUsuario*, al *ManejadorRegistrTarjeta* el cual es instanciado por el *ManejadorRegistroUsuario* y a la clase *InterfaceBaseDatosRegistro* correspondiente a la clase borde que accesará la base de datos de registro. El resto de los atributos son heredados de la clase *Manejador*. La tarjeta de clase para *ManejadorRegistroUsuario* se muestra en la Tabla 8.123.

Clase: ManejadorRegistroUsuario
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para poder utilizar el sistema.
Módulo: Registro.Usuario
Estereotipo: Control
Propiedades: Concreta
Superclase: Manejador
Subclases:

Atributos: PantallaCrearRegUsuario, PantallaObtenerRegUsuario, RegistroUsuario, ManejadorRegistrTarjeta, InterfaceBaseDatosRegistro	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void	
2. Registrar Usuario	
crearRegistroUsuario() devuelve void	InterfaceBaseDatosRegistro (1)
validarRegistroUsuario(String,String) devuelve boolean	InterfaceBaseDatosRegistro (1)
obtenerRegistroUsuario() devuelve void	InterfaceBaseDatosRegistro (1)
Responsabilidades Privadas	
actualizarRegistroUsuario() devuelve void	InterfaceBaseDatosRegistro (1)
eliminarRegistroUsuario() devuelve void	InterfaceBaseDatosRegistro (1)
registrarTarjeta() devuelve void	ManejadorRegistroTarjeta (2)

Tabla 8.123. Tarjeta para la clase *ManejadoRegistroUsuario* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos y atributos identificados de los casos de uso *RegistrarUsuario* y *ValidarUsuario*.

La descripción de las clases *PantallaRegUsuario*, *PantallaCrearRegUsuario* y *PantallaObtenerRegUsuario* se mantienen igual.

La clase *RegistroUsuario* incluye los atributos asignados durante la identificación del dominio del problema y descrita en los casos de uso, como se muestra en la Tabla 8.124.

Clase: RegistroUsuario	
Descripción: Para poder utilizar el sistema de reservaciones, el usuario debe estar registrado con el sistema. El registro contiene información acerca del usuario que incluye nombre, dirección, colonia, ciudad, país, código postal, teléfono de casa, teléfono de oficina, fax, email, login y password.	
Módulo: Registro.Usuario	
Estereotipo: Entidad	
Propiedades: Concreta	
Superclases: Datos	
Subclases:	
Atributos: login, password, nombre, apellido, dirección, colonia, ciudad, país, CP, telCasa, telOficina, fax, email	

Tabla 8.124. Tarjeta para la clase *RegistroUsuario* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos y atributos para el caso de uso *RegistrarUsuario*.

Tarjeta

Los atributos para la clase *ManejadoRegistroTarjeta* serán considerados de manera similar al *ManejadorRegistroUsuario*. Incluiremos referencias a sus dos pantallas, *PantallaCrearRegTarjeta* y *PantallaObtenerRegTarjeta*, al *RegistroTarjeta* donde se guardará la información manipulada por el manejador, y a la *InterfaceBaseDatosRegistro* encargada de aaccesar a la base de datos. La tarjeta con atributos se muestra en la Tabla 8.125.

Clase: ManejadorRegistroTarjeta	
Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.	
Módulo: Registro.Tarjeta	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos: PantallaCrearRegTarjeta, PantallaObtenerRegTarjeta, RegistroTarjeta, InterfaceBaseDatosRegistro	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void	
2. Registrar Tarjeta	

registrarTarjeta(String) devuelve void	
Responsabilidades Privadas	
crearRegistroTarjeta() devuelve void	InterfaceBaseDatosRegistro (2)
obtenerRegistroTarjeta() devuelve void	InterfaceBaseDatosRegistro (2)
actualizarRegistroTarjeta() devuelve void	InterfaceBaseDatosRegistro (2)
eliminarRegistroTarjeta() devuelve void	InterfaceBaseDatosRegistro (2)

Tabla 8.125. Tarjeta para la clase *ManejadoRegistroTarjeta* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos y atributos identificados del caso de uso *RegistrarTarjeta*.

La descripción de las clases *PantallaRegTarjeta*, *PantallaCrearRegTarjeta* y *PantallaObtenerRegTarjeta* son similares a las anteriores.

La clase *RegistroTarjeta* incluye atributos también identificados durante la especificación del dominio del problema y descritos nuevamente en los casos de uso. Estos atributos se mueatran en la Tabla 8.126.

Clase: RegistroTarjeta	
Descripción: Para poder hacer un pago con una tarjeta de crédito, se debe tener un registro de tarjeta. El registro contiene información acerca de la tarjeta incluyendo nombre, número, expedidor y vencimiento. La tarjeta está ligada a un registro de usuario.	
Módulo: Registro.Tarjeta	
Estereotipo: Entidad	
Propiedades: Concreta	
Superclases: Datos	
Subclases:	
Atributos: login, nombre, número, tipo, fecha	

Tabla 8.126. Tarjeta para la clase *RegistroTarjeta* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos y atributos para el caso de uso *RegistrarTarjeta*.

Interface Base Datos Registro

La clase *InterfaceBaseDatosRegistro* no requiere atributos por el momento.

Servicios

La clase *ManejadorServicio* requiere únicamente una referencia a la clase *ManejadorRegistroUsuario* para poder referirle solicitudes relacionadas con registro, como se muestra en la Tabla 8.127.

Clase: ManejadorServicio	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Servicios	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void	
2. Ofrecer Servicio	
ofrecerServicio() devuelve void	
Responsablidades Privadas	
registrar() devuelve void	SubsistemaRegistro (2)

Tabla 8.127. Tarjeta para la clase *ManejadorServicio* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos y atributos a partir de los casos de uso *RegistrarUsuario* y *RegistrarTarjeta*.

La descripción de la clase *PantallaServicio* se mantiene igual, como se hizo con las demás pantallas.

Algoritmos

Los *algoritmos* definen la lógica utilizada por cada operación para resolver la responsabilidad a la cual corresponden. En general, muchas responsabilidades son suficientemente simples que no requieren de algoritmos, como son las responsabilidades de delegar entre los objetos para consultar o modificar los valores de los atributos. Este es mayormente el caso en nuestro ejemplo del sistema de reservaciones vuelos. Sin embargo, es especialmente importante especificar los algoritmos para implementar funciones de lógica más compleja, como ordenar un conjunto de números o cadenas, o hacer cálculos de intereses sobre cuentas bancarias. Los algoritmos pueden especificarse de manera declarativa o procedural dependiendo de su complejidad. Cuando se especifica el aloritmo de manera procedural, esto típicamente se hace mediante un diagrama de flujo. Por el contrario, un algoritmo especificado de manera declarativa, típicamente se hace mediante una especificación textual en cuyo caso se debe usar algún conocimiento adicional para implementar el algoritmo. Vale la pena resaltar que los aspectos algorítmicos pueden llegar a ser una de las fuentes de mayor complejidad en un sistema.
A continuación especificamos nuestros algoritmos de manera declarativa, principalmente coo un comentario sencillo.

InterfaceUsuario

La clase *InterfaceUsuario* se muestra en la Tabla 8.128.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos: Manejador, Pantalla	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla(Pantalla) devuelve void *Método encargado de desplegar las pantallas enviadas como parámetros. Se delega el despliegue particular a cada pantalla.*	Pantalla (1) : PantallaPrincipal (1), PantallaServicio (1), PantallaCrearRegUsuario (1), PantallaObtenerRegUsuario (1), PantallaCrearRegTarjeta (1), PantallaObtenerRegTarjeta (1)
2. Enviar Evento	
enviarEvento(Evento, Manejador) devuelve void *Método encargado de recibir eventos del sistema de ventanas a través de las diversas pantallas. Se envía el evento recibido a los distintos manejadores.*	Manejador (1) : SubsistemaPrincipal (1), SubsistemaServicio (1), SubsistemaRegistro (1)

Tabla 8.128. Tarjeta para la clase *InterfaceUsuario* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos, atributos y especificación de algoritmos identificados de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

La clase *Pantalla* se muestra en la Tabla 8.129.

Clase: Pantalla	
Descripción: Pantalla heredada por las demás clases de tipo pantalla.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Abstracta	
Superclases:	
Subclases: PantallaPrincipal, PantallaServicio, PantallaRegUsuario, PantallaRegTarjeta	
Atributos: InterfaceUsuario, Manejador	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla() devuelve void *Método encargado de desplegar la pantalla actual.*	
Responsabilidades Privadas	

enviarEvento() devuelve void *Método encargado de recibir eventos del sistema de ventanas. Se envía el evento recibido a la InterfaceUsuario.*	InterfaceUsuario (2)

Tabla 8.129. Tarjeta para la superclase *Pantalla* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos, atributos y especificación de algoritmos identificados de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Principal

La clase *Manejador* se muestra en la Tabla 8.130.

Clase: Manejador	
Descripción: Pantalla heredada por las demás clases de tipo pantalla.	
Módulo: Principal	
Estereotipo: Control	
Propiedades: Abstracta	
Superclases:	
Subclases: ManejadorPrincipal, ManejadorServicio, ManejadorRegistroUsuario, ManejadorRegistroTarjeta	
Atributos: InterfaceUsuario, Pantalla, ManejadorServicio, Manejador	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void *Método encargado de recibir eventos del sistema de ventanas a través de la InterfaceUsuario.*	
Responsablidades Privadas	
desplegarPantalla() devuelve void *Método encargado de desplegar las pantallas administradas por los manejadores. Se solicita al SubsistemaInterfaceUsuario que las despliegue.*	SubsistemaInterfaceUsuario (1)
ofrecerServicio() devuelve void *Método encargado de solicitar al SubsistemaServicio que ofrezca los servicios correspondientes.*	SubsistemaServicio (2)
salir() devuelve void *Método encargado de salir del sistema.*	

Tabla 8.130. Tarjeta para la clase *Manejador* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos, atributos y especificación de algoritmos identificados de los diversos manejadores para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

La clase *ManejadorPrincipal* se muestra en la Tabla 8.131.

Clase: ManejadorPrincipal	
Descripción: El manejador principal es el encargado de desplegar la pantalla principal de interacción con el usuario, y luego delegar las diferentes funciones a los manejadores especializados apropiados.	
Módulo: Principal	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos:	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void *Método sobrescrito de la clase Manejador, encargado de recibir eventos del sistema de ventanas a través de la InterfaceUsuario.*	
Responsablidades Privadas	
crearRegistroUsuario() devuelve void	SubsistemaRegistro (2)

Método encargado de solicitar al SubsistemaRegistro que de servicio al contrato de "Registrar Usuario".	
validarRegistroUsuario() devuelve void *Método encargado de solicitar al SubsistemaServicio que de servicio al contrato de "Ofrecer Servicio".*	SubsistemaRegistro (2)

Tabla 8.131. Tarjeta para la clase *ManejadorPrincipal* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos, atributos y especificación de algoritmos identificados de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

La descripción de la clase *PantallaPrincipal* se mantiene igual ya que no incluye responsabilidades al igual que las demás pantallas.

Dominio

La descripción de la clase *Datos* se mantiene igual, ya que no incluye hasta el momento responsabilidades.

Registro

En el módulo de *Registro*, compuesto de los módulos *Usuario*, *Tarjeta* e *InterfaceBD*, sólo se agregan especificaciones algorítmicas a las clases de control pero no las de interface o entidad, como se verá a continuación.

Usuario

La clase *ManejadoRegistroUsuario* se muestra en la Tabla 8.132.

Clase: ManejadorRegistroUsuario	
Descripción: El manejador de registro de usuario se encarga de todo lo relacionado con registro del usuario para poder utilizar el sistema.	
Módulo: Registro.Usuario	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos: PantallaCrearRegUsuario, PantallaObtenerRegUsuario, RegistroUsuario, ManejadorRegistrTarjeta, InterfaceBaseDatosRegistro	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void *Método sobrescrito de la clase Manejador, encargado de recibir eventos del sistema de ventanas a través de la InterfaceUsuario.*	
2. Registrar Usuario	
crearRegistroUsuario() devuelve void *Método encargado de solicitar a la InterfaceBaseDatosRegistro la creación de un nuevo RegistroUsuario a través del contrato de "Registrar Usuario"*	InterfaceBaseDatosRegistro (1)
validarRegistroUsuario(String,String) devuelve Boolean *Método encargado de solicitar a la InterfaceBaseDatosRegistro la validación de un usuario a través del contrato de "Registrar Usuario"*	InterfaceBaseDatosRegistro (1)
obtenerRegistroUsuario() devuelve void *Método encargado de solicitar a la InterfaceBaseDatosRegistro la obtención de un RegistroUsuario a través del contrato de "Registrar Usuario"*	InterfaceBaseDatosRegistro (1)
Responsabilidades Privadas	
actualizarRegistroUsuario() devuelve void *Método encargado de solicitar a la InterfaceBaseDatosRegistro la actualización de un RegistroUsuario a través del contrato de "Registrar Usuario"*	InterfaceBaseDatosRegistro (1)
eliminarRegistroUsuario() devuelve void *Método encargado de solicitar a la InterfaceBaseDatosRegistro la eliminación de un RegistroUsuario a través del contrato de "Registrar Usuario"*	InterfaceBaseDatosRegistro (1)
registrarTarjeta() devuelve void	ManejadorRegistroTarjeta (2)

Método encargado de solicitar a la ManejadorRegistroTarjeta que procese el contrato "Registrar Tarjeta"	

Tabla 8.132. Tarjeta para la clase *ManejadoRegistroUsuario* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos, atributos y especificación de algoritmos identificados de los casos de uso *RegistrarUsuario* y *ValidarUsuario*.

La descripción de las clases *PantallaRegUsuario, PantallaCrearRegUsuario, PantallaObtenerRegUsuario* y *RegistroUsuario* se mantienen igual.

Tarjeta

La clase *ManejadoRegistroTarjeta* se muestra en la Tabla 8.133.

Clase: ManejadorRegistroTarjeta	
Descripción: El manejador de registro de tarjeta se encarga de todo lo relacionado con registro de la tarjeta del usuario para poder pagar las reservaciones.	
Módulo: Registro.Tarjeta	
Estereotipo: Control	
Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Atributos: PantallaCrearRegTarjeta, PantallaObtenerRegTarjeta, RegistroTarjeta, InterfaceRegistro	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void *Método sobrescrito de la clase Manejador, encargado de recibir eventos del sistema de ventanas a través de la InterfaceUsuario.*	
2. Registrar Tarjeta	
registrarTarjeta(String) devuelve void *Método encargado de crear u obtener un RegistroTarjeta*	
Responsabilidades Privadas	
crearRegistroTarjeta() devuelve void *Método encargado de solicitar a la InterfaceBaseDatosRegistro la creación de un nuevo RegistroTarjeta a través del contrato de "Registrar Tarjeta"*	InterfaceBaseDatosRegistro (2)
obtenerRegistroTarjeta() devuelve void *Método encargado de solicitar a la InterfaceBaseDatosRegistro la obtención de un RegistroTarjeta a través del contrato de "Registrar Tarjeta"*	InterfaceBaseDatosRegistro (2)
actualizarRegistroTarjeta() devuelve void *Método encargado de solicitar a la InterfaceBaseDatosRegistro la actualización de un RegistroTarjeta a través del contrato de "Registrar Tarjeta"*	InterfaceBaseDatosRegistro (2)
eliminarRegistroTarjeta() devuelve void *Método encargado de solicitar a la InterfaceBaseDatosRegistro la eliminación de un RegistroTarjeta a través del contrato de "Registrar Tarjeta"*	InterfaceBaseDatosRegistro (2)

Tabla 8.133. Tarjeta para la clase *ManejadoRegistroTarjeta* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos, atributos y especificación de algoritmos identificados del caso de uso *RegistrarTarjeta*.

La descripción de las clases *PantallaRegTarjeta, PantallaCrearRegTarjeta, PantallaObtenerRegTarjeta* y *RegistroTarjeta* se mantienen igual.

Interface Base Datos Registro

La clase *InterfaceBaseDatosRegistro* se muestra en la Tabla 8.134.

Clase: InterfaceBaseDatosRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceDB	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos:	
Contratos	
1. Registrar Usuario	
crearRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la creación de un nuevo RegistroUsuario*	BaseDatosRegistro
obtenerRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la obtención de un RegistroUsuario*	BaseDatosRegistro
actualizarRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la actualización de un RegistroUsuario*	BaseDatosRegistro
eliminarRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la eliminación de un RegistroUsuario*	BaseDatosRegistro
validarRegistro(String, String) devuelve bolean *Método encargado de solicitar a la BaseDatosRegistro la validación de un usuario*	BaseDatosRegistro
2. Registrar Tarjeta	
crearRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la creación de un nuevo RegistroTarjeta*	BaseDatosRegistro
obtenerRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la obtención de un RegistroTarjeta*	BaseDatosRegistro
actualizarRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la actualización de un RegistroTarjeta*	BaseDatosRegistro
eliminarRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la eliminación de un RegistroTarjeta*	BaseDatosRegistro

Tabla 8.134. Tarjeta para la clase *InterfaceBaseDatosRegistro* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos, atributos y especificación de algoritmos identificados para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

Servicios

La clase *ManejadorServicio* se muestra en la Tabla 8.135.

Clase: ManejadorServicio
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.
Módulo: Servicios
Estereotipo: Control

Propiedades: Concreta	
Superclases: Manejador	
Subclases:	
Contratos	
1. Manejar Evento	
manejarEvento(Evento) devuelve void *Método sobrescrito de la clase Manejador, encargado de recibir eventos del sistema de ventanas a través de la InterfaceUsuario.*	
2. Ofrecer Servicio	
ofrecerServicio() devuelve void *Método encargado de hacer solicitudes de consultar, reservar y administración de registros*	
Responsablidades Privadas	
registrar() devuelve void *Método encargado de hacer la solicitud de administración de registros al SubsistemaRegistro*	SubsistemaRegistro (2)

Tabla 8.135. Tarjeta para la clase *ManejadorServicio* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos, atributos y especificación de algoritmos identificados a partir de los casos de uso *RegistrarUsuario* y *RegistrarTarjeta*.

La descripción de la clase *PantallaServicio* se mantiene de manera similar a las demás pantallas.

8.3 Diseño de Sistema

Hasta aquí, durante el diseño de objetos, hemos desarrollado un diseño detalaldo en base a la arquitectura de análisis especificada anteriormente. Estos aspectos del diseño fueron especificados de manera independiente al ambiente de implementación. Aunque no entraremos en detalles de implementación en este capítulo, es importante ya considerar dicho ambiente ya que este afectará la implementación final. En general, el diseño de sistema incluye diversos aspectos como:

?? Selección del lenguaje de programación a utilizarse, típicamente estructurados u orientados a objetos;
?? Incorporación de bibliotecas, como por ejemplo, interfaces gráficas (GUI), bibliotecas numéricas y de estructuras de datos;
?? Incorporación de una base de datos, típicamente relacionales, relacionales extendidos u orientados a objetos;
?? Incorporación de archivos, en sus diferentes formatos;
?? Consideraciones de procesamiento, como concurrencia, paralelismo, distribución y tiempo real;

Estos aspectos pueden variar radicalmente entre uno y otro sistema y también pueden afectar de manera importante la arquitectura final del sistema. En general existen diversos enfoques para la incorporación del ambiente de implementación a la arquitectura del sistema: (i) agregando clases abstractas o interfaces que luego serán especializadas según el ambiente de implementación particular; (ii) instanciando objetos especializados que administren los aspectos particulares del ambiente de implementación particular; y (iii) configurando múltiples versiones del sistema correspondientes a diferentes plataformas. Este es el enfoque más flexible, aunque por lo general el de mayor costo de desarrollo.

En este capítulo simplificaremos de gran manera el efecto del ambiente de implementación. Utilizaremos a Java como lenguaje de programación bajo un procesamiento secuencial y con interfaces gráficas sencillas escritas en Java. También mostraremos cómo integrar el sistema con bases de datos y archivos externos.

Lenguajes de Programación

Aunque no es obligatorio implementar un desarrollo orientado a objetos mediante lenguajes de programación orientada a objetos, es obviamente más natural hacerlo de tal modo. En el caso de los lenguajes orientados a objetos, existe un apoyo directo a los conceptos y mecanismos fundamentales del análisis orientado a objetos: encapsulamiento, clasificación, generalización y polimorfismo. Sin embargo, se puede generalmente traducir cualquier concepto o mecanismo orientado a objetos a otros existentes en los lenguajes no orientados a objetos. El problema con otros tipos de lenguajes no es su poder de computación, si no en su expresibilidad, conveniencia, protección contra errores, y mantenimiento. Un lenguaje orientado a objetos hace que la escritura, mantenimiento, y

extensión de los programas sea más fácil y segura, ya que ejecuta tareas que un programador de un lenguaje no orientado a objetos tendría que hacer manualmente.
Aunque esta decisión se ha simiplificado hoy en día gracias a lenguajes como Java, no todos los lenguajes de programación orientados a objetos implementan de la misma forma los diferentes conceptos de la orientación a objetos. Existen aspectos, como el manejo del encapsulamiento, referencias, herencia múltiple, y otros aspectos que varían de manera importante entre lenguajes.
Como parte de la implementación de las clases en Java definiremos los siguientes aspectos:

?? Encapsulamiento o *visibilidad* de los métodos tanto como los atributos mediante modificadores de tipo *public* en el caso de contratos y responsabilidades públicas, *private* en el caso de responsabilidades y atributos privados, y *protected* en el caso de responsabilidades y atributos privados definidos a nivel de una superclase y que sean necesarios de acceder a nivel de sus subclase.
?? Protocolos correspondientes a tipos primitivos existentes en Java, como *int, float, boolean*, y objetos con tipos predefinidos en Java, como *String*, *Vector*.

De manera general, la implementación se basará en definiciones y manejos estándares de Java.

Interfaces Gráficas

Las interfaces gráficas tienen como objetivo esencial administrar la interacción con el usuario mediante elementos gráficos, como lo son los botones, menús y textos. En general, aplicaciones interactivas donde el control del ratón y el teclado juegan un papel primordial de control, son conocidos como sistemas controlados o dirigidos por *eventos*. Estos eventos corresponden al movimiento del ratón, como oprimir o soltar uno de sus botones, oprimir una tecla, junto con eventos que no son directamente iniciados por el usuario, como los eventos de desplegar una pantalla o interrumpir un programa. Desarrollar un sistema dirigido por eventos significa que la aplicación debe desde un inicio considerar un diseño adecuado. Por ejemplo, en el caso de Java, se debe inicialmente escoger una de sus bibliotecas gráficas, como AWT o Swing, para luego utilizar el manejo apropiado a través de clases como *Frame*, *Canvas*, *Panel*, *Button*, *Event*, etc. Más allá de los elementos o clase particuales de estas bibliotecas, también se afecta la lógica de diseño, ya que se debe contemplar, por ejemplo, en que momentos es apropiado procesar nuevos eventos y cómo se inicializará el sistema.
Para evitar mayor complejidad, nos limitaremos a ventanas relativamente sencillas, tanto a nivel de diseño gráfico como del tipo de elementos internos que incorporaremos. El diseño se basará en el prototipo gráfico descrito anteriormente en el Capítulo 5, donde creamos un solo marco de ventana (*frame*) el cual muestra las diferentes pantallas en diferentes momentos, todas dentro del mismo marco. Dada esta decisión, que obviamente no es la única alternativa, tendremos básicamente una clase controladora única de la ventana, la clase *InterfaceUsuario* anteriormente definida. Si hubieran múltiples ventanas independientes (en el "desktop") entonces deberíamos definir a cada marco como una clase controladora, correspondiente a cada pantalla, para luego tener una clase controladora para los múltiples marcos. Dada nuestra decisión, las pantallas se vuelven elementos internos de la ventana única y bajo el control de la *InterfaceUsuario*. ¿Cómo afecta esto a diseño de objetos? Se afecta de la siguiente manera. En lugar de que cada pantalla reciba un evento por parte del usuario (administrado a través el sistema de ventanas del sistema operativo), podemos hacer que todos los eventos sean recibidos directamente por la *InterfaceUsuario*, volviendo a las pantallas más pasivas, con menor conocimiento de su entorno y más sencillas de manipular. En otras palabras convertimos las pantallas en elementos exclusivamente visuales quitándole todo el conocimiento necesario para manipular eventos generados externamente.
Esto directamente afecta las responsabilidades de la clase *InterfaceUsuario* y de las las pantallas. Los cambios principales serán en relación al cliente del contrato "2" el cual será el manejador de ventanas del sistema en lugar de las distintas pantallas. Esto significa que el protocolo de la responsabilidad *enviarEvento* correspondiente al contrato "2" de la clase *InterfaceUsuario* deberá modificarse, tomado de su estado anterior definido en la Tabla 8.103 y mostrado nuevamente en la Tabla 8.136.

2. Enviar Evento	
enviarEvento(Evento, Manejador) devuelve void	Manejador (1) : SubsistemaPrincipal (1), SubsistemaServicio (1), SubsistemaRegistro (1)

Tabla 8.136. Contrato *enviarEvento* para la clase *InterfaceUsuario* definido en la Tabla 8.103.

Dado que la llamada al contrato "2" es externa a nuestro sistema, no habrá conocimiento sobre el manejador que controla la pantalla actual. Por lo tanto omitiremos el parámetro *Manejador* en el protocolo, como se muestra en la Tabla 8.137.

2. Enviar Evento	
enviarEvento(Evento) devuelve void	Manejador (1) : SubsistemaPrincipal (1),

	SubsistemaServicio (1), SubsistemaRegistro (1)

Tabla 8.137. Contrato *enviarEvento* con protocolo revisado para la clase *InterfaceUsuario* definido en la Tabla 8.103.

La clase *InterfaceUsuario* modificada se muestra en la Tabla 8.138.

Clase: InterfaceUsuario	
Descripción: Toda la interacción con el usuario se hace por medio de la interface de usuario.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos: Manejador manejador, Pantalla pantalla	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla(Pantalla) devuelve void *Método encargado de desplegar las pantallas enviadas como parámetros. Se delega el despliegue particular a cada pantalla.*	Pantalla (1) : PantallaPrincipal (1), PantallaServicio (1), PantallaCrearRegUsuario (1), PantallaObtenerRegUsuario (1), PantallaCrearRegTarjeta (1), PantallaObtenerRegTarjeta (1)
2. Enviar Evento	
enviarEvento(Evento) devuelve void *Método encargado de recibir eventos del sistema de ventanas a través de las diversas pantallas. Se envía el evento recibido a los distintos manejadores.*	Manejador (1) : SubsistemaPrincipal (1), SubsistemaServicio (1), SubsistemaRegistro (1)

Tabla 8.138. Tarjeta para la clase *InterfaceUsuario* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos, atributos y algoritmos identificados de los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

La clase *Pantalla* incluía una responsabilidad privada *enviarEvento*, la cual colaboraba con el contrato "2", con el mismo nombre, perteneciente a la clase *InterfaceUsuario*. Dado nuestras últimas modificaciones en el diseño gráfico, esta responsabilidad privada ya no será necesaria y la clase *InterfaceUsuario* recibirá directamente la llamada al contrato "2" por parte del manejador de ventanas de la máquina. (De por si, la responsabilidad privada *enviarEvento* definida en la clase *Pantalla*, tendría que ser pública para que el manejador de ventanas pueda enviar los eventos.) En resumen, se eliminará la responsabilidad *enviarEvento* en la clase *Pantalla*, como se muestra en la Tabla 8.139.

Clase: Pantalla	
Descripción: Pantalla heredada por las demás clases de tipo pantalla.	
Módulo: InterfaceUsuario	
Estereotipo: Borde	
Propiedades: Abstracta	
Superclases:	
Subclases: PantallaPrincipal, PantallaServicio, PantallaRegUsuario, PantallaRegTarjeta	
Atributos:	
Contratos	
1. Desplegar Pantalla	
desplegarPantalla() devuelve void *Método encargado de desplegar la pantalla actual.*	

Tabla 8.139. Tarjeta para la superclase clase *Pantalla* modificada de acuerdo al diseño gráfico. Se eliminó la responsabilidad privada *enviarEvento*.

Bases de Datos

Las bases de datos siempre juegan un papel fundamental en los sistemas de información. Una decisión estratégica importante en tal contexto es si utilizar bases de datos relacionales u orientadas a objetos. Dado su amplia

utilización, seleccionaremos para nuestro desarrollo una base de datos relacional, utilizando el lenguaje SQL para su interacción. En general, se consideran tres los modelos de bases de datos principales:

?? **Modelo Relacional** definiendo una colección de tablas donde cada tabla tiene un número específico de columnas y un número arbitrario de filas. Cada elemento de la tabla guarda un valor de tipo primitivo, como enteros o cadenas. De tal manera, cada objeto se representa como una fila en una tabla y donde cada columna corresponde a un atributo distinto en el objeto. El lenguaje de consultas se basa en operaciones simples, donde la simplicidad del modelo es un beneficio pero también es una limitación.

?? **Modelo Relacional Extendido** extendiendo el modelo relacional mediante procedimientos, objetos, versiones, y otras nuevas capabilidades. No hay un solo modelo relacional extendido, hay más bien una variedad de ellos, aunque todos comparten las tablas y consultas básicas del modelo relacional. Todos incorporan algo de "objetos" y todos tienen la habilidad de guardar procedimientos al igual que datos en la base de datos.

?? **Modelo Orientado a Objetos** definiendo un modelo orientado a objetos donde varía el tipo de encapsulamiento de los datos y los procedimientos en el objeto. El termino "orientado a objetos" varía también según los autores. En el caso del modelo orientado a objetos, los objetos pasan a tener *persistencia* más allá de su existencia durante la ejecución del programa.

En general, el diseño de bases de datos es un aspecto crucial en los sistemas de información. Las base de datos son repositorios de datos guardados en uno o más archivos. Existen sistemas de manejo de bases de datos (DBMS, OODBMS en el caso de objetos) para la administranción de los repositorios de datos permanentes, lo cual da un apoyo importante en los siguientes aspectos:

?? recuperación de caída: protección ante fallas de hardware y errores de usuarios.

?? múltiples usuarios: acceso concurrente para diversos usuarios.

?? múltiples aplicaciones: acceso concurrente de lectura y escritura de datos, facilitando la comunicación entre las diferentes aplicaciones.

?? seguridad: protección contra acceso no autorizados de lectura o escritura.

?? integridad: reglas que deben ser satisfechas para controlar la calidad de los datos más allá del control particular de la aplicación.

?? extensibilidad: mecanismos que permitan extender la arquitectura de la base de datos sin interrumpir su ejecución.

?? distribución de datos: distribución de los datos en diferentes lugares, organizaciones y plataformas de hardware.

Durante el diseño de la base de datos se debe traducir el modelo del dominio del problema a un modelos de tablas. Existen diversos enfoques de diseño, incluyendo aspectos relacionados con las correspondencia de asociaciones y generalización a tablas. Cada tabla derivada de una clase debe incluir un identificador para la llave primaria, y uno o más identificadores para la llave primaria de la tabla derivada de las asociaciones. La estrategia es compatible con la orientación a objetos, donde los objetos tienen una identidad aparte de sus propiedades. Existen beneficios en el uso de identificadors, ya que éstos son inmutables y completamente independientes a cambios en los valores de los datos y su lugar físico. Aunque cada clase puede corresponder a una o más tablas, por lo general se diseñan tablas correspondientes a un objeto completo. Simplificaremos la tarea de diseño, creando una tabla correspondiente a cada clase entidad del dominio del problema y manteniendo las asociaciones entre clases. Obviamente, el diseño de tablas puede optimizarse para un acceso más eficiente, algo que no trataremos aquí.

Dado que nuestra descripción se ha concentrado hasta el momento a lo referente a registro de usuario y tarjeta, nos limitaremos a mostrar el diseño de dichas tablas. Por ejemplo, en la Tabla 8.140, se muestra el diseño de la tabla para el *RegistroUsuario*, donde *login* es definida como la llave primaria de la tabla. La primera fila representa los nombres de los campos, mientras que la segunda muestra un ejemplo de datos.

login	password	nombre	apellido	dirección	ciudad	país	CP	telCasa	telOf	fax	email
alfredo	awr	Alfredo	Weitzenfeld	Río Hondo	México DF	México	01000	1234	5678	9012	alfredo@itam.mx

Tabla 8.140. Diseño de la base de datos para la tabla correspondiente a la clase *RegistroUsuario*.

En la Tabla 8.141, se muestra el diseño de la tabla para el *RegistroTarjeta*, donde se incluye *login* como referencia a la llave primaria de la tabla de *RegistroUsuario*.

login	nombre	número	tipo	fecha
alfredo	Alfredo Weitzenfeld	1234-5678-9012	Visa	120205

Tabla 8.141. Diseño de la base de datos para la tabla correspondiente a la clase *RegistroTarjeta*.

En la Figura 8.32 se muestra de manera esquemática la relación entre las tablas *RegistroUsuario* y *RegistroTarjeta*.

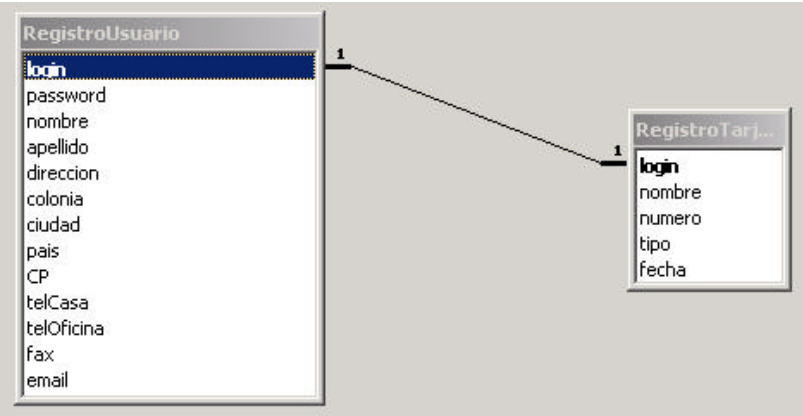

Figura 8.32. Diagrama mostrando la relación esquemática entre las tablas *RegistroUsuario* y *RegistroTarjeta*.

Archivos

Aunque es siempre mas efectivo trabajar con bases de datos, siempre es posible utilizar archivos, en particular cuando la especificación del sistema así lo requiera. De tal forma, aprovecharemos para extender el manejo de base de datos y mostrar como se puede lograr un objetivo similar a través de archivos. Este ejemplo también servirá para mostrar el poder de la extensibilidad definiendo una jerarquía de herencia a nivel de las clases borde para accesar tanto bases de datos como archivos de manera casi transparente. Para ello crearemos una nueva superclase *InterfaceRegistro*, de la cual heredará la clase *InterfaceBaseDatosRegistro* y nuestra nueva clase *InterfaceArchivoRegistro*. La superclase la definiremos como abstracta y deberá tener contratos y responsabilidades públicas similares a las ya definidas en la clase *InterfaceBaseDatosRegistro*. Esta clase se muestra en la Tabla 8.142.

Clase: InterfaceRegistro	
Descripción: Superclase para las interfaces a base de datos de registro y archivos.	
Módulo: Registro.InterfaceDB	
Estereotipo: Borde	
Propiedades: Abstracta	
Superclases:	
Subclases:	
Atributos:	
Contratos	
1. Registrar Usuario	
crearRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la creación de un nuevo RegistroUsuario*	BaseDatosRegistro
obtenerRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la obtención de un RegistroUsuario*	BaseDatosRegistro
actualizarRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la actualización de un RegistroUsuario*	BaseDatosRegistro
eliminarRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la eliminación de un RegistroUsuario*	BaseDatosRegistro
validarRegistro(String, String) devuelve bolean *Método encargado de solicitar a la BaseDatosRegistro la validación de un usuario*	BaseDatosRegistro
2. Registrar Tarjeta	
crearRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la creación de un nuevo RegistroTarjeta*	BaseDatosRegistro

obtenerRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la obtención de un RegistroTarjeta*	BaseDatosRegistro
actualizarRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la actualización de un RegistroTarjeta*	BaseDatosRegistro
eliminarRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la eliminación de un RegistroTarjeta*	BaseDatosRegistro

Tabla 8.142. Tarjeta para la clase *InterfaceRegistro* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos, atributos y algoritmos para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

En el caso de la base de datos, la clase *InterfaceBaseDatosRegistro* se comunica con el DBMS (manejador del sistema de la base de datos) para hacer solicitudes a cualquier tabla. Sin embargo, en el caso de los archivos, tal manejador no existe y será necesario hacerlo de manera manual. Por tal motivo, de manera adicional a la clase *InterfaceArchivoRegistro*, crearemos otra nueva clase *ArchivoRegistro* la cual se encargará de adminsitrar los diferentes archivos, instanciando una por archivo manipulado. De tal manera modificaremos las colaboraciones en la *InterfaceArchivoRegistro* para hacerlas con la clase *ArchivoRegistro* en lugar de *BaseDatosRegistro*, como se muestra en la Tabla 8.143. Nótese que todos los contratos deben ser identicos a los de la superclase *InterfaceRegistro* al igual que *InterfaceBaseDatosRegistro* para poder lograr la sobrescritura y aprovechar el polimorfismo.

Clase: InterfaceArchivoRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceDB	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos:	
Contratos	
1. Registrar Usuario	
crearRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la creación de un nuevo RegistroUsuario*	ArchivoRegistro
obtenerRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la obtención de un RegistroUsuario*	ArchivoRegistro
actualizarRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la actualización de un RegistroUsuario*	ArchivoRegistro
eliminarRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la eliminación de un RegistroUsuario*	ArchivoRegistro
validarRegistro(String, String) devuelve bolean *Método encargado de solicitar a la BaseDatosRegistro la validación de un usuario*	ArchivoRegistro
2. Registrar Tarjeta	
crearRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la creación de un nuevo RegistroTarjeta*	ArchivoRegistro
obtenerRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la obtención de un RegistroTarjeta*	ArchivoRegistro

actualizarRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la actualización de un RegistroTarjeta*	ArchivoRegistro
eliminarRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la eliminación de un RegistroTarjeta*	ArchivoRegistro

Tabla 8.143. Tarjeta para la clase *InterfaceArchivoRegistro* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos, atributos y algoritmos para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

La clase ArchivoRegistro de se muestra en la Tabla 8.144.

Clase: ArchivoRegistro	
Descripción: La información de cada usuario se almacena en la base de datos de registro la cual se accesa mediante la interface de la base de datos de registro. Esto permite validar a los distintos usuarios además de guardar información sobre la tarjeta de crédito para pagos en línea.	
Módulo: Registro.InterfaceDB	
Estereotipo: Borde	
Propiedades: Concreta	
Superclases:	
Subclases:	
Atributos:	
Contratos	
1. Registrar Usuario	
crearRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la creación de un nuevo RegistroUsuario*	BaseDatosRegistro
obtenerRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la obtención de un RegistroUsuario*	BaseDatosRegistro
actualizarRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la actualización de un RegistroUsuario*	BaseDatosRegistro
eliminarRegistro(RegistroUsuario) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la eliminación de un RegistroUsuario*	BaseDatosRegistro
validarRegistro(String, String) devuelve bolean *Método encargado de solicitar a la BaseDatosRegistro la validación de un usuario*	BaseDatosRegistro
2. Registrar Tarjeta	
crearRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la creación de un nuevo RegistroTarjeta*	BaseDatosRegistro
obtenerRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la obtención de un RegistroTarjeta*	BaseDatosRegistro
actualizarRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la actualización de un RegistroTarjeta*	BaseDatosRegistro
eliminarRegistro(RegistroTarjeta) devuelve void *Método encargado de solicitar a la BaseDatosRegistro la eliminación de un RegistroTarjeta*	BaseDatosRegistro

Tabla 8.144. Tarjeta para la clase *InterfaceArchivoRegistro* con responsabilidades, colaboraciones, jerarquías, contratos, protocolos, atributos y algoritmos para los casos de uso *RegistrarUsuario*, *ValidarUsuario* y *RegistrarTarjeta*.

A continuación mostramos un ejemplo de cómo se vería un archivo correspondiente al *RegistroUsuario*. La primera línea especifica el número de registros guardado en el archivo, en este caso "2". Los diversos valores de cada registro pueden guardarse como texto, en este caso delimitado mediante una líena vertical "|".

```
2
alfredo|awr         |Alfredo|Weitzenfeld|Rio         Hondo         #1|San         Angel
Tizapan|Mexico|MEXICO|01000|6284000|6284000 x3614|6162211|alfredo@itam.mx|
reservas|awr        |Sistema|Reservaciones|Rio        Hondo        #1|San        Angel
Tizapan|Mexico|MEXICO|01000|6284000|6284000 x3614|6162211|alfredo@itam.mx|
```

De manera similar, el archivo correspondiente al *RegistroTarjeta* se muestra a continuación,

```
2
alfredo|Alfredo Weitzenfeld|123456789|MasterCard|01/05|
reservas|Alfredo Weitzenfeld|987654321|Visa|02/4|
```

Nótese que la asociación entre ambos archivos es mediante el campo de "login", el primer campo en ambos archivos.

8.4 Revisión Final del Diseño

Como parte de la revisión final, decidimos hacer ciertas optimizaciones menores en el diseño. Una de estas optimizaciones es a nivel de acceso a la base de datos de registro. En lugar de hacer primero una validación de usuario y luego obtener el registro de la base de datos, podemos incroporar ambas responsabildiades en una sola. Por lo tanto, al momento de validar un registro, obtenemos los datos completos registrados para el usuario de manera inmediata.

8.5 Diagramas de Secuencias del Diseño

Una vez completado tanto el diseño de objetos como el del sistema, es posible describir los casos de uso del análisis en base a los protocolos de clases anteriormente definidos. Esto es muy importante ya que permita revisar que el diseño esté lógicamente completo. Para ello se describen los casos de uso mediante diagramas de secuencia, los cuales se pueden referir directamente a las clases, o incluso a partir de la interacción entre subsistemas. Esto es a menudo una técnica muy útil ya que se puede diseñar un subsistema mostrando sólo las interfaces de los subsistemas relacionados. A un nivel más detallado se pueden mostrar las interacciones internas del subsistema. Normalmente, los *eventos* en el diagrama corresponden a los protocolos antriromente diseñados y se especifican exactamente como se verían en el código final.

A continuación se muestran los diagramas de secuencia para el sistema de reservaciones de vuelo para los casos de uso *Registrar Usuario* y *Registrar Tarjeta*. Nótese como estos diagramas extienden con detalles adicionales los diagramas de secuencias generados anteriormente durante el modelo de análisis.

Registrar Usuario: Crear Registro Usuario

El diagrama de secuencia para el caso de uso *Registrar Usuario*, subflujo *Registrarse por Primera Vez* se muestra en la Figura 8.33.

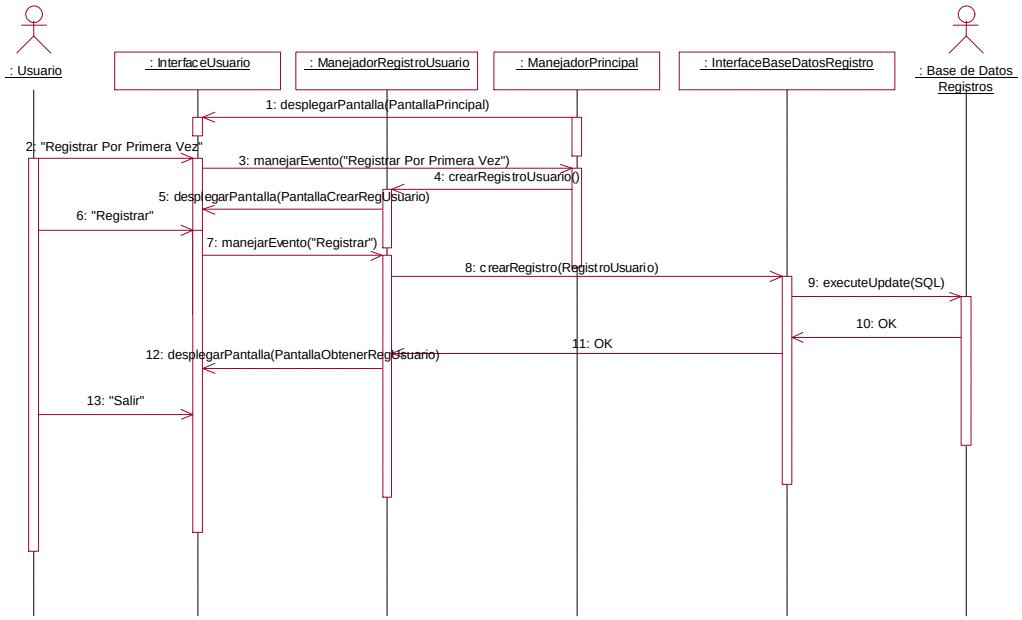

Figura 8.33. Diagrama de secuencias para el caso de uso *Registrar Usuario*, subflujo *Registrarse por Primera Vez.*

Registrar Usuario: Actualizar Registro Usuario

El diagrama de secuencia para el caso de uso *Registrar Usuario*, subflujo *Actualizar Registro Usuario* se muestra en la Figura 8.34.

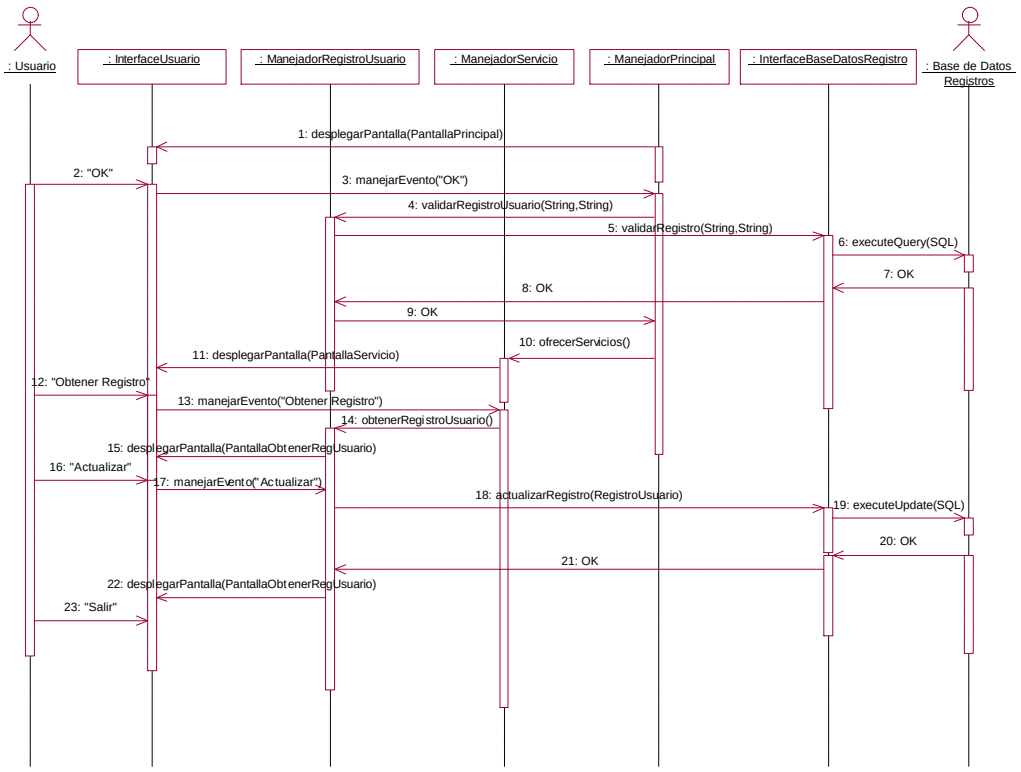

Figura 8.34. Diagrama de secuencias para el caso de uso *Registrar Usuario*, subflujo *Actualizar Registro Usuario*.

Registrar Usuario: Eliminar Registro Usuario

El diagrama de secuencia para el caso de uso *Registrar Usuario*, subflujo *Eliminar Registro Usuario* se muestra en la Figura 8.35.

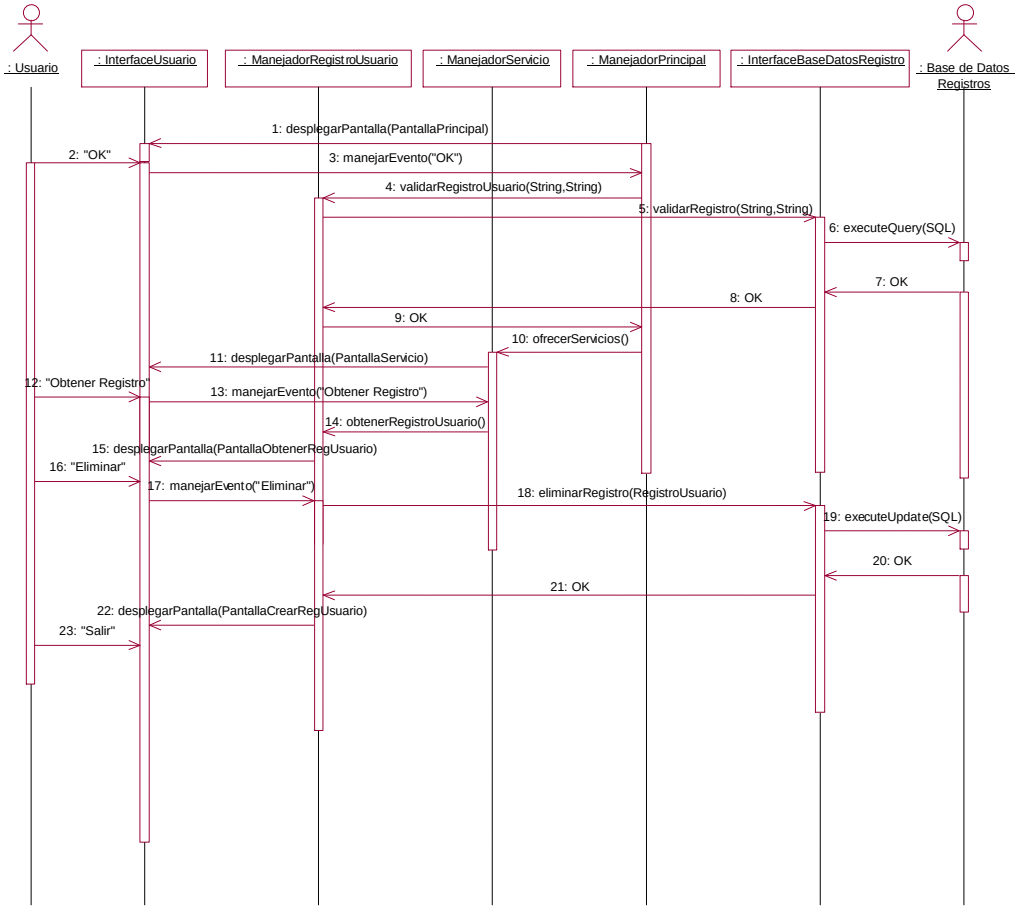

Figura 8.35. Diagrama de secuencias para el caso de uso *Registrar Usuario*, subflujo *Eliminar Registro Usuario*.

Validar Usuario

El diagrama de secuencia para el caso de uso *Validar Usuario*, se muestra en la Figura 8.36.

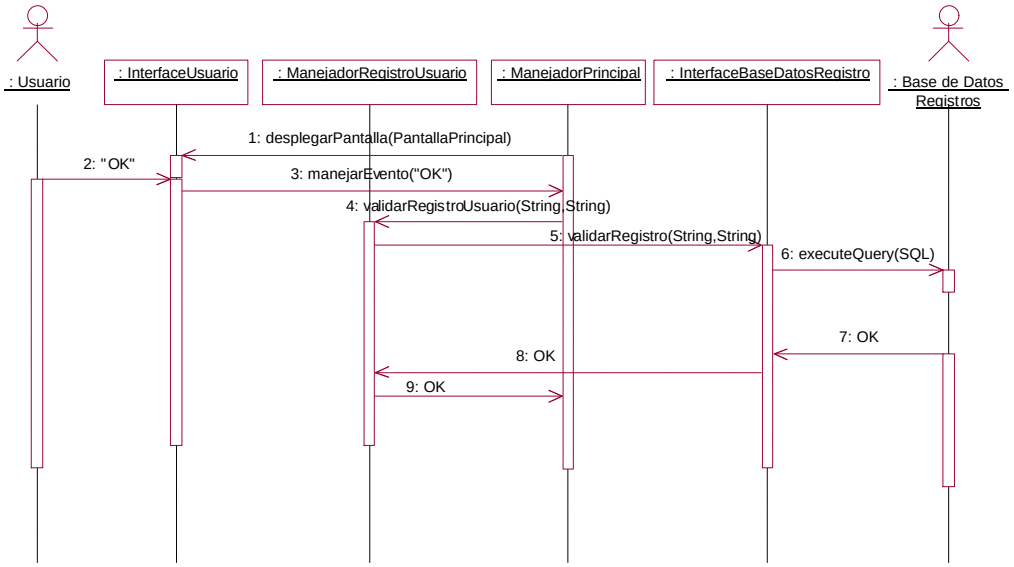

Figura 8.36. Diagrama de secuencias para el caso de uso *Validar Usuario*.

Registrar Tarjeta: Crear Registro Tarjeta

El diagrama de secuencia para el caso de uso *Registrar Tarjeta*, subflujo *Crear Registro Tarjeta* se muestra en la Figura 8.37.

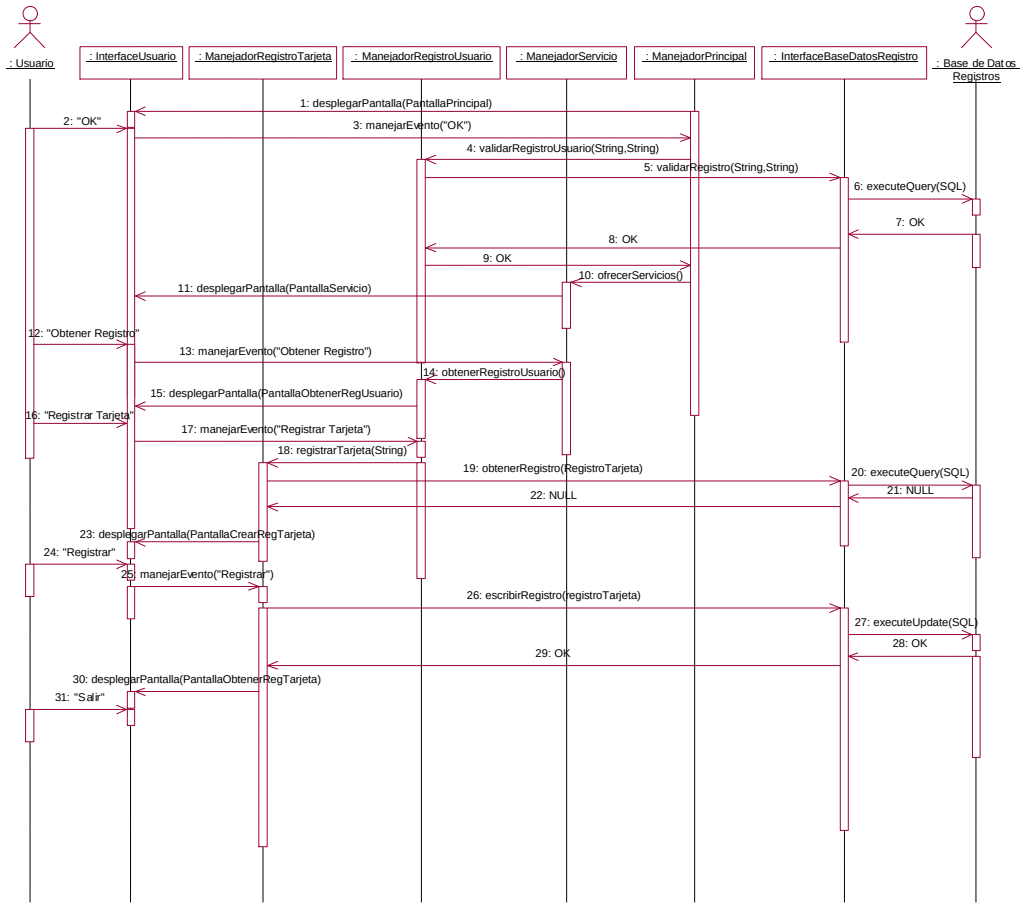

Figura 8.37. Diagrama de secuencias para el caso de uso *Registrar Tarjeta*, subflujo *Crear Registro Tarjeta*.

Registrar Tarjeta: Actualizar Registro Tarjeta

El diagrama de secuencia para el caso de uso *Registrar Tarjeta*, subflujo *Actualizar Registro Tarjeta* se muestra en la Figura 8.38.

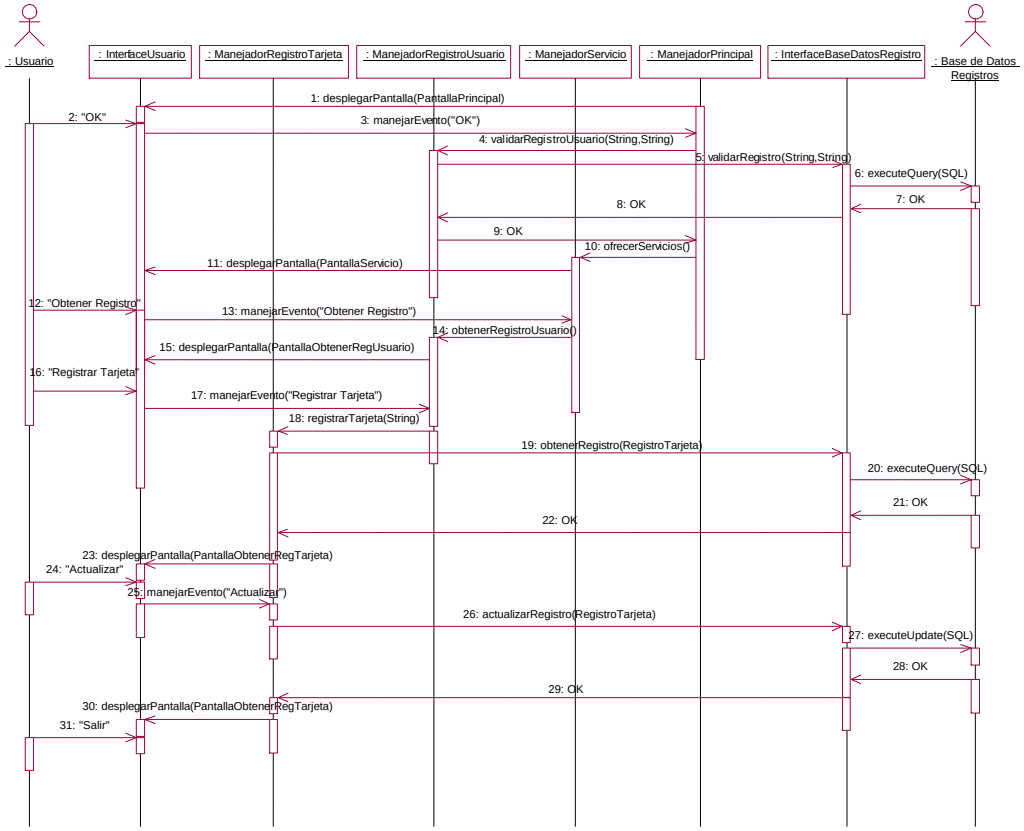

Figura 8.38. Diagrama de secuencias para el caso de uso *Registrar Tarjeta*, subflujo *Actualizar Registro Tarjeta*.

Registrar Tarjeta: Eliminar Registro Tarjeta

El diagrama de secuencia para el caso de uso *Registrar Tarjeta*, subflujo *Eliminar Registro Tarjeta* se muestra en la Figura 8.39.

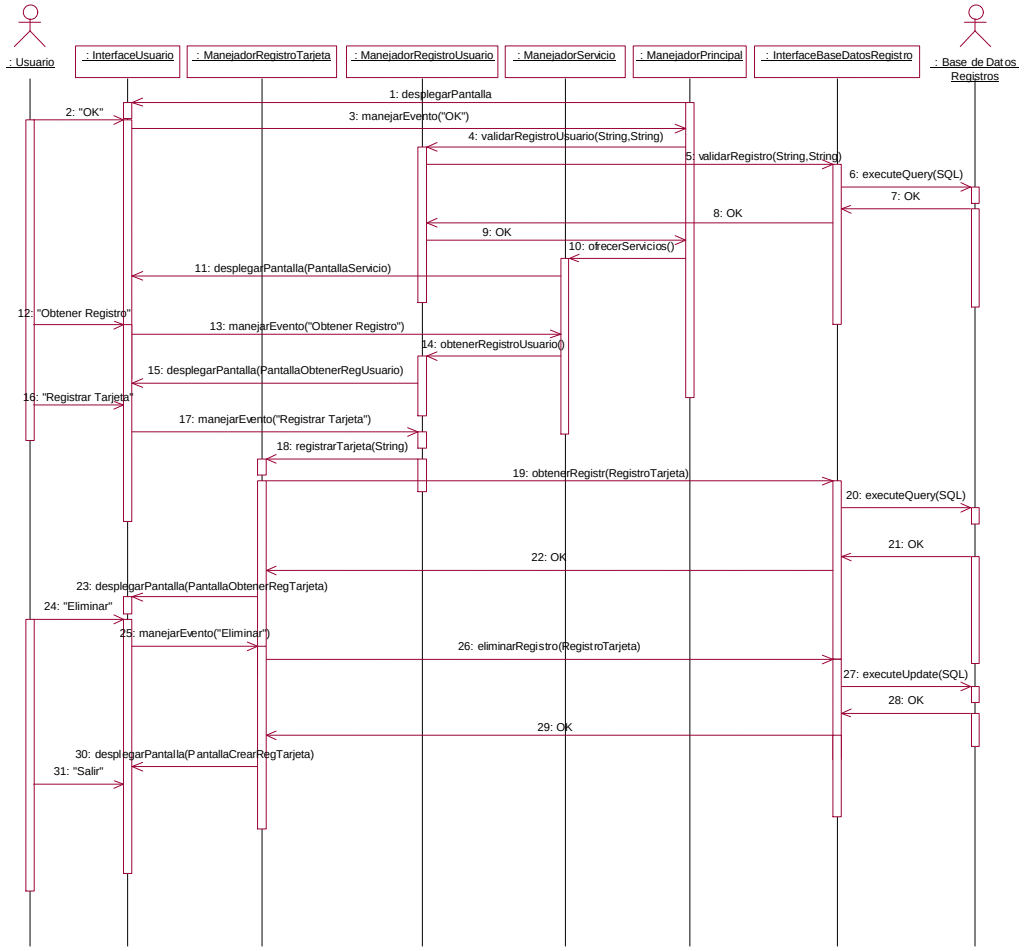

Figura 8.39. Diagrama de secuencias para el caso de uso *Registrar Tarjeta*, subflujo *Eliminar Registro Tarjeta*.