

Análisis y Diseño de Sistemas II – Teoría

Computación e Informática

ÍNDICE

Presentación	5
Red de contenidos	6
UNIDAD 1 : Análisis Orientado a Objetos	7
Tema 1 : Análisis orientado a objetos	8
1.1. Flujo de trabajo del AOO	9
1.2. Artefactos del análisis	10
1.3. Modelo de análisis	11
Tema 2 : Análisis de la arquitectura	14
2.1. Pasos en el análisis de la arquitectura	14
Tema 3 : Análisis de casos de uso	20
3.1. Pasos en el análisis de casos de uso	20
3.2. Caso práctico	26
Tema 4 : Análisis de clases	38
4.1. Pasos a realizar	38
4.2. Tarjetas CRC	39
4.3. Caso práctico	41
UNIDAD 2 : Modelo de Datos	
Tema 1 : Modelo de datos	50
1.1. Modelo conceptual	50
1.2. Construcción del Modelo Conceptual	51
1.3. Caso práctico	59
Tema 2 : Modelo Lógico	63
2.1. Tipos de datos	63
2.2. Tipos de relaciones	65
Tema 3 : Modelo físico	67
3.1. Campos	67
3.2. Indices	68
3.3. Sistemas manejadores de Base de datos (SMBD)	68
3.4. Caso práctico	70

Tema 4	: Recomendación en el manejo de herencia	72
Tema 5	: Auditoría de base de datos	77
 UNIDAD 3 : Diseño Orientado a Objetos		81
Tema 1	: Diseño orientado a objetos	82
	2.1 Flujo de trabajo	82
	2.2. Artefactos del diseño	82
	2.3. Modelo de análisis Vs. Modelo de diseño	85
Tema 2	: Arquitectura de software	87
	2.1. Por qué es importante la arquitectura	89
	2.2. Evolución de la arquitectura	89
	2.3. Fases de la arquitectura	90
	2.4. Estilos arquitectónicos	91
Tema 3	: Diseño de casos de uso	99
	1. Patrones de diseño	99
	2. Extensiones de UML para aplicaciones web (WAE)	104
	3. Realización de diseño de casos de uso con patrón arquitectónico MVC	108
	4. Realización de diseño de casos de uso con patrón arquitectónico MVC y patrón de diseño DAO	114
Tema 4	: Modelo de Implementación	126
	1.Componentes y tipos	126
	2.Diagrama componentes	127
	3.Nodo	127
	4.Servidor de aplicaciones	127
	5.Cluster	128
	6.Diagrama de despliegue	129
	7.Aplicaciones desatendidas	129

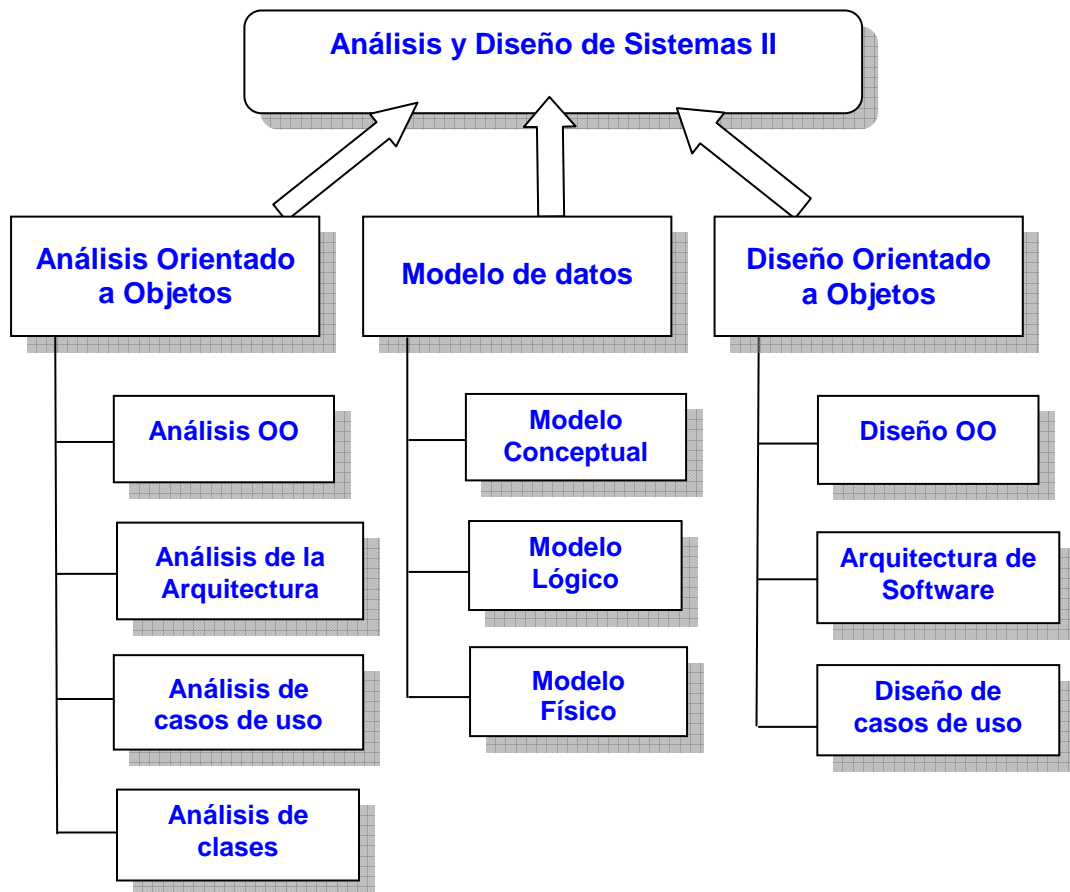
PRESENTACIÓN

Análisis y Diseño de Sistemas II pertenece a la línea formativa y se dicta en la carrera de Computación e Informática. El curso imparte conocimientos relacionados con la disciplina de análisis y diseño, y el modelo de datos.

El manual para el curso ha sido diseñado bajo la modalidad de unidades de aprendizaje, las que se desarrollan durante semanas determinadas. En cada una de ellas, hallará los logros, que debe alcanzar al final de la unidad; el tema tratado, el cual será ampliamente desarrollado; y los contenidos, que debe desarrollar, es decir, los subtemas. Por último, encontrará las actividades que deberá desarrollar en cada sesión, que le permitirán reforzar lo aprendido en la clase.

El curso es teórico - práctico: consiste en un taller de desarrollo de proyectos de software. En primer lugar, se describe el flujo de trabajo del análisis orientado a objetos. A continuación, se explica el modelo de datos. Por último, se presenta el flujo de trabajo del diseño orientado a objetos.

RED DE CONTENIDOS



UNIDAD DE
APRENDIZAJE

1

ANÁLISIS ORIENTADO A OBJETOS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al finalizar la primera unidad, el alumno modula la arquitectura de análisis que da soporte a los procesos del negocio, diagrama la estructura y el comportamiento de sus funcionalidades mediante diagramas de clases y diagramas de comunicación respectivamente. Los artefactos serán creados utilizando la herramienta *CASE IBM Rational Software Architect (RSA)*.

TEMARIO

Tema 1: Análisis orientado a objetos

- 1.1. Flujo de trabajo del AOO
- 1.2. Artefactos del análisis
- 1.3. Modelo de análisis

Tema 2: Análisis de la arquitectura

- 2.1. Pasos en el análisis de la arquitectura
- 2.2. Caso práctico

Tema 3: Análisis de casos de uso

- 3.1. Pasos en el análisis de casos de uso
- 3.2. Caso práctico

Tema 4: Análisis de clases

- 4.1. Análisis de clases
- 4.2. Tarjetas CRC
- 4.3. Caso práctico

ACTIVIDADES PROPUESTAS

1. Los alumnos crean la arquitectura de análisis a partir de un diagrama de casos de uso estructurado.
2. Los alumnos crean la realización de análisis de un caso de uso a partir de una ECU.
3. Los alumnos confeccionan las tarjetas CRC de las clases que se identifican a partir de una ECU.

1. ANÁLISIS ORIENTADO A OBJETOS

El Análisis Orientado a Objetos (AOO) es parte de la disciplina Análisis y Diseño de RUP; esta disciplina tiene como propósito:

- Transformar los requisitos en un diseño del sistema a crear.
- Definir una arquitectura robusta para el sistema.
- Adaptar el diseño para que funcione en el ambiente de implementación, diseñándolo para un desempeño esperado.

El flujo de trabajo de Análisis y Diseño toma los casos de uso documentados del flujo de trabajo de la Captura de Requisitos y los traslada a elementos de diseño que serán usados para construir el sistema. Por medio de varias actividades y modelos, el flujo de trabajo de Análisis y Diseño busca transformar la información obtenida de los *stakeholders* en información que los programadores podrán usar. El flujo de trabajo de esta disciplina se muestra en la figura No. 1.1.

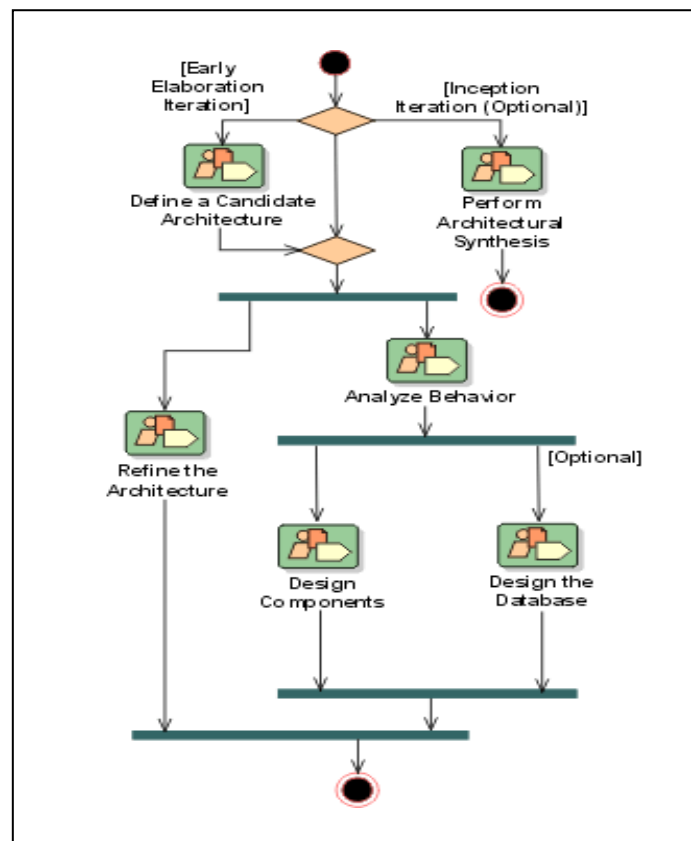


Figura 1.1. Flujo de Trabajo de la disciplina Análisis y Diseño.

En la fase de Inicio, la disciplina de Análisis y Diseño se preocupa por establecer si la visión del sistema es factible, y en determinar las tecnologías potenciales para la solución de software (dentro de la actividad **Perform Architectural Synthesis**). Si se considera que pocos riesgos se asocian al desarrollo (porque el dominio se entiende muy bien, el sistema no es novedoso o cualquier otra razón parecida), entonces, éste aspecto se omite del flujo de trabajo.

En la parte inicial de la fase de Elaboración, se enfoca el esfuerzo en crear una arquitectura inicial del sistema (en la actividad **Define a Candidate Architecture**)

la cual proporcionará un punto inicial para todo el trabajo de análisis. Si la arquitectura ya existe, porque fue creada en iteraciones anteriores o en proyectos anteriores, el trabajo se enfoca en cambios para refinar la arquitectura (actividad ***Refine the Architecture***) y en analizar el comportamiento, y crear un conjunto inicial de elementos los cuales proporcionarán un comportamiento apropiado (en la actividad ***Analyze Behavior***).

Después de que los elementos iniciales son identificados, se refinan en iteraciones futuras. La actividad ***Design Components*** produce un conjunto de componentes los cuales proveen un comportamiento adecuado para satisfacer los requisitos del sistema. Si el sistema incluye una base de datos, entonces, la actividad de ***Design the Database*** se ejecuta en paralelo. El resultado es un conjunto inicial de componentes los cuales en un futuro son refinados dentro de la Implementación.

Con el fin de entender el flujo de trabajo realizado en esta disciplina en forma más detallada se ha dividido su descripción en dos partes: Análisis orientado a objetos y Diseño orientado a objetos. A continuación, se empezará a estudiar el AOO.

1.1. Flujo de trabajo del AOO

El objetivo del análisis es **comprender el problema** y comenzar a desarrollar un modelo visual de lo que se está tratando de construir, independiente de la tecnología a utilizar en la aplicación, como el lenguaje de programación. El análisis se centra en la traducción de los requisitos funcionales en conceptos de *software*. La idea es identificar los objetos que conforman el sistema, centrándose en el comportamiento.

En la siguiente figura se resalta las actividades que se llevan a cabo en el AOO.

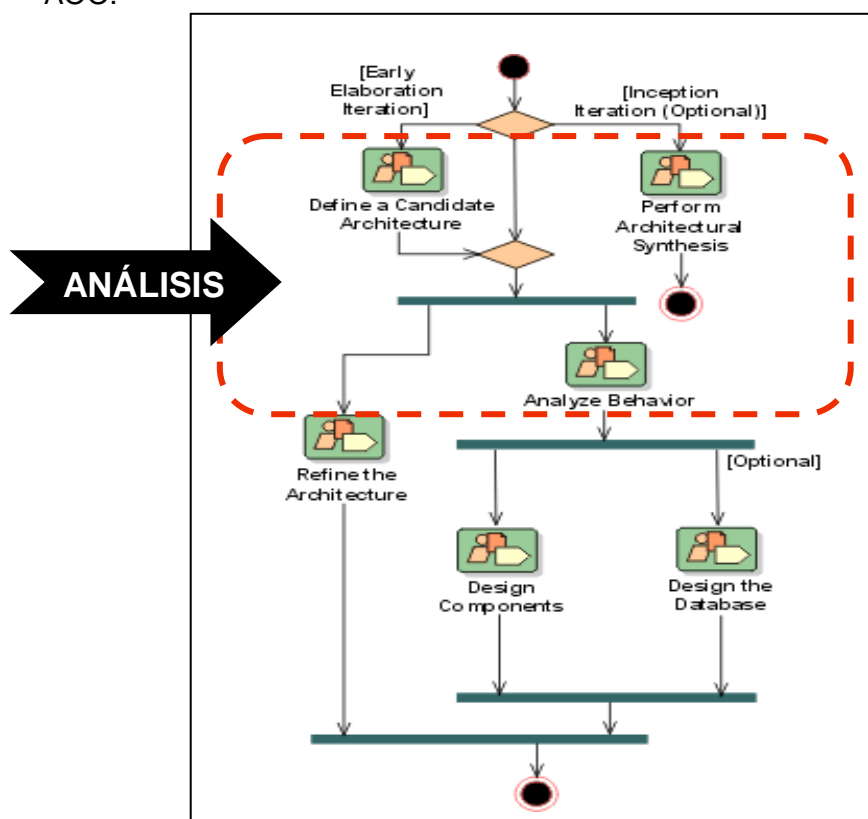


Figura 1.2. Flujo de trabajo del AOO.

1.2. Artefactos del Análisis

Son muchos los artefactos que se elaboran en el flujo de trabajo del AOO. En el curso, consideraremos los siguientes artefactos:

- Modelo de Análisis.
- Elementos del modelo de análisis: paquetes de análisis, clases de análisis y realizaciones de análisis de casos de uso.
- Diagramas de realizaciones de análisis de casos de uso: diagrama de clases y diagramas de comunicación.

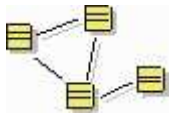





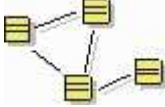
Artefacto	Descripción
 Modelo de Análisis	Representa la vista interna del sistema. Define un modelo de objetos que describe la realización de casos de uso, y que sirve como una abstracción del modelo de diseño.
 Paquete de Análisis	Los paquetes del análisis proporcionan un medio para organizar los artefactos del modelo de análisis en piezas manejables. Un paquete de análisis contiene clases y realizaciones de casos de uso a nivel de análisis.
 Clase de Interfaz	Es una clase utilizada para modelar la interacción entre el entorno del sistema y su funcionamiento interno. Modela las partes del sistema que dependen de su entorno.
 Clase Control	Representa la lógica de negocio de la aplicación, es decir, el control, la coordinación y la secuencia entre objetos. Encapsula el comportamiento de uno o más casos de uso.
 Clase Entidad	La entidad es una clase utilizada para modelar la información y comportamiento asociado que deben ser almacenados. Modela las partes del sistema que son independientes de su entorno.
 Realización de Caso de Uso	La realización de análisis de un caso de uso es una colaboración que describe cómo se realiza el caso de uso en términos de clases de análisis y sus interacciones.
 Diagrama de Clases	El diagrama de clases describe la estructura de un caso de uso.

Tabla 1.1. Artefactos del Análisis.

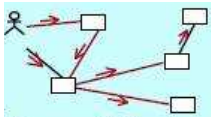
 <p>Diagrama de Comunicación</p>	<p>Diagrama de interacción que describe el comportamiento del caso de uso centrado en la responsabilidades y colaboraciones entre los objetos.</p>
---	--

Tabla 1.1. Artefactos del Análisis (Continuación).

1.3. Modelo de Análisis

El **análisis orientado a objetos se traduce en el modelo de análisis**, el cual es usado para representar la estructura global del sistema, describe la realización de casos de uso y sirve como una abstracción del modelo de diseño.

Durante el análisis, se identifica, de manera continua, nuevos paquetes del análisis, clases y requisitos comunes a medida que el modelo de análisis evoluciona, y los paquetes de análisis concretos continuamente se refinan y mantienen.

Las actividades que se realizan para elaborar el modelo de análisis son las siguientes:

- Análisis de la Arquitectura
- Análisis de Casos de Uso
- Análisis de Clases
- Análisis de Paquetes.

Según Ivar Jacobson, *“El modelo de análisis es un nivel de diseño intermedio entre las etapas de captura de requisitos y la de diseño.”*

Este modelo de análisis no es un diagrama final que describe todos los posibles conceptos y sus relaciones, es un primer intento por definir los conceptos claves que describen el sistema. Este artefacto es opcional, pero también tiene a su vez la propiedad de ser temporal en el caso en que se planea su desarrollo. Su utilidad radica en que permite una apreciación global conceptual del sistema.

El modelo de análisis puede contener las clases y paquetes de análisis, las realizaciones de los casos de uso, las relaciones y los diagramas. Es opcional detallar aquí las realizaciones de los casos de uso, ya que estas pueden estar en el modelo de diseño donde se recomienda que se encuentre.

A diferencia del modelo de casos de uso que captura la funcionalidad del sistema, el modelo de análisis da forma a la arquitectura para soportar las funcionalidades que en el anterior modelo se expresan.

1.3.1. Características principales

- Proporciona un diseño preliminar, pues contiene paquetes que se usan para organizar el modelo de análisis en piezas más manejables, que representan abstracciones o subsistemas y una primera vista del diseño.
- Puede ayudar a descubrir la necesidad de clases adicionales.
- Proporciona una prueba de completitud a los casos de uso, antes de pasar al diseño.
- Proporciona un diseño preliminar de la arquitectura del sistema, denotando los paquetes de análisis de alto nivel.

1.3.2. Modelo de casos de uso Vs. Modelo de análisis

El siguiente cuadro muestra una comparación entre el Modelo de Casos de Uso y el Modelo de Análisis:


<i>Modelo de Casos de Uso</i>	<i>Modelo de Análisis</i>
Descrito con el lenguaje del cliente. Vista externa del sistema.	Descrita en el lenguaje de los desarrolladores. Vista interna del sistema.
Estructurado por los casos de uso.	Estructurado por clases y paquetes estereotipados.
Utilizado como contrato entre el cliente y los desarrolladores sobre qué debería y que NO debería hacer el sistema.	Utilizado por los desarrolladores para comprender cómo debería darse forma al sistema, es decir, cómo debe estar diseñado e implementado.
Puede contener redundancias e inconsistencias entre los requisitos.	No debería contener redundancias, inconsistencias, entre los requisitos.
Captura la funcionalidad del sistema, incluida la funcionalidad significativa para la arquitectura.	Esboza cómo llevar a cabo la funcionalidad dentro del sistema, incluida la funcionalidad significativa para la arquitectura.
Define los casos de uso que se analizarán con más profundidad en el modelo de análisis.	Define las realizaciones de casos de uso, y cada una de ellas representa el análisis de un caso de uso del modelo de casos de uso.


Cuadro 1.1. Comparación del MCU Vs. MA

Resumen

 La disciplina Análisis y Diseño de RUP tiene como propósito:

- Transformar los requisitos en un diseño del sistema a crear.
- Definir una arquitectura robusta para el sistema.
- Adaptar el diseño para que funcione en el ambiente de implementación, diseñándolo para un desempeño esperado.

 El objetivo del análisis es comprender el problema y comenzar a desarrollar un modelo visual de lo que se está tratando de construir, independiente de la tecnología a utilizar en la aplicación, como el lenguaje de programación. El análisis se centra en la traducción de los requisitos funcionales en conceptos de *software*. La idea es identificar los objetos que conforman el sistema, centrándose en el comportamiento.

 El modelo de análisis es usado para representar la estructura global del sistema, describe la realización de casos de uso y sirve como una abstracción del modelo de diseño.

2. ANÁLISIS DE LA ARQUITECTURA

El rol responsable de esta tarea es el Arquitecto de software. Esta tarea permite definir una arquitectura candidata basada en la experiencia obtenida de sistemas similares o en dominios del problema similares, y restringir las técnicas arquitectónicas a ser usadas en el sistema. Se definen los diagramas de las vistas arquitectónicas, mecanismos claves y los modelos para el sistema. Cabe destacar que analizar la arquitectura resulta beneficioso en el caso donde se desarrollen sistemas que no se hayan hecho antes.

El propósito del análisis de la arquitectura es esbozar el modelo de análisis y la arquitectura mediante la identificación de paquetes del análisis, clases de análisis de entidad evidentes y requisitos especiales comunes.

2.1. Pasos en el Análisis de la arquitectura

El Análisis de la Arquitectura se realiza como sigue:

- Identificación de los paquetes de análisis
- Definición de las dependencias entre los paquetes de análisis
- Identificación de clases de entidad obvias por cada paquete de análisis
- Identificación de mecanismos de análisis
- Identificación de las características fundamentales de un requisito especial.

2.1.1. Identificación de paquetes de análisis

Los paquetes de análisis constituyen una división del sistema de *software* que tiene sentido desde el punto de vista de los expertos en el dominio. La descomposición del *software* en paquetes se establece cuando uno tiene una idea que considera suficientemente fiable de la cantidad de trabajo y del número, y la complejidad de los diagramas, situación a la cual se puede haber llegado tanto al principio de la etapa de análisis como un tiempo después. Una identificación inicial de los paquetes del análisis se hace de manera natural basándonos en los requisitos funcionales y en el dominio del problema, es decir, en la aplicación o negocio que estamos considerando.

Debido a que los requisitos funcionales se capturan en la forma de casos de uso, una forma directa de identificar paquetes del análisis es asignar la mayor parte de un cierto número de casos de uso a un paquete concreto.

Entre las asignaciones adecuadas de casos de uso a un paquete en concreto se tiene los siguientes criterios:

1. Los casos de uso requeridos para dar soporte a un determinado proceso de negocio.
2. Los casos de uso requeridos para dar soporte a un determinado actor del sistema.

Para identificar los paquetes se basa en lo siguiente:

1. Tener un diagrama de casos de uso con los roles bien definidos.

2. Los casos de uso que estén bajo la responsabilidad de un actor deben tener contenidos estrechamente relacionados.
3. Los casos de uso que están relacionados mediante relaciones de generalización deben pertenecer al mismo paquete.

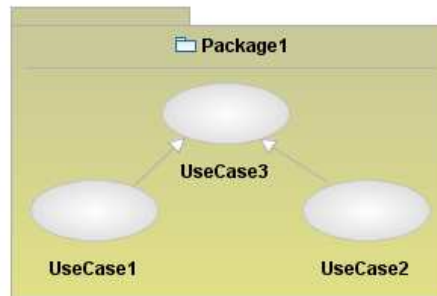


Figura 1.3. Casos de Uso con relación de generalización.

4. Los casos de uso relacionados mediante relaciones de extensión y que solo se extienden a partir de un caso de uso base deben pertenecer al mismo paquete del caso de uso base.

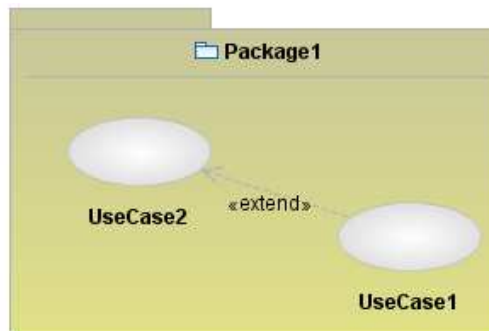


Figura 1.4. Casos de Uso con relación *extend*.

5. Los casos de uso incluidos tienden a generar su propio paquete la mayor parte de veces. Si los casos de uso base que incluyen al caso de uso son funcionalidades con distintos contenidos, entonces, se debe crear un paquete para el caso de uso incluido.

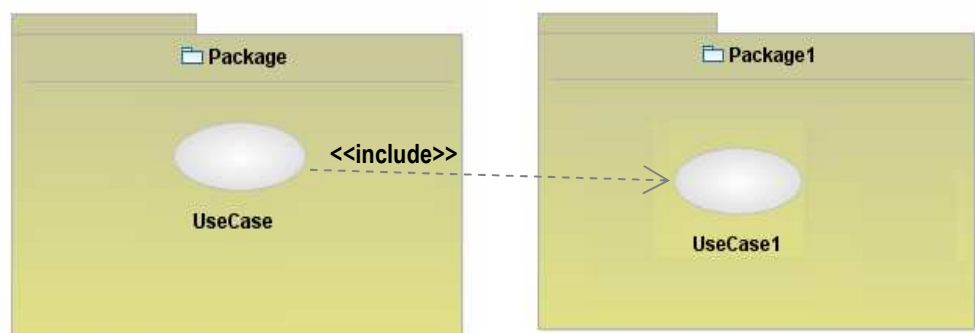


Figura 1.5. Casos de uso con relación *incluye*.

2.1.2. Definición de dependencias entre paquetes de análisis

El objetivo es conseguir paquetes que sean relativamente independiente y débilmente acoplados, pero que posean una cohesión interna alta. Es inteligente intentar reducir el número de relaciones entre clases en paquetes diferentes, debido a que ello reduce las dependencias entre paquetes.

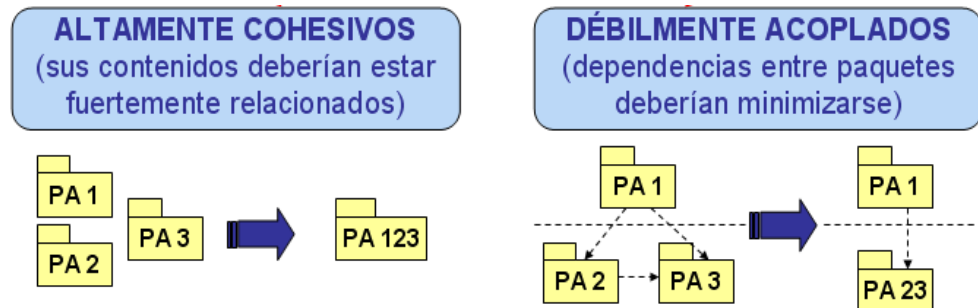


Figura 1.6. Características de los paquetes de análisis.

Los paquetes identificados se organizarán en la Capa de Aplicación, la cual se subdivide en dos capas internas:

a) Capa Específica: Aquí se ubican los paquetes correspondientes al proceso del negocio (core) de la empresa identificados (Nivel Superior).

b) Capa General: Aquí se ubican los paquetes de maestros de información, paquetes de servicio, seguridad y casos de apoyo del sistema (Nivel Inferior).

Para identificar las dependencias entre paquetes es conveniente revisar el diagrama de casos de uso estructurado según análisis, esto con el fin de ubicar las relaciones que existen entre los casos de uso. Las dependencias se crean a partir de los paquetes de análisis que contienen los casos de uso base.

A continuación, se muestra un ejemplo de distribución de paquetes en las capas de la aplicación y sus dependencias para definir la arquitectura de análisis.

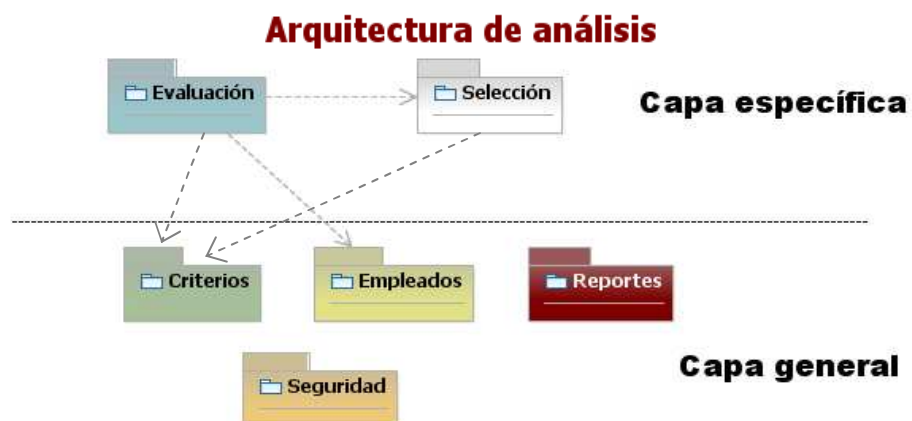


Figura 1.7. Capas y dependencias entre paquetes de análisis.

2.1.3. Identificación de clases de entidad obvias

Solo se debe concentrar en identificar las clases del tipo *entity* por cada caso de uso. La mayoría de las clases se identificarán al crear las realizaciones de caso de uso (en la actividad del análisis de caso de uso).

Se debe tener cuidado de no identificar demasiadas clases en esta etapa y quedar atrapados en demasiados detalles, pues ese trabajo es para el análisis de casos de uso.

Ejemplo:

“Reserva”, “Habitación” → Clases del dominio del problema

“Detalle Reserva” → Clase de la asociación entre Reserva y Habitación

2.1.4. Identificación de mecanismos de análisis

El arquitecto es el responsable de identificar los mecanismos de análisis comunes de forma que los desarrolladores puedan referirse a ellos como requisitos especiales sobre realizaciones de CU y clases del análisis determinadas.

Los mecanismos de análisis a considerar son sobre los siguientes requisitos especiales:

- Persistencia
- Comunicación
- Distribución y concurrencia
- Gestión de transacciones
- Sincronización y control de procesos
- Intercambio de información
- Formato de conversión
- Característica de seguridad
- Tolerancia de fallos
- Redundancia

2.1.5. Identificación de características fundamentales de mecanismos de análisis

En esta etapa, se debe indicar las características de cada mecanismo de análisis. Por ejemplo, las características de un requisito de persistencia son los siguientes:

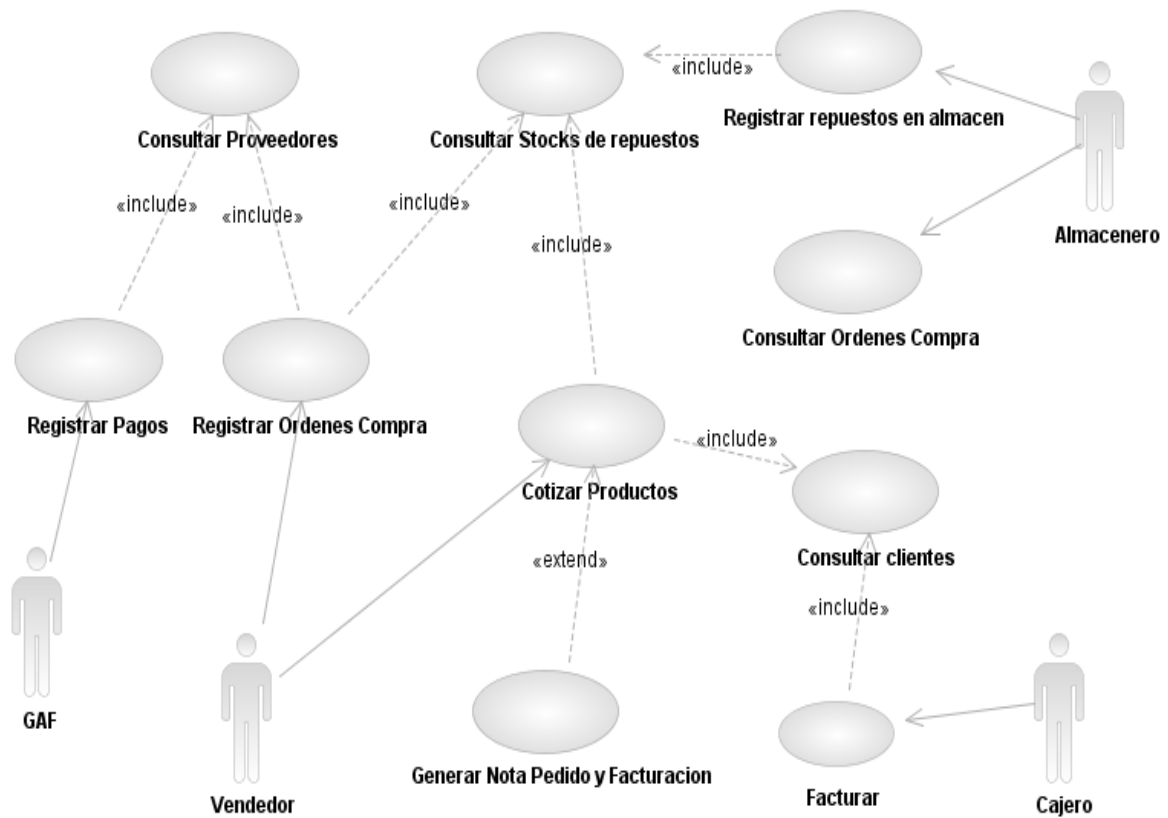
- Granularidad: Rango de tamaño de objetos persistentes.
- Volumen: Número de objetos persistentes.
- Duración: Periodo de persistencia.
- Mecanismo de acceso: Cómo identificar a un objeto.
- Frecuencia de acceso: Identificar qué objetos son más o menos constantes y qué objetos son actualizados frecuentemente.
- Confiabilidad.

De la misma manera, se trabaja con los otros requisitos especiales identificados en el paso anterior.

ACTIVIDADES PROPUESTAS

Identifique los paquetes de análisis y sus dependencias para realizar la Arquitectura de Análisis a partir del siguiente Diagrama de Caso de Uso Estructurado:

Caso: Cotización



Resumen

📖 El propósito del análisis de la arquitectura es esbozar el modelo de análisis y la arquitectura mediante la identificación de paquetes del análisis, clases análisis del tipo entidad evidentes y requisitos especiales comunes.

📖 Las actividades del Análisis de la Arquitectura son como sigue:

- Identificación de paquetes de análisis
- Definición de las dependencias entre los paquetes de análisis
- Identificación de clases de entidad obvias por cada paquete de análisis
- Identificación de los requisitos especiales comunes
- Identificación de las características fundamentales de un requisito especial.

📖 Los paquetes se usan para organizar el modelo de análisis en piezas más manejables, que representan abstracciones o subsistemas y una primera vista del diseño.

Un paquete de análisis debe ser débilmente acoplado y altamente cohesivo.

Los paquetes de análisis identificados se organizarán en la Capa de Aplicación, la cual se subdivide en dos capas internas:

- **Capa Específica:** Aquí se ubican los paquetes correspondientes al proceso del negocio (core) de la empresa identificados (Nivel Superior).
- **Capa General:** Aquí se ubican los paquetes de maestros de información, paquetes de servicio, seguridad y casos de apoyo del sistema (Nivel Inferior).

3. ANÁLISIS DE CASOS DE USO

El Análisis de Caso de Uso es el proceso de examinar los casos de uso para descubrir los objetos y clases de análisis del sistema a desarrollar. Las clases identificadas deben agruparse en los paquetes según criterios de Arquitectura de Software.

El rol responsable de esta tarea es el Diseñador. Esta tarea describe cómo desarrollar las Realizaciones de los Casos de Uso del nivel de análisis de un caso de uso particular. Tiene los siguientes propósitos:

- Identificar las clases que llevan a cabo el flujo de eventos de un caso de uso.
- Distribuir el comportamiento de casos de uso a las clases identificadas, usando realizaciones de casos de uso a nivel de análisis.
- Identificar atributos, responsabilidades y relaciones entre las clases.
- Observar los mecanismos arquitectónicos.

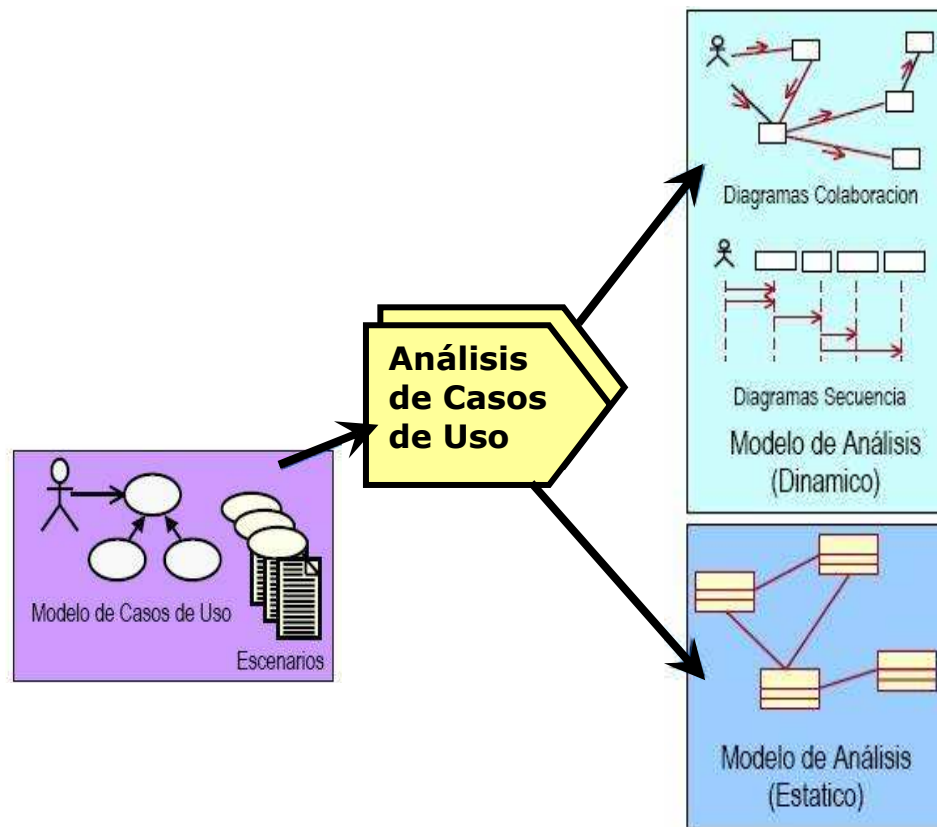


Figura 1.8. Análisis de Casos de Uso.

3.1. Pasos en el Análisis de casos de uso

Para llevar a cabo el análisis de casos de uso se realiza lo siguiente:

- Crear la realización de análisis de casos de uso.
- Encontrar clases de análisis del comportamiento de casos de uso.
- Crear el diagrama de clases (estructura del caso de uso).
- Crear el diagrama de comunicación (comportamiento del caso de uso).

3.1.1. Crear la realización de análisis de casos de uso

Una realización de caso de uso describe cómo un caso de uso en particular es modelado, primero en el modelo de análisis y, después, en el modelo de diseño en términos de objetos colaboradores.

En una realización de caso de uso se especifica qué clases deben ser construidas para implementar cada caso de uso.

En UML, las realizaciones de caso de uso se representan con colaboraciones estereotipadas. El ícono para una colaboración es una elipse con líneas punteadas que se sitúa al lado izquierdo del nombre de la colaboración, tal como se ilustra en la siguiente figura.



Figura 1.9. Colaboración.

3.1.2. Encontrar clases de análisis

Este paso se realiza por cada caso de uso. Para ello, se analizan los escenarios de Caso de Uso para identificar las clases que participan en ellos.

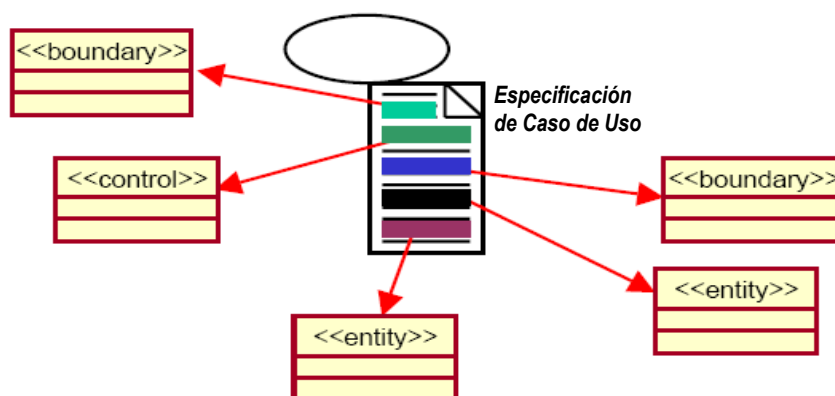


Figura 1.10. Un flujo completo de un caso de uso debe distribuirse en clases de análisis.

Tal como se muestra en la figura 2.3, a partir de una especificación de caso de uso (ECU) se pueden obtener las clases de análisis. Existen tres tipos de clases de análisis: *boundary*, *control* y *entity*. A continuación, se describe a cada una de ellas:

- **Boundary.** Describe una interacción entre el sistema con los usuarios y con otros sistemas. Pueden representar abstracciones de formularios, de protocolos de comunicaciones con otros sistemas o interfaces de dispositivos.

Las características importantes de este tipo de clase cuando modela un API con otro sistema son los siguientes:

- a. Las funciones que provee el otro sistema

- b. La información a ser pasada al otro sistema
- c. El “protocolo” de comunicación usado para “hablar” con el otro sistema.

Por regla general, al menos una clase boundary sirve como medio de comunicación entre un actor y el correspondiente caso de uso.

Ejemplo:

En el caso de uso “*Procesar Facturación*” hay información que debe ser enviada a un Sistema de Facturación externo. Se puede crear una clase de interfaz llamada *CI_SistemaFacturacion* para representar la interfaz al sistema externo.

- **Entity.** Se emplean para modelar aquella información o comportamiento que posee una vida larga en el sistema. Normalmente, están asociadas a algún fenómeno o concepto, como una persona, un objeto del mundo real o un suceso del mundo real.

El número de clases entidad variará dependiendo de los conceptos que requieren almacenamiento persistente dentro del caso de uso. Estas clases sufren un proceso de refinamiento a medida que se ubica a la misma clase entidad dentro de distintas realizaciones de caso de uso.

Ejemplo:

En el caso de uso “**Mantener empleados**” en el cual se puede registrar, modificar o desactivar empleados es evidente que la información que debe ser manipulada es del empleado. Para ello, se crea una clase entidad *Empleado*.

- **Control.** Se utilizan para modelar la coordinación, secuencia, transacciones y control de otros objetos. También para representar derivaciones y cálculos complejos, cómo la lógica de negocio, que no pueden asociarse a ninguna información concreta de larga duración almacenada por el sistema.

Por regla general, se trata de encapsular la lógica de control de un caso de uso dentro de una clase Control. Suele ser un buen hábito de diseño utilizar únicamente una clase control por cada caso de uso, y así encapsular en un único elemento la lógica del caso de uso correspondiente. Por otro lado, todos los casos de uso ubicados en un mismo paquete de análisis comparten la misma clase control.

Ejemplo:

En un paquete de análisis denominado Evaluación se ubica los casos de uso “Evaluar empleado”, “Procesar evaluación de desempeño” y “Consultar estadísticas de Evaluaciones”. Para los tres casos de uso se crea una clase control *CC_Evaluacion* que coordina el trabajo de los tres casos de uso.

Según la metodología OOSE de Ivan Jacobson, las clases de análisis son clases estereotipadas para crear modelos ideales de objetos. Esta metodología se basa en el patrón MVC (*Model-View-Controller* / Modelo Vista Controlador), que define clases enfocadas a la separación de responsabilidades para conseguir componentes extensibles y reutilizables.

En la siguiente figura se muestran los tipos de clases enfocados a los elementos del patrón MVC.

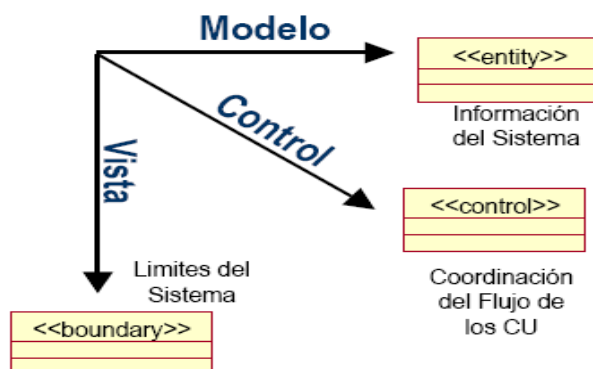


Figura 1.11. Clases de análisis enfocadas al patrón MVC.

Cada clase de análisis debe tener un estereotipo¹, como mecanismo que el UML provee para extender la notación. Los estereotipos se muestran en el compartimiento del nombre de la clase encerrado entre `<<>>` (*guillemets*) o con un icono; también, se pueden representar con la forma de la imagen del icono en lugar de una clase. Estas representaciones se muestran a continuación:



Figura 1.12. Clases de interfaz (*boundary*).

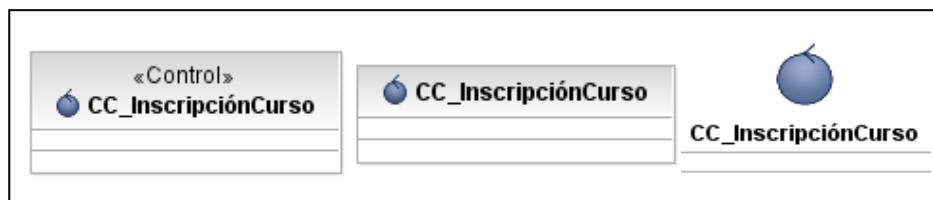


Figura 1.13. Clases de control (*control*).

¹ Un estereotipo (*Stereotype* en inglés) es un nuevo tipo de elemento de modelado que extiende la semántica del meta modelo, basados en tipos existentes o clases del meta modelo.



Figura 1.14. Clases de entidad (*entity*).

3.1.3. Crear el diagrama de clases

Este paso permite representar las clases participantes y sus relaciones para un determinado caso de uso.

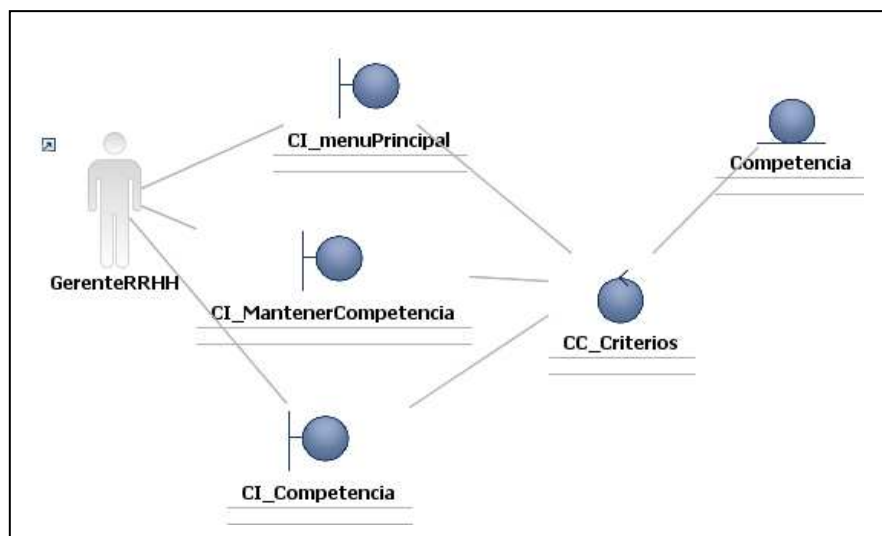


Figura 1.15. Diagrama de clases de análisis.

3.1.4. Crear diagramas de comunicación

El diagrama de comunicación es un tipo de diagrama de interacción. En esta etapa, no se usa diagramas de secuencia, porque no es importante la cronología de las interacciones. Un diagrama de comunicación muestra la colaboración dinámica entre los objetos, es decir, describe el comportamiento de un caso de uso mostrando explícitamente las relaciones de los objetos participantes.

La realización de un caso de uso puede tener uno o más diagramas de comunicación, esto es debido a que se representa el flujo básico, sub-flujos y flujos alternativos.

Por ejemplo, a continuación, se muestra el diagrama de comunicación para describir el comportamiento del caso de uso Mantener competencias.

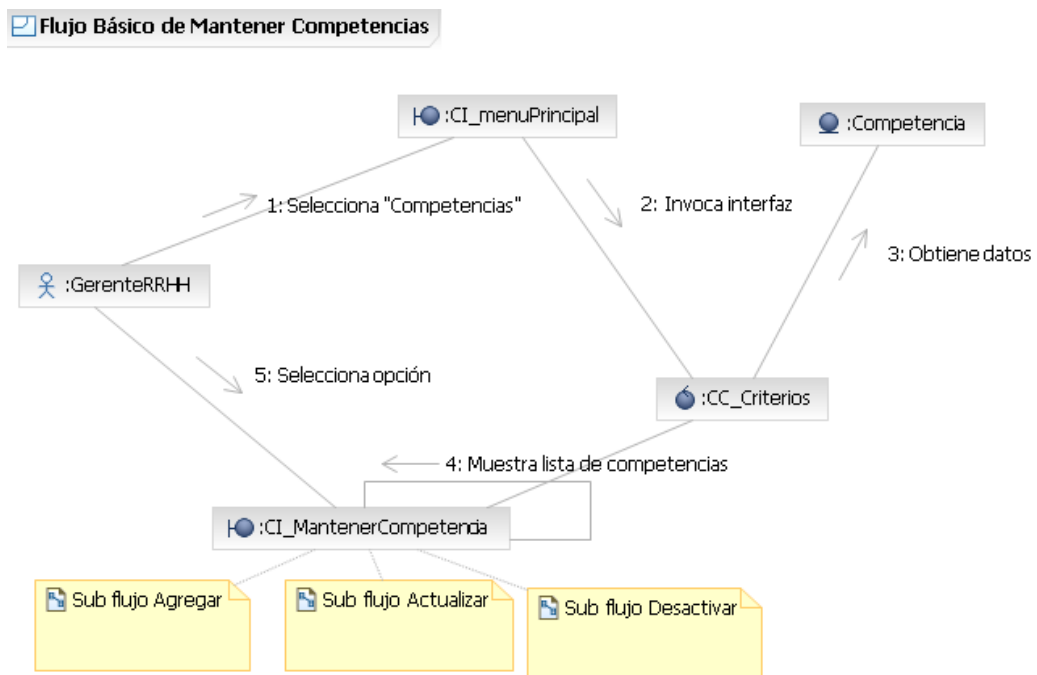


Figura 1.16. Diagrama de comunicación.

3.2. Caso práctico

A continuación, se presenta un caso práctico para identificar clases de análisis a partir de una especificación de casos de uso.

CASO: Sistema de Ventas

La secretaria de Gerencia registra los pedidos al crédito del equipo de Ventas, asignando a cada pedido el vendedor, además, verifica si el cliente existe, si no lo registra en ese momento y obtiene los productos. Además, es la encargada de la administración de los registros de los vendedores, de los Productos y de los clientes.

El Jefe de ventas posee una opción para evaluar los pedidos al crédito; otra opción, para Anular facturas; otra opción, para dar como pagadas las facturas y hay una asistente que genera las facturas valiéndose de los pedidos aprobados

A continuación, se muestra la ECU de Evaluar pedidos al crédito:

Especificación de Caso de uso: Evaluar pedidos al crédito

1. Breve descripción

El caso de uso permite al Jefe de venta evaluar los pedidos, pudiendo aprobar o denegar el pedido.

2. Actor(es)

Jefe de venta

3. Flujo de Eventos

3.1. Flujo Básico

1. El caso de uso comienza cuando el Jefe de Venta solicita “Evaluar Pedidos al Crédito” en el menú principal.
2. El sistema muestra la interfaz “Evaluar Pedidos” con la lista de todas los Pedidos que aun no fueron evaluados. La lista de pedidos contiene los siguientes datos: Nro del pedido, Código y Nombre del Cliente, Vendedor, Fecha y Monto total. Además, posee las opciones Aceptar y Salir.
3. El Jefe de venta selecciona uno de los pedidos a evaluar.
4. El Jefe de venta selecciona Aceptar.
5. El sistema muestra la Interfaz “Pedido al Crédito” donde se muestra detalladamente los datos del Pedido (Número del pedido, fecha del pedido, monto total del pedido y nombre del vendedor). Asimismo:
Datos del cliente: código, DNI, nombres, apellidos y línea de crédito;
Lista con el detalle del pedido (Código y nombre del producto, cantidad, precio Total Item) y Sub total) IGV y Monto Total;
Además, incluye un cuadro de texto para colocar las observaciones y 2 opciones de selección (Aprobar y Denegar) y los botones Grabar y Salir.
6. El Jefe de ventas ingresa las observaciones.
7. El Jefe de Ventas selecciona Aprobar.
8. El jefe de ventas selecciona grabar.
9. El sistema actualiza el Pedido como aprobado.
10. El sistema muestra el MSG “Pedido Aprobado” y cierra la interfaz automáticamente.

3.2. Flujos Alternativos

<Solicitud Denegada>

Si en el punto 7, el Jefe de ventas selecciona Denegar.

1. El sistema actualiza el Pedido como Denegado

4. Requerimientos Especiales

Ninguno

5. Pre Condiciones

1. Jefe de ventas logeado en el sistema.
2. Lista de Pedidos pendientes.

6. Post Condiciones

1. En el sistema queda actualizada el pedido

7. Puntos de extensión**8. Prototipo**

Evaluar Pedido

Nro PEDIDO	COD CLIENTE	NOMBRE DEL CLIENTE	VENDEDOR	FECHA	MONTO TOTAL
00000001	00000001	LUIS POLO	LUIS ZARATE	10/06/2010	120,610
00000002	00000001	LUIS POLO	LUIS SANCHEZ	10/05/2010	105,010
00000003	00000002	ANA PEREZ	LUISA NAVARRO	11/06/2010	116,010
00000004	00000002	ANA PEREZ	JUANA NAVARRO	11/06/2010	110,200

Pedido al credito

Datos del pedido

Numero Fecha Monto total Vendedor

Datos del Cliente

Codigo DNI Linea de Credito

Nombre Apellido

Detalle del Pedido

Codigo	Nombre del Producto	Precio	Cantidad	Total Item

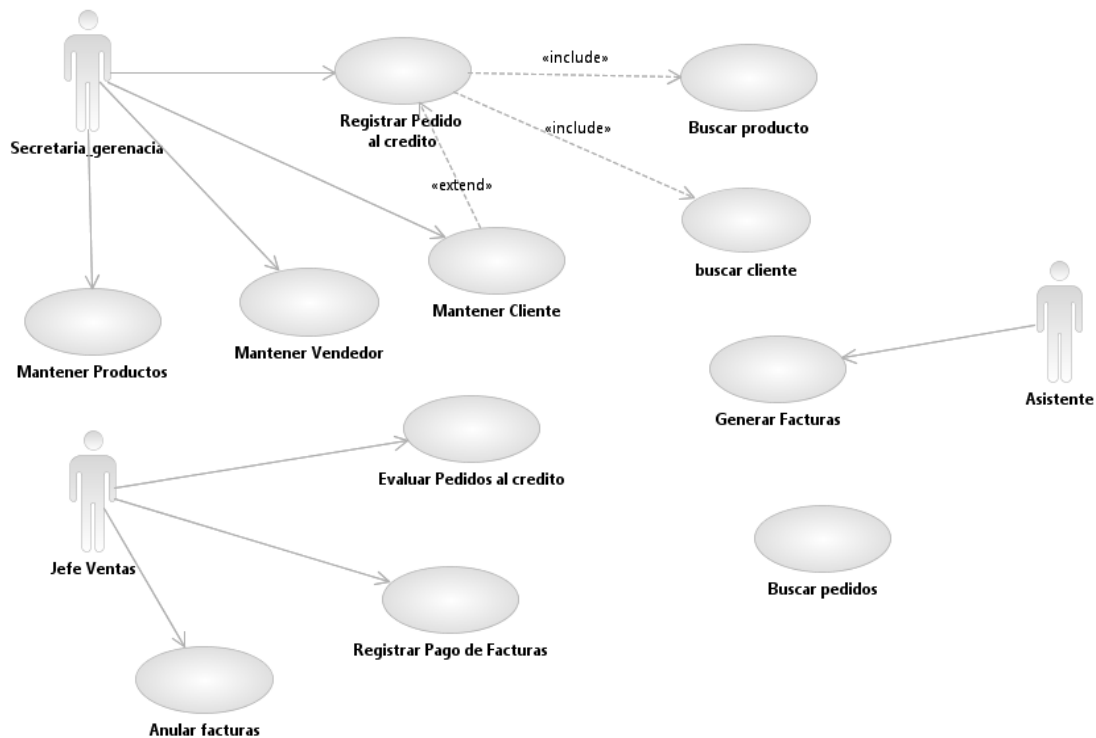
Sub total

IGV

Monto Total

Observaciones

☐ Aprobar
☐ Denegar



1. Escenario : Evaluar pedidos al crédito

1. El caso de uso comienza cuando el Jefe de Venta Solicita “Evaluar Pedidos al Crédito” en el menú principal.
2. El sistema muestra la interfaz “Evaluar Pedidos” con la lista de todas los Pedidos que aún no fueron evaluados. La lista de pedidos contiene los siguientes datos: Nro del pedido, Código y Nombre del Cliente, Vendedor, Fecha y Monto total. Además, posee las opciones Aceptar y Salir.
3. El Jefe de Ventas selecciona uno de los pedidos a evaluar.
4. El jefe de ventas selecciona Aceptar.
5. El sistema muestra la Interfaz “Pedido al Crédito” donde se muestra, detalladamente, los datos del Pedido (Número del pedido, fecha del pedido, monto total del pedido y nombre del vendedor).
 Datos del cliente: código, DNI, nombres, apellidos y línea de crédito,
 Lista con el detalle del pedido (Código y nombre del producto, cantidad, precio Total Item) y Subtotal) IGV y Monto Total.
 Además, incluye un cuadro de texto para colocar las observaciones y 2 opciones de selección (Aprobar y Denegar), y los botones Grabar y Salir.
6. El Jefe de ventas ingresa las observaciones.
7. El Jefe de Ventas selecciona Aprobar.
8. El jefe de ventas selecciona grabar.
9. El sistema actualiza el Pedido como aprobado.
10. El sistema muestra el MSG “Pedido Aprobado” y cierra la interfaz automáticamente.

2. Encontrando objetos entidad

- Los objetos de Entidad se identifican examinando los sustantivos y frases de sustantivos en los escenarios
- Los sustantivos encontrados pueden ser los siguientes:
 - Objetos específicos o genéricos
 - Descripciones del estado de un objeto
 - Entidades externas y/o actores

3. Filtrando sustantivos

- Cuando se identifican sustantivos tengan presente lo siguiente:
 - Varios términos pueden referirse al mismo objeto
 - Un término puede referirse a más de un objeto
 - El lenguaje natural es muy ambiguo.
- De esta manera, se pueden identificar muchos objetos sin importancia.
 - La lista de sustantivos se debe filtrar
- Advertencia
 - Cualquier sustantivo puede ser convertido en verbo; cualquier verbo puede ser convertido en sustantivo.
 - Los resultados dependen, en gran parte, de la capacidad de redacción del o los autores.

4. Sustantivos del Escenario

- Jefe de Venta
- Pedidos al Crédito
- Lista de todas los Pedidos
- Nro del pedido
- Código y Nombre del Cliente
- Vendedor
- Fecha
- Monto total
- Pedidos a evaluar
- Datos del Pedido (Número del pedido, fecha del pedido, monto total del pedido y nombre del vendedor)
- Datos del cliente: código, DNI, nombres, apellidos y línea de crédito
- Lista con el detalle del pedido (Código y nombre del producto, cantidad precio Total Item) y Subtotal) IGV y Monto Total
- Además, incluye un cuadro de texto para colocar las observaciones
- Actualiza el Pedido como aprobado.

5. Filtrando el escenario

- Jefe de Venta
- Pedidos al Crédito
- Lista de todos los Pedidos
- Nro del pedido
- Código y Nombre del Cliente
- Vendedor
- Fecha
- Monto total
- Pedidos a evaluar
- Datos del Pedido (Número del pedido, fecha del pedido, monto total del pedido y nombre del vendedor)
- Datos del cliente: código, DNI, nombres, apellidos y línea de crédito
- Lista con el detalle del pedido (Código y nombre del producto, cantidad, precio Total ítem) y Subtotal) IGV y Monto Total
- Además, incluye un cuadro de texto para colocar las observaciones
- Actualiza el Pedido como aprobado.


6. Análisis de los objetos filtrados en escenario

- | | |
|---|--------------------------|
| • Pedidos al Crédito | Estado de Objetos Pedido |
| • Lista de todas los Pedidos | Varios objetos Pedido |
| • Código y Nombre del Cliente | Objeto Cliente |
| • Vendedor | Objeto vendedor |
| • Pedidos a evaluar | Estado de Objetos Pedido |
| • Nombre del vendedor | Objeto vendedor |
| • Datos del cliente: | Objeto Cliente |
| • Detalle del pedido | Objeto detalle |
| • Código y nombre del producto, cantidad y precio | Objeto Producto |
| • Actualiza el Pedido como aprobado. | Estado de Objetos Pedido |

7. Creando Clases Entidad

- Los objetos de entidad encontrados se agrupan en clases.
 - Los objetos genéricos representan clases.
 - Las instancias de objetos con estructura y/o comportamiento similar se agrupan en clases.
- Este es solo un intento inicial
 - Las clases pueden cambiar a medida que se examinan más escenarios.

8. Clases entidad identificadas en el escenario del caso de uso.

 <p>Pedido</p>	<p>Pedido es la solicitud de productos que se le hacen a un fabricante o vendedor; representa la venta que un artículo tiene.</p>
--	---

 Producto	Producto es el bien ofrecido en el pedido.
 DetallePedido	DetallePedido es la relación de productos en un pedido.
 Vendedor	Vendedor persona que tiene por oficio vender cosas y realiza el pedido.
 Cliente	Cliente persona que compra el producto.

9. Encontrando Clases *Boundary* (límite)

- Para cada par de actor físico y escenario cree una clase *boundary*.
 - Durante el diseño, esta clase se transformará dependiendo de los mecanismos de interfase escogidos.
- Añada más clases *boundary* para modelar navegación entre interfaces (por ejemplo pantallas) en el mismo caso de uso.
- Ejemplo :
 - El sistema muestra la interfaz “Evaluar Pedidos”
Se crea una clase *boundary* llamada CI_Evaluar_pedido para permitir al jefe seleccionar el pedido a evaluar.
 - El sistema muestra la Interfaz “Pedido al Crédito”
Se crea una clase *boundary* llamada CI_Pedido para evaluar el pedido solicitado.

10. Candidatos a Clase *Boundary* en el escenario del CU



11. Bosquejo de pantalla

Se pueden crear “*storyboards*”, prototipos y bosquejos de pantallas para comunicar el “*look & feel*” de la clase *boundary* al usuario. Por ejemplo, a continuación, se muestra la interfaz CI_Evaluar_pedido.



Nro PEDIDO	COD CLIENTE	NOMBRE DEL CLIENTE	VENDEDEDOR	FECHA	MONTO TOTAL
00000001	00000001	LUIS POLO	LUIS ZARATE	10/06/2010	120,610
00000002	00000001	LUIS POLO	LUIS SANCHEZ	10/05/2010	105,010
00000003	00000002	ANA PEREZ	LUISA NAVARRO	11/06/2010	116,010
00000004	00000002	ANA PEREZ	JUANA NAVARRO	11/06/2010	110,200

12. Encontrando Clases Control

- Las clases de control contienen información de secuencia para coordinar los casos de uso.
- En este nivel de análisis, típicamente se añade una clase control para cada caso de uso. Es responsable del flujo de eventos en el caso de uso.
- Las clases de control NO deben ejecutar funciones cuya responsabilidad pertenece a clases de entidad o de límite.
- Esto es solo un corte inicial. A medida que se desarrollan más casos de uso y escenarios, las clases de control pueden eliminarse, dividirse o combinarse.

13. Clases Control en el CU

- Se añade una clase de control llamada CC_pedido.

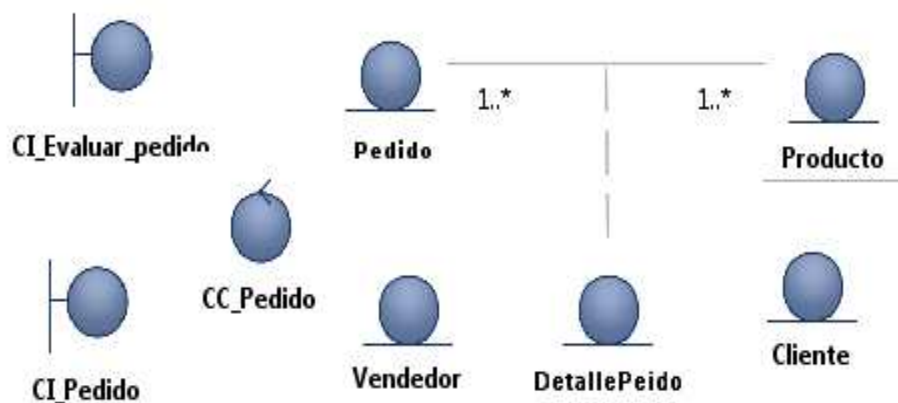


- Recibe información de la clase *boundary* CI_Evaluar_pedido y de la *boundary* CI_pedido.
- Para cada opción ingresada:
 - Valida el estado de los pedidos.
 - Obtiene la información del pedido.
 - Obtiene la información del cliente.
 - Obtiene la información de los productos que pertenecen al pedido.
 - Obtiene el vendedor que realizó el pedido.
 - Actualiza el estado del pedido.

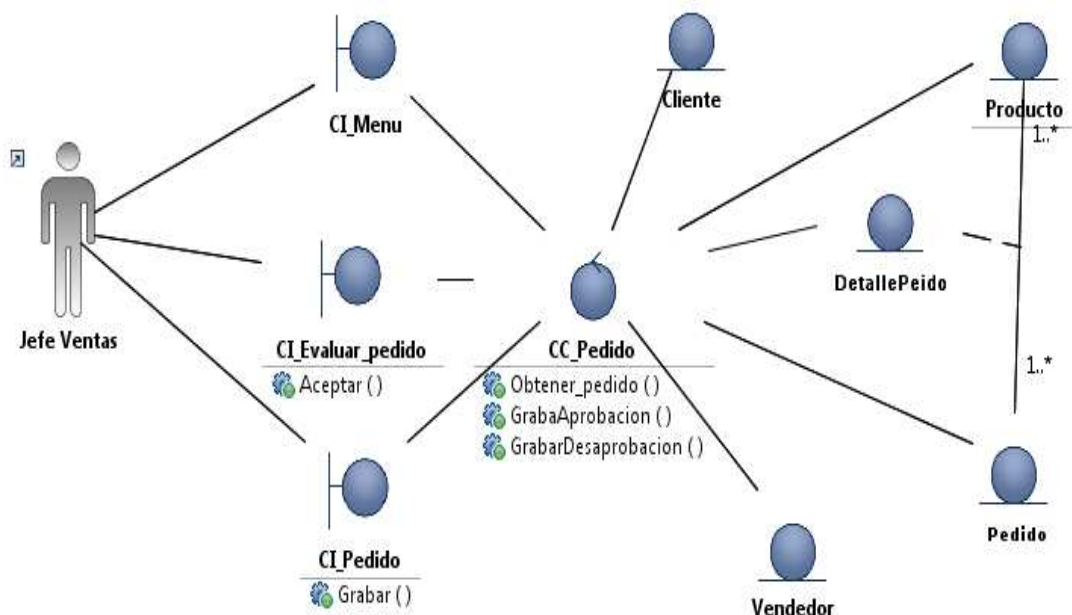
14. Diagramas de Vistas de Clases Participantes (VOPC)

- Un diagrama de clases muestra una o más clases en un mismo plano, usando la nomenclatura que se ha presentado antes.
- Cada Realización de Caso de Uso tiene uno o más diagramas de clases que muestran las clases participantes en el CU y sus relaciones.
- Tales diagramas son llamados Vista de Clases Participantes (*“View of Participating Classes”*) lo que se resume como VOPC.

15. En el siguiente gráfico, se muestra las clases participantes en el caso de uso “Evaluar Pedido al crédito”.



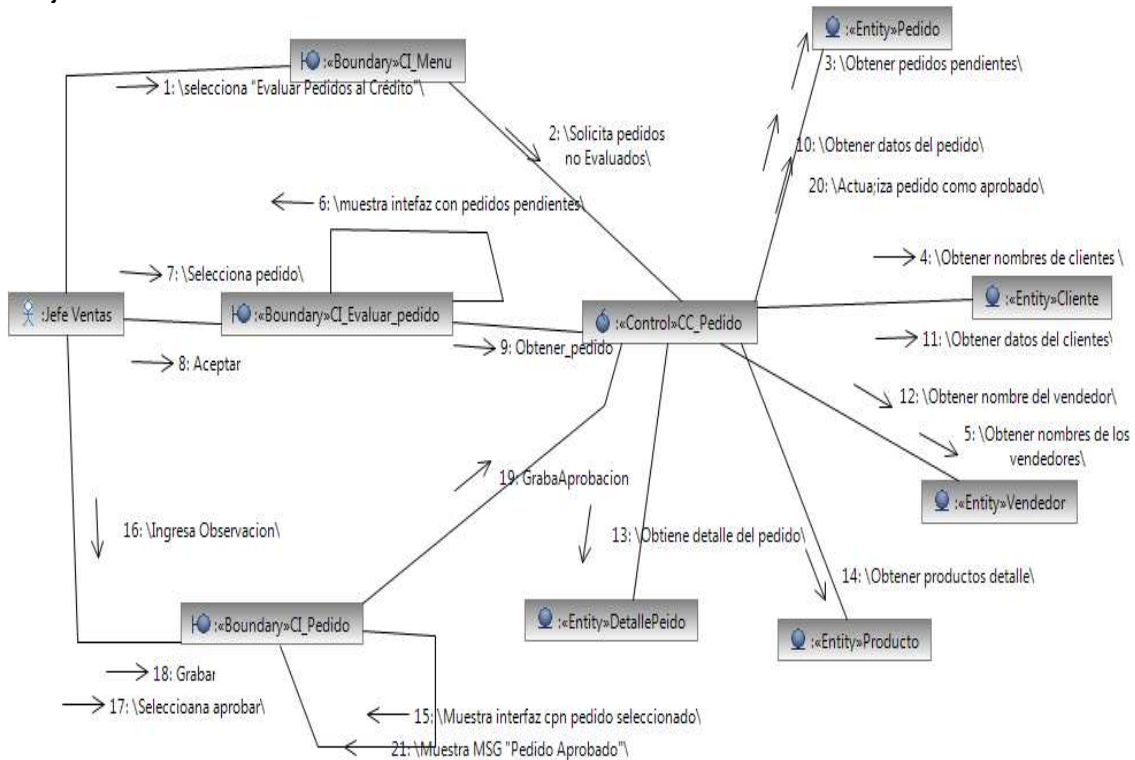
16. A continuación, se ordena y relaciona para mostrar el diagrama de clases. Si la especificación menciona un menú, se debe de agregar en el diagrama de clases y especificar las operaciones necesarias.



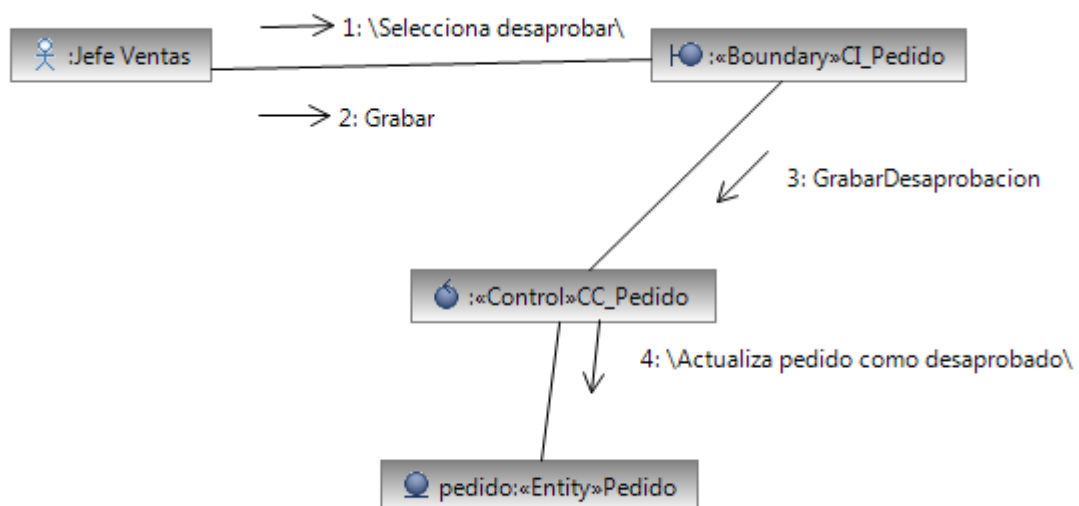
17. Diagrama de comunicación

Se deberán de crear tantos diagramas como escenarios existan. Para nuestro caso, tendremos 2 escenarios; en consecuencia, se crearan 2 diagramas de comunicación.

Flujo básico



Flujo denegar



ACTIVIDADES PROPUESTAS

1. A partir de la Especificación de Caso de Uso realice los siguientes diagramas:
 - a) Diagrama de clases de análisis
 - b) Diagrama de comunicación del flujo básico

ESPECIFICACIÓN DE CASO DE USO: Registrar Pedido

1 Actores

Secretaria

2 Propósito

El propósito es que la secretaria registre los pedidos a crédito de la distribuidora mayorista.

3 Breve Descripción

El sistema permite registrar un pedido al crédito.

4 Flujo Básico de Eventos

1. El caso de uso se inicia cuando la Secretaria selecciona la opción **"Solicitar Pedido al crédito"** del Menú Principal.
2. El sistema muestra la interfaz **"Registrar Pedido"** con los siguientes campos:
Numero de Pedido y Fecha , Nombre de Cliente y la opción **Buscar cliente**.
Nombre del Producto, la Opción **Buscar producto**, el campo cantidad y la **Opcion Agregar Producto**.
Una cuadrícula de detalle del pedido, que se cargará, con los productos, la cuadrícula contiene los siguientes campos (código, nombre, precio y cantidad). Se muestra, adicionalmente, una lista precargada de los vendedores y la opción **"Grabar Pedido"**.
3. La secretaria selecciona "Buscar Cliente".
4. El sistema incluye el Caso de Uso Buscar Cliente.
5. El sistema muestra los datos del cliente.
6. La secretaria solicita "Buscar producto" disponible.
7. El sistema incluye el Caso de Uso Buscar producto.
8. El sistema muestra los datos del producto.
9. La secretaria ingresa la cantidad del producto.
10. La secretaria selecciona agregar producto
11. El sistema agrega el producto a la cuadrícula del detalle del pedido.
12. Si la secretaria quiere seleccionar otro producto, se repite del paso 6 al 12.
13. La secretaria selecciona vendedor que atendió al cliente.
14. La secretaria selecciona "Grabar pedido".
15. El Sistema valida datos.
16. El sistema obtiene el último número de pedido y autogenera un nuevo número.
17. El sistema graba el pedido con su detalle en estado "Pendiente" y muestra el número del pedido.

5 Flujos Alternativos

<Vendedor no seleccionado>

En el paso 15, si el sistema verifica que no seleccionó al vendedor, muestra un mensaje de error. El flujo continúa en el paso 13 del flujo básico.

6 Precondiciones

La secretaria se debe haber logueado en el sistema.

Lista de vendedores disponible.

7 Poscondiciones

El sistema graba los datos del pedido en estado “**Pendiente**”.

8 Prototipos

http://localhost:8080/Ejemplo_Dojo/Pedido_credito.html

Nro de Pedido : ##### Fecha : 01/07/2011

Nombre del Cliente

Nombre del Producto

Cantidad de Productos

Codigo	Nombre del producto	Precio	Cantidad
00001	Tomacorriente	12.34	100
00002	enchufe de Plancha	12.45	300
00012	Alambre TW 14	45.5	300

Nombre de los vendedores

Resumen

- 📖 Para llevar a cabo el análisis de casos de uso se realiza lo siguiente:
 - Crear la realización de análisis de casos de uso.
 - Encontrar clases de análisis del comportamiento de casos de uso.
 - Crear el diagrama de clases (estructura del caso de uso).
 - Crear el diagrama de comunicación (comportamiento del caso de uso).
 - 📖 Una realización de casos de uso describe cómo un caso de uso en particular es modelado dentro del modelo de análisis, primero; y, después, dentro del modelo de diseño, en términos de objetos colaboradores.
 - 📖 Existen tres tipos de clases de análisis: Interfaz (boundary), Control (control) y Entidad (entity).
 - 📖 Si desea saber más acerca de estos temas, puede consultar el siguiente libro.
 - 🖱️ VISUAL MODELING WITH IBM RATIONAL ARCHITECT SOFTWARE ARCHITECT AND UML por Terry Quatrani y Jim Palistrant
- En el capítulo 6 de este libro se describe cómo crear un modelo de análisis.

4. ANÁLISIS DE CLASES

El objetivo de esta actividad es describir cada una de las clases que se ha encontrado en el análisis de casos de uso, identificando las responsabilidades que tienen asociadas, sus atributos y las relaciones entre ellas.

4.1. Pasos a realizar

Los pasos que se realizan en esta actividad son los siguientes:

- Identificación de responsabilidades y atributos
- Identificación de asociaciones y agregaciones
- Identificación de generalizaciones

4.1.1. Identificación de responsabilidades y atributos

El objetivo de esta tarea es identificar las responsabilidades y atributos relevantes de una clase.

Las responsabilidades de una clase definen la funcionalidad de esa clase y están basadas en el estudio de los papeles que desempeñan sus objetos dentro de los distintos casos de uso. A partir de estas responsabilidades, se puede comenzar a encontrar las operaciones que van a pertenecer a la clase. Éstas deben ser relevantes, simples y participar en la descripción de la responsabilidad.

Los atributos de una clase especifican propiedades de la clase, y se identifican por estar implicados en sus responsabilidades. Los tipos de estos atributos deberían ser conceptuales y conocidos en el dominio.

De manera opcional, se elabora una especificación para cada clase que incluye: la lista de sus operaciones y las clases que colaboran para cubrir esas operaciones y una descripción de las responsabilidades, atributos y operaciones de esa clase.

Para aquellas clases, cuyo comportamiento dependa del estado en el que se encuentren, se realiza, también de manera opcional, un diagrama de transición de estados.

4.1.2. Identificación de asociaciones y agregaciones

En esta tarea, se estudian los mensajes establecidos entre los objetos del diagrama de interacción para determinar qué asociaciones existen entre las clases correspondientes. Estas asociaciones suelen corresponderse con expresiones verbales incluidas en las especificaciones.

Las relaciones surgen como respuesta a las demandas en los distintos casos de uso y, para ello, puede existir la necesidad de definir agregaciones y herencia entre objetos.

Una asociación está caracterizada por los siguientes aspectos:

- Los papeles que desempeña.
- Su direccionalidad, que representa el sentido en el que se debe interpretar.

- Su cardinalidad, que representa el número de instancias implicadas en la asociación.

Dichas características pueden obtenerse a partir de la especificación de los casos de uso.

4.1.3. Identificación de generalizaciones

El objetivo de esta tarea es representar una organización de las clases que permita una implementación sencilla de la herencia y una agrupación semántica de las diferentes clases.

4.2. Tarjetas Clase – Responsabilidad – Colaboración (CRC)

A fines de la década de 1980, uno de los centros más grandes de tecnología de objetos era el laboratorio de investigación de Tektronix, en Portland, Oregon, Estados Unidos. Este laboratorio tenía algunos de los principales usuarios de Smalltalk y muchas de las ideas clave de la tecnología de objetos se desarrollaron allí. Dos de sus programadores renombrados de Smalltalk eran Ward Cunningham y Kent Beck.

Tanto Cunningham como Beck estaban y siguen preocupados por cómo enseñar los profundos conocimientos de *Smalltalk* que habían logrado. De esta pregunta sobre cómo enseñar objetos surgió la sencilla técnica de las tarjetas de Clase-Responsabilidad-Colaboración (CRC).

En lugar de utilizar diagramas para desarrollar modelos, como lo hacían la mayoría de los metodólogos, Cunningham y Beck representaron las clases en tarjetas 4 x 6 [pulgadas]. Y, en lugar de indicar atributos y métodos en las tarjetas, escribieron responsabilidades y colaboradores.

4.2.1. Descripción de las tarjetas CRC

La tarjeta se divide en tres compartimientos, tal como se muestra en la figura 4.1. En el compartimiento superior izquierdo, se escribe el nombre de la clase candidata; en el compartimiento inferior izquierdo, las responsabilidades; y, en el derecho, los colaboradores.

Nombre de Clase:	
Responsabilidades	Colaboradores

Figura 1.17. Tarjeta de Clase-Responsabilidad-Colaboración (CRC).

Las responsabilidades de una clase pueden recopilarse combinando todos los roles que cumple en las diferentes realizaciones de caso de uso. Identificar las realizaciones de caso de uso en las cuales participa mediante el estudio de sus diagramas de clases e interacción.

En el diagrama de comunicación cuando se tiene que describir los flujos básico y alternativos, un mensaje debería reflejar el propósito del objeto invocante en la interacción con el objeto invocado. Este propósito es el origen de una responsabilidad del objeto receptor y podría llegar hacer el nombre de una responsabilidad.

Los colaboradores son otras clases que pueden colaborar con esta clase para realizar una parte de la funcionalidad del sistema. El compartimiento de colabores proporciona una forma de grabar reacciones entre clases.

Uno de los principales beneficios de las tarjetas de CRC es que alientan la disertación animada entre los desarrolladores. Son especialmente eficaces cuando se está en medio de un caso de uso para ver cómo lo van a implementar las clases. Los desarrolladores escogen tarjetas a medida que cada clase colabora en el caso de uso. Conforme se van formando ideas sobre las responsabilidades, se pueden escribir en las tarjetas. Es importante pensar en las responsabilidades, ya que evita pensar en las clases como simples repositorios de datos, y ayuda a que el equipo se centre en comprender el comportamiento de alto nivel de cada clase.

4.2.2. Limitaciones de las tarjetas CRC

Aunque las tarjetas CRC pueden ser una herramienta poderosa para empezar el diseño, tienen limitaciones inherentes. El primer problema es que no tienen un escalamiento exitoso. En un proyecto muy complejo, puede agobiarse con las tarjetas CRC; el simple hecho de mantener un registro de todas ellas puede ser difícil. El problema se torna más difícil aún al tratar de mantener sincronizados entre sí, a las tarjetas y los modelos de UML de las clases.

Por otro lado, las tarjetas CRC tampoco capturan la interrelación entre clases. Aunque es cierto que todas las colaboraciones se toman en cuenta, la naturaleza de la colaboración no se modela bien. Al ver las tarjetas CRC no se puede saber quién crea a quién, si las clases se agregan unas a otras, o si una responsabilidad corresponde a una o más clases colaboradoras.

En resumen, las tarjetas CRC son un buen inicio por mantener documentadas las clases de análisis, pues a partir de ellas se tendrá una visión general del comportamiento de una clase con respecto de otras, pero una vez que se tenga conocimiento de las clases y lleguen a su etapa de diseño, estas tarjetas ya no les será útil y quizá ya no necesitará actualizarlas.

4.3. Caso práctico

A continuación, se explicará cómo confeccionar las tarjetas CRC para las clases participantes del caso de uso Pagar préstamos de terceros.

PASO 1: Revisar la Especificación del caso de uso

Especificación de Caso de uso: Pagar Préstamos de Terceros

1. Breve descripción

El caso de uso permite a un cliente del banco efectuar el pago de préstamos de terceros con cargo en su cuenta de ahorros. Además, actualiza el saldo contable en el “Sistema Contable”.

2. Actores

Cliente

3. Flujo de Eventos

3.1. Flujo Básico

1. El caso de uso comienza cuando el cliente selecciona la opción pago de préstamos y la sub opción Préstamos de Terceros en la interfaz del menú principal.
2. El sistema muestra la interfaz “Pago de Préstamos de Terceros” con los campos número de cuenta de préstamo, monto y moneda. Además, muestra una lista con las cuentas de cargo (cuenta de ahorros).
3. El Cliente ingresa cuenta de préstamo, monto y selecciona moneda.
4. El Cliente selecciona una cuenta de cargo.
5. El Cliente selecciona continuar.
6. El sistema verifica saldo de cuenta.
7. El sistema muestra la interfaz “Confirmación de Pago” con los datos: cuenta del préstamo, cuenta de cargo, monto y moneda del préstamo. Además, muestra el mensaje “Ingresar su clave para confirmar”.
8. El cliente ingresa su clave de acceso.
9. El cliente confirma la operación.
10. El sistema registra la transacción con el número de operación, cuenta de préstamo, cuenta de cargo, monto, moneda del préstamo, y fecha y hora de la transacción.
11. El sistema actualiza el saldo contable en el “Sistema Contable” y actualiza el saldo de la cuenta de cargo.
12. El sistema muestra el número de la operación y el MSG: “Pago de préstamo efectuado correctamente”.
13. El cliente selecciona “Salir”, se cierra la interfaz y el caso de uso finaliza.

3.2. Flujos Alternativos

3.2.1. Pago no efectuado

Si el sistema detecta que el saldo de la cuenta de cargo es insuficiente para realizar el pago, el sistema muestra el MSG “Saldo insuficiente para efectuar pago” y el caso de uso continúa en el paso 2.

4. Pre Condiciones

1. El cliente logeado al sistema con su número de tarjeta y clave.
2. Disponible las cuentas de ahorro (cuentas de cargo).

5. Post Condiciones

1. En el sistema quedará registrado la transacción.
2. En el sistema quedará actualizado el saldo de la cuenta de ahorros.
3. Se actualizará el saldo contable en el “Sistema Contable”.

6. Puntos de extensión

1. En el paso 9, si la moneda de la cuenta de cargo es diferente a la del préstamo, el sistema extiende al caso de uso “Convertir moneda” a una cuenta de ahorros del cliente.

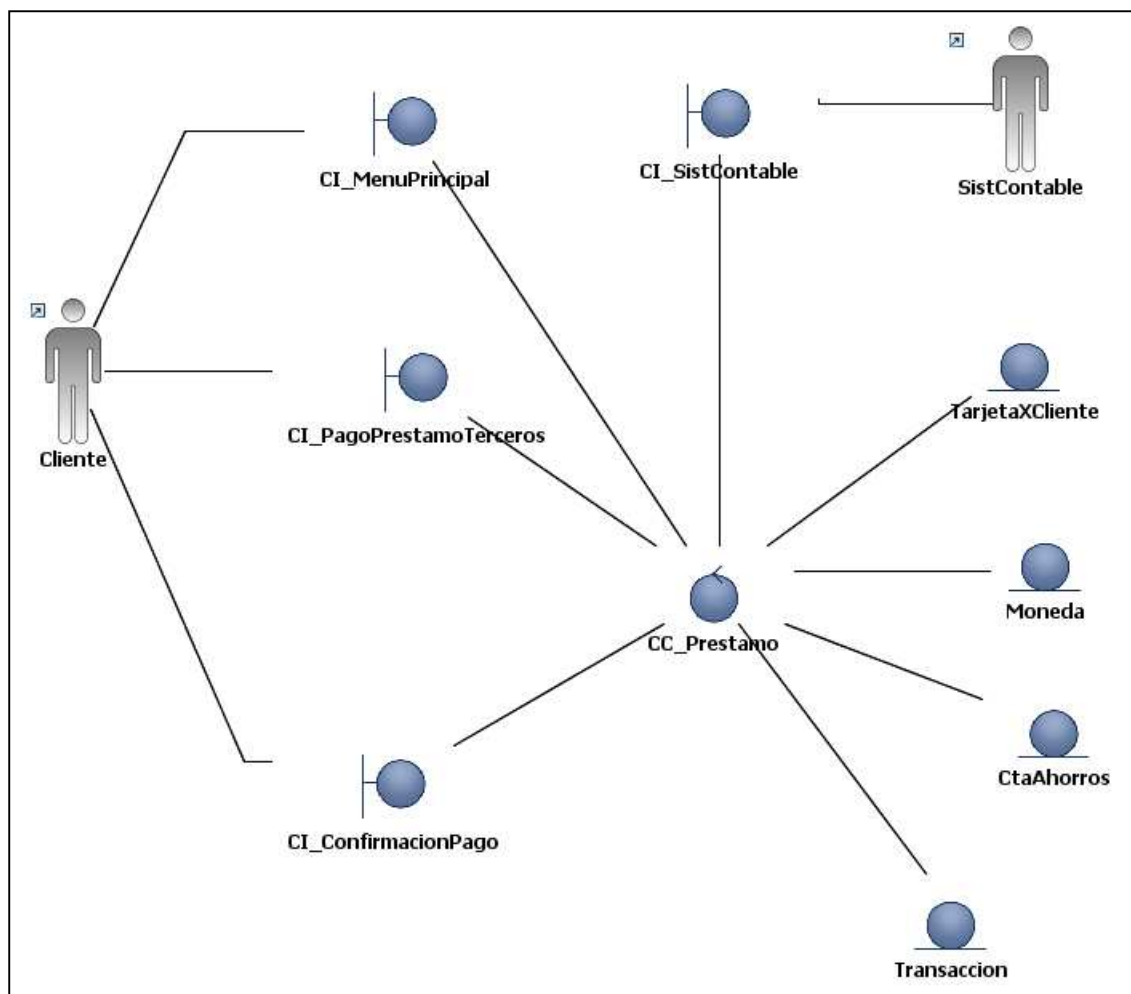
7. Requisitos Especiales

1. Alta seguridad en el manejo de las claves y cuentas.

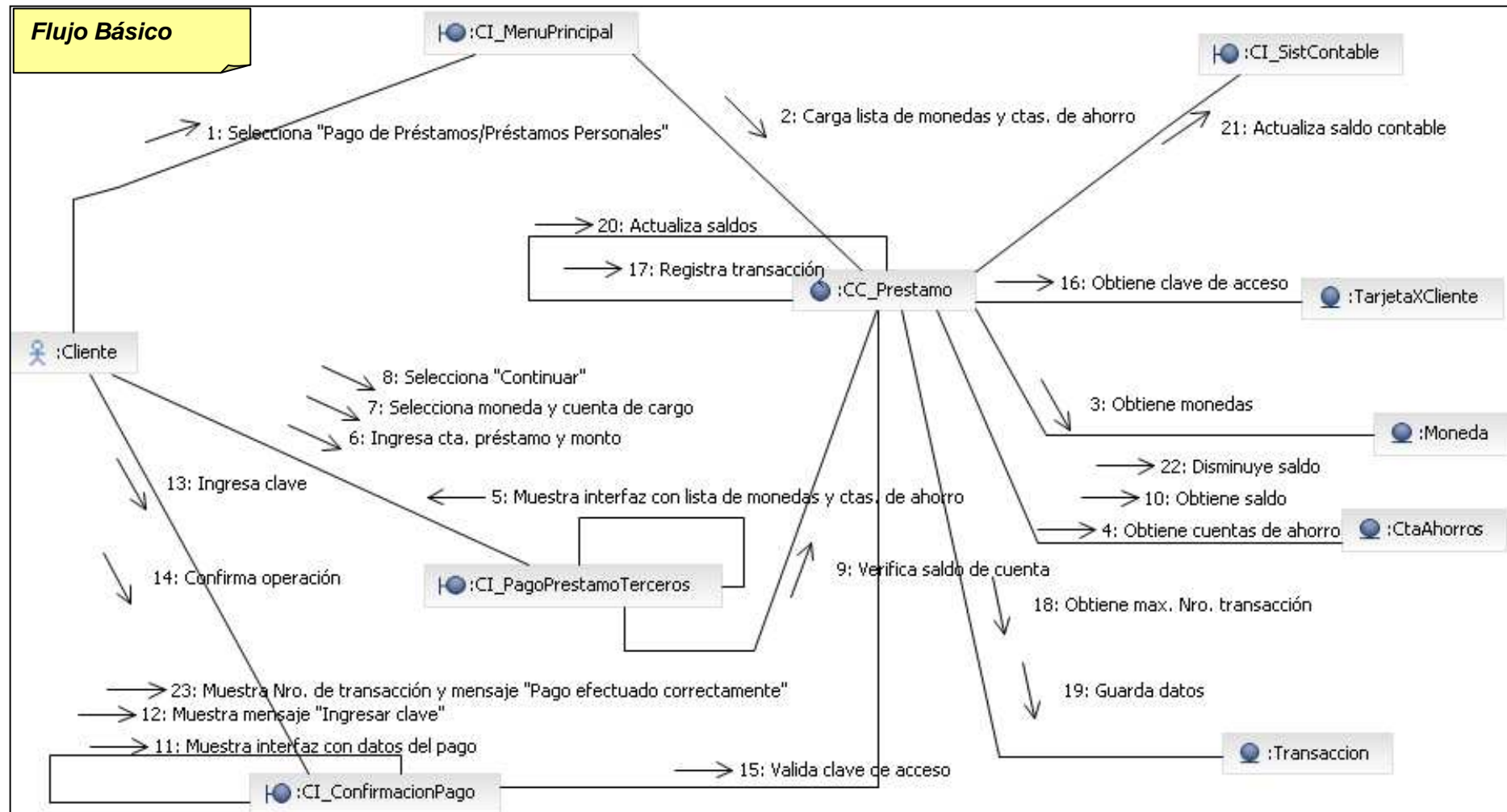
8. Prototipos

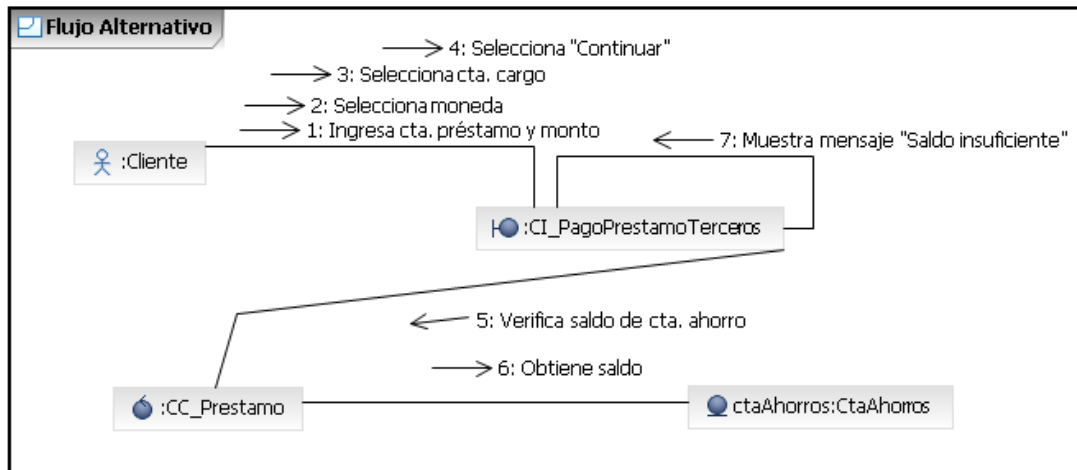
-

PASO 2: Realizar el diagrama de clases



PASO 3: Realizar el diagrama de comunicación del flujo básico y del flujo alternativo





PASO 4: Confeccionar las tarjetas CRC por cada clase de análisis

Se empezará a realizar las tarjetas CRC en orden alfabético por tipo de clase. Primero empezaremos con los *boundary*, luego con la clase control y, al final, con los *entity*.

Nombre de Clase: CI_ConfirmacionPago

Responsabilidad	Colaboradores
1. Solicitar clave de acceso 2. Solicitar la confirmación de operación 3. Mostrar interfaz con datos del pago 4. Mostrar mensaje "Ingresar clave" 5. Mostrar nro. de transacción 6. Mostrar mensaje "Pago efectuado correctamente"	CC_Prestamo

Nombre de Clase: CI_MenuPrincipal

Responsabilidad	Colaboradores
1. Solicitar la selección de "Pago de Préstamos/Préstamos Personales"	CC_Prestamo

Nombre de Clase: CI_PagoPrestamoTerceros

Responsabilidad	Colaboradores
1. Solicitar la selección de "Continuar" 2. Solicitar la selección de moneda 3. Solicitar la selección de cuenta de cargo 4. Solicitar cuenta de préstamos 5. Solicitar monto 6. Mostrar lista de monedas 7. Mostrar cuentas de ahorro 8. Mostrar mensaje "Saldo insuficiente"	CC_Prestamo

Nombre de Clase: CI_SistContable	
Responsabilidad	Colaboradores
1. Actualizar saldo contable	CC_Prestamo

Nombre de Clase: CC_Prestamo	
Responsabilidad	Colaboradores
1. Cargar monedas 2. Cargar cuentas de ahorros 3. Verificar saldo de cuenta de ahorro 4. Verificar clave de acceso 5. Registrar transacción 6. Actualizar saldos de cuentas	CI_MenuPrincipal CI_PagoPrestamoTerceros CI_ConfirmacionPago CI_SistemaContable TarjetaXCliente Moneda CtaAhorros Transacción

Nombre de Clase: CtaAhorros	
Responsabilidad	Colaboradores
1. Obtener cuentas de ahorro 2. Obtener saldos 3. Disminuir saldos	CC_Prestamo

Nombre de Clase: Moneda	
Responsabilidad	Colaboradores
1. Obtener monedas	CC_Prestamo

Nombre de Clase: TarjetaXCliente	
Responsabilidad	Colaboradores
1. Obtener clave de acceso	CC_Prestamo

Nombre de Clase: Transaccion	
Responsabilidad	Colaboradores
1. Obtener máximo número de transacción 2. Guardar datos	CC_Prestamo

ACTIVIDADES PROPUESTAS

A partir de la Especificación de Caso de uso realice los siguientes artefactos:

1. Diagrama de clases de análisis
2. Diagrama de comunicación del flujo básico
3. Diagrama de comunicación de los flujos alternativos
4. Tarjeta CRC de las clases

Especificación de caso de uso: Reservar pistas de juego

1. Descripción:

El caso de uso permite, al encargado de reservas de un club de deportes, registrar una reserva de pista para un socio. El sistema se comunica con el Sistema de Cuentas por Cobrar (SCC) para registrar las reservas pendientes de pago del socio.

2. Actores

Encargado de reservas

3. Flujo de Eventos

3.1. Flujo Básico

1. El caso de uso comienza cuando el encargado de reservas selecciona la subopción “Reservar pistas” de la opción “Reservas” de la interfaz del menú principal.
2. El sistema muestra la interfaz “RESERVAS PISTAS” con una Lista de todas las reservas de pistas del día, ordenadas por hora prevista, y campos para realizar una reserva.
 - Datos de la lista: número de reserva, nombre completo del socio que reservó, número de pista, hora de entrada y número de horas que estará ocupada la pista.
 - Datos de la Reserva: código de socio, fecha y hora de reserva, número de pista y número de horas que estará ocupada la pista.
 - Además, presenta las opciones: Grabar reserva, Buscar socio y Consultar disponibilidad de pistas.
3. El Encargado de reservas ingresa el código del socio.
4. El Encargado de reservas selecciona “Consultar disponibilidad de pistas”.
5. El sistema **incluye el caso de uso Consultar disponibilidad de pistas.**
6. El sistema muestra la fecha y hora de reserva, Nro. de pista y Nro. de horas que estará ocupada la pista.
7. El encargado de reservas selecciona “Grabar reserva”.
8. El sistema valida los datos ingresados.
9. El sistema genera el número de reserva y registra la reserva de pista, registra el estado de la pista como reservada en la fecha y horas indicadas.
10. El sistema se comunica con el SCC para registrar la reserva pendiente de pago del socio con los siguientes datos: número de reserva, fecha de registro, código de socio, número de pista, horas de uso y monto a pagar.
11. El sistema muestra el MSG “Reserva generada” y el caso de uso termina.

3.2. Flujos Alternativos**3.2.1. Código de Socio no Válido**

Si el sistema detecta que el código del socio no es válido, muestra el MSG “Código de Socio incorrecto” y el sistema ofrecerá la posibilidad de buscar al socio y el caso de uso continúa en el punto 7.

4. Pre Condiciones

1. El Encargado de reservas logeado al sistema.
2. Comunicación activa con el SCC.
3. Lista de Socios registrados.
4. Lista de pistas de juego disponibles.

5. Post Condiciones

1. El sistema quedará registrado la reserva de pista de juego.
2. La disponibilidad de la pista de juego quedará registrada como reservada en la fecha y horas especificadas.
3. En el SCC quedará registrado la reserva del socio pendiente de pago.

6. Puntos de Extensión

2. En el punto 3, si el socio no muestra su carné en el cual se indica su código, el Sistema extiende al caso de uso “Buscar socio”.

7. Requisitos Especiales

1. Comunicación con SCC.

8. Prototipos

(Diseñe el prototipo)

Resumen

- 📖 Para llevar a cabo el análisis de clases, se realiza lo siguiente:
- Identificación de responsabilidades y atributos
 - Identificación de Asociaciones y Agregaciones
 - Identificación de Generalizaciones
- 📖 La tarjeta CRC es una técnica para identificar responsabilidades y colaboradores de una clase. Su objetivo es facilitar la comunicación y documentar los resultados del análisis de clases.
- 📖 La tarjeta se divide en tres compartimientos. En el compartimiento superior izquierdo, se escribe el nombre de la clase candidata; en el compartimiento inferior izquierdo, las responsabilidades; y, en el derecho, los colaboradores.

Nombre de Clase:	
Responsabilidades	Colaboradores

- 📖 Si desea saber más acerca de estos temas, puede consultar el siguiente enlace.

🔗 <http://c2.com/doc/oopsla89/paper.html>

En este link muestra un *paper* sobre las Tarjetas CRC.

UNIDAD DE
APRENDIZAJE

2

MODELO DE DATOS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al finalizar la segunda unidad, el alumno crea el modelo de datos de un *software* el cual incluye: el modelo conceptual, el modelo lógico y el modelo físico de la base de datos. Los artefactos serán creados utilizando la herramienta *CASE IBM Rational Software Architect (RSA)* e *IBM Rational InfoSphere Data Architect (IDA)*.

TEMARIO

Tema 1: Modelo de Datos

- 1.1. Modelo Conceptual
- 1.2. Modelo Lógico
- 1.3. Modelo físico
- 1.4. Recomendación en el manejo de Herencia
- 1.5. Auditoria en base de datos

ACTIVIDADES PROPUESTAS

- 1. Los alumnos desarrollan el modelo conceptual de un caso propuesto.
- 2. Los alumnos desarrollan el modelo físico de un caso propuesto.

1. MODELO CONCEPTUAL

Las clases del modelo conceptual se obtienen a partir de los objetos de información que fluyen entre las actividades. Una característica importante que resaltar es que el modelado de los casos de uso del sistema y el modelado conceptual se realizan en paralelo, esto es crucial para obtener casos de uso correctos, puesto que es necesario entender bien el dominio para poder escribir casos de uso que sean realmente útiles.

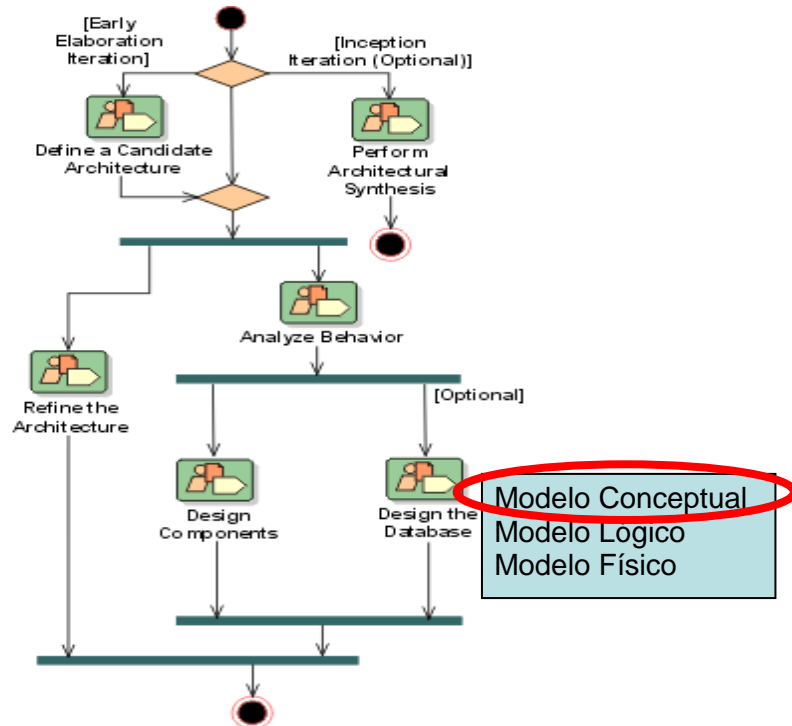


Figura 2.1. El Modelo Conceptual en el Flujo de Trabajo de Análisis y Diseño.

1.1. Importancia del Modelo Conceptual

El Modelo Conceptual Orientado a Objetos beneficiará a dos equipos de trabajo:




- Equipo de Desarrolladores**
 En esta etapa del desarrollo, es conveniente detenerse en la identificación de los conceptos y no tanto en las relaciones entre ellos. Este modelo incluirá los conceptos y sus relaciones y se describirá mediante un diagrama de clases UML, en el que los conceptos se representan mediante clases (del dominio).
- Equipo de Base de Datos**
 En esta etapa, luego del modelo conceptual, se obtiene el proceso del modelo lógico al diseño físico donde se podrá identificar las tablas relacionales del proyecto de Base de Datos como componente RUP.

1.2. Construcción del Modelo Conceptual

A continuación, se describe los pasos que se realizan para la construcción del modelo conceptual, los cuales son los siguientes:

- Identificar clases persistentes con sus atributos
- Asociar clases
- Identificar agregaciones
- Definir jerarquías de clases

1.2.1. Identificar clases persistentes con sus atributos

- La Clase es la unidad básica que encapsula toda la información de un objeto (un objeto es una instancia de una clase). A través de ella, podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.).
- Los atributos representan las propiedades de la clase que se encuentran en todas las instancias de la clase. Definen la estructura de una clase y de sus objetos.
- Los atributos corresponden a sustantivos y sus valores pueden ser sustantivos o adjetivos.
- Dentro de una clase, los nombres de los atributos deben ser únicos (aunque puede aparecer el mismo nombre de atributo en diferentes clases).
- Los atributos pueden representarse solo mostrando su nombre, su tipo e, incluso, su valor por defecto.
 - **Public:** Indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados. 
 - **Private:** Indica que el atributo solo será accesible desde dentro de la clase (solo sus métodos lo pueden acceder). 
 - **Protected:** Indica que el atributo no será accesible desde fuera de la clase, pero sí podrá estar disponible para métodos de la clase, además, de las subclases que se deriven. 

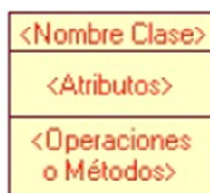


Figura 2.2. Miembros de una clase.

- Para los Identificadores, en el momento de incluir atributos en la descripción de una clase se debe distinguir entre los atributos que reflejan las características de los objetos en el mundo real y los identificadores que son utilizados por razones de implementación.

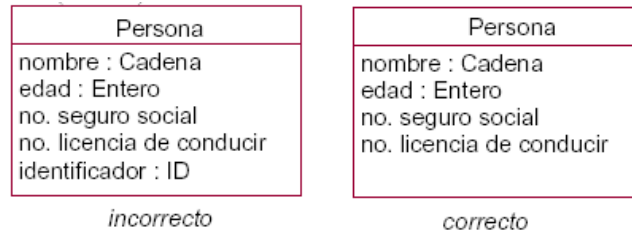


Figura 2.3. Clase con identificador Vs. clase sin identificador.

- Guías prácticas para la definición de Clases y Atributos
 - Las clases poseen información descriptivas; los atributos, no.
 - Los atributos multivaluados deben ser clasificados como clases.
 - Convertir, en una clase, a un atributo que tenga una relación muchos a uno con otra clase.
 - Asociar atributos a las clases que ellos describen más directamente. Los atributos deben ser inherentes a la clase.
 - Evitar los identificadores compuestos en la medida que sea posible.
- Existen algunas categorías de clases que podríamos utilizar para identificarlas correctamente.

Categoría	Ejemplo
Tangibles o físicos	Edificio, Producto
Especificaciones o descripciones	EspecificacionProducto, DescripcionVuelo
Lugares	Tienda, Aula, Laboratorio
Transacciones	Venta, Pago, Reserva
Líneas o detalle de transacción	LineaVenta, DetalleReserva
Registros de finanzas, expedientes	CDP, Factura, Ticket, HistoriaClinica
Roles de personas	Cajero, Piloto
Organizaciones	Departamento, Sucursal
Historiales	PrecioProducto, PrecioDolar, AtencionCitas
Registros de cambios de estados	DisponibilidadHabitacion, DisponibilidadButaca
Conceptos abstractos	RangoHora, UnidadAprendizaje
Relaciones	Amistad, Parentesco

Tabla 2.1. Categoría de clases del dominio del problema.

1.2.2. Asociar clases

- La asociación es una relación entre clases que indica una conexión significativa e interesante.
- Está representada como una línea entre clases con nombre. La asociación es inherentemente bidireccional.
- Es convencional leer la asociación de izquierda a derecha o de arriba hacia abajo.
- Criterios para identificar asociaciones
 - Enfocarse en aquellas asociaciones para las cuales el conocimiento de la relación necesita ser conservado en el tiempo. (Asociaciones que se necesitan saber).
 - Demasiadas asociaciones tienden a confundir.
 - Evitar mostrar asociaciones redundantes o derivables.
 - Pueden existir múltiples asociaciones entre dos clases.

a) Tipos de Asociaciones

- Asociación Binaria: Asociación entre dos clases.

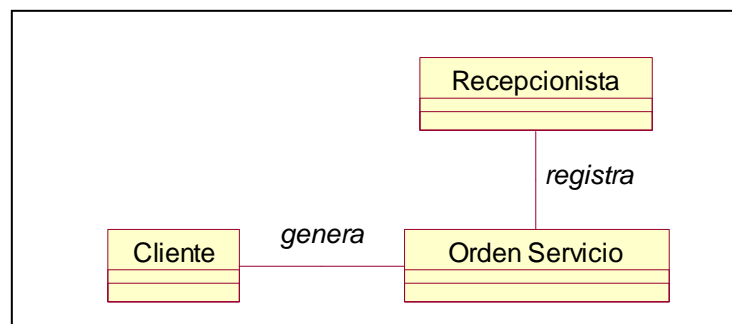


Figura 2.4. Asociación Binaria.

- Asociación de clase: Asociación entre dos clases que contiene otra entidad. Generalmente, este tipo de asociación se utiliza para representar una relación de muchos a muchos entre dos clases.

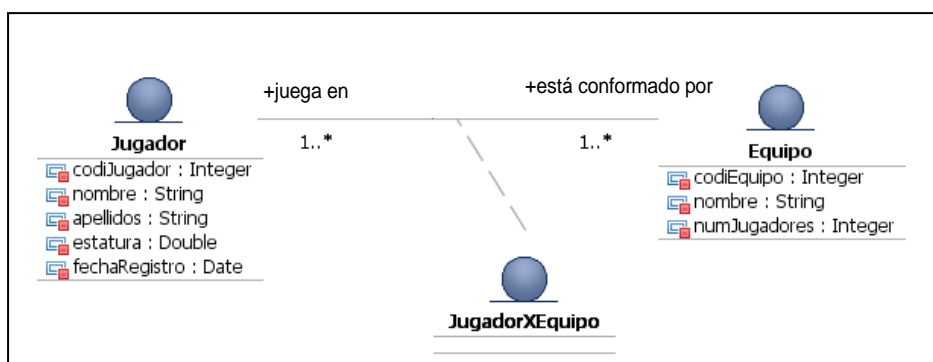


Figura 2.5. Asociación de clase.

- Asociación Reflexiva: Se da en la misma clase.

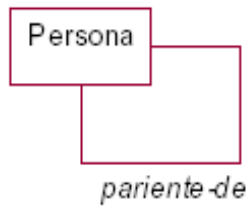


Figura 2.6. Asociación Reflexiva.

b) Clase Asociativa

- Se da cuando uno o más atributos están relacionados con la asociación.
- Las instancias de la clase asociativa dependen del tiempo de vida de la asociación.
- Se da en una asociación de muchos a muchos entre dos clases y existe información asociada con la propia relación de asociación.

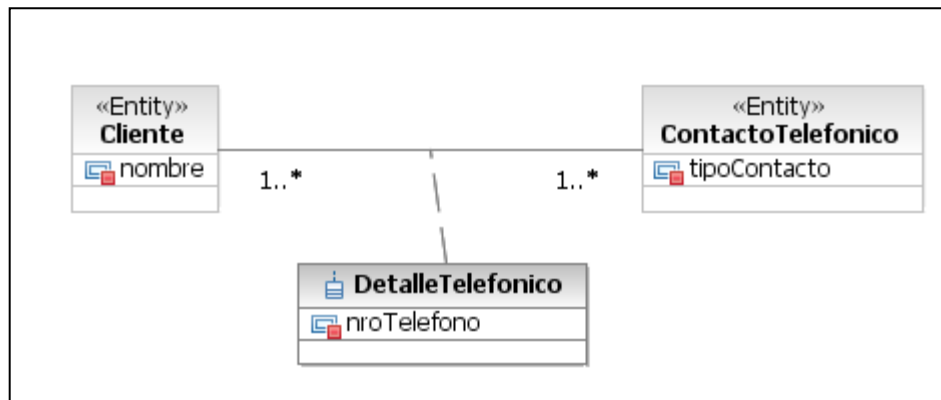


Figura 2.7. Clase asociativa.

c) Roles

- Dada una asociación entre dos entidades, decimos que **cada entidad representa un rol en dicha asociación**.
- Muchas veces, según el punto de vista de cada entidad, **es posible nombrar a la asociación de manera diferente**.

d) Multiplicidad

- Restringe el número de objetos de una clase que se pueden implicar en una relación determinada en cualquier momento en el tiempo. La frase "en cualquier momento en el tiempo" es vital para entender las multiplicidades.
- Define cuántas instancias de la clase A pueden estar asociadas con una instancia de la clase B.
- La multiplicidad presenta las relaciones con valores de datos de acuerdo al detalle siguiente :

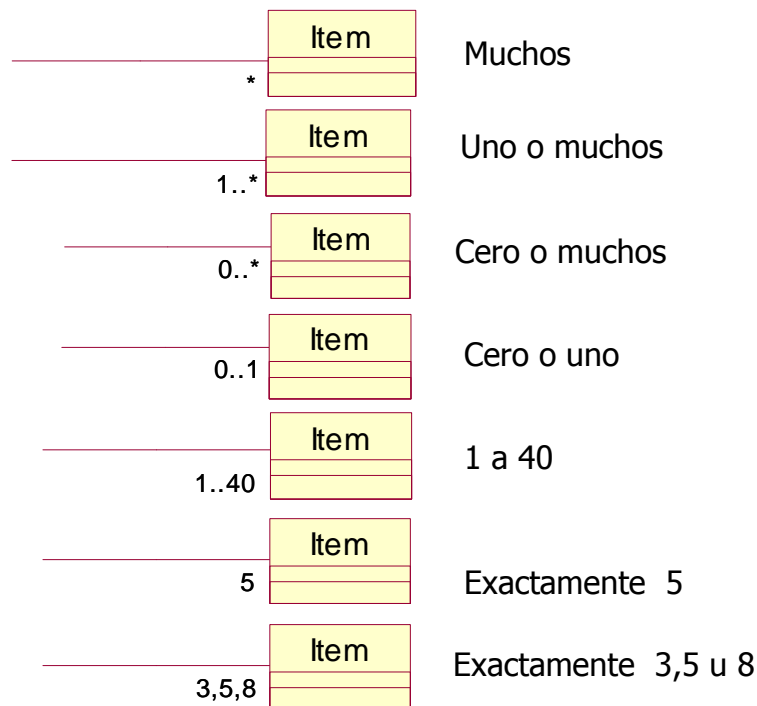


Figura 2.8. Multiplicidad.

Ejemplo:

En el siguiente ejemplo, se representa las siguientes relaciones:

- Un jugador **“juega en”** muchos equipos
- Un equipo **“está conformado por”** varios jugadores.
- Cada jugador, dependiendo del equipo en que se encuentre tendrá un rol diferente.

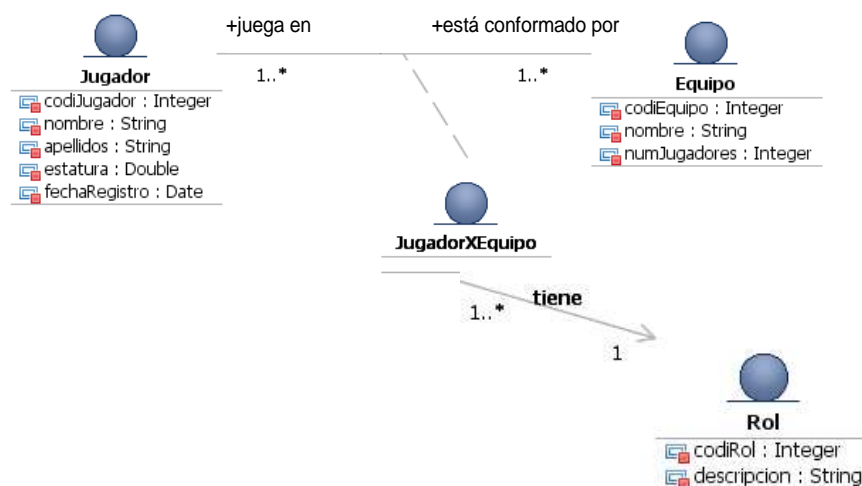


Figura 2.9. Asociaciones y multiplicidades.

1.2.3. Identificar agregaciones

- La agregación indica una relación de “un todo conformado por partes”
- Existen 2 tipos de agregaciones:
 - Agregación Débil o Compartida
 - Agregación Fuerte o Compuesta
- La agregación representa una relación *parte de* entre objetos.
- En UML se proporciona una escasa caracterización de la agregación.
- Puede ser caracterizada con precisión determinando las relaciones de comportamiento y estructura que existen entre el objeto agregado y cada uno de sus objetos componentes.

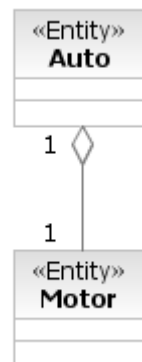


Figura 2.10. Agregaciones entre clases.

a) Agregación Compartida

Es un tipo de relación utilizada para modelar la relación todo-parte entre objetos. La parte puede estar simultáneamente en varias instancias del todo.

La agregación existe de preferencia entre conceptos no físicos.

Ejemplo:

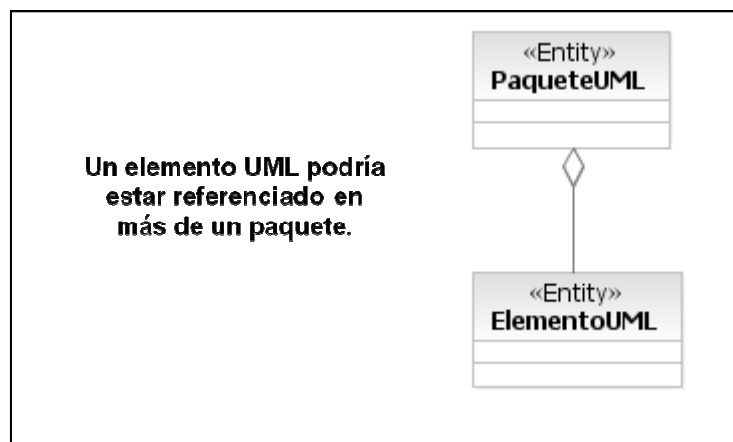


Figura 2.11. Agregación compartida.

b) Agregación Compuesta

Es un tipo de relación utilizada para modelar la relación todo-parte entre objetos. Significa que la parte es miembro de solamente un objeto todo, es decir, la existencia de la parte depende del todo. La composición se representa con un diamante relleno.

El objeto todo es el único dueño del objeto parte.

Ejemplo:

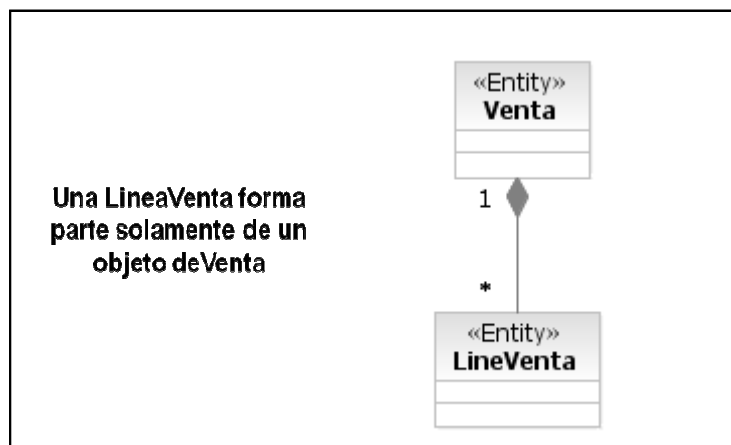


Figura 2.12. Agregación compuesta.

1.2.4. Definir jerarquías de clases

Permiten gestionar la complejidad mediante un ordenamiento lógico. Se obtiene usando los mecanismos de abstracción de Generalización y/o Especialización.

Generalización y Especialización son conceptos fundamentales en el Modelo Conceptual que permiten la reducción de la expresión. Las jerarquías de clases son a menudo las bases de inspiración para las jerarquías de las clases de software que exploten la herencia y reduzcan la duplicación de código.

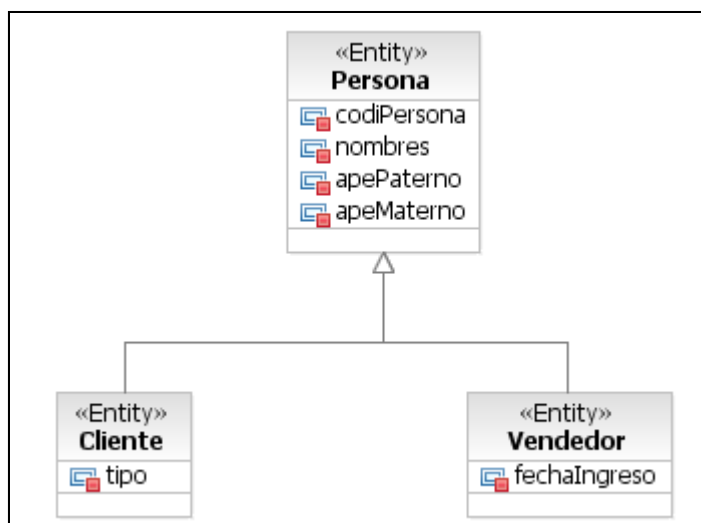


Figura 2.13. Jerarquía de clases.

a) La Generalización

- Consiste en factorizar las propiedades comunes de un conjunto de clases en una clase más general.
- Nombres usados: clase padre - clase hija, superclase - subclase, clase base - clase derivada
- Las subclases **heredan** características de sus superclases, es decir, los atributos y operaciones (y asociaciones) de la superclase están disponibles en sus subclases.

Ejemplo:



Figura 2.14. Generalización.

b) La Especialización

- Es el resultado de tomar un subconjunto de entidades de alto nivel para formar un conjunto de entidades de más bajo nivel.
- Las entidades de bajo nivel presentan atributos adicionales diferenciándolos de las otras entidades que se han creado producto de la especialización.

Ejemplo:

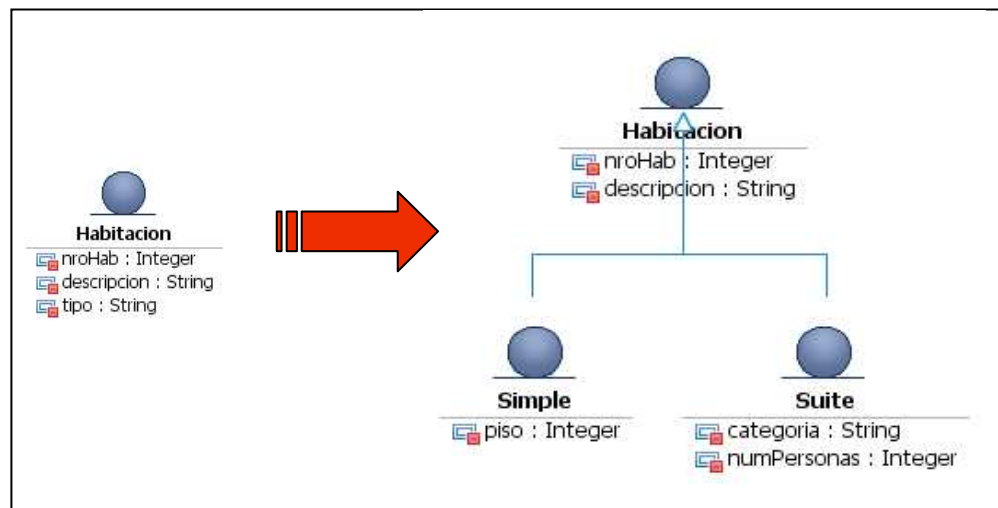


Figura 2.15. Especialización.

1.3. Caso práctico

A continuación, se mostrará el modelo conceptual obtenido a partir de la ECU Generar reserva.

Especificación de Caso de Uso: Generar Reserva

1. Descripción

El caso de uso permite a la Recepcionista de un Hotel generar reserva de habitación(es).

2. Actor(es)

Recepcionista

3. Flujo de Eventos

3.1. Flujo Básico

1. El Caso de uso se inicia cuando la recepcionista selecciona la opción “Generar Reserva” en la interfaz del menú principal.
2. El sistema muestra la interfaz RESERVA con los siguientes datos:
 - Datos del cliente: Código, Nombre.
 - Datos de la Reserva: fecha de llegada, fecha de salida y cantidad de días a hospedarse.
 - Datos de las habitaciones: Número de habitación, Tipo, Costo por día, Nombre del huésped de la Habitación y una opción para Agregar.
 - Además, incluye una cuadrícula que contiene la lista de todas las habitaciones seleccionadas y las opciones: **Buscar cliente, Agregar cliente, consultar disponibilidad de habitación, eliminar habitación, Grabar y Salir.**
3. La recepcionista selecciona “Buscar cliente”.
4. El sistema **incluye el Caso de Uso Buscar Cliente.**
5. El sistema muestra los datos del cliente.
6. La recepcionista ingresa la fecha de llegada y la fecha de salida.
7. El sistema calcula la cantidad de días.
8. La recepcionista solicita “Consultar disponibilidad de habitación”.
9. El sistema **incluye el caso de uso consultar disponibilidad de habitación.**
10. El sistema muestra la habitación seleccionada.
11. La Recepcionista ingresa el nombre de la persona para la habitación seleccionada.
12. La Recepcionista selecciona: agregar
13. El sistema calcula el pago de la habitación, el subtotal, el monto total y lo agrega a la cuadrícula del detalle de la reserva.
14. Si la Recepcionista quiere seleccionar otra habitación, se repite del paso 7 al 12.
15. La recepcionista selecciona “Grabar”.
16. El sistema autogenera un número de reserva.
17. El sistema graba la reserva con su detalle y registra la(s) disponibilidad(es) de la(s) habitación(es) en estado “Reservado”.
18. El sistema muestra el número de reserva y el MSG “Reserva generada” con el Nro. 99999”.
19. La recepcionista cierra la interfaz RESERVA y regresa a la interfaz del menú principal del sistema y finaliza el caso de uso.

3.2. Subflujos

Ninguno.

3.3. Flujos Alternativos

1. Cliente no existe

En el paso 4, si el sistema detecta que el cliente no existe, muestra el MSG: “Cliente no existe” y ofrecerá la posibilidad de registrar al nuevo cliente.

2. Habitaciones no disponibles

En el paso 9, si el sistema detecta que no hay habitaciones disponibles muestra el MSG: “No hay habitaciones disponibles” y finaliza el caso de uso.

3. Eliminar Habitación de Cuadrícula

La recepcionista selecciona una habitación de la cuadrícula y selecciona eliminar, el sistema elimina de la cuadrícula la habitación selecciona y el caso de uso continúa.

4. Precondiciones

4.1. El Recepcionista está logeado en el sistema.

4.2. Lista de clientes disponibles.

4.3. Lista de habitaciones disponibles.

5. Poscondiciones

5.1. En el sistema quedará registrada la reserva con su detalle.

5.2. Las disponibilidades de las habitaciones seleccionadas se registraran en estado “Reservadas”.

6. Puntos de Extensión

En el paso 5, el sistema extiende al caso de uso **Mantener Clientes – subflujo “Agregar Cliente”**.

7. Requerimientos Especiales



Ninguno.

8. Prototipos

Interfaz RESERVA

Reserva

Datos del Cliente


Cliente  


Nombre

Datos de la Reserva

Fecha de Llegada Fecha de Salida Cantidad Dias

Datos de las Habitaciones

Numero de Habitacion  Tipo Costo x día

Nombre del Huesped 



Detalle de la Reserva

Numero	Tipo	Huesped	Costo diario	Días	Monto a Pagar

Sub total

IGV

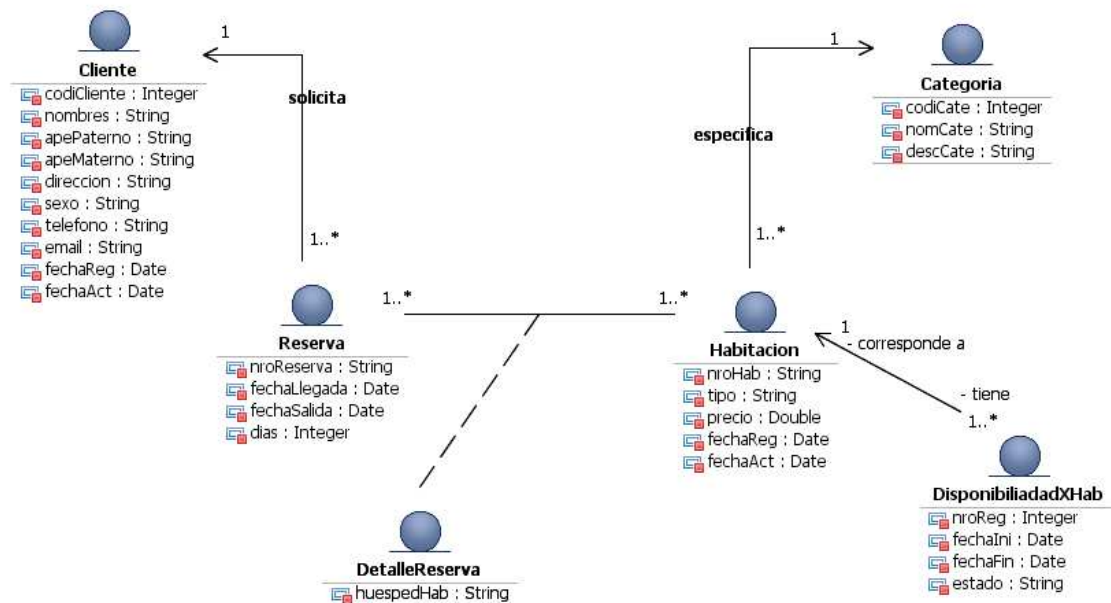
Monto Total

Solución:

El modelo conceptual nos muestra las siguientes relaciones:

- Asociación de clase: Reserva-Habitación
- Asociaciones unidireccionales (con navegabilidad):
 - Reserva-Cliente
 - Habitación-Categoría
 - DisponibilidadXHab- Habitación



ACTIVIDADES PROPUESTAS

1. A partir de la Especificación del Caso de Uso “Reservar pistas de juego”, de la sesión anterior, confeccione el Modelo Conceptual.

2. MODELO LÓGICO

Un modelo lógico de datos es un modelo que no es específico de una base de datos que describe aspectos relacionados con las necesidades de una organización para recopilar datos y las relaciones entre estos aspectos.

El modelo lógico de datos es el refinamiento del modelo conceptual. En este modelo no es necesario especificar las llaves primarias y foráneas de las entidades, pues es trabajo que se recomienda realizar en el modelo físico.

2.1. Tipos de datos

Existen muchas herramientas que permiten realizar transformaciones de un modelo lógico a partir de un modelo conceptual (con notación UML). La transformación UML a modelo lógico de datos genera tipos de datos de modelo lógico de datos a partir de los tipos primitivos de UML. En la siguiente tabla, se muestra la correspondencia entre tipos primitivos de UML y tipos de datos de modelo lógico de datos que se obtienen al utilizar la herramienta InfoSphere Data Architect (IDA) de IBM:

Tipos primitivos de UML	Tipos de datos de modelo lógico de datos que la transformación genera
Boolean, boolean	BOOLEAN
Byte, byte o char	CHAR
Date	DATE
Double, double	DOUBLE
float	FLOAT
Integer, int	INTEGER
Long, long	LONG
short	SHORT
Cadena de caracteres	VARCHAR(32672)

Tabla 2.2. Correlaciones entre tipos de datos de modelo lógico de datos y UML en IDA.

2.2. Tipos de relaciones

Los tipos de relaciones que se pueden crear para un modelo lógico son los siguientes:

2.1.1. Relación Identificada

Las relaciones identificadas **migran la llave primaria de la entidad padre a la llave primaria de la entidad hija**. Su representación es una línea nítida, tal como se ilustra en la siguiente figura.

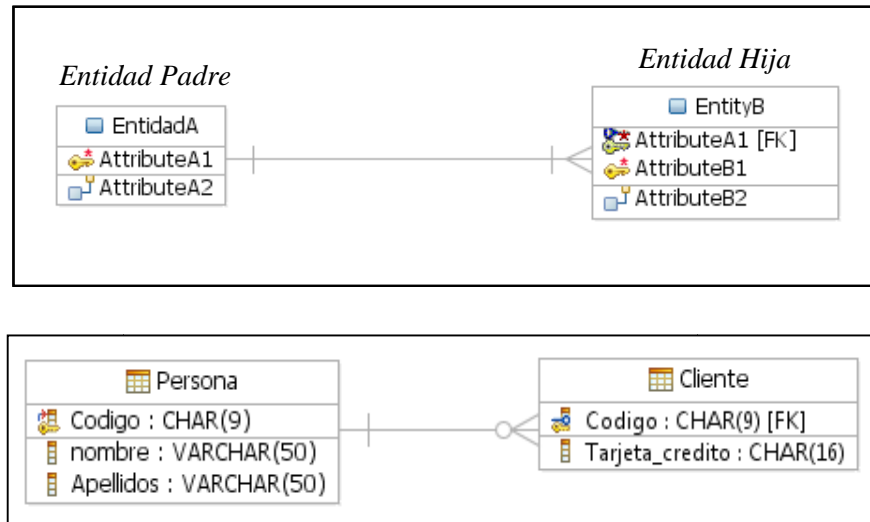


Figura 2.16. Relación Identificada.

2.1.2. Relación No identificada

En una relación no identificada **la llave primaria de la entidad padre es migrada como un atributo más de la entidad hija**, es decir, no formará parte de su llave primaria.

Además, en este tipo de relación se representa otra característica: la existencia. La existencia describe la relación entre un par de entidades desde la perspectiva de la entidad hija. Fundamentalmente, haciendo la pregunta, ¿Es el valor de una llave foránea siempre requerida en la entidad hija? Las posibles respuestas son las siguientes:

- a) Opcional. Cuando el valor de una llave foránea no es siempre requerido en la entidad hija. Sin embargo, si un valor existe, el valor de la llave foránea debe encontrarse en la llave primaria de la entidad padre.

La representación de la relación No identificada Opcional es una línea entrecortada y en el extremo de la entidad padre aparece una línea con un círculo. Así:

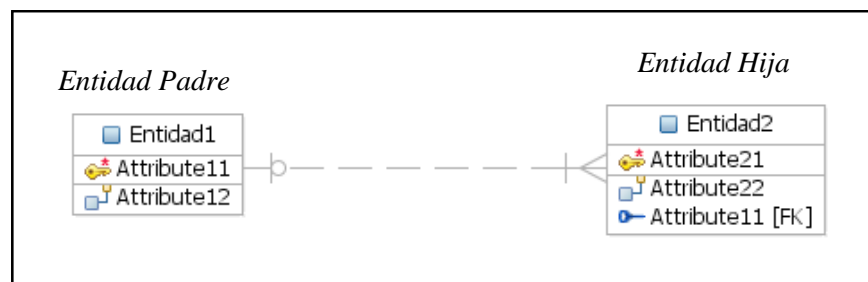


Figura 2.17. Relación No Identificada Opcional.

- b) Obligatoria. Cuando el valor de una llave foránea debe existir en la entidad hija y el valor de la llave foránea debe encontrarse en la llave primaria de la entidad padre.

La representación de la relación No identificada obligatoria es una línea entrecortada y en el extremo de la entidad padre aparece una línea. Así:

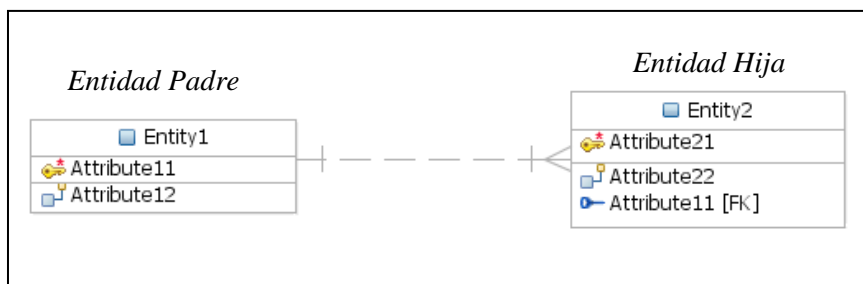


Figura 2.18. Relación No Identificada Obligatoria.

Un proceso de transformación del modelo conceptual (con notación UML) a modelo lógico de datos, genera tipos de relación, existencia y cardinalidad para el modelo lógico de datos de acuerdo a los tipos de asociación y multiplicidad de rol UML.

En la siguiente tabla, se muestran las correlaciones entre un tipo de asociación o multiplicidad de rol UML y el tipo de relación, existencia y cardinalidad de un modelo lógico de datos que se crean en un proceso de transformación en el entorno de IDA.

UML			Modelo lógico de datos		
Tipo de asociación	Multiplicidad de rol padre	Multiplicidad de rol hijo	Tipo de relación	Existencia	Cardinalidad
Simple o agregación	(0..1)	(0..1) /1/*/(1..*)	No identificada	Opcional	(0..1) /1/*/(1..*)
Simple o agregación	1	(0..1) /1/*/(1..*)	No identificada	Obligatoria	(0..1) /1/*/(1..*)
Composición	(0..1) /1	(0..1) /1/*/(1..*)	Identificada	Siempre son Obligatorias	(0..1) /1/*/(1..*)

Tabla 2.3. Correlaciones entre asociaciones UML y relaciones de modelo lógico de datos en IDA.

3. MODELO FÍSICO

El modelo físico es un modelo de datos de bajo nivel. Proporcionan conceptos que describen los detalles de cómo se almacenan los datos en el ordenador.

El paso de un modelo lógico a uno físico requiere un profundo entendimiento del manejador de bases de datos que se desea emplear, incluyendo características como:

- Conocimiento a fondo de los tipos de objetos (elementos) soportados
- Detalles acerca del indexamiento, integridad referencial, restricciones, tipos de datos, etc
- Detalles y variaciones de las versiones
- Parámetros de configuración
- Data Definition Language (DDL)
- El paso de convertir el modelo lógico de datos a tablas hace que las entidades pasen a ser tablas (más las derivadas de las relaciones) y los atributos se convierten en las columnas de dichas tablas.

Las relaciones entre las tablas del modelo físico son las mismas que se describieron en el modelo lógico:

- Relación Identificada
- Relación No Identificada Opcional
- Relación No Identificada Obligatoria

3.1. Campos

3.1.1. Tipos de Datos

Los tipos de datos son según los tipos de datos disponibles en el sistema manejador de base de datos (SMBD). En la mayoría de casos, se considera lo siguiente:

- Número de dígitos en números enteros
- La precisión de los flotantes
- Cadenas de caracteres de longitud fija. Ejemplo: CHAR(5)
- Cadenas de caracteres de longitud variable. Ejemplo: VARCHAR(50)
- Para archivos binarios (imágenes) se utiliza el tipo BLOB (Binary large objects)
- Para cadenas de longitudes grandes se utiliza el tipo CLOB (Character large objects)

3.1.2. Llaves primarias

Representa el identificador de una tabla y está formada por uno o más campos de la tabla.

Algunos SMBD poseen la capacidad de "autoincrement" o "identity property" con la cual pueden automáticamente manipular algún atributo para generar llaves primarias de forma incremental. Pero es importante verificar:

- Cómo se manejan internamente
- Si se pueden reiniciar
- Si se permite especificar algún valor inicial

3.1.3. Orden de los campos o columnas

Por lo general, la secuencia es la siguiente:

- Columnas de longitud fija que no se actualizan frecuentemente.
- Aquellas que nunca se actualizan y que, por lo general, tendrán longitud variable.
- Las que se actualizan frecuentemente.

3.1.4. Integridad Referencial

- En la medida de lo posible, indicar qué columnas brindan o sirven de vínculo entre 2 tablas.
- El administrador de base de datos puede hacerse cargo de esto, pero es mejor que el SMBD lo haga.
- No se recomienda en ambientes de desarrollo.

3.2. Índices

Un índice es un atajo desde un campo llave hacia la localización real de los datos. Es el punto clave de la optimización de velocidad de toda base de datos.

Si se busca algún registro en base a un atributo que no tiene un índice, entonces, se realiza un escaneo de la tabla completa lo cual es demasiado costoso, por eso es recomendable usar índices en...

- Llaves primarias
- Llaves foráneas
- Índices de acceso
- Ordenamiento

No olvidar que el uso de un índice implica:

- *Overhead*, debido a la actualización de los mismos
- Espacio adicional en disco
- Procesos *batch* de muchos datos pueden volverse demasiado lentos
- Manipulación de archivos adicionales por el sistema operativo

3.3. Sistemas manejadores de bases de datos (SMBD) más populares




Logo	Nombre	URL	Productos
	Sybase	www.sybase.com	Adaptive Server
	Oracle	www.oracle.com	Oracle8, Oracle8i, Oracle8iEE, Oracle9i, Oracle 10g
	PostgreSQL	www.postgresql.org	PostgreSQL

Tabla 2.4. SMBD más populares.








Logo	Nombre	URL	Productos
	Microsoft	www.microsoft.com	Access, MS-SQL Server
	MySQL	www.mysql.com	MySQL
	Informix	www.informix.com	Illustra, Universal Server, Dynamic Server
	IBM	www.ibm.com	DB2
	Apache	http://db.apache.org/derby	Derby
	SQLite	http://www.sqlite.org	SQLite
	Firebird	http://firebird.sourceforge.net	Firebird

Tabla 2.4. SDBD más populares (Continuación).

3.4. Caso práctico

A continuación, se mostrará el modelo conceptual obtenido a partir de la ECU Generar reserva.

Especificación de Caso de Uso: Generar Reserva

1. Descripción

El caso de uso permite a la Recepcionista de un Hotel generar reserva de habitación(es).

2. Actor(es)

Recepcionista

3. Flujo de Eventos

3.1. Flujo Básico

1. El Caso de uso se inicia cuando la recepcionista selecciona la opción “Generar Reserva” en la interfaz del menú principal.
2. El sistema muestra la interfaz RESERVA con los siguientes datos:
 - Datos del cliente: Código, Nombre.
 - Datos de la Reserva: fecha de llegada, fecha de salida y cantidad de días a hospedarse.
 - Datos de las habitaciones: Número de habitación, Tipo, Costo por día, Nombre del huésped de la Habitación y una opción para Agregar.
 - Además, incluye una cuadrícula que contiene la lista de todas las habitaciones seleccionadas y las opciones: **Buscar cliente, Agregar cliente, consultar disponibilidad de habitación, eliminar habitación, Grabar y Salir.**
3. La recepcionista selecciona “Buscar cliente”.
4. El sistema **incluye el Caso de Uso Buscar Cliente.**
5. El sistema muestra los datos del cliente.
6. La recepcionista ingresa la fecha de llegada y la fecha de salida.
7. El sistema calcula la cantidad de días.
8. La recepcionista solicita “Consultar disponibilidad de habitación”.
9. El sistema **incluye el caso de uso consultar disponibilidad de habitación.**
10. El sistema muestra la habitación seleccionada.
11. La Recepcionista ingresa el nombre de la persona para la habitación seleccionada.
12. La Recepcionista selecciona: agregar
13. El sistema calcula el pago de la habitación, el subtotal, el monto total y lo agrega a la cuadrícula del detalle de la reserva.
14. Si la Recepcionista quiere seleccionar otra habitación, se repite del paso 7 al 12.
15. La recepcionista selecciona “Grabar”.
16. El sistema autogenera un número de reserva.
17. El sistema graba la reserva con su detalle y registra la(s) disponibilidad(es) de la(s) habitación(es) en estado “Reservado”.
18. El sistema muestra el número de reserva y el MSG “Reserva generada” con el Nro. 99999”.
19. La recepcionista cierra la interfaz RESERVA y regresa a la interfaz del menú principal del sistema y finaliza el caso de uso.

3.2. Subflujos

Ninguno.

3.3. Flujos Alternativos

1. Cliente no existe

En el paso 4, si el sistema detecta que el cliente no existe, muestra el MSG: “Cliente no existe” y ofrecerá la posibilidad de registrar al nuevo cliente.

2. Habitaciones no disponibles

En el paso 9, si el sistema detecta que no hay habitaciones disponibles muestra el MSG: “No hay habitaciones disponibles” y finaliza el caso de uso.

3. Eliminar Habitación de Cuadrícula

La recepcionista selecciona una habitación de la cuadrícula y selecciona eliminar, el sistema elimina de la cuadrícula la habitación selecciona y el caso de uso continúa.

4. Precondiciones

4.4. El Recepcionista está logeado en el sistema.

4.5. Lista de clientes disponibles.

4.6. Lista de habitaciones disponibles.

5. Poscondiciones

5.3. En el sistema quedará registrada la reserva con su detalle.

5.4. Las disponibilidades de las habitaciones seleccionadas se registraran en estado “Reservadas”.

6. Puntos de Extensión



En el paso 5, el sistema extiende al caso de uso **Mantener Clientes – subflujo “Agregar Cliente”**.

7. Prototipos

Interfaz RESERVA

Reserva

Datos del Cliente


Cliente  

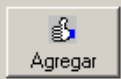
Nombre

Datos de la Reserva

Fecha de Llegada Fecha de Salida Cantidad Dias

Datos de las Habitaciones

Numero de Habitacion  Tipo Costo x día

Nombre del Huesped 



Detalle de la Reserva

Numero	Tipo	Huesped	Costo diario	Días	Moto a Pagar

Sub total

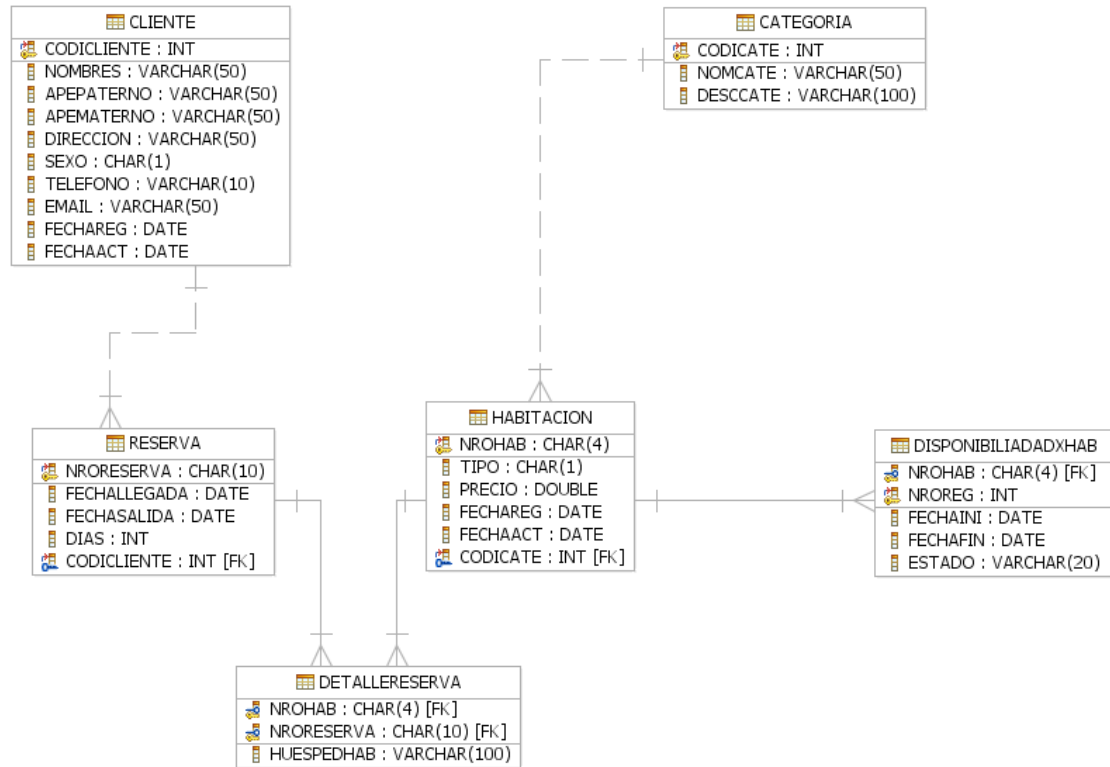
IGV

Monto Total

Solución:

A continuación se muestra el modelo físico del caso.

**ACTIVIDADES PROPUESTAS**

1. A partir de la Especificación del Caso de Uso "Reservar pistas de juego", de la sesión anterior, confeccione el Modelo Físico.

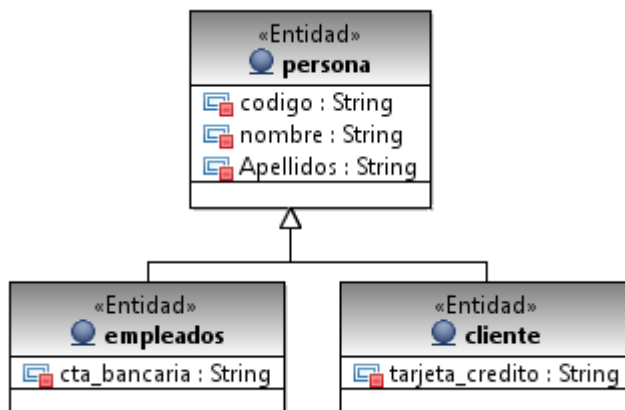
4.-Recomendación en el manejo de Herencia

Vamos a detallar las estrategias que existen para el manejo de la persistencia de herencia en una base de datos relacional. Existen tres estrategias que nos ayudaran para realizar un manejo adecuado de la persistencia.

- Una tabla por cada clase.
- Una tabla por cada clase concreta (no abstracta).
- Una tabla para la jerarquía de herencia y una tabla tipo.

Ejemplo

Vamos a trabajar con la siguiente jerarquía de herencia con tres clases: Persona, cliente y empleado. La clase persona es abstracta y las otras son clases concretas que extienden de la primera. De un cliente vamos a saber su número de tarjeta de crédito y de un empleado, vamos a saber la cuenta bancaria donde se depositan su sueldo.



Esta es la clase Persona (Java):

```

public abstract class Persona {

    private String codigo;
    private String nombre;
    private String apellidos;
    /* métodos get y set */
}
  
```

La clase Cliente solo añade un campo más:

```

public class Cliente extends Persona {
    private String tarjetaCredito;
    /* métodos get y set */
}
  
```

Y la clase Empleado:

```

public class Empleado extends Persona {
    private String ctaBancaria;
    /* métodos get y set */
}
  
```


Tenemos tres estrategias para persistir estas clases en una base de datos relacional:

- **Una tabla por cada clase.** Por tanto, tres tablas: personas, clientes, empleados.
- **Una tabla por cada clase concreta** (no abstracta). Dos tablas: clientes y empleados.
- **Una tabla para la jerarquía de herencia y una tabla tipo**, dos tablas: personas y tipo.

Una tabla por cada clase

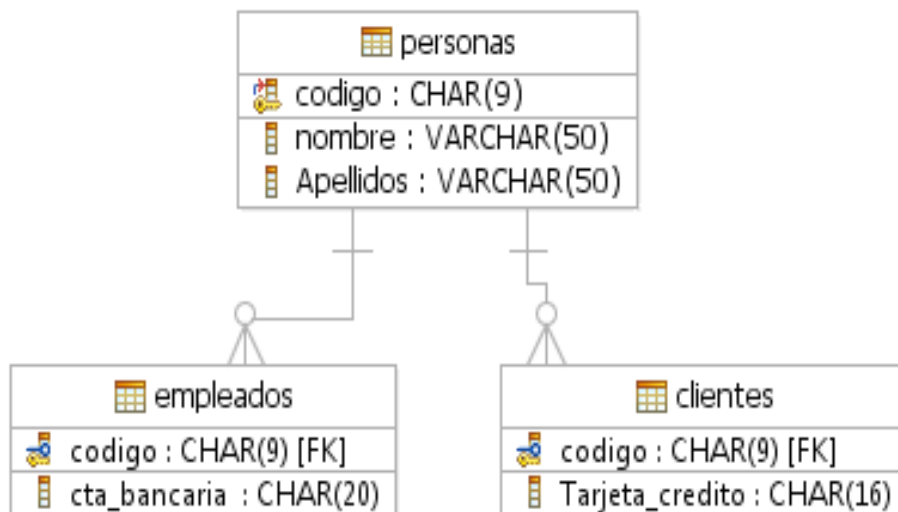
Cada tabla contendrá los campos de cada clase. Los únicos campos que se repetirán serán los relativos a las claves primarias. Estas serían las sentencias para crear las tablas:

```
CREATE TABLE personas (codigo CHAR(9) NOT NULL,
                        nombre VARCHAR(50),
                        apellidos VARCHAR(50),
                        PRIMARY KEY(codigo));
```

```
CREATE TABLE clientes (codigo CHAR(9) NOT NULL,
                        tarjeta_credito CHAR(16) NOT NULL,
                        PRIMARY KEY(código));
```

```
CREATE TABLE empleados (codigo CHAR(9) NOT NULL,
                         cta_bancaria CHAR(20) NOT NULL,
                         PRIMARY KEY(código));
```

```
ALTER TABLE clientes ADD FOREIGN KEY (codigo) REFERENCES personas (codigo);
ALTER TABLE empleados ADD FOREIGN KEY (codigo) REFERENCES personas (codigo);
```



Los beneficios de esta estrategia son los siguientes:

- En caso de que se añadan campos a la clase Persona, solo hay que añadir columnas a la tabla personas.
- Si en ciertas consultas solo necesitamos los datos comunes de clientes y empleados, sólo necesitamos un SELECT.

Los inconvenientes son los siguientes:

- Cuando se inserta un cliente o un empleado, hay que insertar en dos tablas.
- Cuando se consultan datos de un cliente o un empleado, hay que hacer un JOIN.
- La integridad referencial (claves foráneas/ajenas) influyen en el rendimiento de la base de datos.

Una tabla por cada clase concreta

Con esta estrategia las sentencias para crear las tablas serían las siguientes:

```
CREATE TABLE clientes (codigo CHAR(9) NOT NULL,
    nombre VARCHAR(50),
    apellidos VARCHAR(50),
    tarjeta_credito CHAR(16) NOT NULL,
    PRIMARY KEY(codigo));
```

```
CREATE TABLE empleados (codigo CHAR(9) NOT NULL,
    nombre VARCHAR(50),
    apellidos VARCHAR(50),
    cta_bancaria CHAR(20) NOT NULL,
    PRIMARY KEY(codigo));
```

clientes
codigo : CHAR(9)
nombre : VARCHAR(50)
apellidos : VARCHAR(50)
tarjeta_credito : CHAR(16)

empleados
codigo : CHAR(9)
nombre : VARCHAR(50)
apellidos : VARCHAR(50)
cta_bancaria : CHAR(20)

Los beneficios de esta estrategia son los siguientes:

- Cuando se inserta un cliente o un empleado solo se actualiza una tabla.
- Cuando se consultan datos de un cliente o un empleado solo se consulta una tabla.
- No hay restricciones referenciales.

Los inconvenientes son los siguientes:

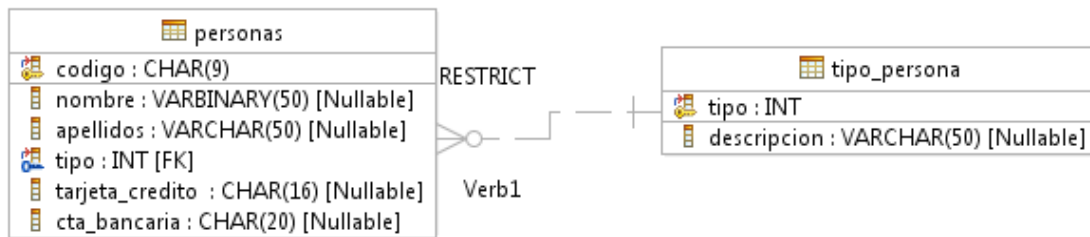
- En caso de que se añadan campos a la clase Persona, habría que modificar varias tablas.
- Si en alguna consulta necesitamos datos de clientes y empleados, tendríamos que hacer dos SELECT y unir los resultados. O hacer un UNIÓN.

Una tabla para la jerarquía de herencia y una tabla tipo

Con esta estrategia, tendríamos una tabla persona y una tipo para manejar restricción referencial:

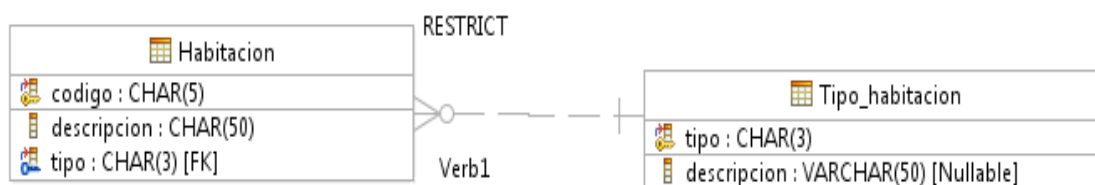
```
CREATE TABLE personas (codigo CHAR(9) NOT NULL,
    nombre VARCHAR(50),
    apellidos VARCHAR(50),
    tipo INT NOT NULL,
    tarjeta_credito CHAR(16),
    cta_bancaria CHAR(20),
    PRIMARY KEY(codigo));
```

```
CREATE TABLE tipo_persona(tipo INT NOT NULL,
                           nombre VARCHAR(50)
                           PRIMARY KEY (tipo));
```



Se añade una columna "tipo" a la tabla persona y se crea la tabla tipo_persona. Esa columna permitirá saber si el registro se refiere a un cliente o a un empleado relacionado con la tabla tipo_persona. También se ha eliminado las restricciones "NOT NULL" tanto de la columna tarjeta_credito como de la columna cta_bancaria, debido a que ahora podrán ser NULL. Si una persona es cliente, tendrá cta_bancaria a NULL, y viceversa con el campo tarjeta_credito si es un empleado.

Otro ejemplo



Los beneficios de esta estrategia son los siguientes:

- Cuando se inserta un cliente o un empleado solo se actualiza una tabla.
- Cuando se consultan datos de un cliente o un empleado solo se consulta una tabla.
- La restricción referencial es controlado por la tabla tipo_persona.
- En caso de que se añadan campos a la clase Persona solo habría que cambiar una tabla.

Los inconvenientes son los siguientes:

- Deberemos utilizar características avanzadas de la base de datos si queremos mantener las restricciones NOT NULL de ciertas columnas.

Conclusiones

En conclusión, lo más recomendable es utilizar la segunda o la tercera estrategia. Elegir, entre ellas, dependerá de las consultas que se han de hacer a la base de datos. Si, por ejemplo, en ningún momento, hacemos una consulta que involucre tanto a clientes y empleados, quizá lo mejor será tener una tabla por cada clase concreta. Si, por el contrario, es común hacer listados que involucren a ambas clases, entonces, lo mejor será utilizar la tabla de jerarquía y la tabla tipo.

5.-Auditoría en Bases de Datos

La información que se trata y almacena en una base de datos cada día cobra más y más importancia, así como se han debido implementar políticas y ver más detalladamente el manejo de la información, la transparencia, fiabilidad y seguridad en una base de datos. Un punto importante en donde hay que detenerse y aplicar políticas de control, seguimiento de cambios de datos que alteran la base de datos, así respaldar y encontrar el hilo de la trayectoria desde que se conecta un usuario a una base de datos determinada, los cambios que realice y el término de su sesión como tal usuario. Tratando así en resguardar el 100% de la integridad de la información tratada en las bases de datos, es aquí en donde entra en marcha la auditoría de base de datos.

Auditoría de las bases de datos: Consiste en el control de acceso, de actualización, de integridad y calidad de los datos.

¿Qué es la Auditoría de BD?

Es el proceso que permite medir, asegurar, demostrar, monitorear y registrar los accesos a la información almacenada en la base de datos, incluyendo la capacidad de determinar:

- Quién accede a los datos.
- Cuándo se accedió a los datos
- Desde qué tipo de dispositivo/aplicación
- Desde qué ubicación en la Red
- Cuál fue la sentencia SQL ejecutada
- Cuál fue el efecto del acceso a la base de datos

Objetivos Generales de la Auditoría de BD

Disponer de mecanismos que permitan tener trazas de auditoría completas y automáticas relacionadas con el acceso a las bases de datos incluyendo la capacidad de generar alertas con el objetivo de...:

- Mitigar los riesgos asociados con el manejo inadecuado de los datos
- Apoyar el cumplimiento regulatorio.
- Satisfacer los requerimientos de los auditores
- Evitar acciones criminales
- Evitar multas por incumplimiento

La importancia de la auditoría del entorno de bases de datos radica en que es el punto de partida para poder realizar la auditoría de las aplicaciones que utiliza esta tecnología.

La Auditoría de BD es importante por las siguientes razones:

- Toda la información financiera de la organización reside en bases de datos y deben existir controles relacionados con el acceso a las mismas.
- Se debe poder demostrar la integridad de la información almacenada en las bases de datos.
- Las organizaciones deben mitigar los riesgos asociados a la pérdida de datos y a la fuga de información.

- La información confidencial de los clientes son responsabilidad de las organizaciones.
- Los datos convertidos en información a través de bases de datos y procesos de negocios representan el negocio.
- Las organizaciones deben tomar medidas mucho más allá de asegurar sus datos. Deben monitorearse perfectamente a fin de conocer quién o qué les hizo exactamente: qué, cuándo y cómo.

Para poder realizar una auditoría de la base de datos, tenemos 4 técnicas que nos ayudarán a solucionar nuestro trabajo, estas técnicas se pueden combinar para tener una solución adecuada

1. Auditoría nativa de las bases de datos

Todos los motores de datos actuales (SQL Server, Oracle, Informix, Sybase, etc.) tienen herramientas para realizar la auditoría de la base de datos o la creación de logs de acceso de usuarios a la base de datos. Esta técnica es muy buena, pero ocupa mucho espacio de datos.

2. Añadir campos a una tabla

El modo más simple de auditar una tabla es registrando cada cambio fila por fila; sin embargo, también es la menos recomendable. Se basa en añadir uno o dos campos testigos que registren la fecha en que se ejecuten cambios y el usuario que los efectúe, sin embargo, cambios en cada columna no son registrados ni auditados, solo se registra el último usuario que haya hecho un cambio. Por ejemplo, si Pilar hace un cambio en la columna "Precio" y luego Juan cambia el valor de la columna "Cantidad" solo quedará registrado que Juan hizo un cambio. Este tipo de auditoría es algo ingenuo, pero que puede servir en algunos casos.

3. Triggers y Tablas Espejo

Todos los motores tienen en los triggers (desencadenadores o disparadores) una ayuda para llevar a cabo registros de auditoría. Esta forma tiene el siguiente procedimiento:

1. Crear una tabla espejo con los mismos campos que la tabla auditada, pero con campos adicionales, como el usuario que haya hecho el cambio, la fecha y la operación realizada que puede ser fila agregada, fila modificada o fila eliminada.
2. Añadir triggers INSERT, UPDATE y DELETE a la tabla origen, de modo que al efectuarse cualquiera de las operaciones indicadas, grabe el mismo registro en la tabla de auditoría, incluyendo los campos de auditoría (usuario, fecha, tipo de operación).

Esta forma de auditoría, si bien es mejor que la anterior, ya que registra cada cambio realizado y almacena un historial completo para cada fila, impone una sobrecarga de operaciones por cada fila, debido a que se vuelve a registrar la misma fila con datos adicionales en caso sea insertada, actualizada o eliminada. Y a la vez aumenta el espacio a ocupar.

4. Creación de tablas de movimiento

Es un modo simple de auditar una tabla, se lleva a cabo registrando cada cambio fila por fila, en una tabla de operación o movimiento esta tabla permitirá reconstruir la información.

Manejar datos de auditoría

El almacenar datos de auditoría ocupa una gran cantidad de espacio en disco duro, y manejar esa información se hace muy complicado, especialmente, si se realiza con tablas espejo. Hay que saber manejar esa información, y auditar las tablas necesarias. Una forma de manejarla es almacenando la información de auditoría en otra base de datos. Se crea una base de datos, y preferiblemente almacenarla en otra partición u otro disco duro si la actividad es intensa. Esto aumenta la complejidad de la auditoría, ya que hay que proporcionar permisos a los usuarios para grabar información en la segunda base de datos. Se podría crear una vista en la base de datos auditada que muestre los datos en la base de datos de auditoría para darle un nivel de abstracción.

Archivamiento






Generar información de auditoría es un problema para el espacio, y generar mucha data es peor aún. Hay que diseñar un plan de archivamiento de esos datos, no tenerlo es lo peor que se puede hacer.

Un plan de archivamiento podría ser simplemente separar la base de datos de auditoría y crear una nueva. La desventaja sería el extraer información de la base de datos separada. Otra sería programar un script que extraiga o inserte la información, según sea el caso, y grabarla en cintas o CD. En cualquier caso, la recuperación y volcado se hace manual y complica la tarea.

Conclusión

Generar data de auditoría en una base de datos es una ventaja y, a la vez, un problema, ya que acarrea costos en espacio de almacenamiento y tiempo en manejarla. Pero aún así es necesaria en ciertas actividades, especialmente, la financiera. Como recomendación, la auditoría debería realizarse a ciertas tablas de una base de datos y no a todas, ya que complicaría aún más el proceso de administración.

Resumen

-  El modelo conceptual es un modelo de datos de alto nivel en el que disponen de conceptos muy cercanos al modo en que la mayoría de los usuarios percibe los datos.
-  En el modelo conceptual, las clases se obtienen a partir de los objetos de información que fluyen entre las actividades. El modelado de los casos de uso del sistema y el modelado conceptual se realizan en paralelo, esto es crucial para obtener casos de uso correctos, puesto que es necesario entender bien el dominio para poder escribir casos de uso que sean realmente útiles.
-  El modelo conceptual orientado a objetos beneficiará a dos equipos de trabajo :
 - **Equipo de Desarrolladores**
En esta etapa del desarrollo, es conveniente detenerse en la identificación de los conceptos y no tanto en las relaciones entre ellos. Este modelo incluirá los conceptos y sus relaciones y se describirá mediante un diagrama de clases UML, en el que los conceptos se representan mediante clases (del dominio).
 - **Equipo de Base de Datos**
En esta etapa, luego del modelo conceptual, se obtiene el proceso del modelo lógico al diseño físico donde se podrá identificar las tablas relacionales del proyecto de Base de Datos como componente RUP
-  Un modelo lógico de datos es un modelo que no es específico de una base de datos que describe aspectos relacionados con las necesidades de una organización para recopilar datos y las relaciones entre estos aspectos. Es el refinamiento del modelo conceptual; no es necesario especificar las llaves primarias y foráneas de las entidades, pues es trabajo que se recomienda realizar en el modelo físico.
-  El modelo físico es un modelo de datos de bajo nivel. Proporcionan conceptos que describen los detalles de cómo se almacenan los datos en el ordenador.

DISEÑO ORIENTADO A OBJETOS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al finalizar la tercera unidad, el alumno diseña la arquitectura del *software* identificando las capas, subsistemas y componentes de la aplicación. Los artefactos serán creados utilizando la herramienta *CASE IBM Rational Software Architect (RSA)*.

Tema 1: Diseño orientado a objetos

- 1.1. Flujo de trabajo del DOO
- 1.2. Artefactos del diseño
- 1.3. Modelo de análisis Vs. Modelo de diseño

Tema 2: Arquitectura de software

- 2.1. Por qué es importante la arquitectura
- 2.2. Evolución de la arquitectura
- 2.3. Fases de la arquitectura
- 2.4. Estilos arquitectónicos

Tema 3: Diseño de casos de uso

- 3.1. Patrones de diseño
- 3.2. Extensiones de UML para aplicaciones web (WAE)
- 3.3. Realización de diseño de casos de uso con patrón arquitectónico MVC
- 3.4. Realización de diseño de casos de uso con patrón arquitectónico MVC y patrón de diseño DAO

ACTIVIDADES PROPUESTAS

1. Los alumnos desarrollan la realización de diseño de un caso de uso a partir de su especificación aplicando el patrón arquitectónico MVC.
2. Los alumnos desarrollan la realización de diseño de un caso de uso a partir de su especificación aplicando el patrón arquitectónico MVC y DAO.

1. DISEÑO ORIENTADO A OBJETOS

1.1. Flujo de trabajo del DOO

El objetivo del diseño es **entender la solución** refinando el modelo de análisis con la intención de desarrollar un modelo de diseño que permita una transición sin problemas a la fase de construcción. En el diseño, nos adaptamos al entorno de implementación y despliegue.

En la siguiente figura se resalta las actividades que se llevan a cabo en el DOO.

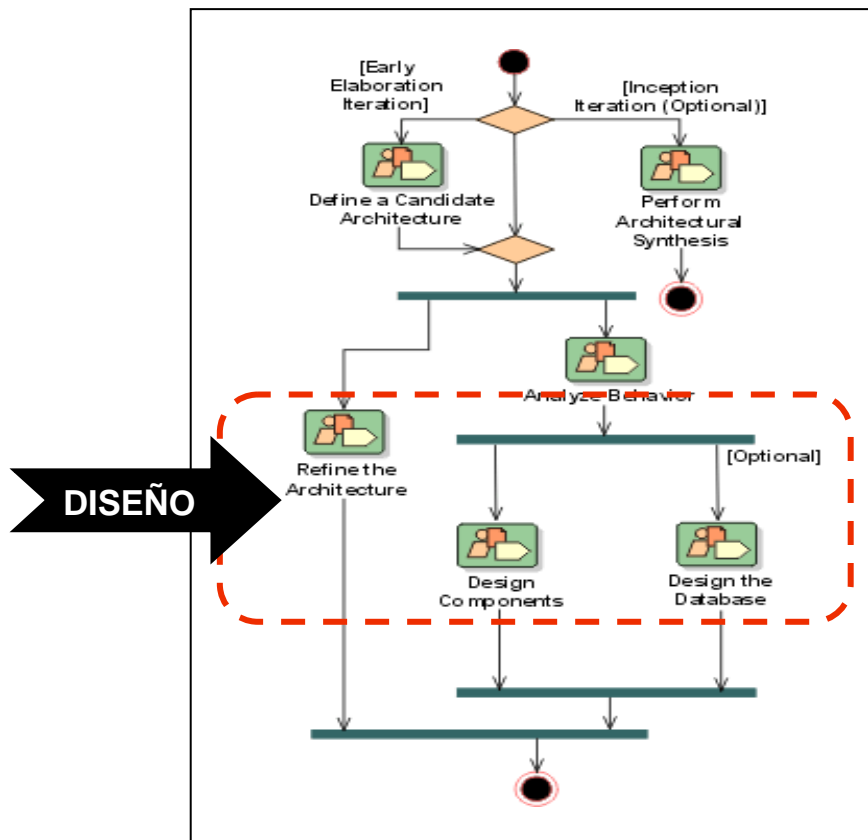


Figura 3.1. Flujo de trabajo del DOO.

1.2. Artefactos del Diseño

Son muchos los artefactos que se elaboran en el flujo de trabajo del DOO. En el curso, consideraremos los siguientes artefactos:

- Modelo de Diseño.
- Elementos del modelo de diseño: capas, subsistemas con clases de diseño (como servlets, beans y otras) e interfaces, librerías con clases utilitarias, y realizaciones de diseño de casos de uso.
- Diagramas de realizaciones de diseño de casos de uso: diagrama de clases y diagramas de secuencia.
- Diagrama de componentes, cuyos elementos son los componentes de la aplicación.
- Modelo de Despliegue.
- Diagrama de despliegue, cuyos elementos son artefactos y nodos.

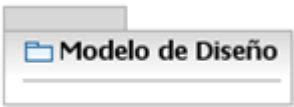

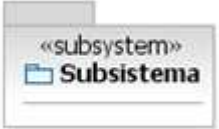


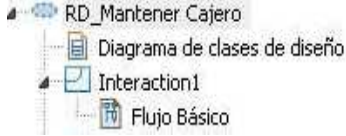
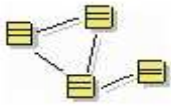
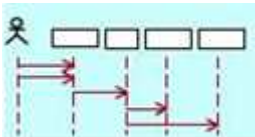
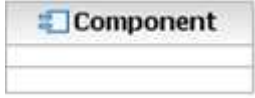
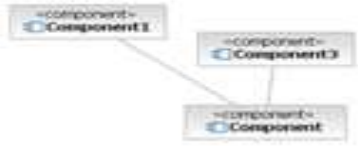

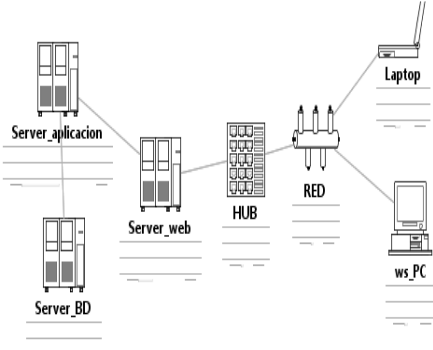

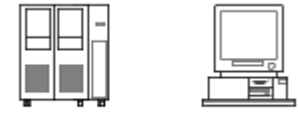
Artefacto	Descripción
 Modelo de Diseño	Representa la vista interna del sistema. Conjunto formado por las clases de diseño y las realizaciones de casos de uso. El modelo de diseño se convertirá en la materia prima que nuestra disciplina de implementación transformará en código ejecutable.
 Capa	Representan un medio para organizar los artefactos del modelo de diseño. Dependiendo del estilo arquitectónico, una capa agrupa un conjunto de subsistemas junto con sus clases de diseño.
 Subsistema	Un subsistema contiene las clases de diseño de un grupo de casos de uso. Tiene correspondencia directa con los paquetes de análisis.
 Librería	Una librería contiene clases utilitarias, adicionales a las del API del lenguaje de programación, implementadas por el equipo de desarrollo.
 Clases de diseño	Son abstracciones de clase directamente utilizadas en la implementación, es decir, estas clases junto con sus atributos y operaciones se mapean directamente en el lenguaje de programación.
 Realización de Diseño de Caso de Uso	Describe cómo un caso de uso se lleva a cabo en términos de clases de diseño y sus objetos. Hay una correspondencia directa entre Realización de Diseño de Casos de Uso y Realización de Análisis de Casos de Uso.
 Diagrama de Clases	El diagrama de clases describe la estructura de un caso de uso. Contiene las clases que participan en el caso de uso, aunque algunas de ellas puedan participar en varios.
 Diagramas de Secuencia	Muestra una secuencia detallada de interacción entre los objetos de diseño. Visualizan el intercambio de mensajes entre objetos. Se crea un diagrama de secuencia por cada flujo de trabajo del caso de uso: flujo básico, subflujos y flujos alternativos.

Tabla 3.1. Artefactos del Diseño.

Artefacto	Descripción
 <p>Componente</p>	<p>Un componente representa una pieza del software reusable. Por ejemplo, si se está desarrollando una aplicación web estas piezas pueden ser recursos estáticos (páginas HTML) y recursos dinámicos (JSP y servlets) representados como componentes.</p>
 <p>Diagrama de componentes</p>	<p>Un diagrama de componentes muestra la estructura de un sistema <i>software</i>, el cual describe los componentes <i>software</i>, sus interfaces y sus dependencias.</p>
 <p>Modelo de Despliegue</p>	<p>Describe la distribución física del sistema en términos de cómo las funcionalidades se distribuyen entre los nodos de computación sobre los que se va a instalar el sistema.</p>
 <p>Diagrama de Despliegue</p>	<p>Un diagrama de despliegue se puede utilizar para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes.</p>
 <p>Artefacto</p>	<p>Los artefactos son elementos que representan las entidades físicas de un sistema <i>software</i>. Los artefactos representan unidades de implementación física como archivos ejecutables, librerías, componentes de <i>software</i>, documentos, y bases de datos.</p>
<p>T a b l a 3</p>  <p>Nodo</p>	<p>Representa un recurso de computación. Los nodos tienen relaciones entre ellos que representan los medios de comunicación que hay entre ellos como una Intranet o Internet. La funcionalidad de un nodo viene representada por los componentes que se ejecutan en él.</p>

1. Artefactos del Diseño. (Continuación)

1.3. Modelo de análisis Vs. Modelo de diseño

El siguiente cuadro muestra una comparación entre el Modelo de análisis y el Modelo de diseño:

<i>Modelo de Análisis</i>	<i>Modelo de Diseño</i>
Es un modelo conceptual y genérico, es una abstracción del sistema.	Es un modelo físico y concreto, es un plano de la implementación.
Es menos formal.	Es más formal.
Es un bosquejo del diseño del sistema.	Es una realización del diseño del sistema.
Puede no mantenerse durante todo el ciclo de vida del <i>software</i> .	Debe ser mantenido durante todo el ciclo de vida del <i>software</i> .
Define una estructura para modelar el sistema.	Da forma al sistema.

Cuadro 3.1. Comparación del MA Vs. MD.

Resumen

📖 El objetivo del diseño es entender la solución refinando el modelo de análisis con la intención de desarrollar un modelo de diseño que permita una transición sin problemas a la fase de construcción. En el diseño, nos adaptamos al entorno de implementación y despliegue.

📖 Son muchos los artefactos que se elaboran en el flujo de trabajo del DOO. En el curso, consideraremos los siguientes artefactos:

- Modelo de Diseño.
- Elementos del modelo de diseño: capas, subsistemas con clases de diseño (como servlets, beans y otras) e interfaces, librerías con clases utilitarias, y realizaciones de diseño de casos de uso.
- Diagramas de realizaciones de diseño de casos de uso: diagrama de clases y diagramas de secuencia.
- Diagrama de componentes, cuyos elementos son los componentes de la aplicación.
- Modelo de Despliegue.
- Diagrama de despliegue, cuyos elementos son nodos.

📖 Si desea saber más acerca de estos temas, puede consultar los siguientes libros.

🔗 “OBJECT-ORIENTED ANALYSIS AND DESIGN WITH APPLICATIONS.” de Grady Booch, Jim Conallen y otros, 3era edición.

En el capítulo 12, encontrará la solución de una aplicación web, explicando cómo se construyen los modelos de las disciplinas de RUP para alcanzar los objetivos de cada fase.

2. ARQUITECTURA DEL SOFTWARE

La Arquitectura del Software AS es la parte de la ingeniería del software que se ocupa de la descripción y el tratamiento de un sistema como un conjunto de componentes, que facilite su organización en los diferentes subsistemas que lo forman. Esto es útil, entre otros aspectos, para poder asignarlos a equipos de trabajo y que puedan llevar a cabo el desarrollo del sistema de una manera organizada y eficiente. Este campo surge ante la necesidad de describir sistemas *software* complejas, descomponerlos en un conjunto de componentes y ver qué relaciones existen entre ellos. Por lo tanto, la AS aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño.

Aunque intuitivamente el concepto de AS es bastante claro, no se ha dado hasta el momento una definición que satisfaga por completo a todos aquellos que trabajan en este campo. Quizás, para obtener una visión de conjunto, lo más adecuado sea considerar varias definiciones a la vez. A continuación, se mencionan varias de ellas ordenadas cronológicamente:

- Posiblemente, la primera definición fue la dada en 1993 por David Garlan y Mary Shaw en la que exponen objetivos y aspectos a tener en cuenta en la AS:

“Más allá de los algoritmos y estructuras de datos de la computación, el diseño y especificación de la estructura global del sistema emerge como un nuevo tipo de problema. Los detalles estructurales incluyen: la organización general y la estructura de control global; los protocolos de comunicación, sincronización y acceso a datos; la asignación de funcionalidad a los elementos de diseño; la distribución física; la composición de los elementos de diseño; su escalabilidad y los aspectos de rendimiento; y la selección entre alternativas de diseño”.

- Se puede considerar una segunda definición, quizás la más breve y sencilla, y a la vez la más aceptada, dada por David Garlan y Dewayne Perry en 1995:

“La arquitectura del software está compuesta por la estructura de los componentes de un programa o sistema, sus interrelaciones, y los principios y reglas que gobiernan su diseño y evolución a lo largo del tiempo”.

- A continuación, se presenta la definición dada por Clements en 1996 que puede considerarse como una definición general, amplia y flexible, aunque por ello pueda parecer incompleta y algo ambigua. Sin embargo, es válida tanto para aquellos que desean una visión descriptiva de la AS (punto de vista representado por Garlan) como para los que utilizan una visión de proceso (representada por Perry):

“La arquitectura del software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se percibe desde el resto del sistema, y las formas en que los componentes interactúan y se coordinan para alcanzar el objetivo del sistema. La vista arquitectónica es una vista abstracta de un sistema, aportando el más alto nivel de comprensión y la supresión del detalle inherente a la mayor parte de abstracciones”.

- Otra definición que ha sido utilizada por la comunidad de ingeniería de software es la de RUP, citada por Phillipe Kruchten en 1995, que define lo siguiente:

“La arquitectura de software abarca un conjunto de decisiones significativas sobre la organización del sistema de software. Estas decisiones abarcan: la selección de los elementos estructurales y sus interfaces, con los que se compone el sistema, junto con su comportamiento tal como se especifica en las colaboraciones entre esos elementos, la composición de esos elementos estructurales y de comportamiento en subsistemas progresivamente más grandes, y el estilo arquitectónico que guía esta organización”.

- En el año 2000 se acordó definir oficialmente la AS según figura en el documento IEEE Std 1471-2000:

“La arquitectura del software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el entorno, y los principios que dirigen su diseño y evolución”.

- En el número de marzo/abril de 2006 de IEEE Software, en el artículo de Kruchten, Obbink y Stafford figura la siguiente definición más elaborada:

“La arquitectura del software captura y preserva las intenciones de los diseñadores sobre la estructura y el comportamiento de los sistemas, proporcionando un mecanismo de defensa contra el deterioro de los sistemas antiguos. Esta es la clave para lograr el control intelectual sobre la creciente complejidad de los sistemas...”

... Paradójicamente... aún no hay un consenso sobre cuál es la respuesta a la pregunta de qué es la arquitectura del software y que sea ampliamente aceptada... Llegar a un acuerdo sobre la definición más adecuada para esta disciplina fue uno de los objetivos de definir el estándar IEEE. Sin embargo, esta falta de acuerdo no ha sido impedimento para que la arquitectura del software progrese”.

A partir de todas las definiciones que se han dado sobre AS, se ha llegado a una especie de consenso por el que esta disciplina se refiere a la **estructura de un sistema a grandes rasgos, formada por componentes y relaciones entre ellos**. Estas cuestiones se tienen en cuenta durante el diseño, puesto que la AS se refiere al diseño del software que se realiza en fases tempranas del desarrollo software y a alto nivel de abstracción.



2.1. Por qué es importante la arquitectura

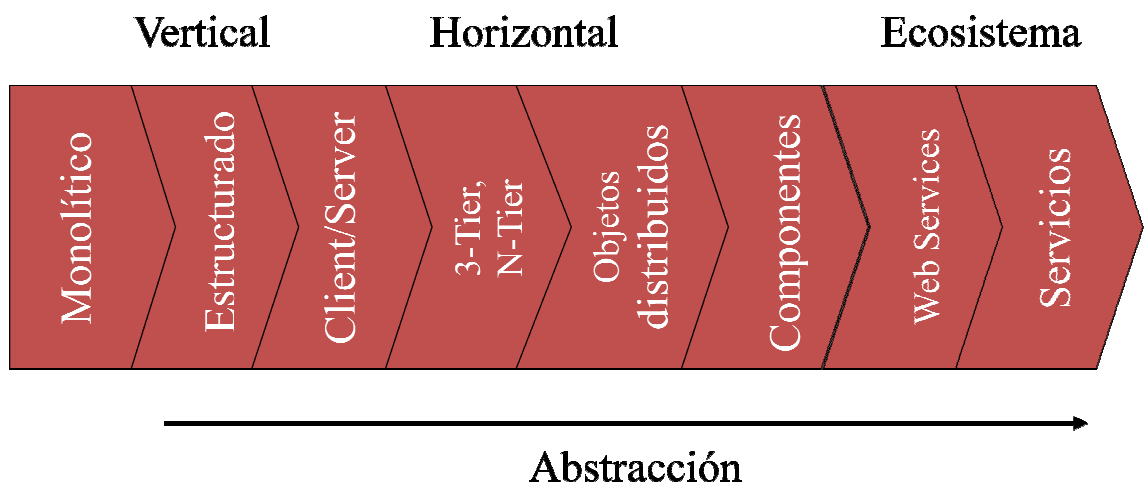
Para empezar, de una u otra forma, muchos stakeholders están interesados en la arquitectura:

- El analista de sistemas, que lo utiliza para organizar y articular los requisitos y comprender las limitaciones tecnológicas y riesgos.
- Los usuarios finales o clientes, que lo utilizan para visualizar en un alto nivel lo que están comprando.
- El gerente de proyecto de software, que lo utiliza para organizar el equipo y el plan de desarrollo.
- Los diseñadores, que la utilizan para comprender los principios subyacentes y localizar los límites de su propio diseño.
- Otras organizaciones de desarrollo (si el sistema es abierto), que lo utilizan para comprender cómo interactuar con él.
- Arquitectos, que la usan para razonar acerca de la evolución o la reutilización.

Al respecto, Bass y sus colegas (2003), en un libro dedicado a la arquitectura de software, identifican tres razones clave por las cuales la arquitectura del software es importante:

- Las representaciones de la arquitectura del software **permiten la comunicación entre los diferentes stakeholders.**
- La arquitectura resalta las **decisiones iniciales relacionadas con el diseño** que tendrán un impacto en todo el proceso de desarrollo posterior.
- La arquitectura encarna un modelo relativamente pequeño, intelectualmente tratable, de la forma en que un sistema se estructura y sus componentes se entienden entre sí; este modelo es **transferible a través de sistemas**; en particular, se puede aplicar a otros sistemas que exhiben requisitos parecidos **y puede promover reutilización en gran escala.**

2.2. Evolución de la arquitectura



2.3. Fases de la arquitectura

Un ingeniero de software define la arquitectura del sistema en tres etapas, tal como se muestra en la figura 3.2.

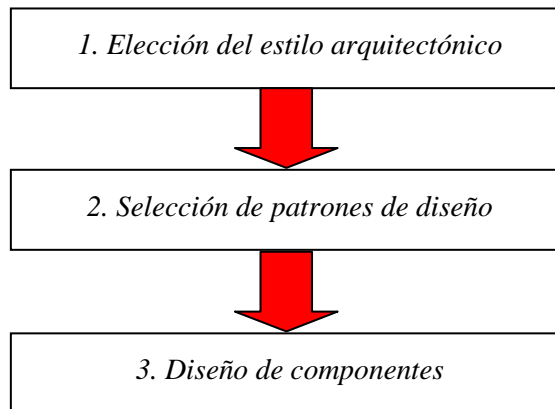


Figura 3.2. Fases de la arquitectura del sistema

2.2.1. Elección de estilos arquitectónicos

Una vez que el equipo de desarrollo define los requisitos funcionales y no funcionales del sistema que habrá de construirse, **podrá elegirse un estilo arquitectónico o la combinación de estilos que mejor combinen con los requisitos del sistema.**

Un estilo arquitectónico es un patrón de organización de un sistema. El objetivo es establecer una estructura para todos los componentes del sistema.

Existe un número relativamente pequeño de estilos arquitectónicos. La primera taxonomía la propusieron Shaw y Garlan, en 1993, considerando los siguientes tipos:

- Tuberías-filtros.
- Organización de abstracción de datos y orientación a objetos.
- Invocación implícita, basada en eventos.
- Sistemas en capas.
- Repositorios.
- Procesos distribuidos, ya sea en función de la topología (anillo, estrella, etc.) o de los protocolos entre procesos. Una forma particular de proceso distribuido es, por ejemplo, la arquitectura cliente-servidor.
- Organizaciones programa principal / subrutina.
- Arquitecturas software específicas de dominio.
- Sistemas de transición de estados.
- Sistemas de procesos de control.
- Estilos heterogéneos.

Más adelante, el grupo de Buschmann en 1996, quienes denominaron a los estilos como patrones arquitectónicos, los enumeraron agrupándolos en los siguientes sistemas:

- Sistemas estructurales.
- Sistemas distribuidos.
- Sistemas interactivos
- Sistemas adaptables.

Posteriormente, Bass, Clements y Kazman en el año 2003, definieron un conjunto de clases de estilo en los siguientes grupos:

- Flujo de datos.
- Llamada y retorno.
- Componentes independientes centrados en datos.
- Máquina virtual.

En los últimos años, la relación de estilos se ha hecho más detallada y exhaustiva. Considerando solamente los estilos que contemplan alguna forma de distribución o topología de red, tal como la taxonomía propuesta por Roy Fielding en el año 2000:

- Estilos de flujo de datos.
- Estilos de replicación.
- Estilos jerárquicos.
- Estilos de código móvil.
- Estilo peer-to-peer.
- Estilos de transferencia de estado de representación (REST).

2.2.2. Selección de patrones de diseño

Un patrón es una solución de diseño de software a un problema, aceptada como correcta, a la que se ha dado un nombre y que puede ser aplicada en otros contextos.

Se debe **seleccionar uno o más patrones de diseño para aplicarlo en el diseño de los componentes**. Actualmente, existen varias categorías de patrones, pero en el curso describiremos los patrones GOF y el catálogo de patrones de J2EE.

2.2.3. Diseño de componentes

Consiste en el diseño de los módulos del sistema utilizando una notación gráfica (como UML) para expresar lo que se va a implementar. **El diseño de componentes tiene como artefacto principal la colaboración de casos de uso**, la cual contiene un diagrama de clases y diagramas de interacción para representar la estructura y comportamiento del caso de uso respectivamente.

2.4. Estilos arquitectónicos

Los estilos que se describirán a continuación son los más representativos y vigentes, los que contemplan alguna forma de distribución de componentes o topología de red.

2.4.1. Estilos de flujos de datos

Es apropiada para sistemas que implementan transformaciones de datos frecuentemente. Ejemplares de la misma serían las arquitecturas de tubería-filtros y las de proceso secuencial en lote.

a) Tuberías y filtros

Este estilo consiste en una tubería (*pipeline*) que conecta componentes computacionales (filtros) a través de conectores (*pipes*), de modo que los procesos se ejecutan a la manera de un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas.

El sistema tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes.

b) Proceso secuencial en lote

En el estilo secuencial por lotes (*batch sequential*) los componentes son programas independientes; el supuesto es que cada paso se ejecuta hasta completarse antes que se inicie el paso siguiente.

2.4.2. Estilos centrados en datos

Esta familia de estilos es apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

a) Arquitecturas de Pizarra o Repositorio

En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. En base a esta distinción, se han definido dos subcategorías principales del estilo:

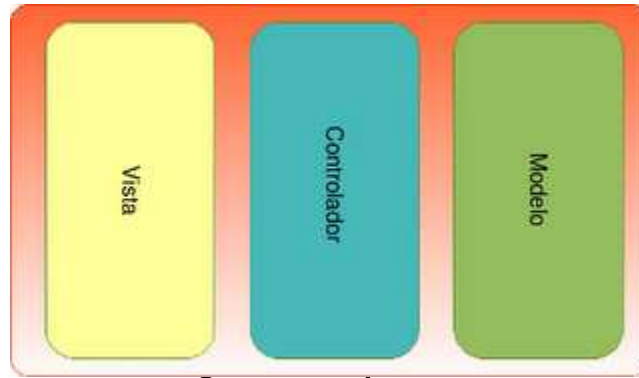
- Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional (implícitamente no cliente-servidor).
- Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control.

2.4.3. Estilos de llamada y retorno

Son los estilos más generalizados en sistemas en gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

a) Model-View-Controller (MVC)

Algunos lo definen como patrón de diseño, pero otros autores precisan que debe ser tratado como estilo arquitectónico, debido a que está referido a la estructura global del *software*.



El estilo o patrón arquitectónico MVC separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres capas:

- **Modelo.** El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- **Vista.** Maneja la visualización de la información.
- **Controlador.** Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

Entre las ventajas de este estilo se encuentran:

- **Soporte de vistas múltiples.** Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Por ejemplo, múltiples páginas de una aplicación de Web pueden utilizar el mismo modelo de objetos, mostrado de maneras diferentes.
- **Adaptación al cambio.** Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares o PDAs. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación, generalmente, no afecta al modelo. Este patrón sentó las bases para especializaciones posteriores, tales como Page Controller y Front Controller.

b) Arquitectura en capas

Los sistemas o arquitecturas en capas constituyen uno de los estilos que aparecen con mayor frecuencia en las estructuras de sistemas. Consiste en una organización jerárquica de capas, de modo tal que cada una proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En algunos ejemplares, las capas internas están ocultas a todas las demás, menos para las

capas externas adyacentes, y excepto para funciones puntuales de exportación; en estos sistemas, los componentes implementan máquinas virtuales en alguna de las capas de la jerarquía. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas.

Casos representativos de este estilo son muchos de los protocolos de comunicación en capas. En ellos, cada capa proporciona un sustrato para la comunicación a algún nivel de abstracción, y los niveles más bajos suelen estar asociados con conexiones de hardware.

El ejemplo más característico es el modelo OSI con siete niveles: nivel físico, vínculo de datos, red, transporte, sesión, presentación y aplicación. El estilo también se encuentra en forma más o menos pura en arquitecturas de bases de datos y sistemas operativos, así como en las especificaciones relacionadas con XML.

El número mínimo de capas es obviamente dos, y en ese umbral la literatura arquitectónica sitúa a veces al sub-estilo cliente-servidor como el modelo arquetípico del estilo de capas y el que se encuentra con mayor frecuencia en las aplicaciones en red. Un componente servidor, que ofrece ciertos servicios, escucha que algún otro componente requiera uno; un componente cliente solicita ese servicio al servidor a través de un conector. El servidor ejecuta el requerimiento (o lo rechaza) y devuelve una respuesta.

c) Arquitectura orientada a objetos

Nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos.

Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.

d) Arquitectura basada en componentes

En este estilo arquitectónico los componentes son las unidades de modelado, diseño e implementación. Las interfaces están separadas de las implementaciones, y las interfaces y sus interacciones son el centro de incumbencias en el diseño arquitectónico.

2.4.4. Estilos de código móvil

Esta familia de estilos enfatiza la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando. Fuera de las máquinas virtuales y los intérpretes, los otros miembros del conjunto han sido rara vez estudiados desde el punto de vista estilístico.

a) Arquitectura de máquinas virtuales

La arquitectura de máquinas virtuales se ha llamado también intérpretes basados en tablas. De hecho, todo intérprete involucra una máquina virtual implementada en software. Se puede decir que un intérprete incluye un pseudo-programa a interpretar y una máquina de interpretación. El pseudo-programa, a su vez, incluye el programa mismo y el análogo que hace el intérprete de su estado de ejecución (o registro de activación). La máquina de interpretación incluye tanto la definición del intérprete como el estado actual de su ejecución. De este modo, un intérprete posee, por lo general, cuatro componentes:

- Una máquina de interpretación que lleva a cabo la tarea.
- Una memoria que contiene el pseudo-código a interpretar.
- Una representación del estado de control de la máquina de interpretación.
- Una representación del estado actual del programa que se simula.

Las máquinas virtuales no son una invención reciente ligada a Java, sino que existen desde la década de 1950. En esa década, los precursores de los ingenieros de software sugirieron una máquina virtual basada en un lenguaje de máquina universal de bytcodes (un ejemplo fue UNCOL), de manera que las aplicaciones podían escribirse en las capas más altas y ejecutarse donde fuere sin tener que recompilarse, siempre que hubiera una máquina virtual entre el programa por un lado y el sistema operativo y la máquina real por el otro. En 1968, Alan Kay implementó una máquina virtual vinculada a un sistema orientado a objetos y luego participó con Dan Ingalls en el desarrollo de la máquina virtual de Smalltalk hacia 1972. Numerosos lenguajes y ambientes de scripting utilizan máquinas virtuales: Perl, Javascript, Windows Script Host (WSH), Python, PHP, Pascal.

2.4.5. Estilos heterogéneos

En esta categoría de estilos se encuentran los sistemas de control de procesos industriales, sistemas de transición de estados, arquitecturas específicas de dominios o estilos derivados de otros estilos, como GenVoca, C2 o REST.

a) Sistemas de control de procesos

Desde el punto de vista arquitectónico, mientras casi todos los demás estilos se pueden definir en función de componentes y conectores, los sistemas de control de procesos se caracterizan no sólo por los tipos de componentes, sino por las relaciones que mantienen entre ellos. El objetivo de un sistema de esta clase es mantener ciertos valores dentro de ciertos rangos especificados, llamados puntos fijos o valores de calibración; el caso más clásico es el de los termostatos. Existen mecanismos tanto de retroalimentación (feedback) como de prealimentación (feedforward), y tanto reductores de oscilación como amplificadores; pero el tipo de retroalimentación negativa es el

más común. La ventaja señalada para este estilo radica en su elasticidad ante perturbaciones externas.

b) Arquitectura basada en atributos

En este contexto, los estilos arquitectónicos definen las condiciones en que han de ser usados. Además de especificar los habituales componentes y conectores, los estilos basados en atributos incluyen atributos de calidad específicos que declaran el comportamiento de los componentes en interacción.

2.4.6. Estilos Peer-to-Peer

Esta familia, también llamada de componentes independientes, consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. Los mensajes pueden ser enviados a componentes nominados o propalados mediante broadcast. Miembros de la familia son los estilos basados en eventos, en mensajes (Chiron-2), en servicios y en recursos.

a) Arquitectura basada en eventos

Las arquitecturas basadas en eventos se vinculan históricamente con sistemas basados en actores, *daemons* y redes de conmutación de paquetes (publicación-suscripción). Los conectores de estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos. La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa (como se haría en un estilo orientado a objetos), un componente puede anunciar mediante difusión uno o más eventos. Un componente de un sistema puede anunciar su interés en un evento determinado asociando un procedimiento con la manifestación de dicho evento. Un caso clásico en ambientes Microsoft sería el Servicio de Notificación de SQL Server. Cuando el evento se anuncia, el sistema invoca todos los procedimientos que se han registrado para él. De este modo, el anuncio de un evento implícitamente ocasiona la invocación de determinados procedimientos en otros módulos.

Los ejemplos de sistemas que utilizan esta arquitectura son numerosos. El estilo se utiliza en ambientes de integración de herramientas, en sistemas de gestión de base de datos para asegurar las restricciones de consistencia (bajo la forma de disparadores, por ejemplo), en interfaces de usuario para separar la presentación de los datos de los procedimientos que gestionan datos, y en editores sintácticamente orientados para proporcionar verificación semántica incremental.

b) Arquitectura orientada a servicios

Solo recientemente estas arquitecturas que los conocedores llaman SOA han recibido tratamiento intensivo en el campo de exploración de los estilos. Al mismo tiempo se percibe una

tendencia a promoverlas de un sub-estilo propio de las configuraciones distribuidas que antes eran a un estilo en plenitud. Esta promoción ocurre al compás de las predicciones convergentes de Giga o de Gartner que (después de un par de años de titubeo y consolidación) las visualizan en sus pronósticos y cuadrantes mágicos como la tendencia que habrá de ser dominante en la primera década del nuevo milenio. Ahora bien, este predominio no se funda en la idea de servicios en general, comunicados de cualquier manera, sino que más específicamente va de la mano de la expansión de los *Web services* basados en XML, en los cuales los formatos de intercambio se basan en XML 1.0 *Namespaces* y el protocolo de elección es SOAP. SOAP significa un formato de mensajes que es XML, comunicado sobre un transporte que por defecto es HTTP, pero que puede ser también HTTPS, SMTP, FTP, IIOP, MQ o casi cualquier otro, o puede incluir prestaciones sofisticadas de última generación como WS-Routing, WSAttachment, WS-Referral, etcétera.

c) Arquitecturas Basadas en Recursos

Roy Fielding considera que REST resulta de la composición de varios estilos más básicos, incluyendo repositorio replicado, cache, cliente-servidor, sistema en capas, sistema sin estado, máquina virtual, código a demanda e interfaz uniforme. Fielding no solamente expande más allá de lo habitual y quizá más de lo prudente el catálogo de estilos existentes, sino que su tratamiento estilístico se basa en Perry y Wolf antes que en Garlan y Shaw, debido a que la literatura sobre estilos que se deriva de este último texto sólo considera elementos, conectores y restricciones, sin tomar en consideración los datos, que para el caso de REST al menos constituyen una dimensión esencial.

Resumen

- Una de las definiciones que ha sido utilizada por la comunidad de ingeniería de software es la de RUP, citada por Phillippe Kruchten en 1995, que define lo siguiente:

“La arquitectura de software abarca un conjunto de decisiones significativas sobre la organización del sistema de software. Estas decisiones abarcan: la selección de los elementos estructurales y sus interfaces, con los que se compone el sistema, junto con su comportamiento tal como se especifica en las colaboraciones entre esos elementos, la composición de esos elementos estructurales y de comportamiento en subsistemas progresivamente más grandes, y el estilo arquitectónico que guía esta organización”.

- Un ingeniero de software define la arquitectura del sistema en tres etapas: Elección del estilo arquitectónico, selección de patrones de diseño y diseño de componentes.

- Si desea saber más acerca de estos temas, puede consultar los siguientes libros:

- “SOFTWARE ARCHITECTURE IN PRACTICE” de Len Bass, Paul Clements y Rick Kazman, 2ª edición.

Aquí encontrará la descripción completa de estilos arquitectónicos y patrones de diseño.

- “INGENIERÍA DEL SOFTWARE. UN ENFOQUE PRÁCTICO” de Roger Pressman, 6ª edición.

En el capítulo 10 de este libro se describe el diseño arquitectónico de un sistema.

- Además, puede consultar las siguientes páginas.

- <http://sophia.javeriana.edu.co/~javila/pregrado/arquitectura/estilosArquitectonicos.pdf>

Aquí se presenta un documento que describe los estilos arquitectónicos clasificados por Shaw y Garlan.

- <http://www.ibm.com/developerworks/rational/library/feb06/eeles/>

En este enlace, Peter Eales (Señor IT Architect de IBM) explica lo que es una arquitectura de software.

3. DISEÑO DE CASOS DE USO

El diseño de los casos de uso es la actividad del diseño orientado a objetos que consiste en ...

- Identificar las clases de diseño y/o subsistemas necesarios para la realización del caso de uso.
- Distribuir el comportamiento del caso de uso entre las clases y/o subsistemas de diseño.

Debido a que la clave para la reutilización es anticiparse a los nuevos requisitos y cambios, de modo que los sistemas evolucionen de forma adecuada, es conveniente que para realizar los pasos del diseño de casos de uso, se seleccione el patrón o una combinación de patrones a aplicar en la implementación. Cada patrón permite que algunos aspectos de la estructura del sistema puedan cambiar independientemente de otros aspectos. Facilitan la reutilización, extensibilidad y mantenimiento.

3.1. Patrones de diseño

Como analistas y programadores vamos desarrollando a diario nuestras habilidades para resolver problemas usuales que se presentan en el desarrollo del software. Por cada problema que se nos presenta pensamos distintas formas de resolverlo, incluyendo soluciones exitosas que ya hemos usado anteriormente en problemas similares. Es así que a mayor experiencia que tengamos, nuestro abanico de posibilidades para resolver un problema crece, pero al final siempre habrá una sola solución que mejor se adapte a nuestra aplicación. Si documentamos esta solución, podemos reutilizarla y compartir esa información que hemos aprendido para resolver de la mejor manera un problema específico.

Los patrones del diseño tratan los problemas del diseño que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Por lo tanto, los patrones de diseño son soluciones exitosas a problemas comunes. Asimismo, son descripciones de clases cuyas instancias colaboran entre sí. Cada patrón es adecuado para ser adaptado a un cierto tipo de problema. Para describir un caso debemos especificar:

- Nombre
- Propósito o finalidad
- Sinónimos (otros nombres por los que puede ser conocido)
- Problema al que es aplicable
- Estructura (diagrama de clases)
- Participantes (responsabilidad de cada clase)
- Colaboraciones (diagrama de interacciones)
- Implementación (consejos, notas y ejemplos)
- Otros patrones con los que está relacionado

3.1.1. Patrones GOF

El catálogo más famoso de patrones se encuentra en *“Design Patterns: Elements of Reusable Object-Oriented Software”*, de Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, 1995, Addison-Wesley, también conocido como el libro GOF (Gang-Of-Four).

Los autores de este patrón recopilaron 23 patrones que son utilizados por muchos diseñadores de software orientado a objetos.

		Propósito		
		Creación	Estructural	Comportamiento
Ámbito	Clase	Factory Method	Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Figura 3.4. Patrones de diseño según GOF.

En la figura 3.4 se muestra la clasificación de los patrones GOF considerando dos aspectos: el propósito para el que han sido definidos y el ámbito.

Según el propósito para el que han sido definidos existen 3 tipos:

- **Creacionales:** Solucionan problemas de creación de instancias. Nos ayudan a encapsular y abstraer dicha creación.
- **Estructurales:** Solucionan problemas de composición (agregación) de clases y objetos.
- **De Comportamiento:** Dan soluciones respecto a la interacción y responsabilidades entre clases y objetos, así como los algoritmos que encapsulan.

Según el ámbito, se clasifica en patrones de clase y de objeto, es así que se cuenta con 6 tipos:

Creacionales

- **Creacional de la Clase**
Los patrones creacionales de Clases usan la herencia como un mecanismo para lograr la instanciación de la Clase. Por ejemplo, el método Factoría.
- **Creacional del objeto**
Los patrones creacionales de objetos son más escalables y dinámicos comparados de los patrones creacionales de Clases. Por ejemplo, la Factoría abstracta y el patrón *Singleton*.

Estructurales

- **Estructural de la Clase**

Los patrones estructurales de Clases usan la herencia para proporcionar interfaces más útiles combinando la funcionalidad de múltiples Clases. Por ejemplo, el patrón Adaptador (Clase).

- **Estructural de Objetos**

Los patrones estructurales de objetos crean objetos complejos agregando objetos individuales para construir grandes estructuras. La composición de 1 patrón estructural del objeto puede ser cambiado en tiempo de ejecución, el cual nos da flexibilidad adicional sobre los patrones estructurales de Clases. Por ejemplo el Adaptador (Objeto), *Facade*, *Bridge*, *Composite*.

Comportamiento

- **Comportamiento de Clase**

Los patrones de comportamiento de Clases usan la herencia para distribuir el comportamiento entre Clases. Por ejemplo, *Interpreter*.

- **Comportamiento de Objeto**

Los patrones de comportamiento de objetos nos permiten analizar los patrones de comunicación entre objetos interconectados, como objetos incluidos en un objeto complejo. Ejemplo *Iterator*, *Observer*, *Visitor*.

La figura 3.5 muestra la plantilla que utilizan los autores de patrones GOF para describir un patrón.

<p>Nombre del patrón: Nombre estándar del patrón.</p> <p>Contexto: ¿Cuál es el alcance del patrón?</p> <p>Problema: ¿Qué pretende resolver?</p> <p>Solución</p> <p> Estructura: Diagramas de clases oportunos para describir las clases que intervienen en el patrón.</p> <p> Estrategia: Como funciona.</p> <p>Consecuencias: Consecuencias positivas y negativas en el diseño derivadas de la aplicación del patrón.</p> <p>Patrones relacionados: Referencias cruzadas con otros patrones.</p>

Figura 3.5. Plantilla de Patrones GOF.

3.1.2. Patrones J2EE

Con la aparición del J2EE, todo un nuevo catálogo de patrones de diseño apareció. Desde que J2EE es una arquitectura por sí misma que involucra otras arquitecturas, incluyendo servlets, JavaServer Pages, Enterprise JavaBeans, y más, merece su propio conjunto de patrones específicos para diferentes aplicaciones empresariales.

De acuerdo al libro "J2EE PATTERNS Best Practices and Design Strategies", existen 5 capas en la arquitectura J2EE:

- **Cliente:** Esta capa corresponde a lo que se encuentra en el computador del cliente. Es la interfaz gráfica del sistema y se encarga de interactuar con el usuario. J2EE tiene soporte para diferentes tipos de clientes incluyendo clientes HTML, applets Java y aplicaciones Java.
- **Presentación:** La capa de presentación es la que ve el usuario, comunica y captura la información proveniente de él.
- **Negocio:** Es donde reside el núcleo del sistema. Se comunica con la capa de presentación para recibir y enviar los datos al usuario.
- **Integración:** La capa de integración es donde residen los datos y es la encargada de acceder a los mismos. Contiene uno o más gestores de base de datos. Recibe solicitudes de almacenamiento o recuperación de información desde la capa de negocio.
- **Recurso:** Es la capa donde reside la base de datos y sistemas legados.

Nombre	Quiénes la componen	Dónde se ubica
Capa Cliente	Aplicaciones cliente, applets, aplicaciones y otras GUIs	PC Cliente
Capa de Presentación	JSP, Servlet y otras UIs	Servidor J2EE
Capa de Negocios	EJBs y otros objetos de negocios	Servidor J2EE
Capa de Integración	JMS, JDBC	Servidor J2EE
Capa de Recursos	Bases de Datos, Sistemas Legados	Servidor BD

Tabla 3.1. Capas de la Arquitectura J2EE.

Existen 15 patrones J2EE que están divididos en 3 de las capas descritas: presentación, negocio e integración. Cabe señalar que todos estos patrones poseen tanto características de diseño como de arquitectura.

Capa de Presentación

Patrón	Descripción
Decorating Filter / Intercepting Filter	Un objeto que está entre el cliente y los componentes Web. Este procesa las peticiones y las respuestas.

Front Controller/ Front Component	Un objeto que acepta todos los requerimientos de un cliente y los direcciona a manejadores apropiados. El patrón Front Controller podría dividir la funcionalidad en 2 diferentes objetos: el Front Controller y el Dispatcher. En ese caso, El Front Controller acepta todos los requerimientos de un cliente y realiza la autenticación, y el Dispatcher direcciona los requerimientos a manejadores apropiada.
View Helper	Un objeto helper que encapsula la lógica de acceso a datos en beneficio de los componentes de la presentación. Por ejemplo, los JavaBeans pueden ser usados como patrón View Helper para las páginas JSP.
Composite view	Un objeto vista que está compuesto de otros objetos vista. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva include o el action include es un patrón Composite View.
Service To Worker	El Controlador actúa como Front Controller, pero con un factor importante: aquí el Dispatcher (el cual es parte del Front Controller) usa View Helpers a gran escala y ayuda en el manejo de la vista.
Dispatcher View	Es con el controlador actuando como Front Controller, pero con un asunto importante: aquí el Dispatcher (el cual es parte del Front Controller) no usa View Helpers y realiza muy poco trabajo en el manejo de la vista. El manejo de la vista es manejado por los mismos componentes de la Vista.

Capa de Negocio

Patrón	Descripción
Business Delegate	Un objeto que reside en la capa de presentación y en beneficio de los otros componentes de la capa de presentación llama a métodos remotos en los objetos de la capa de negocios.
Value Object/ Data Transfer Object/ Replicate Object	Un objeto serializable para la transferencia de datos sobre la red.
Session Façade/ Session Entity Façade/ Distributed Façade	El uso de un bean de sesión como una fachada (facade) para encapsular la complejidad de las interacciones entre los objetos de negocio y participantes en un flujo de trabajo. El Session Façade maneja

	los objetos de negocio y proporciona un servicio de acceso uniforme a los clientes.
Aggregate Entity	Un bean entidad que es construido o es agregado a otros beans de entidad.
Value Object Assembler	Un objeto que reside en la capa de negocios y crea Value Objects cuando es requerido.
Value List Handler/ Page-by-Page Iterator/ Paged List	Es un objeto que maneja la ejecución de consultas SQL, caché y procesamiento del resultado. Usualmente implementado como beans de sesión.
Service Locator	Consiste en utilizar un objeto Service Locator para abstraer toda la utilización JNDI y para ocultar las complejidades de la creación del contexto inicial, de búsqueda de objetos home EJB y recreación de objetos EJB. Varios clientes pueden reutilizar el objeto Service Locator para reducir la complejidad del código, proporcionando un punto de control.

Capa de Integración

Patrón	Descripción
Data Access Object	Consiste en utilizar un objeto de acceso a datos para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.
Service Activator	Se utiliza para recibir peticiones y mensajes asíncronos de los clientes. Cuando se recibe un mensaje, el Service Activator localiza e invoca a los métodos de los componentes de negocio necesarios para cumplir la petición de forma asíncrona.

3.2. Extensiones de UML para aplicaciones web (WAE)

En 1998, Jim Conallen (empleado de Rational Software Corporation) definió una extensión de UML para representar aplicaciones Web, a la que denominó WAE (Web Application Extension). Esta extensión es la convención más difundida y aceptada hasta nuestros días y podríamos decir que define el estándar de facto.

Lo primero que se planteó Jim es que las aplicaciones Web presentan elementos que UML no contemplaba para su representación. UML no facilitaba la tarea de diferenciar código cliente (scripts) de código servidor. Así fue que a partir de UML extendió una nueva semántica: estereotipos, listados de etiquetas (tags) y restricciones (constraints).

- Estereotipos: Define una nueva semántica al modelo.
- Lista de etiquetas: Consiste en una lista de campo-valor.
- Restricciones: Definen las reglas para trabajar con determinados estereotipos.

En los siguientes puntos se describirá cómo representar los elementos de una aplicación web en diagramas de clases, diagrama de componentes y diagrama de secuencia.

1.1.1. Diagramas de clases

1. Estereotipos en clases

Estereotipos en clases define los siguientes estereotipos:

- **<<Server Page>>** Son las páginas que contienen scripts o código ejecutable por el servidor. (.php , .asp , .jsp)
- **<<Client Page>>** Son las páginas que están en el lado del cliente, normalmente páginas HTML y scripts (javascript).
- **<<Form>>** Es la representación de un formulario. Es código HTML que contiene etiquetas de formulario como: <input>, <textarea>, <select>, etc.

Estos estereotipos se pueden seguir ampliando a partir de una clase, la cual es un elemento que ya existe en UML. Otros ejemplos de estereotipo pueden ser <<Javascript>>, <<Applet>> o <<Flash>>.

2. Estereotipos en relaciones

Define los siguientes estereotipos para las relaciones de asociación unidireccionales:

- **<<build>>** Es una asociación unidireccional entre una página servidor y una página cliente. La página servidor "construye" a la página cliente.
- **<<link>>** Es una asociación unidireccional entre una página y otra página del sistema (página servidor o página cliente).
- **<<submit>>** Es una asociación unidireccional entre un formulario y una página servidor.
- **<<redirect>>** Es también una asociación unidireccional que indica que una página Web redirige hacia otra. En caso de que la página origen sea una "client page" esta asociación corresponderá con la etiqueta "META" y valor HTTP-EQUIV de "Refresh".
- **<<forward>>** Es una asociación unidireccional entre una página servidor y otra página servidor o una página cliente. Esta asociación representa la delegación de procesamiento de la solicitud de un cliente para un recurso a otra página del lado del servidor.
- **<<object>>** Es una relación entre una página cliente y una clase, normalmente, uno que representa a un applet, el control ActiveX u otro componente.

- **<<include>>** Se da entre una página servidor y otra página servidor o página cliente. Durante la ejecución de la página, esta asociación indica que la página incluida es procesada.

En las siguientes figuras, se muestra dos diagramas de clases que utilizan los diferentes estereotipos descritos para clases y relaciones.

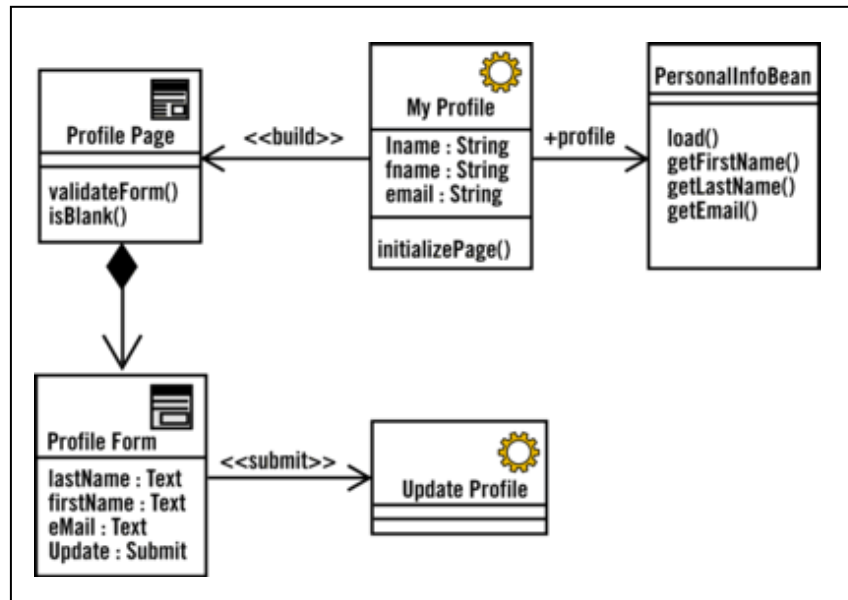


Figura 3.6. Diagrama de clases con páginas y relaciones **<<build>>** y **<<submit>>**.

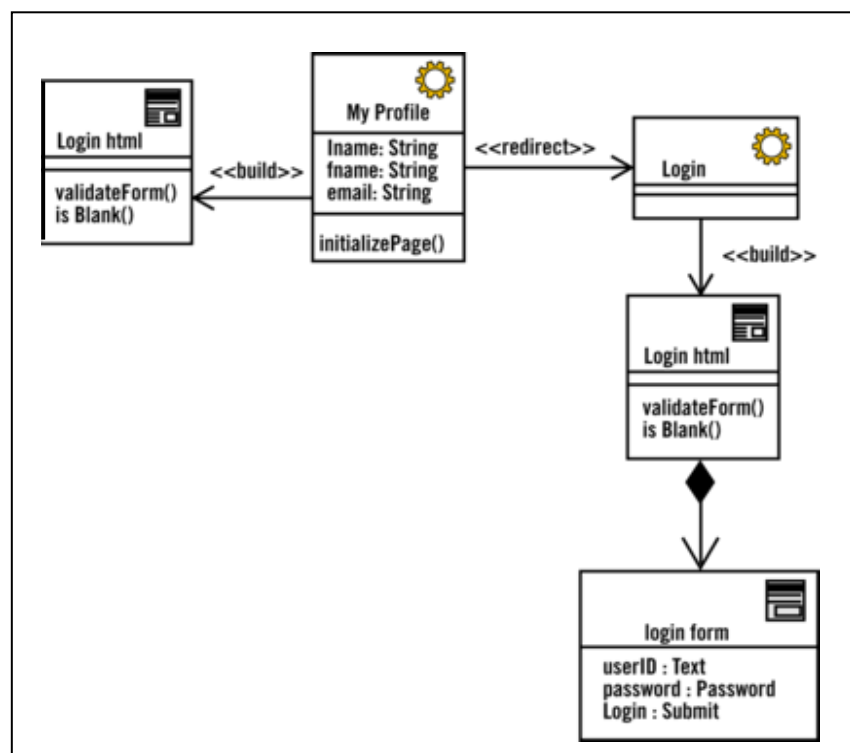


Figura 3.7. Diagrama de clases con páginas y relaciones **<<build>>** y **<<redirect>>**.

1.1.2. Diagrama de Componentes

Ilustran las piezas del software que conformarán un sistema. Un componente puede ser un ejecutable, una librería estática o dinámica, clases de Java, etc. **Un componente abstrae un conjunto de clases para que pueda el componente ser utilizado como una unidad**, esto es el API que todo componente debe proveer. Su funcionamiento es igual que con UML 2.0. En WAE se construye un diagrama de componentes tanto para mostrar el flujo de navegación de la aplicación como para representar los componentes que contiene el servidor.

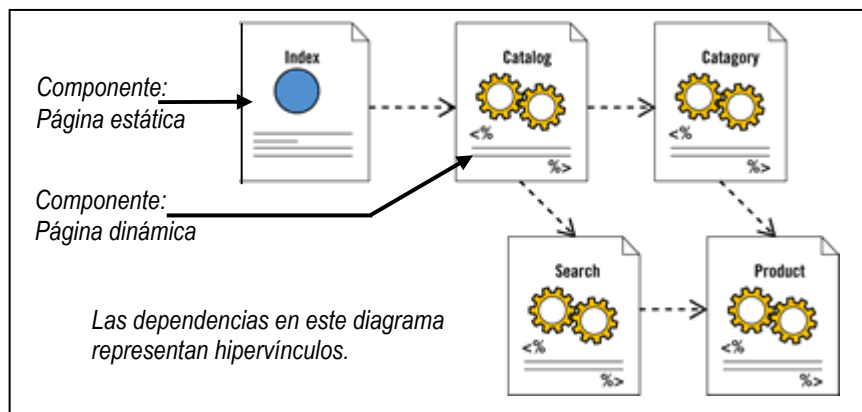


Figura 3.8. Flujo de navegación.

En la siguiente figura, se ilustra un componente para abstraer el hecho de que una página servidor genera una página de cliente; por ello, **<<server page>> y <<client page>> se modela agrupándolos como un componente**. Asimismo, se define el estereotipo <<virtual root>> en un paquete para representar el contenedor de los componentes de la aplicación web.

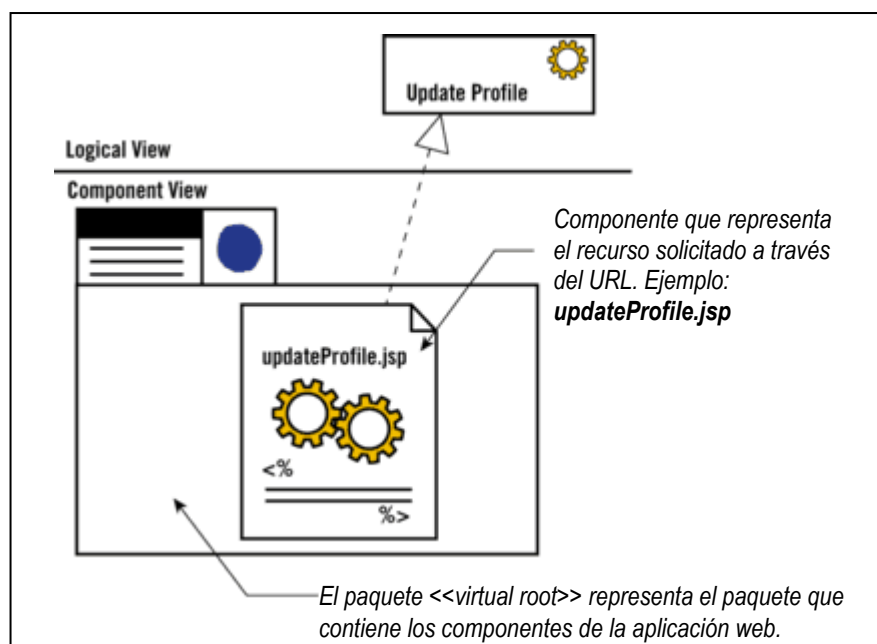


Figura 3.9. Diagrama de componentes de una aplicación web.

1.1.3. Diagrama de secuencia

Describe el comportamiento de los casos de uso en función del tiempo. Su uso respecto a UML no cambia. En este diagrama se escriben las líneas de tiempo de los actores y componentes implicados. Los actores y componentes se envían mensajes entre ellos describiendo el comportamiento del sistema. Las páginas del cliente pueden enviarse mensajes así mismo (funciones javascript donde el servidor no interviene), pero si vemos **fechas asíncronas que van desde el cliente hasta el servidor solo pueden ser interpretadas por el programador como el uso de AJAX**, ya que la tecnología web es síncrona.

3.3. Realización de diseño de casos de uso con patrón arquitectónico MVC

La realización de casos de uso es una colección de varios diagramas UML que validan que nosotros tenemos las clases, responsabilidades e interacciones de los objetos necesarios para dictaminar el comportamiento que debe tener el proceso de nuestro caso de uso.

En el análisis, nuestra realización de casos de uso contiene solo el análisis de clases y objetos, los cuales pueden contar con varios diagramas UML, como pueden ser los diagramas de clases y los de interacción. En nuestro diseño, la realización contendrá información que explica como los pasos dentro de un caso de uso se llevarán a cabo colaborando con el diseño de objetos. Los diagramas de clase, de interacción y descripción derivados de los requisitos detallarán nuestras realizaciones.

Antes de indicar cómo se realiza las realizaciones de diseño de un caso de uso, primero se mostrará la organización de las clases de diseño en capas, subsistemas y librerías que utilizaremos en el curso, aplicando patrón arquitectónico MVC:


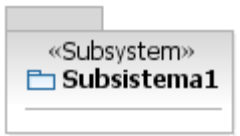

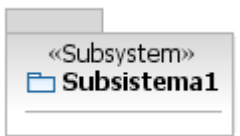
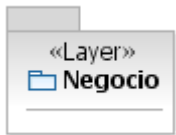
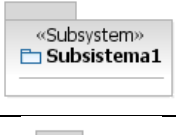

Capa	Subsistema/Librerías	Elementos de diseño
		Clases estereotipadas: <ul style="list-style-type: none"> • Páginas HTML: <<Client Page>> • Páginas JSP: <<Server Page>>, <<Client Page>> y <<HTML Form>>
		Clase estereotipada para servlets: <<Http Servlet>>
		Clases de diseño: <i>beans</i> .
		Clases de diseño: clases utilitarias.

Tabla 3.2. Capas, subsistemas, librerías y elementos de diseño.

A partir de la Especificación del Caso de Uso *Mantener Cajeros* crearemos los diagramas de la realización de diseño del caso de uso.

ESPECIFICACIÓN DE CASO DE USO: Mantener Cajero

1. Descripción

El caso de uso permite mantener actualizado el registro de los cajeros de la clínica. De acuerdo a su necesidad, el administrador de la clínica puede agregar, actualizar y desactivar un cajero.

2. Actor(es)

Administrador

3. Flujo de Eventos

3.1. Flujo Básico

1. El caso de uso se inicia cuando el Administrador selecciona la opción “Cajeros” en la interfaz del menú principal.
2. El sistema muestra la interfaz “MANTENER CAJERO” con la lista de cajeros con los campos: código, nombres, apellido paterno, apellido materno, teléfono, correo, dirección, fecha de registro, fecha de actualización y estado. Además, muestra las opciones: **Agregar Cajero, Actualizar Cajero y Desactivar Cajero.**
3. Si el Administrador elige un cajero
 - a. Si elige “Actualizar” ver el Subflujo Actualizar Cajero.
 - b. Si elige “Desactivar” ver el Subflujo Desactivar Cajero.
4. Si el Administrador NO elige un cajero
 - a. Si elige “Agregar” ver el Subflujo Agregar Cajero.
5. El Administrador selecciona “Salir” y el caso de uso finaliza.

3.2. Subflujos

3.2.1. Agregar Cajero

1. El sistema muestra la interfaz CAJERO con los siguientes campos: código (sólo lectura), nombres, apellido paterno, apellido materno, teléfono, correo, dirección, fecha de registro (sólo lectura) y fecha de actualización (sólo lectura). Además, muestra las opciones: **Aceptar y Cancelar.**
2. El Administrador ingresa los datos del Cajero.
3. El Administrador selecciona la opción Aceptar.
4. El sistema valida los datos ingresados.
5. El sistema genera un nuevo código de cajero y obtiene la fecha del sistema para la fecha de registro y la fecha de actualización
6. El sistema graba un nuevo registro de cajero y muestra el MSG “Cajero creado con código Nro. 999999”.
7. El Administrador cierra la interfaz CAJERO y regresa a la interfaz MANTENER CAJERO con la lista de cajeros actualizada y el subflujo finaliza.

3.2.2. Actualizar Cajero

1. El sistema muestra los datos del cajero seleccionada en la interfaz CAJERO: código (sólo lectura), nombres, apellido paterno, apellido materno, teléfono, correo, dirección, fecha de registro (sólo lectura) y fecha de actualización (sólo lectura). Además, muestra las opciones: **Aceptar y Cancelar.**
2. El Administrador actualiza los datos del cajero.
3. El Administrador selecciona la opción Aceptar.

4. El sistema valida los datos ingresados del cajero.
5. El sistema obtiene la fecha del sistema para la fecha de actualización, actualiza el registro de cajero, y muestra el MSG "Cajero actualizado satisfactoriamente".
6. El Administrador cierra la interfaz CAJERO y regresa a la interfaz MANTENER CAJERO con la lista de cajeros actualizada y el subflujo finaliza.

3.2.3. Desactivar Cajero

1. El sistema muestra el MSG: "¿Está seguro que desea desactivar el(los) cajero(s) seleccionado(s)?".
2. El Administrador selecciona la opción YES para confirmar la desactivación.
3. El sistema actualiza el registro del(los) cajero(s) en estado "Desactivado".
4. El sistema muestra la interfaz MANTENER CAJERO con la lista de cajeros actualizada y termina el subflujo.

3.3. Flujos Alternativos

1. Datos del Cajero Inválidos

Si los datos ingresados son nulos o inválidos, tanto en los subflujos Agregar como en Actualizar Cajero, el sistema muestra el MSG: "Se han encontrado datos inválidos" y los subflujos continúan en el paso 2.

2. Cajero ya existe

Si el sistema detecta que el cajero ya existe en el paso 4 del subflujo Agregar Cajero, muestra el MSG: "Cajero ya existe" y el subflujo finaliza.

3. No confirma Desactivación

Si el Administrador selecciona NO en el paso 2 del subflujo Desactivar Cajero, finaliza el subflujo.

4. Precondiciones

1. El Administrador está identificado en el sistema.
2. Lista disponible de Cajeros.

5. Poscondiciones

1. En el sistema quedará registrado el nuevo Cajero.
2. En el sistema quedará actualizado el registro del Cajero.
3. En el sistema quedará desactivado el Cajero.

6. Puntos de Extensión

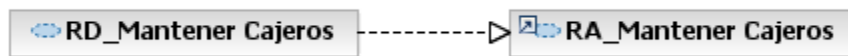
Ninguno.

7. Requisitos Especiales

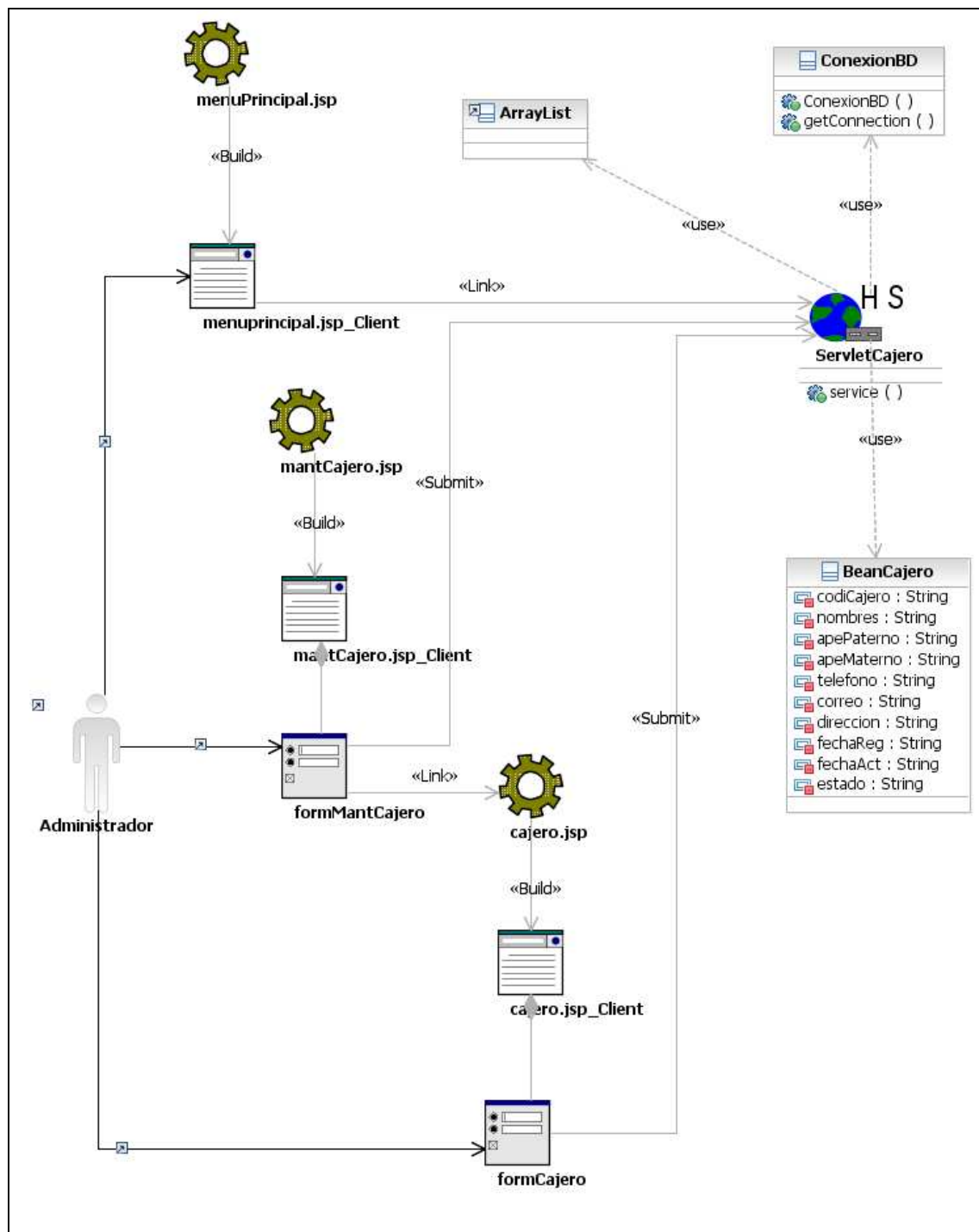
Ninguno.

A continuación, se muestra los diagramas de la realización de diseño de caso de uso.

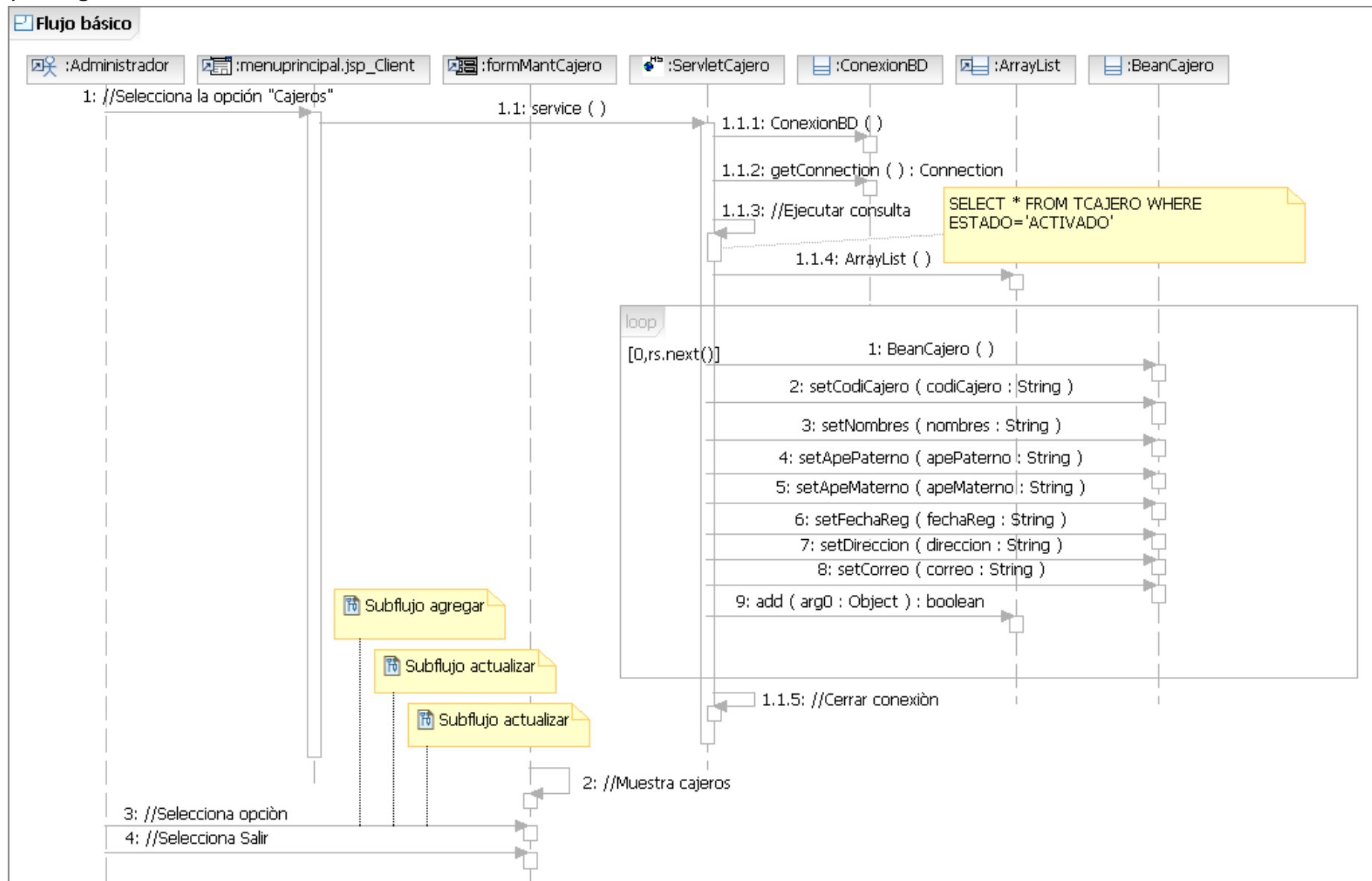
1) Realización de diseño



2) Diagrama de Clases de Diseño



3) Diagramas de Secuencia



ACTIVIDADES PROPUESTAS

Elabore el diagrama de secuencia para los subflujos agregar, actualizar y desactivar cajeros del caso de uso Mantener Cajeros.

3.4. Realización de diseño de casos de uso con patrón arquitectónico MVC y patrón de diseño DAO

En la siguiente tabla, se muestra la organización de las clases de diseño e interfaces en capas, subsistemas y librerías que utilizaremos en el curso, aplicando patrón arquitectónico MVC y patrón de diseño DAO:

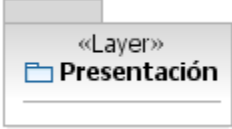
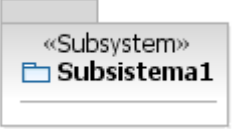

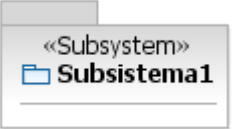
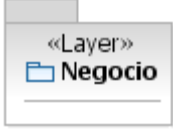
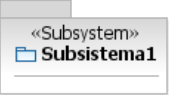


Capa	Subsistema/Librerías	Elementos de diseño
		Clases estereotipadas: <ul style="list-style-type: none"> • Páginas HTML: <<Client Page>> • Páginas JSP: <<Server Page>>, <<Client Page>> y <<HTML Form>>
		Clase estereotipada para servlets: <<Http Servlet>>
		<ul style="list-style-type: none"> • Clases de diseño: servicios, <i>beans</i> y clases DAO. • Interfaces que presentan las operaciones de acceso a una tabla.
		Clases de diseño: clase abstracta <i>DAOFactory</i> y sus clases hijas.
		Clases de diseño: clases utilitarias.

Tabla 3.3. Capas, subsistemas, librerías y elementos de diseño.

A partir de la Especificación del Caso de Uso *Mantener Cajeros*, crearemos los diagramas de la realización de diseño del caso de uso.

ESPECIFICACIÓN DE CASO DE USO: Mantener cajeros

1. Descripción

El caso de uso permite mantener actualizado el registro de los cajeros de la clínica. De acuerdo a su necesidad, el administrador de la clínica puede agregar, actualizar y desactivar un cajero.

2. Actor(es)

Administrador

3. Flujo de Eventos

3.1. Flujo Básico

1. El caso de uso se inicia cuando el Administrador selecciona la opción

- “Cajeros” en la interfaz del menú principal.
2. El sistema muestra la interfaz “MANTENER CAJERO” con la lista de cajeros con los campos: código, nombres, apellido paterno, apellido materno, teléfono, correo, dirección, fecha de registro, fecha de actualización y estado. Además muestra las opciones: **Agregar Cajero, Actualizar Cajero y Desactivar Cajero.**
 3. Si el Administrador elige un cajero
 - a. Si elige “Actualizar” ver el Subflujo Actualizar Cajero.
 - b. Si elige “Desactivar” ver el Subflujo Desactivar Cajero.
 4. Si el Administrador NO elige un cajero
 - a. Si elige “Agregar” ver el Subflujo Agregar Cajero.
 5. El Administrador selecciona “Salir” y el caso de uso finaliza.

3.2. Subflujos

3.2.1. Agregar Cajero

1. El sistema muestra la interfaz CAJERO con los siguientes campos: código (sólo lectura), nombres, apellido paterno, apellido materno, teléfono, correo, dirección, fecha de registro (sólo lectura) y fecha de actualización (sólo lectura). Además, muestra las opciones: **Aceptar y Cancelar.**
2. El Administrador ingresa los datos del Cajero.
3. El Administrador selecciona la opción Aceptar.
4. El sistema valida los datos ingresados.
5. El sistema genera un nuevo código de cajero y obtiene la fecha del sistema para la fecha de registro y la fecha de actualización
6. El sistema graba un nuevo registro de cajero y muestra el MSG “Cajero creado con código Nro. 999999”.
7. El Administrador cierra la interfaz CAJERO y regresa a la interfaz MANTENER CAJERO con la lista de cajeros actualizada y el subflujo finaliza.

3.2.2. Actualizar Cajero

1. El sistema muestra los datos del cajero seleccionada en la interfaz CAJERO: código (sólo lectura), nombres, apellido paterno, apellido materno, teléfono, correo, dirección, fecha de registro (sólo lectura) y fecha de actualización (sólo lectura). Además, muestra las opciones: **Aceptar y Cancelar.**
2. El Administrador actualiza los datos del cajero.
3. El Administrador selecciona la opción Aceptar.
4. El sistema valida los datos ingresados del cajero.
5. El sistema obtiene la fecha del sistema para la fecha de actualización, actualiza el registro de cajero, y muestra el MSG “Cajero actualizado satisfactoriamente”.
6. El Administrador cierra la interfaz CAJERO y regresa a la interfaz MANTENER CAJERO con la lista de cajeros actualizada y el subflujo finaliza.

3.2.3. Desactivar Cajero

1. El sistema muestra el MSG: “¿Está seguro que desea desactivar el(los) cajero(s) seleccionado(s)?”.
2. El Administrador selecciona la opción YES para confirmar la desactivación.
3. El sistema actualiza el registro del(los) cajero(s) en estado “Desactivado”.

4. El sistema muestra la interfaz MANTENER CAJERO con la lista de cajeros actualizada y termina el subflujo.

3.3. Flujos Alternativos

1. Datos del Cajero Inválidos

Si los datos ingresados son nulos o inválidos, tanto en los subflujos Agregar como en Actualizar Cajero, el sistema muestra el MSG: “Se han encontrado datos inválidos” y los subflujos continúan en el paso 2.

2. Cajero ya existe

Si el sistema detecta que el cajero ya existe en el paso 4 del subflujo Agregar Cajero, muestra el MSG: “Cajero ya existe” y el subflujo finaliza.

3. No confirma Desactivación

Si el Administrador selecciona NO en el paso 2 del subflujo Desactivar Cajero, finaliza el subflujo.

4. Precondiciones

1. El Administrador está identificado en el sistema.
2. Lista disponible de Cajeros.

5. Poscondiciones

1. En el sistema quedará registrado el nuevo Cajero.
2. En el sistema quedará actualizado el registro del Cajero.
3. En el sistema quedará desactivado el Cajero.

6. Puntos de Extensión

Ninguno.

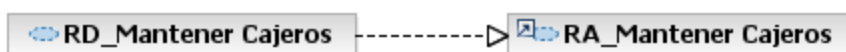
7. Requisitos Especiales

Ninguno.

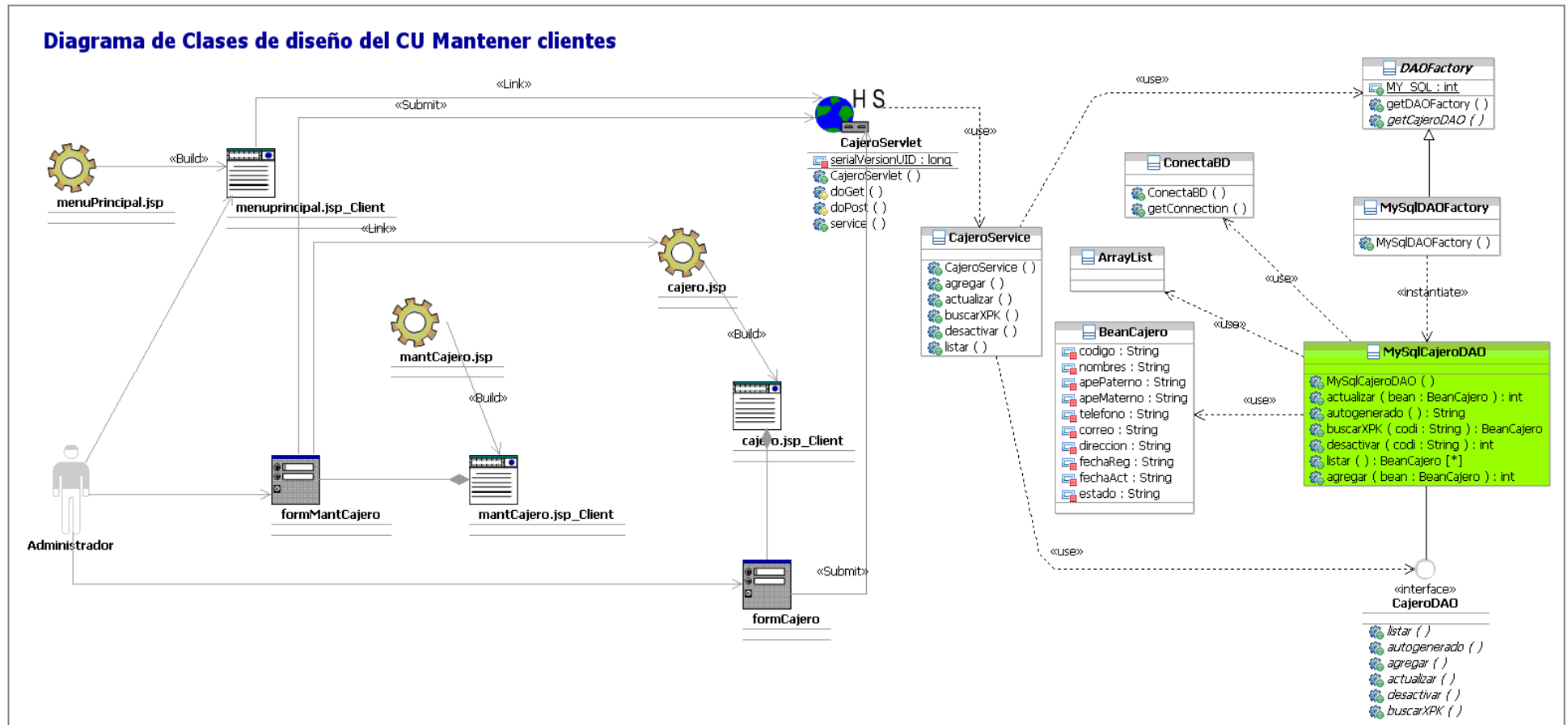
A continuación, se muestra los siguientes artefactos de la realización de diseño de caso de uso:

1. Realización de diseño
2. Diagrama de clases de diseño
3. Diagramas de secuencia
 - a. Flujo básico
 - b. Flujo del método listar
 - c. Subflujo agregar
 - d. Flujo del método autogenerado
 - e. Flujo del método agregar

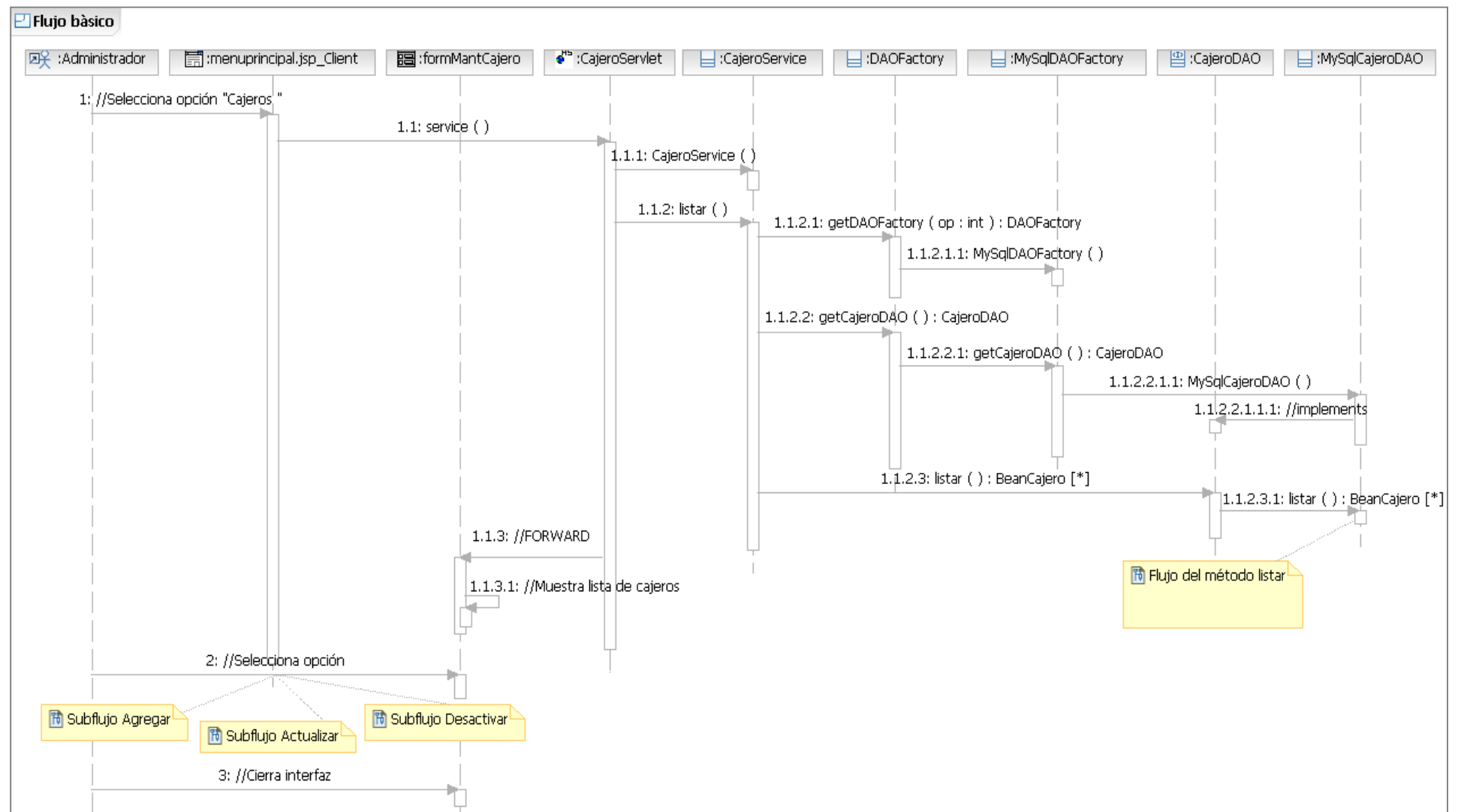
1. Realización de diseño

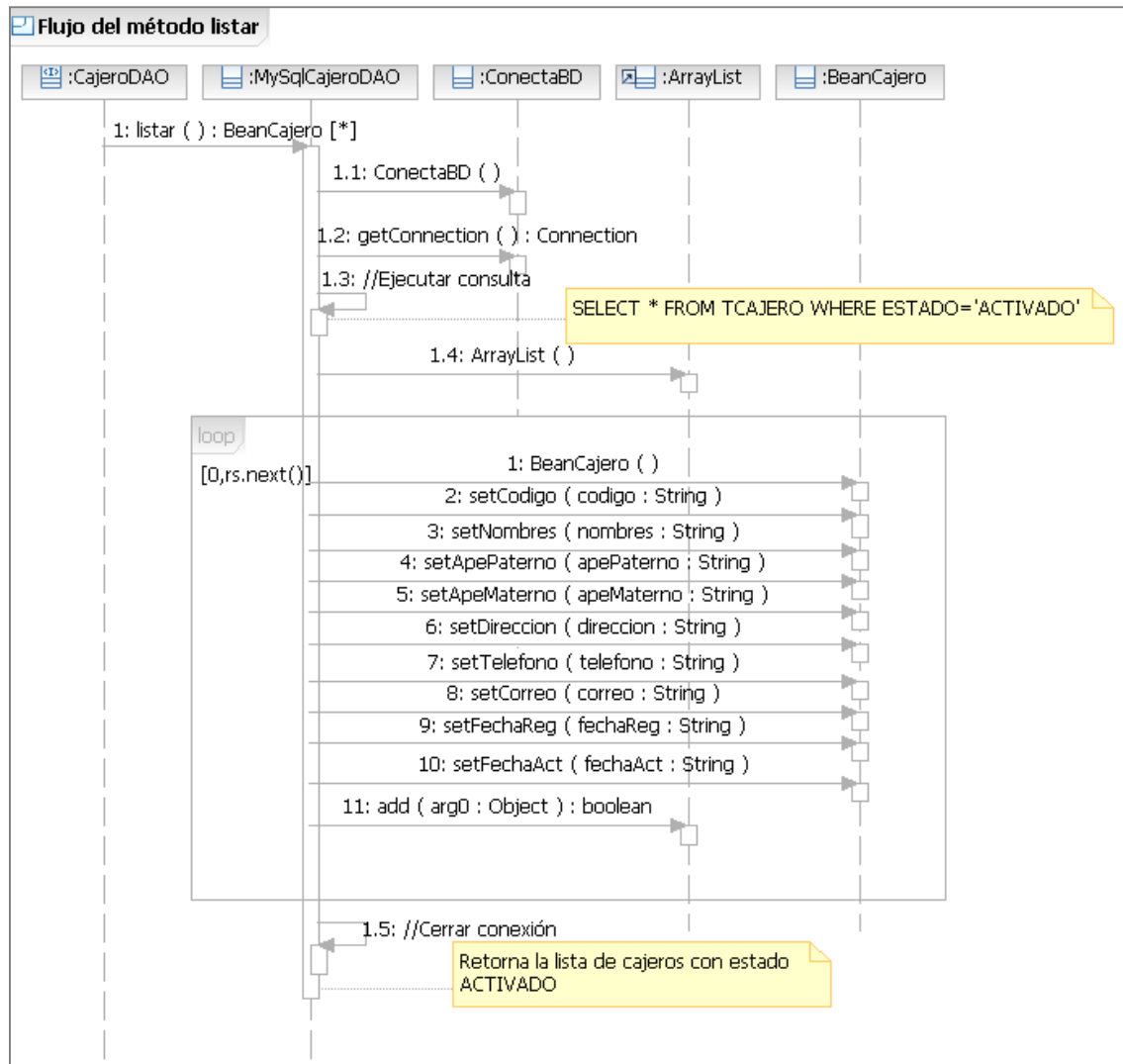


2. Diagrama de clases de diseño



3. Diagrama de secuencia





ACTIVIDADES PROPUESTAS

Elabore el diagrama de secuencia para los subflujos agregar, actualizar y desactivar cajeros y de los métodos del DAO del caso de uso Mantener Cajeros.

Caso práctico

A partir de la Especificación del Caso de Uso crearemos los diagramas de la realización de diseño del caso de uso ECU Generar reserva.

Especificación de Caso de Uso: Generar Reserva

1. Descripción

El caso de uso permite a la Recepcionista de un Hotel generar reserva de habitación(es).

2. Actor(es)

Recepcionista

3. Flujo de Eventos

3.1. Flujo Básico

1. El Caso de uso se inicia cuando la recepcionista selecciona la opción “Generar Reserva” en la interfaz del menú principal.
2. El sistema muestra la interfaz RESERVA con los siguientes datos:
 - Datos del cliente: Código, Nombre.
 - Datos de la Reserva: fecha de llegada, fecha de salida y cantidad de días a hospedarse.
 - Datos de las habitaciones: Número de habitación, Tipo, Costo por día, Nombre del huésped de la Habitación y una opción para Agregar.
 - Además, incluye una cuadrícula que contiene la lista de todas las habitaciones seleccionadas y las opciones: **Buscar cliente, Agregar cliente, consultar disponibilidad de habitación, eliminar habitación, Grabar y Salir.**
3. La recepcionista selecciona “Buscar cliente”.
4. El sistema **incluye el Caso de Uso Buscar Cliente.**
5. El sistema muestra los datos del cliente.
6. La recepcionista ingresa la fecha de llegada y la fecha de salida.
7. El sistema calcula la cantidad de días.
8. La recepcionista solicita “Consultar disponibilidad de habitación”.
9. El sistema **incluye el caso de uso consultar disponibilidad de habitación.**
10. El sistema muestra la habitación seleccionada.
11. La Recepcionista ingresa el nombre de la persona para la habitación seleccionada.
12. La Recepcionista selecciona: agregar
13. El sistema calcula el pago de la habitación, el subtotal, el monto total y lo agrega a la cuadrícula del detalle de la reserva.
14. Si la Recepcionista quiere seleccionar otra habitación, se repite del paso 7 al 12.
15. La recepcionista selecciona “Grabar”.
16. El sistema autogenera un número de reserva.
17. El sistema graba la reserva con su detalle y registra la(s) disponibilidad(es) de la(s) habitación(es) en estado “Reservado”.
18. El sistema muestra el número de reserva y el MSG “Reserva generada” con el Nro. 99999”.
19. La recepcionista cierra la interfaz RESERVA y regresa a la interfaz del menú principal del sistema y finaliza el caso de uso.

3.2. Subflujos

Ninguno.

3.3. Flujos Alternativos

1. Cliente no existe

En el paso 4, si el sistema detecta que el cliente no existe, muestra el MSG: “Cliente no existe” y ofrecerá la posibilidad de registrar al nuevo cliente.

2. Habitaciones no disponibles

En el paso 9, si el sistema detecta que no hay habitaciones disponibles muestra el MSG: “No hay habitaciones disponibles” y finaliza el caso de uso.

3. Eliminar Habitación de Cuadrícula

La recepcionista selecciona una habitación de la cuadrícula y selecciona eliminar, el sistema elimina de la cuadrícula la habitación selecciona y el caso de uso continúa.

4. Precondiciones

4.7. El Recepcionista está logeado en el sistema.

4.8. Lista de clientes disponibles.

4.9. Lista de habitaciones disponibles.

5. Poscondiciones

5.5. En el sistema quedará registrada la reserva con su detalle.

5.6. Las disponibilidades de las habitaciones seleccionadas se registraran en estado “Reservadas”.



6. Puntos de Extensión

En el paso 5, el sistema extiende al caso de uso **Mantener Clientes – subflujo “Agregar Cliente”**.

7. Prototipos

Interfaz RESERVA

Datos del Cliente



Cliente  

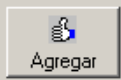
Nombre

Datos de la Reserva

Fecha de Llegada Fecha de Salida Cantidad Dias


Datos de las Habitaciones


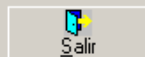
Numero de Habitacion  Tipo Costo x día 

Nombre del Huesped 

Detalle de la Reserva

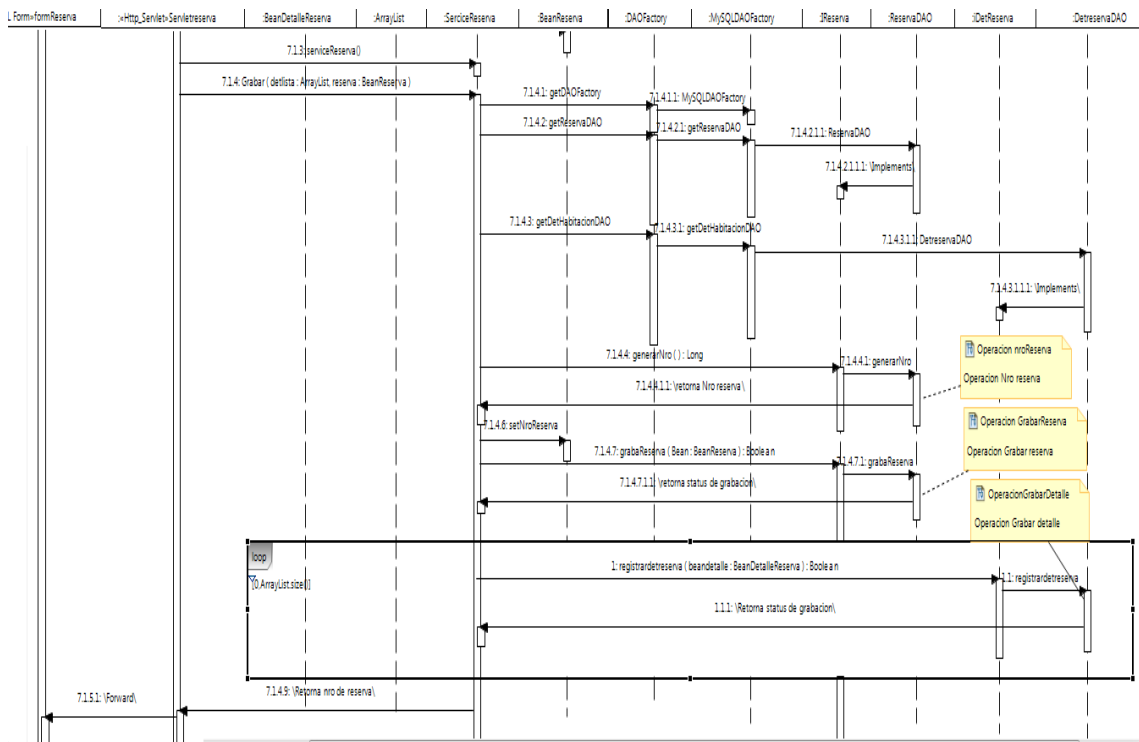
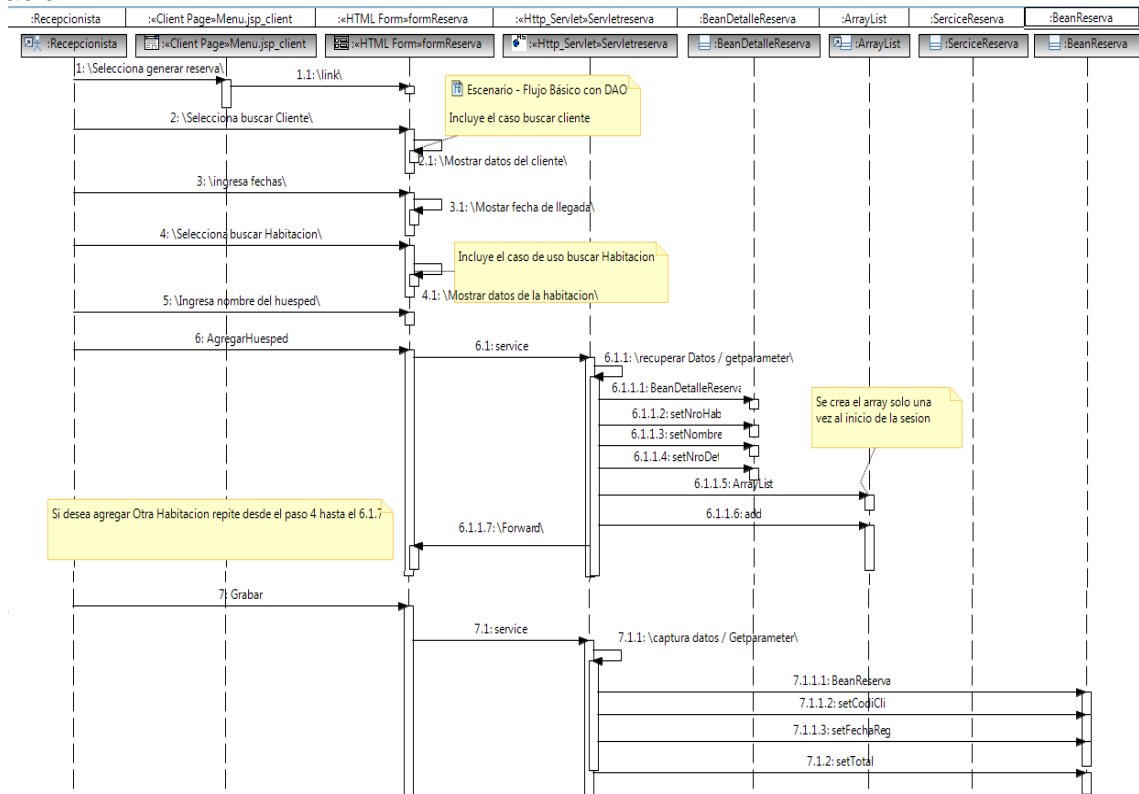
Numero	Tipo	Huesped	Costo diario	Días	Monto a Pagar


 Sub total
 IGV
 Monto Total

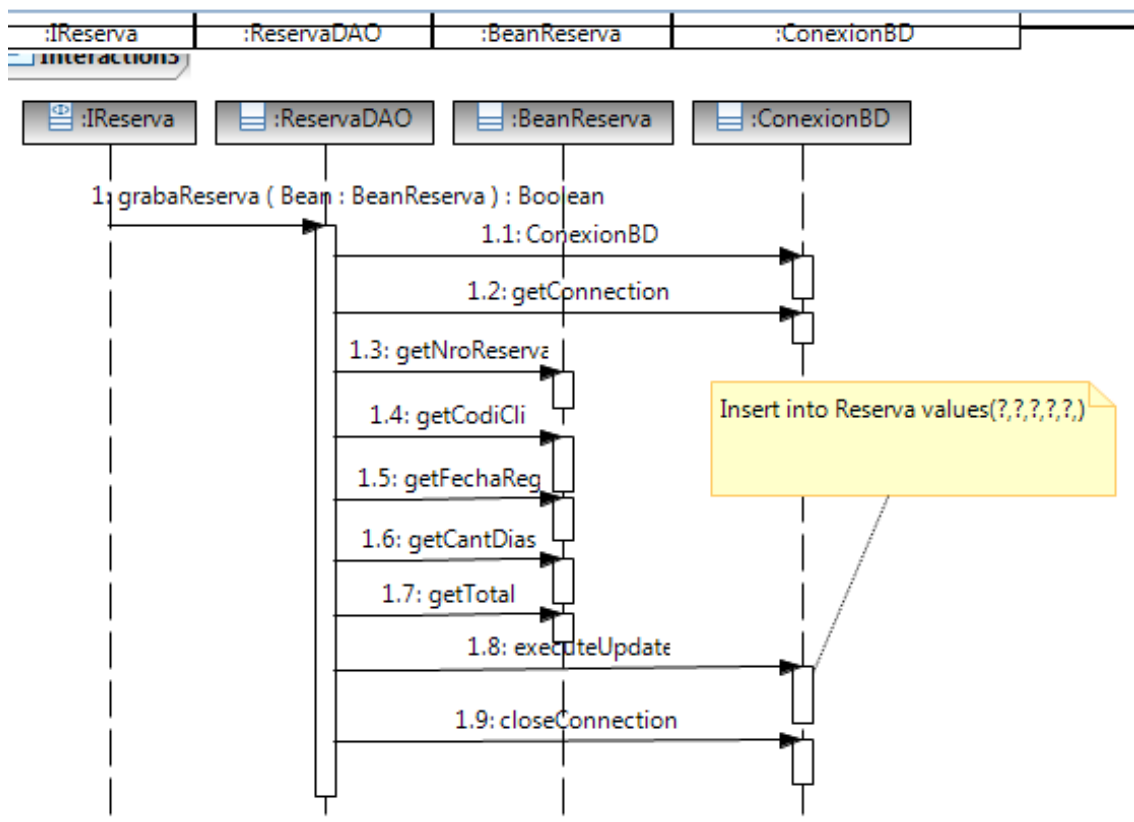
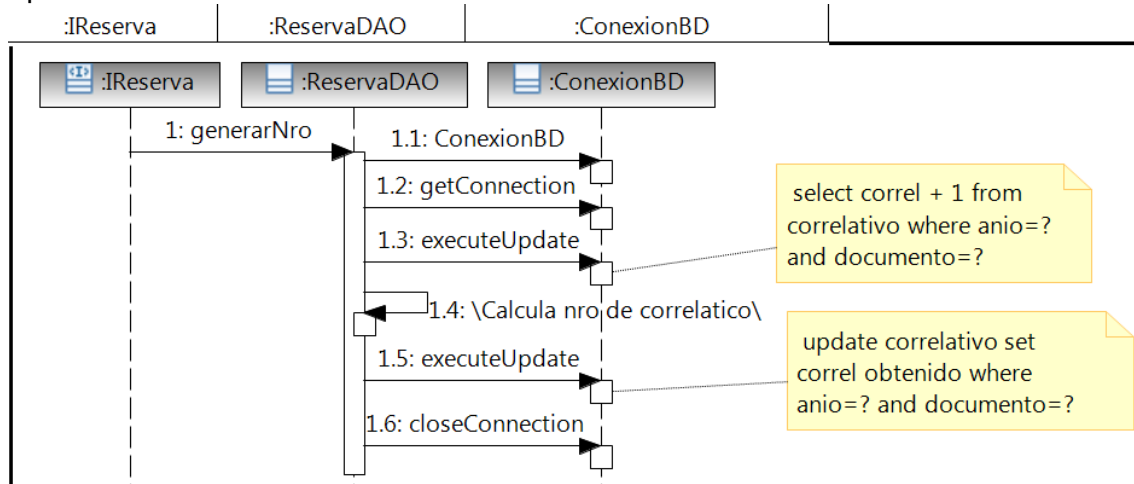
 

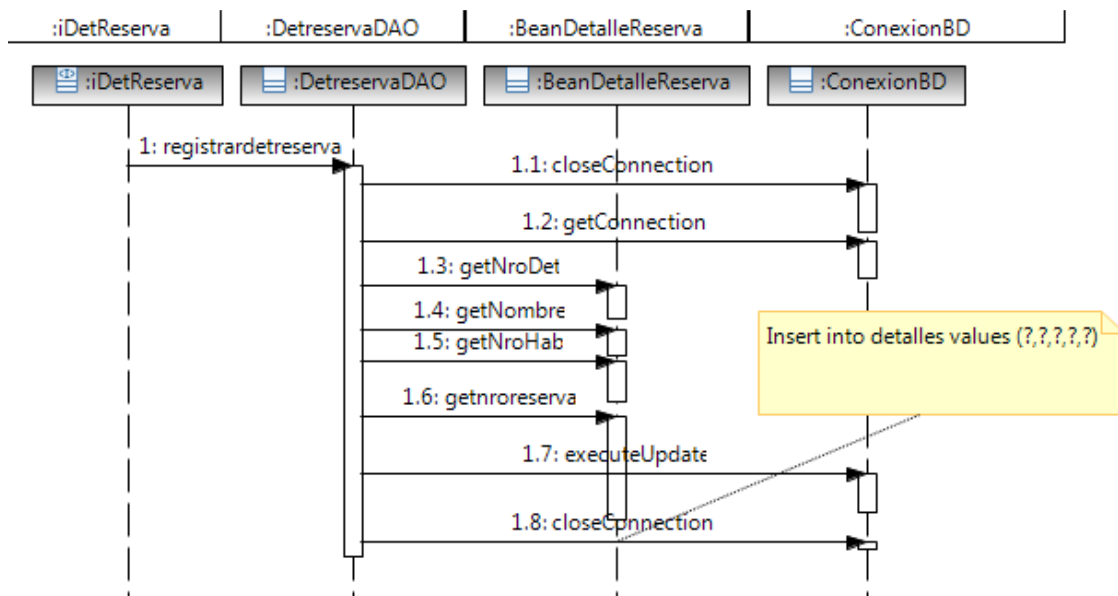
Solución flujo básico:

acción



Operación Correlativo





Resumen

El modelado de aplicaciones web es muy similar al de cualquier otro sistema, solo es cuestión de realizar algunas extensiones a UML para representar ciertas características propias de estas aplicaciones, para lo cual debemos de centrarnos en los detalles de la funcionalidad a la hora de desarrollar aplicaciones web y no tanto en la presentación, ya que si se considera el caso opuesto se pueden crear sitios que pudieran ser muy atractivos en un principio, pero con el tiempo dejan de ser funcionales.

Los patrones de diseño son soluciones recurrentes a problemas dentro de un contexto. Describen con algún nivel de abstracción y soluciones expertas a un problema.

Los patrones de diseño son útiles porque:

- Contribuyen a reutilizar el diseño, permitiendo ser aplicado en una gran cantidad de situaciones.
- Mejoran la flexibilidad, modularidad y extensibilidad.
- Aumentan la escalabilidad del sistema.

Los patrones de diseño GOF son utilizados en aplicaciones orientadas a objetos. Los autores de este patrón recopilaron 23 patrones que son utilizados por muchos diseñadores de software.

Los patrones de diseño J2EE consisten en soluciones recurrentes y documentadas de problemas comunes de diseño de aplicaciones J2EE. Muchas de estas soluciones se basan en los patrones GOF.

Además, puede consultar las siguientes páginas.

http://www.drdobbs.com/article/printableArticle.jhtml?articleID=184414696&dept_url=/architect/

En esta dirección, se describe los elementos de una aplicación web utilizando WAE.

<http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php>

Aquí se presenta una descripción breve sobre patrones de diseño GOF.

<http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html>

En este enlace, encontrará la descripción completa de los patrones J2EE.

<http://www.programacion.com/java/tutorial/patrones2/8/>

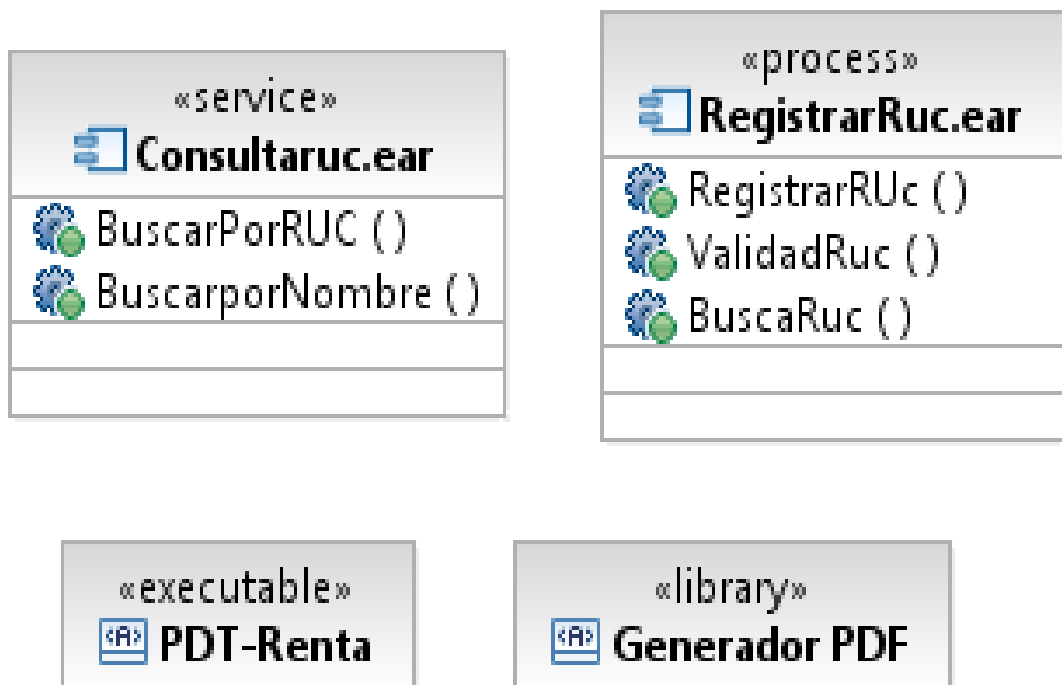
En este enlace, encontrará un ejemplo de aplicación del patrón *DAO* del patrón J2EE con la estrategia *Factory* del patrón GOF.

4. MODELO DE IMPLEMENTACION

El Modelo de Implementación del sistema. Muestra cómo se traduce el Modelo de Diseño en los distintos componentes ejecutables de la aplicación a desarrollar. Este modelo se representa con el diagrama de Componentes y diagrama de despliegue

4.1 Componentes

- Un componente representa una pieza del software reutilizable. que conforma un conjunto de interfaces y proporciona su implementación.
- Modela artefactos tales como ejecutables, bibliotecas, tablas, ficheros, documentos,...
- Representa el empaquetamiento físico de elementos lógicos tales como clases, interfaces,...
- Residirán en los nodos del sistema

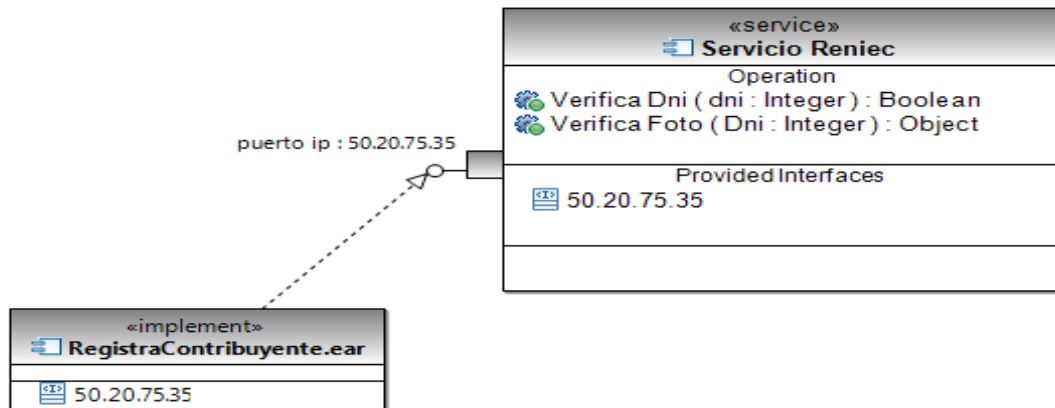


Tipos de componentes

- Despliegue
 - Necesarios y suficientes para formar un sistema ejecutable: librerías dinámicas(dll), ejecutables (exe, ear, jar)
- Productos del trabajo
 - Permanecen al final del proceso de desarrollo: archivos código fuentes, ficheros de datos,...
 - Con ellos, se crean los componentes de despliegue
- De ejecución
 - Se crean durante la ejecución: objeto COM, instanciado a partir de una dll.
 - Servicios web, al momento de instalar un ear.

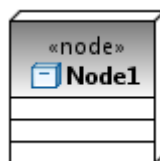
4.2 Diagrama Componentes

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes. Los diagramas de Componentes prevalecen en el campo de la arquitectura de software, pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema.



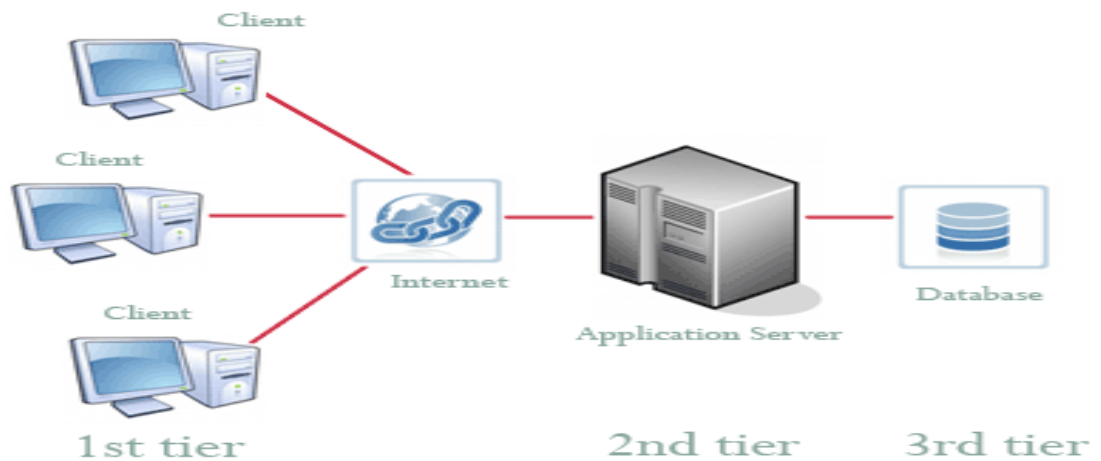
4.3 Nodo

- Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional que puede tener memoria y capacidad de procesamiento.
- En redes de computadoras cada una de las máquinas es un nodo, y si la red es Internet, cada servidor constituye también un nodo.
- Los componentes se ejecutan en nodos.
- Los nodos representan el despliegue físico de los componentes.



4.4 Servidor de Aplicaciones

- Es un recurso computacional dedicada a la ejecución eficaz de los procedimientos (programas, rutinas, secuencias de comandos) para apoyar la construcción de aplicaciones
- Se denomina servidor de aplicaciones a un servidor en una red de computadores que ejecuta ciertas aplicaciones



4.5 Cluster

El término **cluster** se aplica a los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora.

De un cluster se espera que presente combinaciones de los siguientes servicios:

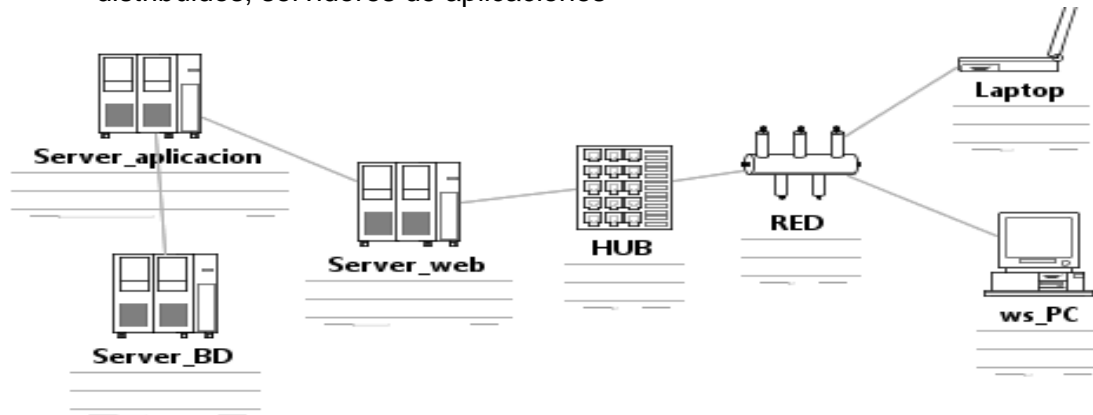
- Alto rendimiento
- Alta eficiencia
- Alta disponibilidad



- Alto rendimiento: Son clusters en los cuales se ejecutan tareas que requieren de gran capacidad computacional. El llevar a cabo estas tareas puede comprometer los recursos del cluster por largos periodos de tiempo.
- Alta eficiencia: Son clusters cuyo objetivo de diseño es el ejecutar la mayor cantidad de tareas en el menor tiempo posible.
- Alta disponibilidad: Son clusters cuyo objetivo de diseño es el de proveer disponibilidad y confiabilidad. Estos clusters tratan de brindar la máxima disponibilidad de los servicios que ofrecen. La confiabilidad se provee mediante software que detecta fallos y permite recuperarse frente a los mismos, mientras que en hardware se evita tener un único punto de fallos.

4.6 Diagrama de despliegue

- Muestra la configuración de los nodos que participan en la ejecución y de los componentes que residen en los nodos.
- Incluye nodos y arcos que representan conexiones físicas entre nodos.
- Modelado de sistemas empotrados, sistemas cliente-servidor, sistemas distribuidos, servidores de aplicaciones



4.7 Aplicaciones desatendidas

Algunas aplicaciones necesitan que ciertos procesos se disparen automáticamente, sin intervención del usuario; en éstos casos, precisamos una herramienta que nos permita planificar su ejecución.

- En Windows se llama tarea programada
- En Unix se llama Cron
- En Java se llama scheduler

Glosario

Abstracción

Características esenciales de una entidad que la distingue de otros tipos de entidades. Define una frontera desde la perspectiva del observador.

API

Una API representa una interfaz de comunicación entre componentes de software. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente), entre los niveles o capas inferiores y los superiores del software.

Blueprints

En el contexto de arquitectura de sistemas de *software*, este término se refiere al conjunto de diagramas que tienen como objetivos representar las diferentes vistas del sistema.

Artefacto

Pieza discreta de información que es utilizada o producida por un proceso de desarrollo de software.

Elemento

Constituyente atómico de un modelo.

Especificación

Descripción textual de la sintaxis y la semántica de un bloque de construcción específico; descripción declarativa de lo que algo es o hace.

Estereotipo

Extensión del vocabulario de UML que permite crear nuevos bloques de construcción derivados a partir de los existentes, pero específicos a un problema concreto.

Estilo arquitectónico

Los estilos arquitectónicos son una generalización y abstracción de los patrones de diseño. También, puede definirse como la descripción de los tipos componente y de los patrones de interacción entre ellos.

Framework

En el desarrollo de software, es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

Heurística

Capacidad de un sistema para realizar de forma inmediata innovaciones positivas para sus fines. La capacidad heurística es un rasgo característico de los humanos, desde cuyo punto de vista puede describirse como el arte y la ciencia del descubrimiento y de la invención o de resolver problemas mediante la creatividad y el pensamiento lateral o pensamiento divergente.

Ingeniería de Software

Rama de la ingeniería que aplica los principios de la ciencia de la computación y las Matemáticas para lograr soluciones costo-efectivas a los proyectos de desarrollo o mantenimiento de software de calidad.

Notación

Sistema de signos convencionales que se adoptan para expresar un conjunto de conceptos sobre el sistema de software por desarrollar.

OMG Object Management Group

Consortio del cual forman parte las empresas más importantes que se dedican al desarrollo de software.

Refinamiento

Relación que representa una especificación más completa de algo que ya ha sido especificado a cierto nivel de detalle.

Requisito

Característica, propiedad o comportamiento deseado de un sistema.

RUP *Rational Unified Process*

Proceso Unificado de Rational, metodología del proceso de ingeniería de *software* que proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización del desarrollo.

Sistema legado

Aquel que es utilizado en un contexto distinto del que en principio fue concebido. La mayoría de sistemas de información distribuidas se convierten en sistemas legados o heredados. En inglés, es conocido como *Legacy System*.

Stakeholder

Persona, grupo u organización que tenga directa o indirecta participación en una organización, ya que puede afectar o ser afectados por la organización de acciones, objetivos y políticas. Actores claves en una organización de negocios incluyen los acreedores, clientes, directores, empleados, gobierno (y sus organismos), los propietarios (accionistas), los proveedores, los sindicatos y la comunidad en la que se basa el negocio de sus recursos.

UML *Unified Modeling Language*

Lenguaje Unificado de Modelado, notación estándar para el modelado de sistemas *Software*.

Vista

Proyección de un modelo, que se ve desde una perspectiva o un punto de vista dado, y que omite entidades que no son relevantes desde esa perspectiva.