**MATERIA: ANÁLISIS Y DISEÑO DE SISTEMAS II**

**1.- CONTENIDO**

1. INTRODUCCIÓN AL MODELADO ORIENTADO A OBJETOS

1.1. Modelado

1.2. Principios Básicos del Modelado

1.3. Orientación a Objetos

1.4. Ventajas de la Orientación a Objetos

1.5. Conceptos Básicos de la Orientación a Objeto

2. LENGUAJE DE MODELADO UNIFICADO UML

2.1. Elementos Estructurales

2.2. Elementos de Comportamiento

2.3. Elementos de Agrupación

2.4. Elementos de Anotación

2.5. Relaciones

2.6. Diagramas

3. HERRAMIENTAS CASE ORIENTADO A OBJETOS

3.1. Introducción

3.2. Barra de Herramientas

3.3. Herramientas para los Diagramas

3.4. Generación de Código

4. PROCESO DE DESARROLLO

4.1. Visión General de la Metodología

5. FASE DE PLANIFICACIÓN Y ESPECIFICACIÓN DE REQUISITOS

5.1. Actividades

5.2. Requisitos

5.3. Casos de Usos

5.4. Construcción del Modelo de Casos de Uso

5.5. Planificación según Ciclos de Desarrollo

6. FASE DE CONSTRUCCIÓN - DISEÑO DE ALTO NIVEL

6.1. Actividades

6.2. Diagramas de Secuencia del Sistema

6.3. Modelo Conceptual

6.4. Glosario

6.5. Contratos de Operaciones

6.6. Post Condiciones

6.7. Diagramas de Estados

7. FASE DE CONSTRUCCIÓN - DISEÑO DE BAJO NIVEL

7.1. Actividades

7.2. Casos de Usos Reales

7.3. Diagramas de Colaboración

7.4. Diagramas de Clases de Diseño

7.5. Otros Aspectos en el Diseño de Sistemas

8. IMPLEMENTACIÓN Y PRUEBAS

8.1. Codificación en un lenguaje Orientado a Objetos

8.2. Pruebas Alfa

8.3. Pruebas Beta

**2.- BIBLIOGRAFÍA**

Diseño Orientado a Objetos con UML - Raúl Alarcón - Editorial EIDOS – 2000.

• The rational unified process - Philippe Kruchten - Tercera Edición - Editorial Addison-Wesley - 2003.

• The unified process inception phase - Scott W. Ambler, Larry L. Constantine - Editorial Focal Press, 2000.

**1. INTRODUCCIÓN AL MODELADO ORIENTADO A OBJETOS**

**1.1. Modelado**

Un modelo representa a un sistema[[1]](#footnote-2) software desde una perspectiva específica. Al igual que la planta y el alzado de una figura en dibujo técnico nos muestran la misma figura vista desde distintos ángulos, cada modelo nos permite fijarnos en un aspecto distinto del sistema.

La importancia del Modelado

* Modelar es probar, es una técnica bien aprobada en la ingeniería.
* Se construye el modelo de una casa para tener una vista previa del producto final.
* Se construyen modelos matemáticos para analizar los efectos de los vientos o terremotos en las construcciones.

Si construimos ....

* Una casa para el perro.
* Sólo necesitaremos unos cuantos materiales y pocas herramientas.
* Probablemente no necesitemos modelar.
* Un edificio de 100 pisos.
* Necesitaremos muchos materiales, muchas herramientas y ayuda de muchas personas.
* Si necesitamos modelar.

¿Qué es entonces un modelo?

* Un modelo es una simplificación de la realidad.
* Un modelo proporciona un anteproyecto del sistema.
* Es una abstracción de la realidad.
* Es una proyección a micro escala.
* Depende del contexto.

¿Por qué debemos modelar?

Debemos modelar para entender mejor el problema que estamos desarrollando. A través del modelado nosotros alcanzamos cuatro puntos:

1. Ayuda a visualizar como es el sistema.

2. Nos ayuda a especificar la conducta de la estructura del sistema.

3. Da una guía base para la construcción del sistema.

4. Documenta las decisiones que debemos tomar.

**1.2. Principios Básicos del Modelado**

El modelaje tiene una rica historia en todas las disciplinas de ingeniería. Los cuatro principios de modelado son derivados de esa historia.

1. El tipo de modelo que se crea influye la manera en que el problema es atacado.- Ej. Si se nos pide utilizar un lenguaje como Java un modelo recomendado por omisión es el OO.
2. Cada modelo puede ser expresado en diferentes niveles de precisión.- Ej. Si se está construyendo un edificio, necesitamos una distancia de muchos metros para visualizar el objeto. Pero si deseamos ver sólo un departamento del edificio la distancia visual se acorta. De igual manera sucede con el software.
3. Los mejores modelos están conectados con la realidad.- Ej. Un modelo físico que no corresponde de igual manera a la realidad tiene valor limitado. Un modelo de un avión que asume sólo condiciones ideales y perfecta manufactura, puede tener fatales errores en el avión real.
4. Un solo modelo no es suficiente.- Ej. Si construimos un edificio, los planos, especificaciones y maquetas no son un conjunto de modelos aislados. Por ejemplo los planos eléctricos, hidráulicos, etc., en un proyecto de ingeniería civil, no son independientes sino que tienen que estar íntegramente relacionados unos con otros.

**1.3. Orientación a Objetos**

* Una vista contemporánea del modelado de software toma la perspectiva de la orientación a objetos.
* En esta perspectiva el bloque principal de todo el sistema de software es la clase y el objeto.
* Un objeto representa una abstracción del problema a solucionar.
* La clase es la descripción del conjunto de objetos similares.
* Todos los objetos tienen una identidad, un estado y un comportamiento.
* La orientación a objetos es una forma principal del modelado de software.
* La orientación a objetos proporciona los conceptos fundamentales para ensamblar sistemas usando tecnologías como JAVA ó COM+.

**1.4. Ventajas de la Orientación a Objetos**

Vamos a ver las ventajas más importantes de la programación orientada a objetos:

* Reusabilidad. Cuando hemos diseñado adecuadamente las clases, se pueden usar en distintas partes del programa y en numerosos proyectos.
* Mantenibilidad. Debido a las sencillez para abstraer el problema, los programas orientados a objetos son más sencillos de leer y comprender, pues nos permiten ocultar detalles de implementación dejando visibles sólo aquellos detalles más relevantes.
* Modificabilidad. La facilidad de añadir, suprimir o modificar nuevos objetos nos permite hacer modificaciones de una forma muy sencilla.
* Fiabilidad. Al dividir el problema en partes más pequeñas podemos probarlas de manera independiente y aislar mucho más fácilmente los posibles errores que puedan surgir.

La programación orientada a objetos presenta también algunas desventajas como pueden ser:

* Cambio en la forma de pensar de la programación tradicional a la orientada a objetos.
* La ejecución de programas orientados a objetos es más lenta.
* La necesidad de utilizar bibliotecas de clases obliga a su aprendizaje y entrenamiento.

**1.5. Conceptos Básicos de la Orientación a Objeto**

La **programación orientada a objetos** es una forma de programar que trata de encontrar una solución a estos problemas. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:

* **Clase**: definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La **instanciación** es la lectura de estas definiciones y la creación de un objeto a partir de ellas.
* **Herencia**: (por ejemplo, herencia de la clase C a la clase D) es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables publicas declaradas en C. Los componentes registrados como "privados" (private) también se heredan, pero como no pertenecen a la clase, se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos. Esto es así para mantener hegemónico el ideal de POO.
* **Objeto**: Instancia de una clase. Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos. Se corresponden con los **objetos reales del mundo que nos rodea**, o con objetos internos del sistema (del programa). Es una instancia a una clase.
* **Método**: algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.
* **Evento**: es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento la reacción que puede desencadenar un objeto; es decir, la acción que genera.
* **Atributos**: características que tiene la clase.
* **Mensaje**: una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.
* **Propiedad o atributo**: contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.
* **Estado interno**: es una variable que se declara privada, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos). No es visible al programador que maneja una instancia de la clase.

**Componentes de un objeto:** atributos, identidad, relaciones y métodos.

**Identificación de un objeto:** un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

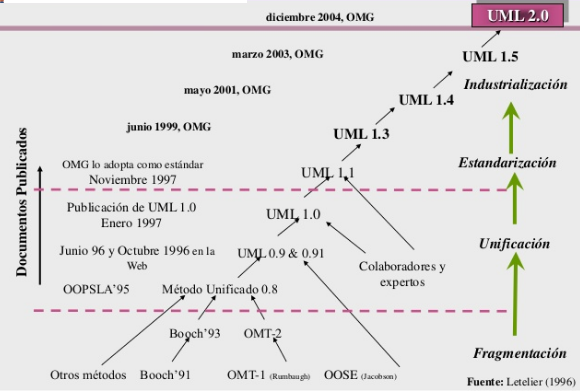
En comparación con un lenguaje imperativo, una "variable" no es más que un contenedor interno del atributo del objeto o de un estado interno, así como la "función" es un procedimiento interno del método del objeto.

**2. LENGUAJE DE MODELADO UNIFICADO UML**

UML (*Unified Modeling Language*) es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Se ha convertido en el estándar de facto de la industria, debido a que ha sido concebido por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh. Estos autores fueron contratados por la empresa Rational Software Co. para crear una notación unificada en la que basar la construcción de sus herramientas CASE. En el proceso de creación de UML han participado, no obstante, otras empresas de gran peso en la industria como Microsoft, Hewlett-Packard, Oracle o IBM, así como grupos de analistas y desarrolladores.

****

***Figura 1. Logo UML.***

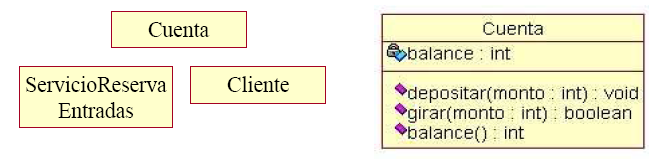
****

***Figura 2. Evolución UML.***

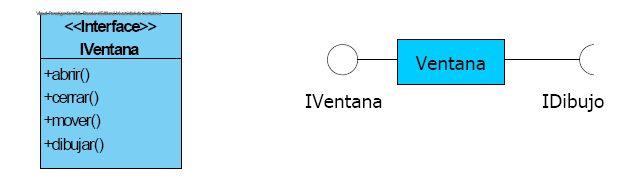
**2.1. Elementos Estructurales**

En UML 2.0 Existen los siguientes elementos estructurales:

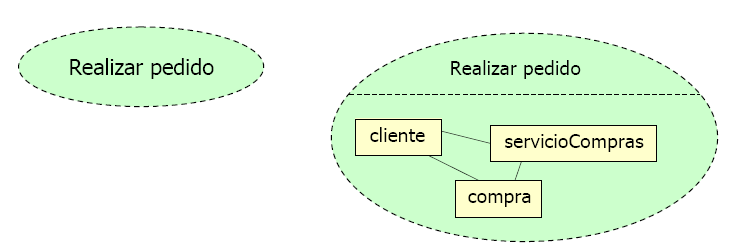
|  |  |  |
| --- | --- | --- |
| **TIPO** | **ELEMENTO** | **DESCRIPCION** |
| **LOGICO/ CONCEPTUAL** | **CLASE** | Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. |
| **INTERFAZ** | Colección de operaciones que especifican un servicio que puede ser ofrecido por una clase o componente. Define un conjunto de especificaciones de operaciones (signaturas), pero no las implementaciones de dichas operaciones. |
| **COLABORACION** | Define una interacción entre una sociedad de roles y otros elementos que colaboran para proporcionar un comportamiento mayor que la  suma de sus comportamientos aislados. |
| **CASO DE USO** | Describe un comportamiento de un sistema, clase o componente, desde el punto de vista del usuario. |
| **OBJETO** | Un objeto representa una instancia de una clase en un determinado contexto |
| **CLASE ACTIVA** | Tipo especial de clase cuyos objetos tienen uno o más procesos o hilos de ejecución => |
| **COMPONENTE** | Parte modular de la arquitectura física de un sistema que oculta su implementación tras un conjunto de interfaces externas. |
| **FISICO** | **ARTEFACTO** | Parte física y reemplazable de un sistema que contiene información física (bits) |
| **NODO** | Elemento físico que existe en tiempo de ejecución y representa un recurso computacional. |

****

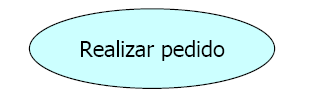
**Figura 3. Clases (Representan Hardware, Software, Personas, Conceptos, ..etc.).**

****

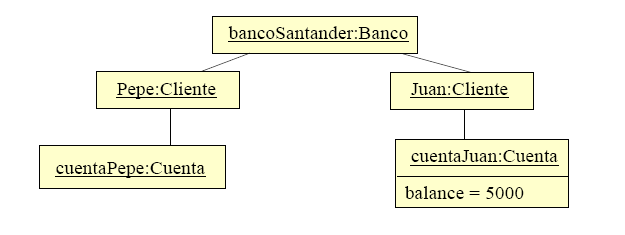
**Figura 4. Interface.**

****

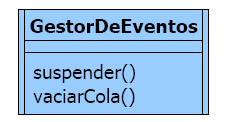
**Figura 5. Colaboración.**

****

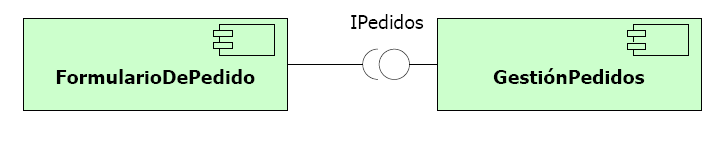
**Figura 6. Caso de Uso.**

****

**Figura 7. Objeto.**

****

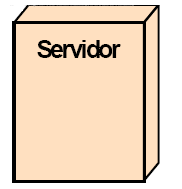
**Figura 8. Clase Activa.**

****

**Figura 9. Componente.**

****

**Figura 10. Artefacto.**

****

**Figura 11. Nodo.**

**2.2. Elementos de Comportamiento**

Son las partes dinámicas de los modelos UML.

|  |  |
| --- | --- |
| **ELEMENTO** | **DESCRIPCION** |
| **ITERACCIONES** | Comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos, dentro de un contexto particular, para un propósito específicos como: mensajes, acciones y enlaces  . |
| **MAQUINA DE ESTADOS** | Comportamiento que especifica las secuencias de estados por las que pasa un objeto o una interacción durante su vida, en respuesta a eventos, junto con sus reacciones a dichos eventos. Involucran a: Estados, Transiciones (flujo de un estado a otro), Eventos (disparan una transición), Actividades (Respuesta a una transición) . |
| **ACTIVIDAD** | Comportamiento que especifica la secuencia de pasos que ejecuta un proceso computacional. Una acción es un paso de una actividad. |

**2.3. Elementos de Agrupación**

Son las partes organizativas de los modelos UML. Son las “cajas” en las que puede dividirse un modelo. Hay un tipo principal: "Paquete" este es el mecanismo de propósito general para organizar el propio diseño.

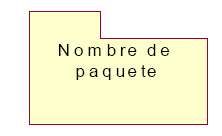


Figura 12 Paquete.

**2.4. Elementos de Anotación**

Son las partes explicativas (usado para comentarios, restricciones, clarificar observaciones) de los modelos UML.

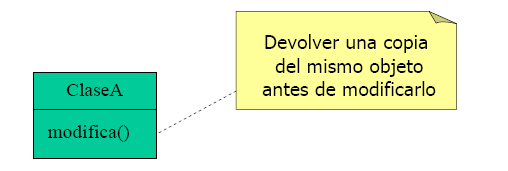
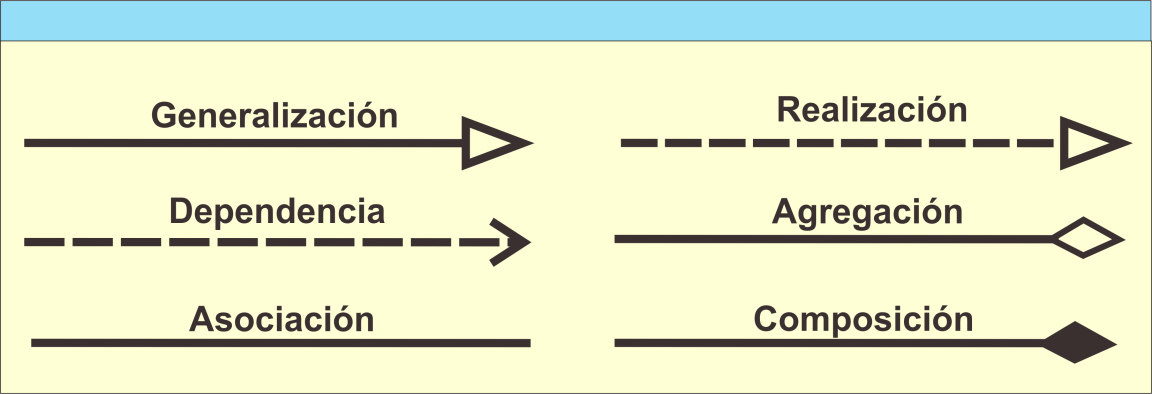


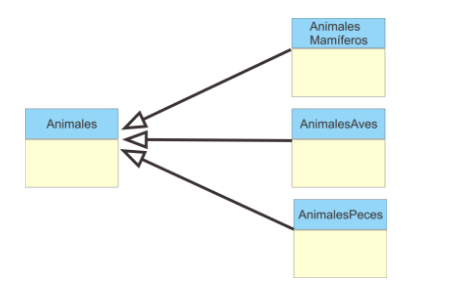
Figura 13 Anotaciones UML.

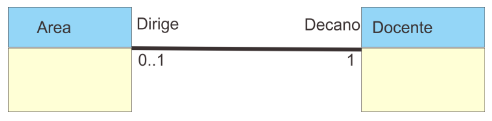
**2.5. Relaciones**

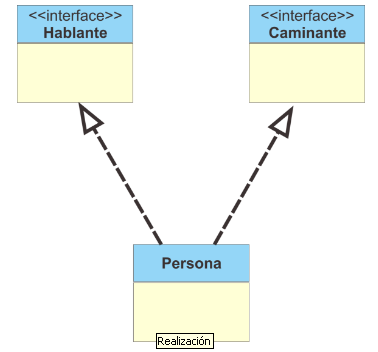
Una relación es una conexión entre elementos estructurales. Existen los siguientes tipos de relaciones en UML2:

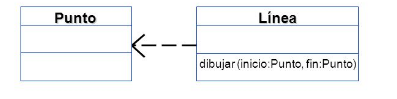
|  |  |
| --- | --- |
| **ELEMENTO** | **DESCRIPCION** |
| **GENERACIONALES** | Esta hace referencia a la relación de una súper clase o clase padre con una subclase o clase hija. |
| **DEPENDENCIA** | Relación semántica en la cual un cambio a un elemento (independiente) puede afectar a la semántica del otro elemento (dependiente). |
| **ASOCIACION** | Es una relación estructural que especifica que los objetos de un elemento están conectados con los objetos de otro. |
| **REALIZACION** | Es una relación semántica entre clasificadores donde este especifica unas normas o un reglamento con otro clasificador que garantiza que se cumplirá. |
| **AGREGACION** | Relación dinámica: el tiempo de vida del objeto que se agrega es independiente del objeto agregador. El cliente usa tarjeta de crédito. |
| **COMPOSICION** | Relación estática: el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del objeto compuesto. La portátil tiene un teclado. |



****

****

****

****

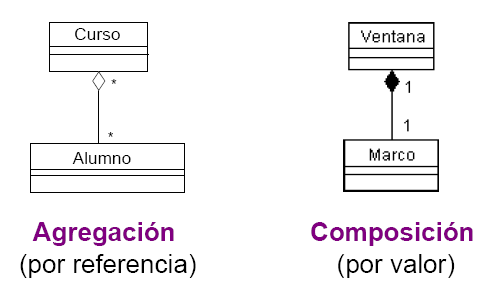
****

Figura 14. Relaciones.

**2.6. Diagramas**

Sirven para visualizar un sistema desde diferentes perspectivas. UML tiene los siguientes tipos de diagramas:

Estructurales (Estáticos):

1. Clases
2. Objetos
3. Componentes
4. Despliegue[[2]](#footnote-3)
5. Paquetes
6. Estructura Compuesta

De Comportamiento (Dinámico):

1. Casos de uso[[3]](#footnote-4)
2. Estados
3. Actividades[[4]](#footnote-5)
4. Interacción
   1. Secuencia
   2. Comunicación
   3. Tiempos[[5]](#footnote-6)
   4. Revisión de Interacciones

|  |  |
| --- | --- |
| **TIPO** | **DIAGRAMA** |
| **UML** |  |
| **REAL** |  |

Figura 15. Diagrama de Despliegue (Cada Nodo debe tener los componentes).

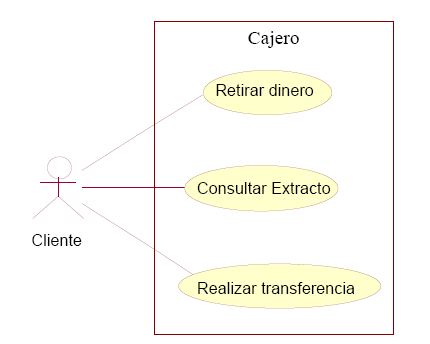


Figura 16. Diagrama de Casos de Uso.

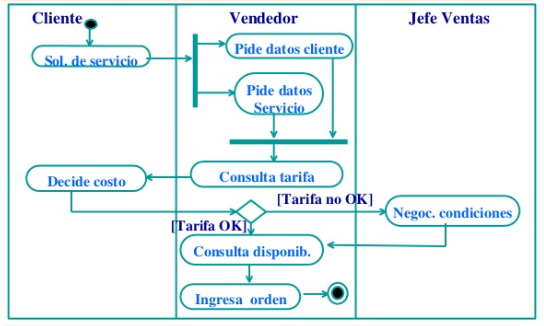


Figura 17. Diagrama de Actividad.

**3. HERRAMIENTAS CASE ORIENTADO A OBJETOS**

**3.1. Introducción**

Herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadoras). Son diversas Aplicaciones informáticas destinadas a aumentar la productividad en el Desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el diseño de proyectos, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, Compilación automática, documentación o detección de errores entre otras.

**3.2. Barra de Herramientas**

Visual Paradigm es una herramienta CASE: Ingeniería de Software Asistida por Computación. La misma propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación.

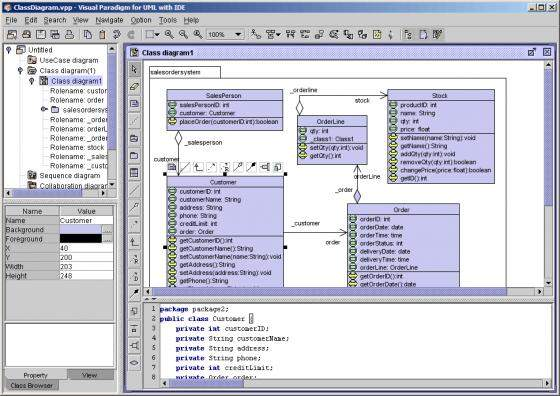
****

Figura 18. Visual Paradigm.

**3.3. Herramientas para los Diagramas**

Otras herramientas y plugins de modelado UML:

* Plataforma Java (Windows/Linux/Mac OS X):
* SDE para Eclipse.
* SDE para NetBeans.
* SDE para Sun ONE.
* SDE para Oracle JDeveloper.
* SDE para JBuilder.
* SDE para IntelliJ IDEA.
* SDE para WebLogic Workshop.
* Plataforma Windows:
* SDE para Microsoft Visual Studio

**3.4. Generación de Código**

La generación automática de código a partir de modelos ha sido una de las promesas parcialmente cumplidas en el desarrollo de software. La experiencia de las herramientas CASE, aún distante del automatismo absoluto, se complementa con algunos trabajos teóricos que se alejan de los estándares de modelamiento. En este artículo se proponen reglas para la generación de código a partir de metamodelos de diagramas de clases, secuencias y máquina de estados de UML. Las reglas están definidas en lógica de primer orden, permitiendo una especificación donde se evitan las ambigüedades y la necesidad de aprender un lenguaje de programación específico.

**4. PROCESO DE DESARROLLO**

**4.1. Visión General de la Metodología**

El proceso a seguir para realizar desarrollo orientado a objetos es complejo, debido a la complejidad que nos vamos a encontrar al intentar desarrollar cualquier sistema software de tamaño medio-alto. El proceso está formado por las fases: planificación, especificación de requisitos, construcción e implementación y pruebas que a su vez contienen una serie de actividades y sub actividades, cuya realización se va repitiendo en el tiempo aplicadas a distintos elementos.

**5. FASE DE PLANIFICACIÓN Y ESPECIFICACIÓN DE REQUISITOS**

Esta fase se corresponde con la Especificación de Requisitos tradicional ampliada con un Borrador de Modelo Conceptual y con una definición de Casos de Uso de alto nivel. En esta

fase se decidiría si se aborda la construcción del sistema mediante desarrollo orientado a objetos o no, por lo que, en principio, es independiente del paradigma empleado posteriormente.

**5.1. Actividades**

Las actividades de esta fase son las siguientes:

1. Definir el Plan-Borrador.

2. Crear el Informe de Investigación Preliminar.

3. Definir los Requisitos.

4. Registrar Términos en el Glosario. (*continuado en posteriores fases*)

5. Implementar un Prototipo. (*opcional*)

6. Definir Casos de Uso (de alto nivel y esenciales).

7. Definir el Modelo Conceptual-Borrador. (*puede retrasarse hasta una fase posterior*)

8. Definir la Arquitectura del Sistema-Borrador. (*puede retrasarse hasta una fase posterior*)

9. Refinar el Plan.

**5.2. Requisitos**

Un requisito es una descripción de necesidades o aspiraciones respecto a un producto. El objetivo principal de la actividad de definición de requisitos consiste en identificar qué es lo que realmente se necesita, separar el grano de la paja. Esto se hace en un modo que sirva de comunicación entre el cliente y el equipo de desarrollo.

Es aconsejable que un documento de Especificación de Requisitos tenga los siguientes puntos:

- Propósito.

- Ámbito del Sistema, Usuarios.

- Funciones del Sistema.

- Atributos del Sistema.

El formato del documento de Especificación de Requisitos no está definido en UML, pero se ha incluido este punto para resaltar que la actividad de definición de requisitos es un paso clave en la creación de cualquier producto software.

Para refinar los requisitos y mejorar la comprensión de los mismos la técnica de casos de uso constituye una valiosa ayuda.

**5.3. Casos de Usos**

Un Caso de Uso es un documento narrativo que describe la secuencia de eventos de un actor (un agente externo) que usa un sistema para completar un proceso. Es una historia o una forma particular de usar un sistema. Los casos de uso no son exactamente requisitos ni especificaciones funcionales, pero ilustran e implican requisitos en las historias que cuentan.

**5.4. Construcción del Modelo de Casos de Uso**

Para construir el Modelo de Casos de Uso en la fase de Planificación y Especificación de

Requisitos se siguen los siguientes pasos:

1. Después de listar las funciones del sistema, se definen los límites del sistema y se identifican los actores y los casos de uso.

2. Se escriben todos los casos de uso en el formato de *alto nivel*. Se categorizan como primarios, secundarios u opcionales.

3. Se dibuja el Diagrama de Casos de Uso.

4. Se relacionan los casos de uso y se ilustran las relaciones en el Diagrama de Casos de Uso (<<extiende>> y <<usa>>).

5. Los casos de uso más críticos, importantes y que conllevan un mayor riesgo, se describen en el formato expandido esencial. Se deja la definición en formato expandido esencial del resto de casos de uso para cuando sean tratados en posteriores ciclos de desarrollo, para no tratar toda la complejidad del problema de una sola vez.

6. Se crean casos de uso reales sólo cuando:

·Descripciones más detalladas ayudan significativamente a incrementar la comprensión

del problema.

· El cliente pide que los procesos se describan de esta forma.

7. Ordenar según prioridad los casos de uso (este paso se va a ver a continuación).

**5.5. Planificación según Ciclos de Desarrollo**

La decisión de qué partes del sistema abordar en cada ciclo de desarrollo se va a tomar

basándose en los casos de uso. Para tomar la decisión de qué casos de uso se van a tratar primero es necesario ordenarlos según prioridad. Las características de un caso de uso específico que van a desarrollar es que el caso de uso tenga una prioridad (primaria, secundara y opcional) más alta.

**6. FASE DE CONSTRUCCIÓN - DISEÑO DE ALTO NIVEL (ANALISIS)**

En la fase de Análisis de un ciclo de desarrollo se *investiga* sobre el problema (se trabaja con los modelos construidos en la fase anterior), sobre los conceptos relacionados con el subconjunto de casos de uso que se esté tratando. Se intenta llegar a una buena comprensión del problema por parte del equipo de desarrollo, sin entrar en cómo va a ser la solución en cuanto a detalles de implementación.

**6.1. Actividades**

Las actividades de la fase de Análisis son las siguientes:

1. Definir Casos de Uso Esenciales en formato expandido. (*si no están definidos* )

2. Refinar los Diagramas de Casos de Uso.

3. Refinar el Modelo Conceptual.

4. Refinar el Glosario. (*continuado en posteriores fases*)

5. Definir los Diagramas de Secuencia del Sistema.

6. Definir Contratos de Operación.

7. Definir Diagramas de Estados. (*opcional*)

**6.2. Diagramas de Secuencia del Sistema**

Además de investigar sobre los conceptos del sistema y su estructura, también es preciso investigar en el Análisis sobre el comportamiento del sistema, visto éste como una caja negra. Una parte de la descripción del comportamiento del sistema se realiza mediante los Diagramas de Secuencia del Sistema.

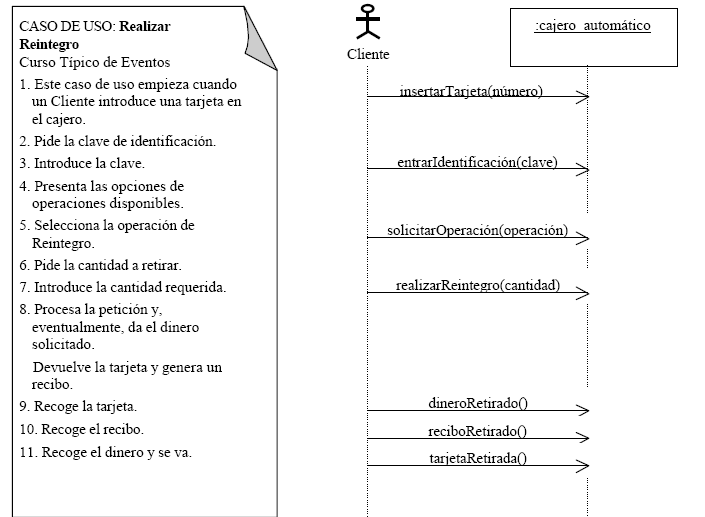


Figura 19. Diagrama de Secuencia.

**6.3. Modelo Conceptual**

Una parte de la investigación sobre el dominio del problema consiste en identificar los conceptos que lo conforman. Para representar estos conceptos se va a usar un Diagrama de

Estructura Estática de UML, al que se va a llamar Modelo Conceptual. Se identifican los objetos, asociaciones y atributos.

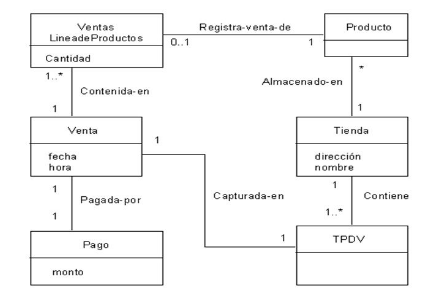
****

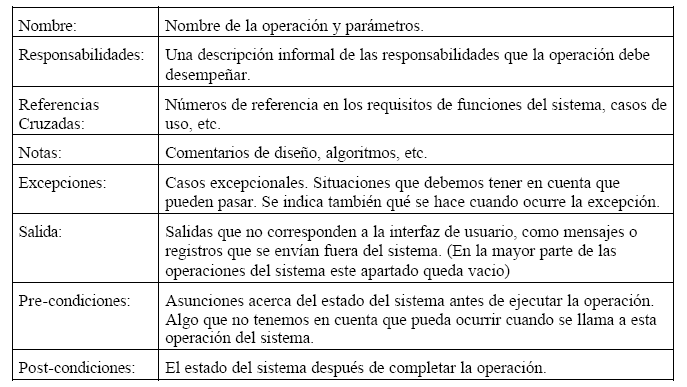
Figura 20. Modelo Conceptual.

**6.4. Glosario**

En el glosario debe aparecer una descripción textual de cualquier elemento de cualquier modelo, para eliminar toda posible ambigüedad.

**6.5. Contratos de Operaciones**

Un Contrato es un documento que describe qué es lo que se espera de una operación. Tiene una redacción en estilo declarativo, enfatizando en el *qué* más que en el *cómo*. Lo más común es expresar los contratos en forma de pre- y post-condiciones en torno a cambios de estado. Una vez se tienen las Operaciones del Sistema identificadas en los Diagramas de Secuencia, se describe mediante el siguiente formato los contratos:



**Ejemplo de Contrato:**

**Nombre:** insertarTarjeta (número\_tarjeta: número)

**Responsabilidades:** Comenzar una sesión con el sistema para realizar una operación. Presentar las opciones disponibles.

Referencias Cruzadas: Funciones del Sistema: R1.2, R1.6, R1.7 Casos de Uso: Reintegro

Notas:

Excepciones: Si la tarjeta es ilegible, indicar que ha habido un error.

Salida:

Pre-condiciones: No hay una sesión activa.

Post-condiciones: · Una nueva *Sesión* se ha creado. (*creación de instancia*).

· La *Sesión* se ha asociado con *Cajero*. (*asociación formada*).

**6.6. Post Condiciones**

Las post-condiciones se basan en el Modelo Conceptual, en los cambios que sufren los

elementos del mismo una vez se ha realizado la operación. Por ejemplo es mejor decir “se ha creado una *Sesión*” que decir “crear una *Sesión*”.

**6.7. Diagramas de Estados**

Para modelar el comportamiento del sistema pueden usarse los Diagramas de Estados. La utilidad de un Diagrama de Estados en el análisis reside en mostrar la secuencia permitida de eventos externos que pueden ser reconocidos y tratados por el sistema Ej.:

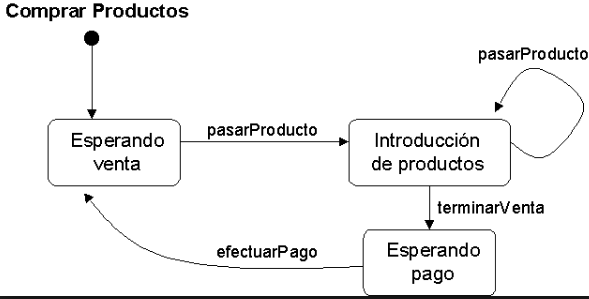
****

Figura 21. Diagrama de Estados.

**7. FASE DE CONSTRUCCIÓN - DISEÑO DE BAJO NIVEL**

En la fase de Diseño se crea una solución a nivel lógico para satisfacer los requisitos, basándose en el conocimiento reunido en la fase de Análisis.

**7.1. Actividades**

Las actividades que se realizan en la etapa de Diseño son las siguientes:

1. Definir los Casos de Uso Reales.

2. Definir Informes e Interfaz de Usuario.

3. Refinar la Arquitectura del Sistema.

4. Definir los Diagramas de Interacción.

5. Definir el Diagrama de Clases de Diseño. (*en paralelo con los Diagramas de Interacción*)

5. Definir la Base de Datos

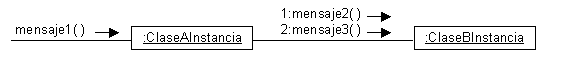
**7.2. Casos de Usos Reales**

Un Caso de Uso Real describe el diseño real del caso de uso según una tecnología concreta de entrada y de salida y su implementación. Si el caso de uso implica una interfaz de usuario, el caso de uso real incluirá bocetos de las ventanas y detalles de la interacción a bajo nivel con los *widgets* (botón, lista seleccionable, campo editable, etc.) de la ventana.

**7.3. Diagramas de Colaboración**

Los Diagramas de Interacción muestran el intercambio de mensajes entre instancias del modelo de clases para cumplir las post-condiciones establecidas en un contrato (en un determinado orden).

Por ejemplo:



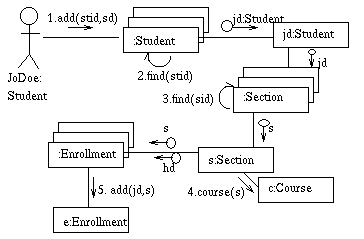


Figura 22. Diagrama de Colaboración.

**7.4. Diagramas de Clases de Diseño**

Al construir los Diagramas de Colaboración se van usando clases procedentes del Modelo Conceptual, junto con otras creadas para encargarse de responsabilidades específicas. El conjunto de todas las clases usadas, junto con sus relaciones, forma el Diagrama de Clases de Diseño. Un Diagrama de Clases de Diseño muestra la especificación para las clases software de una aplicación.

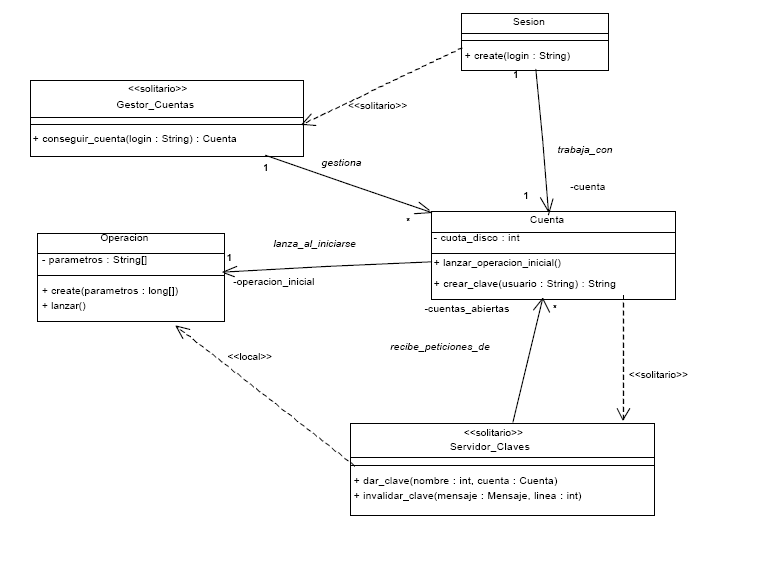


Figura 23. Diagrama de Clases.

**7.5. Otros Aspectos en el Diseño de Sistemas**

En el diseño de un sistema es necesario también tomar decisiones a un nivel más alto sobre la descomposición de un sistema en subsistemas y sobre la arquitectura del sistema. Sí hay que tener en cuenta que las posibles divisiones en subsistemas tienen que hacerse en base a las clases definidas en el Diagrama de Clases del Diseño.

**8. IMPLEMENTACIÓN Y PRUEBAS**

Una vez se tiene completo el Diagrama de Clases de Diseño, se pasa a la implementación en el lenguaje de programación elegido.

**8.1. Codificación en un lenguaje Orientado a Objetos**

La codificación es el elemento fundamental de la programación orientada a objetos OOP es, como su nombre lo indica, el objeto. Podemos definir un objeto como un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización.

**8.2. Pruebas Alfa**

Pruebas alfa son pruebas simuladas u operacionales realizadas por usuarios/clientes potenciales o por un equipo de pruebas independiente en las dependencias de desarrollo, pero fuera de la organización de desarrollo. Las pruebas alfa son utilizadas con frecuencia para software de distribución masiva como una forma de pruebas de aceptación internas.

**8.3. Pruebas Beta**

Pruebas beta son pruebas operacionales realizadas por usuarios/clientes potenciales y/o existentes, en un sitio externo no relacionado de ninguna manera con los desarrolladores, para determinar si un componente o sistema satisface o no las necesidades del usuario/cliente y se ajusta a los procesos de negocio. Con frecuencia las pruebas beta se emplean como una forma de prueba de aceptación externa para software de distribución masiva con el objetivo de obtener la respuesta del mercado.

1. Sistema ,es un conjunto organizado de cosas o partes interactuantes e interdependientes, que se relacionan formando un todo unitario y complejo. [↑](#footnote-ref-2)
2. Muestran un conjunto de nodos y sus relaciones. [↑](#footnote-ref-3)
3. Casos de Uso es una técnica para capturar información respecto de los servicios que un sistema proporciona a su entorno (captura y especificación de requisitos). [↑](#footnote-ref-4)
4. Las actividades se enlazan por transiciones automáticas. [↑](#footnote-ref-5)
5. Muestran los tiempos reales en la interacción entre diferentes objetos o roles. [↑](#footnote-ref-6)