

Análisis y diseño orientado a
objetos

TEMA 56

ABACUS NT

Índice

- 1. Introducción.**
- 2. Programación orientada a objetos**
 - 2.1. Clases y objetos**
 - 2.2. Miembros de una clase (encapsulación)**
 - 2.3. Características**
- 3. Ciclo de Vida del Software en la POO**
- 4. Análisis orientado a objetos**
 - 4.1. Identificación de las necesidades**
 - 4.2. Ámbito funcional y recursos**
 - 4.3. Requisitos de implantación**
 - 4.4. Diagramas de casos de uso**
 - 4.5. Escenarios**
 - 4.6. Prototipado**
- 5. Diseño orientado a objetos**
 - 5.1. Diagramas de Clase**
 - 5.2. Diagramas de Componente**
 - 5.3. Diagramas de Despliegue**
 - 5.4. Diagramas de Paquete**
 - 5.5. Diagramas de Actividad**
 - 5.6. Diagramas de Maquina de Estado**
 - 5.7. Diagramas de Secuencia**
 - 5.8. Diagramas de Comunicación**
 - 5.9. Diagramas de Resumen de Interacción**
 - 5.10. Diagramas de Tiempo**
- 6. Conclusión.**
 - 6.1. Relación con el sistema educativo**
- 7. Bibliografía**

1. Introducción.

La Programación Orientada a Objetos (en adelante POO) se ha convertido en **el nuevo paradigma** de la programación. De hecho, no hay libro de POO que no utilice la palabra “paradigma” para referirse a la orientación a objetos. Algunos autores, no sin razón afirman que es más fácil aprender a programar orientación a objetos desde cero que reconvertir a un programador que no haya programado nunca POO.

Paradigma del griego **παράδειγμα** (traducido como “*ejemplo*”), se refiere a una teoría o conjunto de teorías que sirve de modelo a seguir para resolver problemas o situaciones determinadas que se planteen.

En la programación estructurada -un paradigma clásico, **los datos y las operaciones permanecen claramente separados**. Los **módulos** de nuestros programas se iban pasando de unos a otros los datos que necesitaban a través de **parámetros**.

En la POO tanto los **datos como el código se agrupan formando un módulo que denominamos clase**. Esta abstracción es más **parecida al mundo real**, en el que las “cosas” son objetos con los que interactuamos y **no módulos parametrizados**.

El **análisis y diseño orientado a objetos** (ADOO) es un enfoque de análisis en ingeniería de software que modela un sistema como un grupo de objetos que interactúan entre sí. Todo sistema de información requiere de **artefactos o componentes (clases)** para llevar a cabo tareas.

El **lenguaje unificado de modelado** (UML) se ha convertido en el lenguaje de modelado estándar usado en análisis y diseño orientado a objetos.

ADOO **no está restringido al diseño de programas** de computadora, sino que cubre sistemas enteros de distinto tipo. Las metodologías de análisis y diseño más modernas son “casos de uso” guiados a través de requerimientos, diseño, implementación, pruebas, y despliegue.

2. Programación orientada a objetos

Según el autor **J.Glenn Brookshear**, en su libro “Introducción a la computación (2012)”, en este paradigma el sistema software se ve conceptualmente como un *conjunto de unidades, denominadas objetos, los cuales son capaces de llevar a cabo las acciones que le afectan directamente, así como de solicitar acciones a otros objetos*.

La programación orientada a objetos es un paradigma de programación que consiste en **modelar la realidad como un conjunto de objetos que interactúan** entre sí para resolver un problema.

2.1. Clases y objetos

Una clase, es un modelo a partir del cual se pueden construir tantos elementos como se quiera; a esos elementos los llamamos objetos de la clase. Las clases, por tanto, no son utilizables directamente, sólo sirven para crear instancias u objetos de la clase.

Dos instancias (objetos) diferentes de una misma clase comparten el mismo conjunto de procedimientos y la misma lista de datos, pero con valores diferentes. La **instanciación** es el proceso de creación de un objeto a partir de una clase.

Una clase se caracteriza por:

- Un **identificador**
- Unos **componentes**
- Definición del **nivel de acceso** de cada componente.
- **Relación de la clase con otras clases** existentes

2.2. Miembros de una clase (encapsulación)

Los **miembros de una clase** o componentes que conforman la clase son: los atributos, los métodos y los objetos:

1. **Objetos**: Un **objeto** es una **instancia de una clase**. Se dice que un objeto es **miembro de una clase** cuando es una instancia de la misma.
2. **Atributos**: Determina los tipos de datos que se van a guardar de cada objeto, así los objetos se diferenciarán unos de otros en función del valor que tenga para dichos datos. En el ejemplo anterior, base y altura serían los atributos de la clase.
3. **Métodos**: Son los **procedimientos de la clase** que pueden ser llamados por los objetos para realizar operaciones con sus datos. Los tipos de procedimientos que podemos encontrar son:
 - **Observadores**: Sirven para obtener información del objeto.
 - **Modificadores**: Son utilizados para modificar datos de los objetos.
 - **Constructores**: Son utilizados para asignar valores a los datos de un objeto en el momento de su creación.
 - **Destructores**: Son ejecutados cuando un objeto va a ser eliminado.

2.3. Características

Las **características básicas comunes** que presentan los lenguajes orientados a objetos son:

- **Encapsulamiento**. Las propiedades y el comportamiento de los objetos están definidos en las clases, formadas por estructuras de datos y algoritmos denominadas atributos y métodos.

Un objeto es una instancia de una determinada clase y tiene su propio conjunto de datos. Los métodos se aplican a un objeto concreto y son propias de la clase a la que pertenece.

- **Ocultación.** Para aumentar la modularidad y como medida de protección, un objeto se considera como una caja negra en la que se puede introducir o de la que se puede extraer información sin necesidad de conocer su funcionamiento interno.
- **Envío de mensajes.** Consiste en nombrar y activar con los parámetros adecuados uno de los métodos del objeto al que se envía el mensaje. Es la manera que tienen los objetos de relacionarse, y es equivalente a las llamadas a procedimientos de los lenguajes imperativos.
- **Herencia.** Es un mecanismo mediante el cual pueden crearse clases a partir de otras, transmitiendo todos los atributos y todos los métodos de clases padres a clases hijas, con la ventaja de que en estas últimas pueden añadirse más atributos y métodos.
- **Polimorfismo.** Es la capacidad que tiene un objeto de una determinada clase de hacerse pasar por un objeto de una clase ancestro. Esta característica, junto a la posibilidad de las clases hijas de redefinir los métodos heredados, permiten la reutilización de código.

3. Ciclo de Vida del Software en la POO

ISO/IEC 12207 (2017) - Information Technology / Software Life Cycle Processes es el estándar para los procesos de ciclo de vida del software de la organización ISO.

La creación de software consta generalmente de una serie de pasos claramente diferenciados:

1. **Análisis:** se recopila documentación, se estudian los **requisitos** del usuario final (qué se va a implementar, cuáles son las necesidades del usuario). Se suele empezar con una descripción en lenguaje natural de lo que quiere el usuario. Al final de esta etapa tendremos una descripción clara y precisa de qué producto vamos a construir, qué funcionalidades aportará y qué comportamiento tendrá.
2. **Diseño:** una vez que sabemos qué debemos hacer, debemos determinar cómo lo haremos. Estudiaremos el problema y lo descompondremos en problemas más pequeños; definiremos la estructura de la solución, identificando los módulos que hay que implementar y sus relaciones; definiremos los algoritmos a emplear y el lenguaje de programación a utilizar, etc.
3. **Codificación:** se escribe el programa fuente en el lenguaje de programación establecido y se genera el código ejecutable.
4. **Pruebas:** se comprueba que el programa no tenga errores, y que se cumplen los criterios de calidad.
5. **Implementación:** En esta fase es en la que se instala el software en los equipos finales.
6. **Mantenimiento:** el programador se asegura de que el programa siga funcionando, y lo va adaptando también a nuevos requisitos que puedan ir surgiendo.

En el ciclo de vida, cada proyecto atraviesa por algún tipo de análisis, diseño e **implantación** (implementación, prueba y mantenimiento). El ciclo de vida de proyecto utilizado, por ejemplo, en su organización, puede diferir en una o en todas las formas siguientes:

- La fase de análisis puede dividirse en una fase previa de estudio y otra de análisis (sobre todo en organizaciones en las cuales no aceptan cualquier proyecto).
- Puede no haber fase de análisis de hardware si se cree que cualquier sistema nuevo pudiera instalarse con los ordenadores existentes sin causar mayor problema operacional.
- Las fases de diseño, puede dividirse en dos: diseño preliminar y de diseño de detalles.
- Diversas fases de prueba pueden juntarse en una sola; de hecho, podrían incluirse total o parcialmente de forma paralela a la codificación.

De aquí que **el ciclo de vida del proyecto en una organización puede tener cinco fases o siete o doce**, pero seguir siendo todavía un ciclo de vida estructurado.

La **programación orientada a objetos** engloba mucho más que la codificación: Es necesario un enfoque orientado a objetos desde el principio que abarque todo el ciclo de vida del proyecto; en el tema que nos ocupa expondremos los puntos 1 y 2: **Análisis y Diseño**.

4. Análisis orientado a objetos

El análisis orientado a objetos varía respecto al análisis tradicional (programación estructurada) especialmente en el modelado de los objetos, por lo que es bastante similar en cuanto a los pasos y documentación generada como exponemos a continuación:

4.1. Identificación de las necesidades

La identificación de las necesidades es el punto de partida en la evolución de un sistema basado en ordenador. Esta identificación debe estar basada en “qué se desea hacer” y no en el “cómo se va a hacer”.

Para ello el analista de sistemas debe partir de las entrevistas iniciales y comenzar a definir los objetivos del sistema, es decir el **análisis de requisitos**:

- la información que se va a obtener
- la información que se va a suministrar
- las funciones y el rendimiento requerido.

El analista se asegura de distinguir entre lo que "necesita" el cliente (elementos fundamentales) y lo que el cliente "quiere" (elementos no esenciales).

Para conseguir esto, el analista aborda el problema en cinco fases distintas:

1. Reconocimiento del problema (no existen soluciones preexistentes)

2. Evaluación y síntesis del proyecto
3. Modelado estructurado
4. Especificaciones
5. Revisión de lo anterior

La información recogida durante la etapa de identificación de las necesidades se especifica en un **documento de conceptos del sistema**.

Una vez visto esto, es necesario determinar la **viabilidad del sistema**, abordando el mismo desde varias perspectivas:

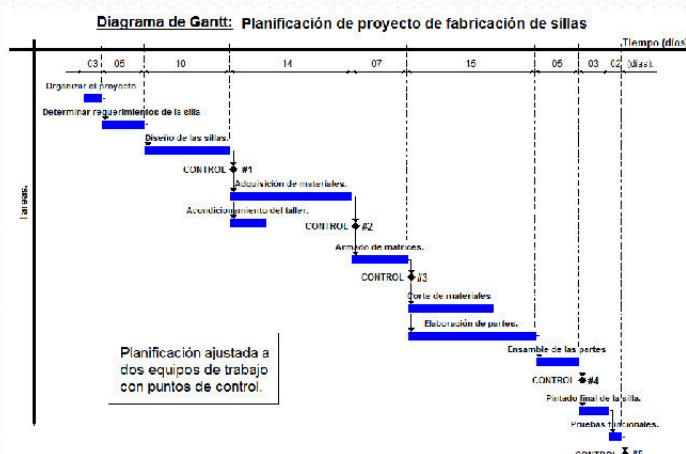
- **Viabilidad económica**: el análisis costo-beneficios debe implicar un beneficio o al menos, una inversión recuperable en un tiempo razonable para la empresa desarrolladora.
- **Viabilidad técnica**: Se cuenta con el personal y los conocimientos necesarios, así como los recursos para abordar el desarrollo, incluyendo el mantenimiento posterior del sistema.
- **Viabilidad legal**: el desarrollo no incumple ninguna ley ni norma legal y además se cuenta con las licencias oportunas del software de desarrollo y del software de terceros que se utilice.
- **Alternativas**. Una evaluación de los enfoques alternativos para el desarrollo del sistema.

4.2. Ámbito funcional y recursos

La determinación del **ámbito funcional** del sistema es una tarea dinámica que se va desarrollando junto al resto de etapas del ciclo de análisis, ya que conforme avanza el estudio del sistema se va detallando y actualizando para incorporar aquellas funcionalidades que en principio pasaron por alto o incluso para aplicar restricciones sobre las mismas.

Así mismo en la etapa de análisis funcional del sistema se debe llevar a cabo una estimación de los **recursos** necesarios para llevar a cabo el desarrollo del sistema.

Para los recursos además, será necesario planificar una agenda detallada de requerimientos y disponibilidad, mediante un cronograma o **diagrama de Gantt**.



Ejemplo de diagrama de Gantt.

4.3. Requisitos de implantación

La implantación del sistema implica la **instalación, la evaluación y la prueba** del mismo; los requisitos de implantación por tanto deben especificar **cómo** se va a realizar dicha implantación y qué tipo de **pruebas** y en qué cantidad y momentos se van a llevar a cabo.

Si hay existe un sistema anterior que va a ser reemplazado, también es necesario tener en cuenta cómo va a ser el **proceso de adaptación** al nuevo sistema.

Otro cometido es definir el **impacto del sistema**, tanto en términos de rendimiento como en el de recursos humanos y formación de los usuarios.

Todos estos requisitos con frecuencia implican:

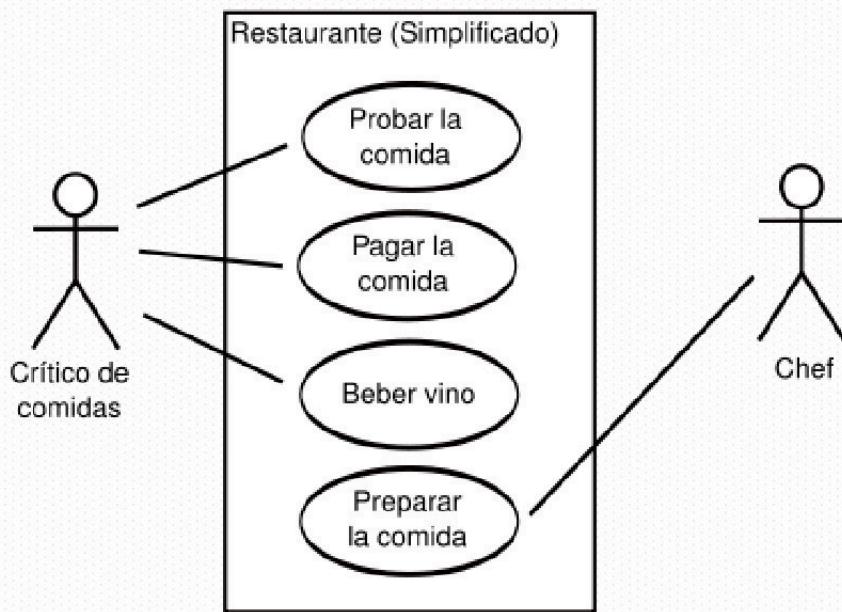
- Desarrollo de software de adaptación: conversores de formato, traspaso de datos, etc
- Reserva de horarios que afecten lo menos posible al funcionamiento normal del sistema a sustituir.
- Desarrollo de manuales y acciones formativas.
- Evaluación operacional
- Estudio de impacto en la empresa
- Calidad del desarrollo
- Diseño de pruebas de pre-instalación

Es común además, que durante un tiempo convivan el sistema antiguo y el nuevo, fase que coincide con la de revisión del producto que permite descubrir y corregir fallos en el mismo.

4.4. Diagramas de casos de uso

Dentro de la metodología UML los diagramas de casos de uso muestran las funcionalidades necesarias para los usuarios. Es una herramienta que permite reflejar los requisitos en el primer nivel de análisis del nuevo sistema.

Estos diagramas representan una vista externa del sistema existente o del nuevo; sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas.



Ejemplo de diagrama de casos de uso.

[Autor: Poor Yorick derivative work: Apokalipsy](#) CC BY-SA 3.0

4.5. Escenarios

El escenario es una herramienta muy útil para la construcción de un **diccionario de las clases** que habrá que implementar en el sistema.

Se parte de los **casos de uso** del sistema y por cada uno se elabora un guion (como en una película) identificando los objetos que intervienen, así como sus responsabilidades y relaciones entre dichos objetos.

4.6. Prototipado

El prototipado del sistema es una modelización que incluye las interfaces (pantallas) del sistema, así como de un sistema de navegación entre ellas. Sin embargo, **c carece completamente de funcionalidad**.

Lo importante es que el cliente pueda ver el aspecto que presentará el proyecto finalizado y evaluarlo.

Los prototipos se utilizan sobre todo cuando los requisitos funcionales no están bien definidos y es necesario indagar más en lo que desea el cliente.

5. Diseño orientado a objetos

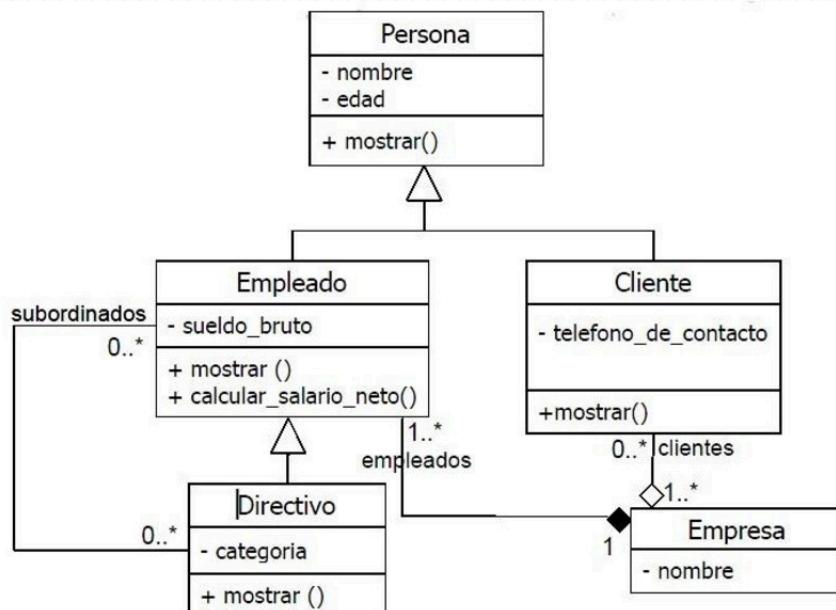
En la fase de diseño se hace un uso intensivo de los diagramas aportados por UML.

UML significa **Lenguaje de Modelado Unificado**. Es un lenguaje estandarizado, que nos permite modelar cualquier aplicación informática, pero realmente nos podría permitir modelar **cualquier tipo de sistema**.

Es el resultado de más de 20 años de esfuerzos para estandarizar varias prácticas en el campo del desarrollo de software, desde los años 90 (siglos desde un punto de vista tecnológico).

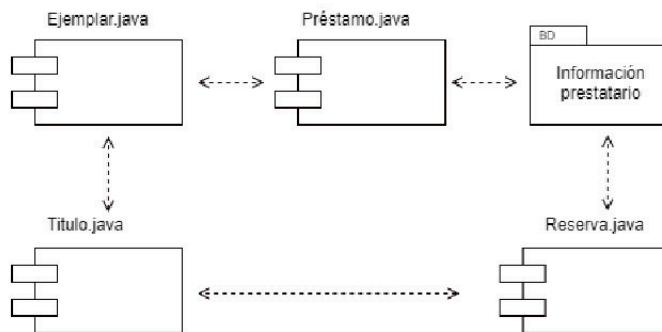
UML está compuesto por un conjunto de diagramas que permiten describir de forma visual cómo funciona un sistema. En total son 14 diagramas, de los que describimos los más importantes:

5.1. Diagramas de Clase



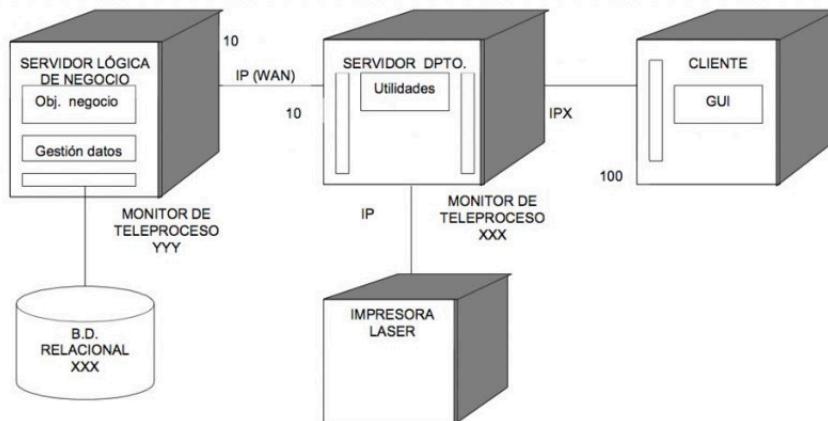
Este diagrama proporciona una vista estática de las clases y las relaciones entre ellas. También reciben el nombre de **diagramas de objeto** si se especifican valores concretos.

5.2. Diagramas de Componente



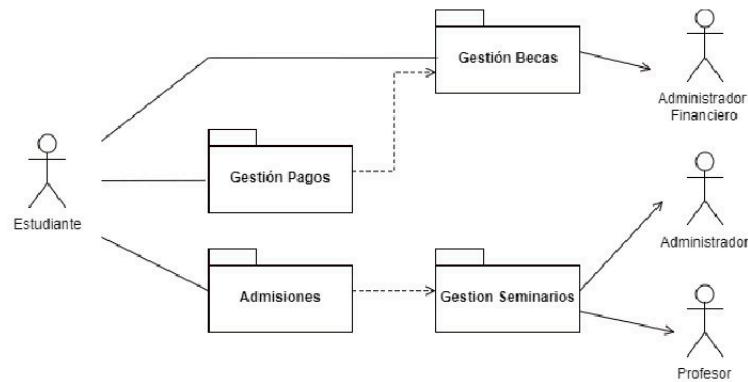
Representan el acceso por parte de las clases a componentes diversos, tales como bases de datos, librerías de funciones, etc.

5.3. Diagramas de Despliegue



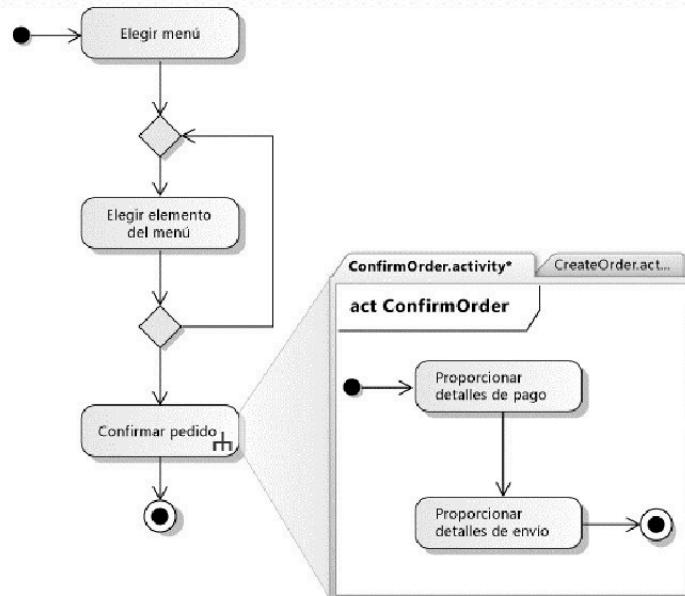
Este diagrama representa la ubicación y el entorno material de un recurso físico necesario para la ejecución del sistema y cómo están instalados sus componentes.

5.4. Diagramas de Paquete



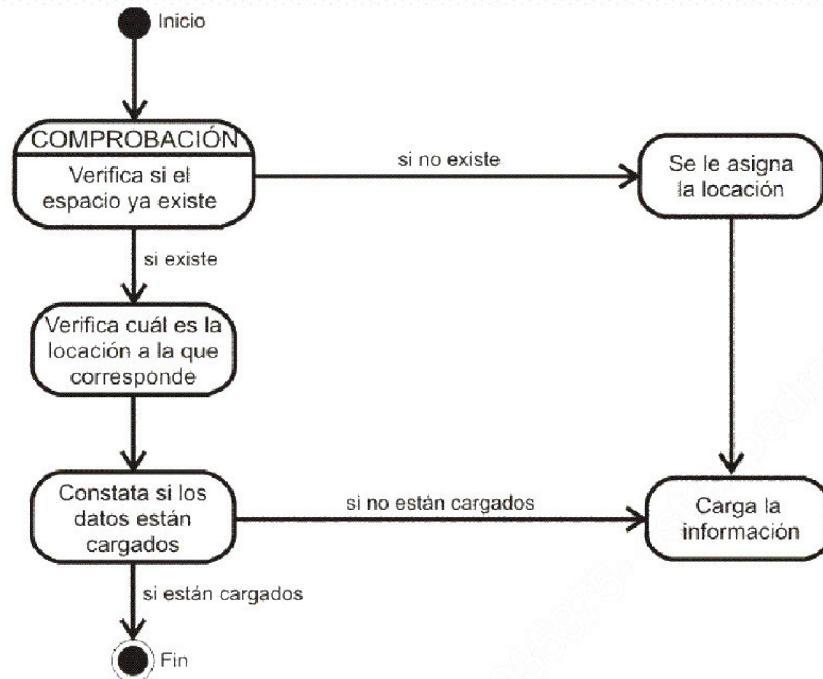
Los diagramas de paquete ofrecen una descomposición del sistema en partes, además de especificar con qué tipo de usuario están relacionados.

5.5. Diagramas de Actividad



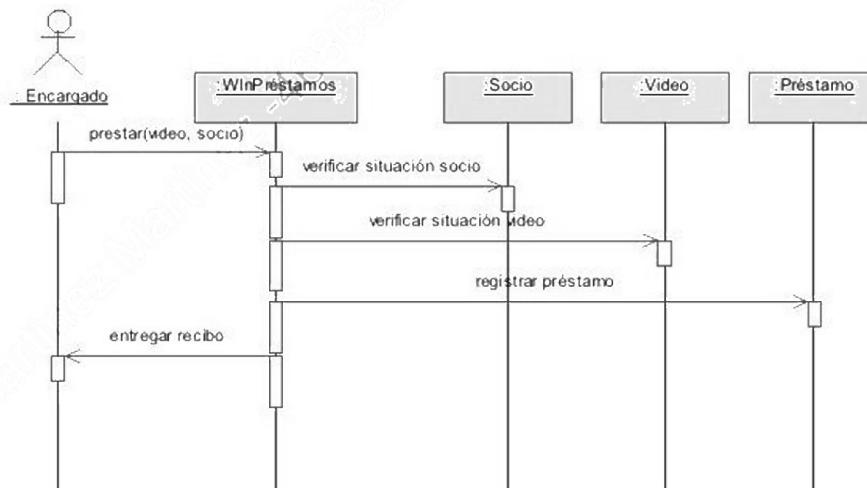
Los diagramas de actividad representan el orden de las acciones, sin especificar qué objetos son necesarios para llevarlas a cabo.

5.6. Diagramas de Maquina de Estado



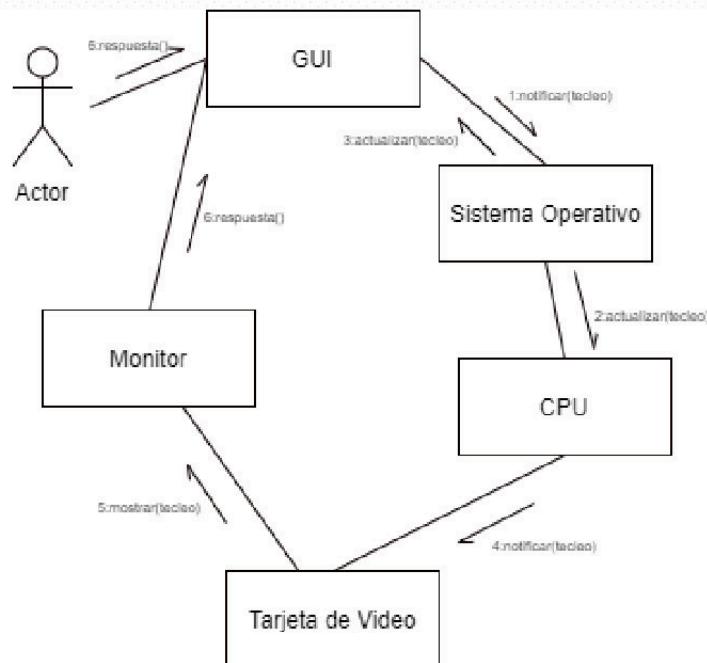
El diagrama de Maquina de estado describe el ciclo de vida de los objetos

5.7. Diagramas de Secuencia



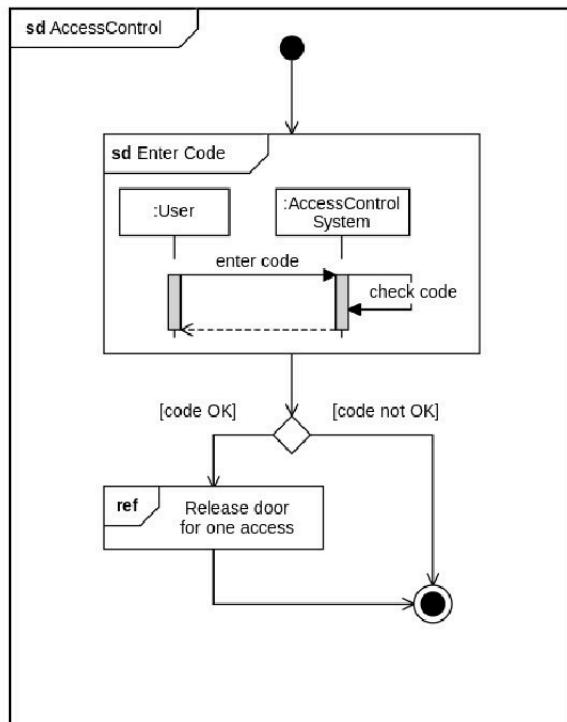
El diagrama de secuencia describe con detalle el orden de ejecución de varias funciones del sistema para llevar a cabo acciones específicas.

5.8. Diagramas de Comunicación



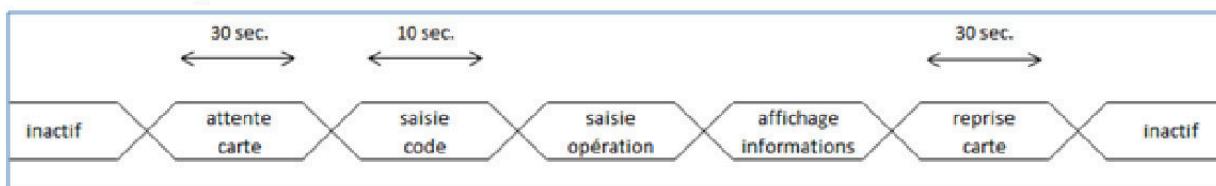
El diagrama de Comunicación permite poner en evidencia los intercambios de mensajes entre objetos del sistema.

5.9. Diagramas de Resumen de Interacción



El Diagrama de Resumen de Interacción da una **vista general de las interacciones del sistema**. Cada elemento puede luego ser detallado con un diagrama de secuencia o un diagrama de actividad.

5.10. Diagramas de Tiempo



El Diagrama de Tiempo es reservado para sistemas con exigencias de funcionamiento en **tiempo real**. Permite describir con precisión el tiempo que debe tardar cada proceso.

6. Conclusión.

6.1. Relación con el sistema educativo

- GS – DAW - DAM –Entornos de Desarrollo
- GS – DAW – DAM – Programación

7. Bibliografía

- Pressman, R. S. Ingeniería del Software. Un enfoque práctico, 3^a edición. Ed. McGraw-Hill, 2000.
- Sommerville, I.: Ingeniería de Software. 6^a Edición. Addison-Wesley Iberoamericana, 2002.
- Piattini, M. y otros: Análisis y diseño detallado de Aplicaciones Informáticas de Gestión. RA-MA, 2003.
- Introduction to Java Programming and Data Structures, Comprehensive Version. Y. Daniel Liang. Pearson, 11^a edición. 2017.
- Java 9. Francisco Javier Moldes Teo. Anaya. 2017.

