

FACULTAD DE INGENIERÍA

ESCUELA ACADEMICA PROFESIONAL DE INGENIERÍA DE SISTEMAS

**INFORME ACADEMICO**

“RUP”

Autores

CISNEROS HARO, Anthony

MEDINA DE LA CRUZ, Júnior

MOLLEAPAZA CHOQUECHAMBI, Milecio

PACULIA ROSADO, Guiancarlos

PUERTAS PINTO, Hanif Omar Alí

VARGAS OSORIO, Christian Andrés

Asesor

CORONEL CASTILLO, Eric Gustavo

Lima-Perú

2017

**ÍNDICE**

**Pág.**

1. **Definición e historia 3**
2. **Proceso unificado de desarrollo de software (RUP)**

**2.1 Flujos de trabajo**

**2.2 Fases**

**2.3 El ciclo de vida del RUP**

1. **UML: El lenguaje unificado de modelado (The Unified Modeling Language) 4**
2. **Diagramas de componentes 5**

**4.1 Estructura de un diagrama de componentes**

1. **Interfaces**
2. **Componentes**
3. **Puertos**
4. **Dependencias**
5. **Diagrama de despliegue 6**

**5.1 Estructura Fundamental**

1. **Nodos**
2. **Artefactos**
3. **Diagrama de casos de usos** **7**

**6.1 ¿Qué es un caso de uso?**

**6.2 Elementos del diagrama**

1. **Actores**
2. **Caso de uso**
3. **include y extends**

**6.3 Ejemplos de caso de uso**

1. **Diagrama de clases 10**

**7.1 Estructura del diagrama de clases**

1. **Nombre de clase**
2. **Atributos**
3. **Objetos**
4. **Operaciones**
5. **Diagrama de secuencias 11**

**8.1 Estructura del diagrama de secuencias**

1. **Línea de vida**
2. **Activación**
3. **Mensajes**
4. **Conclusiones 13**
5. **Referencias 14**
6. **Definición e historia**

**1.1 Definición**

Es un marco de referencia de Procesos influenciado por patrones de Procesos y Análisis bien documentado con un enorme conocimiento de Ingeniería de Software desarrollado por Rational (IBM).

**1.2 Historia**

El antecedente más importante se ubica en 1967 con la Metodología Ericsson (Ericsson Approach) elaborada por Ivar Jacobson. Entre los años de 1987 a 1995 Jacobson lanza el proceso de desarrollo Objectory (abreviación de Object Factory). Posteriormente en 1995 Rational Software Corporation adquiere Objectory AB y entre 1995 y 1997 se desarrolla Rational Objectory Process (ROP). Destacándose especialmente el flujo de trabajo conocido como modelado del negocio. En junio del 1998 se lanza Rational Unified Process.

1. **Proceso unificado de desarrollo de software (RUP)**

La metodología de RUP es de las mejores en el mundo, y es la que se usa en la ingeniería de software.

RUP es el trabajo de muchos años y la unificación de técnicas de desarrollo, usando el UML. Donde sus principales elementos son:

Trabajadores, es el comportamiento y responsabilidades designadas a un solo individuó, grupo o máquina que trabajan en equipos para realizar sus actividades.

Actividades, son las realizadas por trabajador(es) o maquina(as) que tienen una meta.

Artefactos, la producción del proyecto que es el producto final, transformados y finalizados por la actividad.

Flujo de actividades, serie de pasos realizadas por los trabajadores donde se produce un resultado de valor observable.

**2.1 Flujos de trabajo.**

Modelamiento del negocio, describe los procesos del negocio, identifican a quienes participan y las actividades a automatizar.

Requerimientos, lo que le da un propósito al sistema, por lo cual se extraen las funcionalidades requeridas y las restricciones.

Análisis y diseño, describe como el sistema será realizado con sus restricciones y previa vista, a lo que se indica con precisión lo que se va a programar.

Implementación, la organización de las clases y objetos en componentes, los nodos a utilizar y donde se ubican los componentes en ellos y las capas de aplicación.

Prueba (testeo), se buscan errores en el ciclo de vida.

Instalación, produce actividades (empaque, instalación, asistencia a usuarios, etc.) lista para entregar el software a los usuarios finales.

Administración del proyecto, las actividades para satisfacer las necesidades de los clientes en el producto.

Administración de configuración y cambios, que se describe como la utilización y actualización.

Ambiente, donde se contienen los procesos y herramientas en el trabajo del proyecto.

**2.2 Fases**

Conceptualización, descripción del negocio y sus delimitaciones en el proyecto.

Elaboración, la arquitectura del sistema y la aplicación ejecutable definidas.

Construcción, el producto ya se tiene listo para su uso y con un manual de usuario.

Transición, el reléase esta listo para su instalación donde se pueden corregir errores.

**2.3 El ciclo de vida del RUP**

Dirigido por caso de uso, en ellos se reflejan a lo que usuarios futuros requieren y anhelan, lo cual se obtiene cuando de modela el negocio y de da a través de los requerimientos.

Centrado en la arquitectura, la arquitectura muestra la visión común del equipo de proyecto y usuarios plasmados en el sistema completo.

Iterativo e incremental, puntos de control que se evalúan que son:

* Conceptualización – objetivos.
* Elaboración – arquitectura.
* Construcción - funcionalidad operativa.
* Transición – lanzamiento del sistema.

1. **UML: El lenguaje unificado de modelado (The Unified Modeling Language)**

Se deriva y unifica las tres metodologías de análisis y diseños más extendidas.

* Metodología de Grady Booch para la descripción de conjuntos de objetos y sus relaciones.
* Técnica de modelado orientada a objetos de James Rumbaugh (OMT: Object - Modelling Technique).
* Aproximación de Ivar Jacobson (OOSE: Object- Oriented Software Engineering) mediante la metodología de casos de uso (use case).

Comenzó a gestarse en octubre de 1994, cuando Rumbaugh se unió a la compañía Rational fundada por Booch.

El objetivo de ambos era unificar dos métodos que habían desarrollado: el método Booch y el OMT (Object Modelling Tool ). El primer borrador apareció en octubre de 1995. En esa misma época otro reputado investigador, Jacobson, se unió a Rational y se incluyeron ideas suyas. Estas tres personas son conocidas como los “tres amigos”. Además, este lenguaje se abrió a la colaboración de otras empresas para que aportaran sus ideas. Todas estas colaboraciones condujeron a la definición de la primera versión de UML tras el trabajo colectivo y se lanzo en enero de 1997 y fue presentado como versión 1.0 de UML. La gran ventaja de UML es que fue recogiendo aportaciones de los grandes gurús de objetos: David Hasel con sus diagramas de estado ; partes de la notación de fusión , el criterio de responsabilidad colaboración y los diagramas de Rebeca Wirfs-Brock, y el trabajo de patrones y documentación de Gamma-Helm-Johnson-Ulissides.

En 1997, OMG acepto UML como estándar (versión 1.1) y nacio el primer lenguaje de modelado visual orientado a objetos como estándar abierto de la industria. Desde entonces han desaparecido , prácticamente, todas las de mas metodologías y UML se a convertido en el estándar de la industria de software y de muchas otras industrias que requieren el uso de los modelos. En el año 2000 se presento UML 1.4 con la importante aportación de la semántica de acción , que describe el comportamiento de un conjunto de acciones permitidas que pueden implementarse mediante lenguajes de acción.

UML tiene dos categorías importantes de diagramas: Diagramas estructurales ( de estructura) y Diagrama de comportamiento. Las construcciones contenidas en cada uno de los diagramas UML se describen a lado de cada diagrama.

1. **DIAGRAMAS DE COMPONENTES**

Un componente se define como una fracción modular, reemplazable y significativa del sistema que guarda una implementación y exhibe un interfaz. Un componente es una entidad software que absorbe la funcionalidad de la encapsulación, ocultación, modularidad y la reutilización de POO (Programación orientada a objetos)

“los diagramas de componentes nos permiten poder tener una visión estática y arquitectónica de los componentes software utilizados en la aplicación”. (JIMENEZ, 2015)

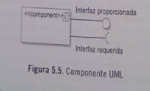
**4.1 Estructura de un diagrama de componentes**

1. **Interfaces**: Agrupación de métodos vacíos en la que se debe implementar una clase.

-Permite el incremento de la abstracción de la implementación

posterior de la clase hija

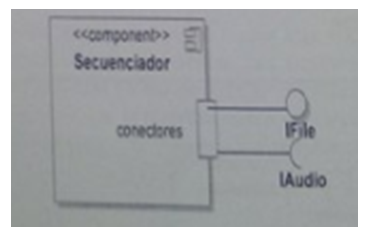
1. **Componentes**: En el diagrama de componentes UML el símbolo se representa como:

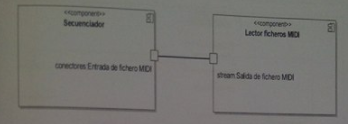


“Donde el componente se comporta como una caja negra, ocultando su información e implementación y exterioriza sus funcionalidades mediante las interfaces. Por ello la interfaz con forma de círculo representa la especificación que ofrece el componente, mientras que la interfaz inferior (marcada con un semicírculo) representa la conexión con una interfaz que proporciona una entrada de datos.

1. **Puertos**:

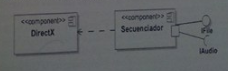
Tienen la funcionalidad de agrupar un conjunto de interfaces funcionan a modo de un punto de conexión del componente con su entorno. La finalidad es clasificar, agrupar y simplificar la representación de las interfaces requeridas y proporcionadas



Ejemplo de simplificación de diagramas y agrupación de interfaces

1. **Dependencias**:

En general indican que un elemento proveedor del modelo tiene un efecto sobre otro elemento. En los diagramas de componentes en particular especifican que el cliente está conectado o invoca a las operaciones proporcionadas por las interfaces del proveedor que dependen de el para su su implementación .las dependencias se representan en una línea discontinua acabada en una flecha abierta en la parte del proveedor.



1. **Diagrama de despliegue**

“El diagrama de despliegue trabaja con las instancias principales del hardware y del software, por lo que se componen de dos elementos fundamentales: nodos y artefactos”. (JIMENEZ, 2015)

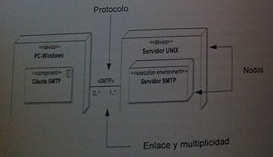
* 1. **Estructura fundamental**

1. **Nodos**:

Son la entidad más fundamental del diagrama de despliegue y representan elementos hardware y software complejos. Se especifican mediante un nombre de nodo y un estereotipo que identifican el tipo de nodo. Existen dos tipos de nodos:

<<Device>> modela lo que correspondía a un dispositivo hardware

<<Execution environment>> representa componentes específicos que son desplegados como artefactos ejecutables.



1. **Artefactos**:

Elementos software que se desplieguen dentro de los nodos. Estos artefactos pueden ser: código fuente, librerías, ficheros ejecutables, ficheros de script, base de datos, ficheros XML, ficheros de configuración y otro tipo.

1. **Diagrama de caso de uso**

Durante mucho tiempo, tanto en el desarrollo orientado a objetos como en el tradicional, las personas utilizaban escenarios típicos que les ayudaban a comprender los requerimientos. Estos escenarios utilizados son métodos muy informales que se construían, pero pocas veces se documentaban. Jacobson quien creo este tipo de método lo llevo a tal punto que lo convirtió en un elemento primario de la planificación y el desarrollo de proyectos.

**6.1 ¿Qué es un caso de uso?**

Un caso de uso es una interacción entre un usuario y un sistema de cómputo. En su forma más simple, el caso de uso se obtiene hablando con los usuarios habituales y analizando con ellos las distintas cosas que deseen realizar el sistema. Se debe abordar cada requerimiento que desee, darle un nombre y escribir un texto descriptivo breve. Durante la elaboración, esto es todo lo que necesitará para empezar.

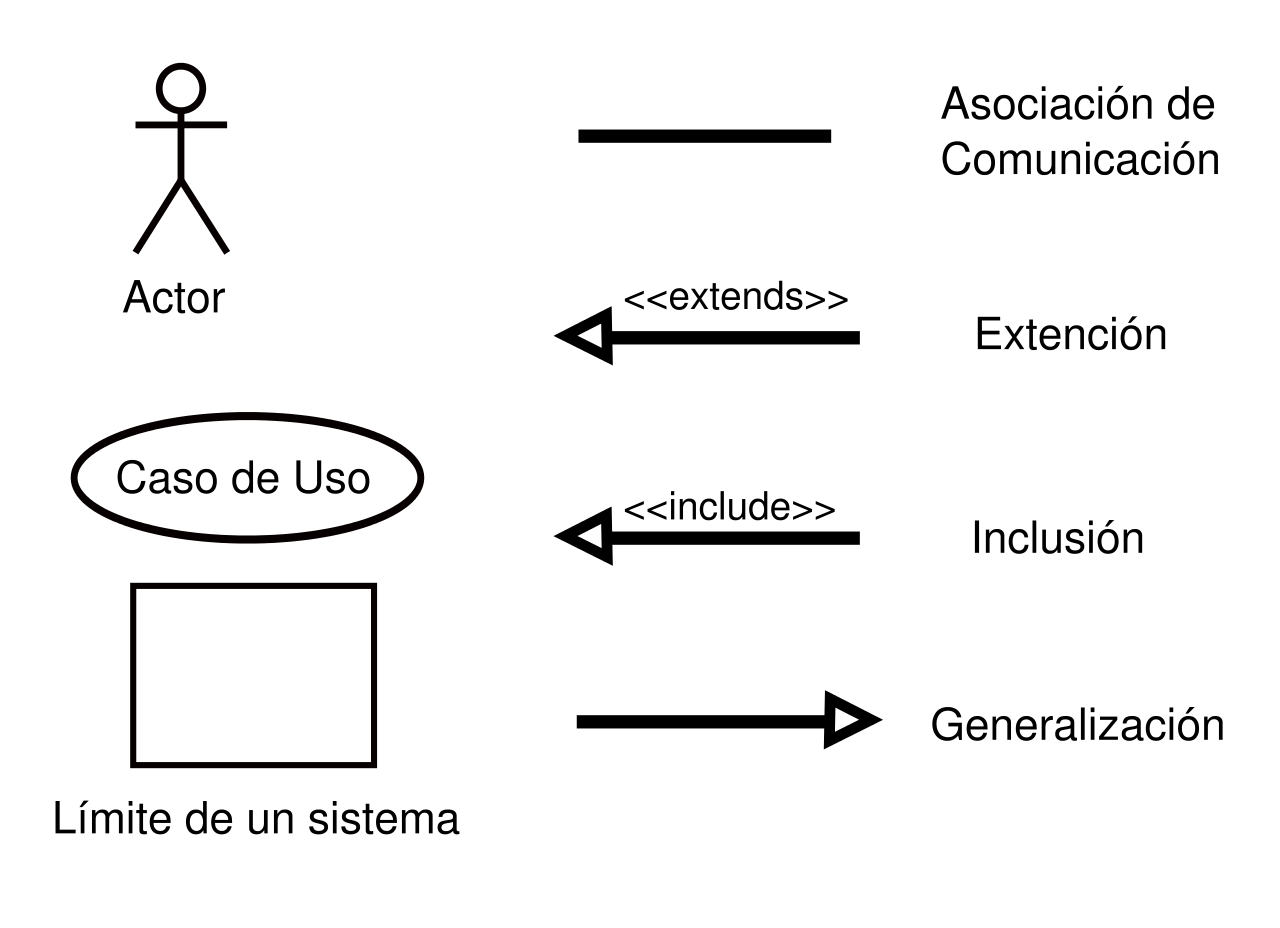
Jacobson además de introducir los casos de uso como elementos primarios del desarrollo del software, también diseñó un diagrama para la representación gráfica de los casos de uso. El diagrama de casos de uso es ya también parte del UML.

* 1. **Elementos del diagrama.**

1. **Actores**

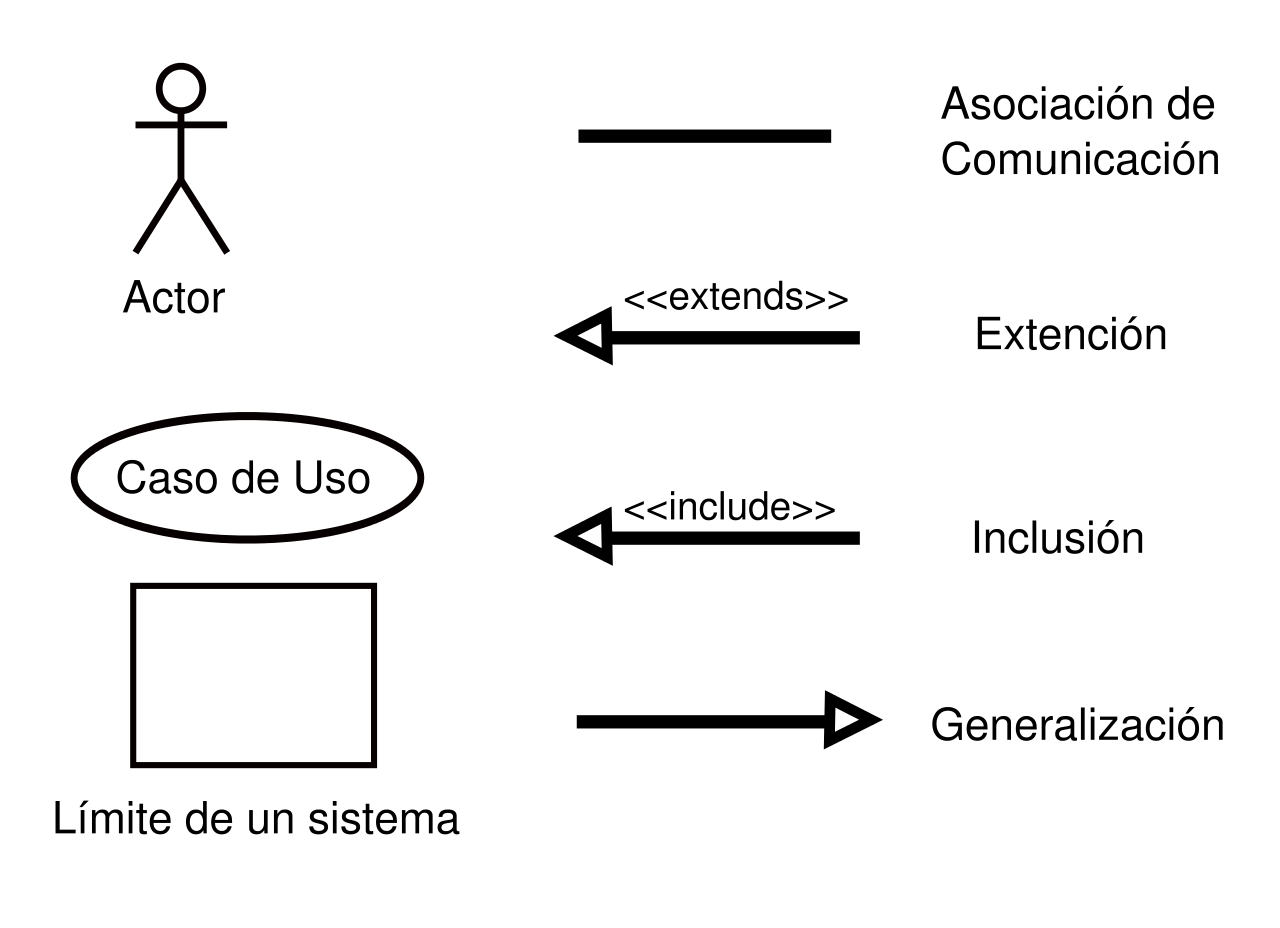
Empleamos el término actor para llamar así al usuario, cuando desempeña ese papel con respecto al sistema. Un usuario puede desempeñar varios papeles. Por ejemplo, un comerciante de edad madura podría desempeñar el papel de gerente de comercio y además ser un comerciante normal Por otra parte, un comerciante puede ser también agente de ventas. Cuando se trata con actores, conviene pensar en 105 papeles, no en las personas ni en los títulos de sus puestos.

Un mismo actor puede realizar muchos casos de uso; a la inversa, un caso de uso puede ser realizado por varios actores. En la práctica, considero que los actores son muy útiles cuando se trata de definir los casos de uso. Al enfrentarse con un sistema grande, puede ser difícil obtener una lista de casos de uso. Es más fácil en tales situaciones definir la lista de los actores y después tratar de determinar los casos de uso de cada actor. Un actor puede ser humano, un sistema informático o algún proceso ejecutable. Un actor se dibuja como una persona pequeña con trazos lineales y el nombre situado debajo de él.



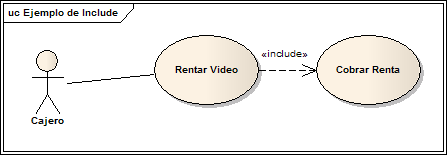
1. **Caso de uso**

El propósito de un caso de uso es definir o describir el comportamiento coherente sin revelar la estructura interna del sistema. La definición de un caso de uso incluye todo el comportamiento que se le supone y las secuencias principales, distintas variaciones del comportamiento normal y todas las condiciones de excepción que pueden darse con dicho comportamiento, junto con la respuesta deseada. Desde el punto de vista del usuario, éstas pueden ser situaciones anormales. Desde el punto de vista del sistema, son variaciones adicionales que deben ser descritas y manejadas.

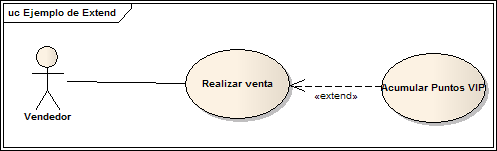


1. **Include y extends**

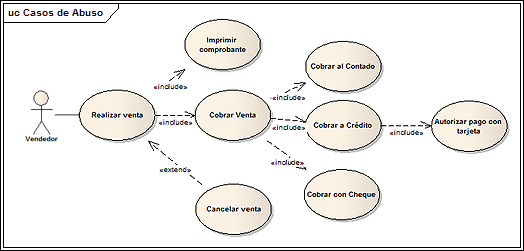
Include. En términos muy simples, cuando relacionamos dos casos de uso con un “include”, estamos diciendo que el primero (el [caso de uso](http://www.abiztar.com.mx/cursos/curso_administracion_levantamiento_de_requerimientos_con_casos_de_uso.html) base) incluye al segundo (el [caso de uso](http://www.abiztar.com.mx/cursos/curso_administracion_levantamiento_de_requerimientos_con_casos_de_uso.html) incluido). Es decir, el segundo es parte esencial del primero. Sin el segundo, el primero no podría funcionar bien; pues no podría cumplir su objetivo.



Extend. La polémica al querer seleccionar una de las dos relaciones es que en el “extend” también podemos ver, desde la perspectiva del usuario, a los dos flujos como si fueran uno sólo. Y en ciertos escenarios el [caso de uso](http://www.abiztar.com.mx/cursos/curso_administracion_levantamiento_de_requerimientos_con_casos_de_uso.html) base no podría cumplir su objetivo si no se ejecutara la extensión. Pero, una de las diferencias básicas es que en el caso del “extend” hay situaciones en que el [caso de uso](http://www.abiztar.com.mx/cursos/curso_administracion_levantamiento_de_requerimientos_con_casos_de_uso.html) de extensión no es indispensable que ocurra, y cuando lo hace ofrece un valor extra (extiende) al objetivo original del caso de uso base. En cambio en el “include” es necesario que ocurra el caso incluido, tan sólo para satisfacer el objetivo del caso de uso base.



* 1. **Ejemplo de caso de uso.**



1. **Diagrama de Clases**

El propósito de este diagrama es el de representar los objetos fundamentales del sistema, es decir los que percibe el usuario y con los que espera tratar para completar su tarea.

Un diagrama de clases se representa mediante una caja dividida en tres partes importantes; en la parte superior se muestra el nombre de la clase; en la media, los atributos y en la parte inferior, las operaciones. Una clase puede representarse con los detalles como atributos y operaciones o simplemente dejándolo con el nombre de la clase según interese o según la fase en la que se este

* 1. **Estructura del diagrama de clases**

1. **Nombre de clase**:

Identifica a la clase dentro del diagrama; el nombre de la clase puede aparecer precedido del nombre del paquete que la contiene.

1. **Atributos:**

Describen las propiedades que representan a una clase y se componen de una cadena de alfanumérica precedida por el tipo de dato; se define a un atributo mediante la siguiente notación:

[alcance] Atributo [: Tipo [multiplicidad]] [=Valor inicial]

Ejemplo: - saldo: float = 200.0

Donde el alcance o visibilidad indica la accesibilidad de una clase al atributo, hay 3 tipos de alcance:

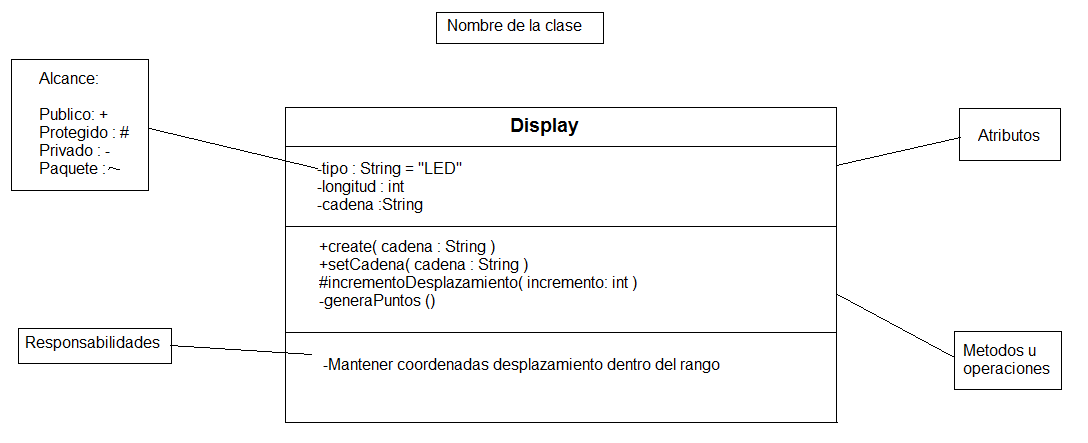
* Atributo público (+): Visible para cualquier clase desde fuera.
* Atributo privado (-): Solo visible para la propia clase y no puede ser heredado ni compartido con otras clases.
* Atributo protegido (#): Es visible para cualquier descendiente en la jerarquía de herencia y un atributo privado.

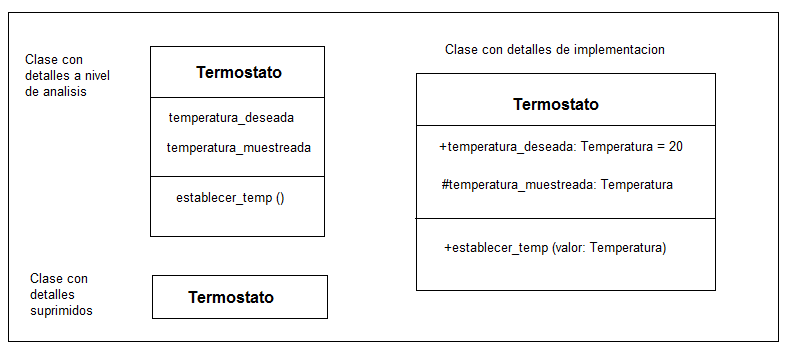
1. **Objeto:**

Son piezas individuales con identidad, estado y comportamiento invocable con los que se construye un sistema ejecutable.

1. **Operaciones**:

Son las acciones básicas que ofrece el objeto de la clase hacia el resto de objetos o para la gestión interna. El resultado de la ejecución de una operación (más conocida como método) producirá un cambio en el objeto al recibir un mensaje generado por otro objeto





1. **Diagrama de secuencias**

Representan información dinámica ya que los objetos interactúan entre si enviando o recibiendo mensajes, representan el flujo de trabajo basado en el intercambio de mensajes realizados durante tiempos transcurridos de manera simultánea; el elemento básico de un diagrama de interacción es el mensaje.

Las invocaciones de operaciones pueden ser de dos tipos

* Síncronas: Las instancias en comunicación tienen que estar sincronizadas, es decir la que invoca se queda bloqueada hasta recibir una respuesta.
* Asíncronas: La instancia que invoca la operación continúa su ejecución sin detenerse.

El eje vertical (línea de vida) representa el tiempo y en el eje horizontal se colocan los objetos y actores participantes; cada objeto o actor tiene una línea vertical y los mensajes se representan mediante flechas entre los distintos objetos; el tiempo se desplaza de arriba abajo, y se pueden colocar restricciones u otras etiquetas al margen izquierdo o junto a las transiciones o activaciones a las que se refieren.

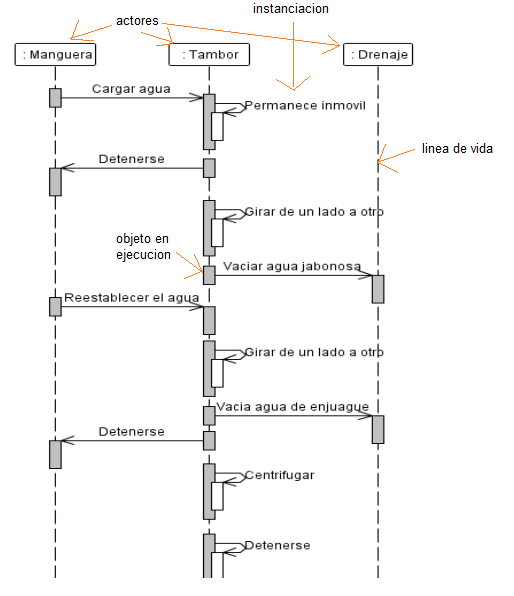
* 1. **Estructura del diagrama de secuencias**

1. **Línea de vida:**

Es el transcurso o evolución del tiempo dentro del escenario donde suceden las interacciones entre objetos. Se representa con una línea vertical y con la instancia del objeto en la parte superior de esta.

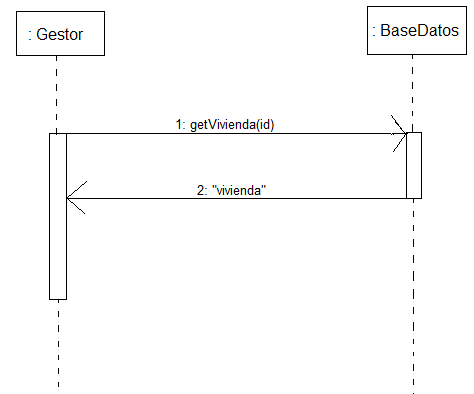
1. **Activación:**

Se produce cuando un objeto recibe un estímulo y procede a llamar a su respectivo método de ejecución.

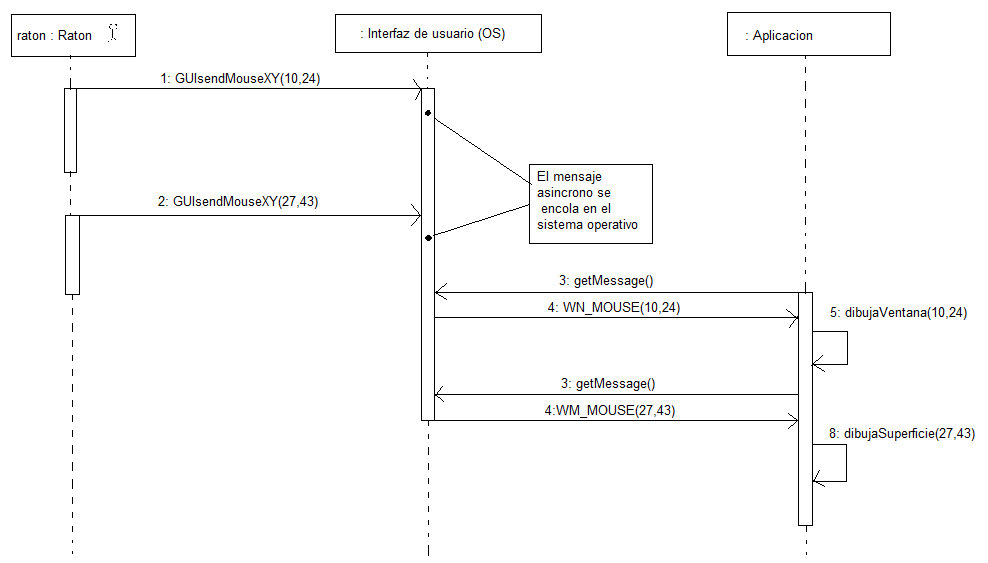


1. **Mensajes:**

* **Síncronos**: Las instancias en comunicación tienen que estar sincronizadas, es decir la entidad que envía el mensaje se queda a esperando la ejecución del mensaje y su retorno. Es preferible usar este tipo de mensaje para indicar la secuencia estricta de las llamadas a operaciones.



* **Asíncronos**: La instancia que invoca la operación continúa su ejecución sin detenerse. Son utilizados por aplicaciones multithread.



1. **Conclusiones**
2. Se puede concluir que, el RUP, como herramienta colaboradora en el desarrollo de software, aumenta la visión de desarrollo del mismo, es decir, el RUP es una herramienta que permite prever los cambios que un software pueda tener de acuerdo a los requerimientos y avance social que se tenga, brindando objetivos más amplios y visión de requerimientos global.
3. Se logró un verdadero aprendizaje de la importancia del Lenguaje de modelado unificado como herramienta de programación. Su éxito y su uso universal será inminente ya que ha sido apoyado por prácticamente todas las empresas importantes de informática. Se ha aceptado como un estándar por la OMG. Prácticamente todas las herramientas CASE y de desarrollo la han adaptado como lenguaje de modelado.
4. Como se ha dicho, el caso de uso es una forma de explicar lo que el usuario quiere o necesita que haga el software, ejemplo actualizar datos, registrar data, dando un panorama más específico de cómo se va comportar el programa en una organización. Así mismo, a la hora de desarrollar el software se tendrá claro a quien estará dirigido.
5. Con el desarrollo del presente trabajo se espera dar a conocer al RUP en funcionamiento al proceso unificado de desarrollo del software con sus fases, ciclo de vida y flujo de trabajo.
6. Podemos concluir, que el objetivo de un diagrama de clase es identificar claramente las relaciones entre objetos, clases y operaciones en una aplicación.
7. Además, reconocemos también el objetivo del diagrama de secuencia que es; la de describir la interacción entre objetos a lo largo de una línea de tiempo.
8. **Referencia**

BAZZ Leen, CLEMENTS Paul y KAZMAN Rick, Rational Unified Process (RUP), [en linea], etsii, 2003.

ISBN: 0321154959

BOOCH, Grady, JACOBSON, Ivar y RUMBAUGH, James. EL LENGUAJE UNIFICADO DE MODELADO. MANUAL DE REFERENCIA [en línea]. Madrid: PEARSON EDUCACION, S.A, 2000 [fecha de consulta: 14 de setiembre de 2017] disponible en: <https://ingenieriasoftware2011.files.wordpress.com/2011/07/el-lenguaje-unificado-de-modelado-manual-de-referencia.pdf>

CARRILO, Anay. Enseñanza de la Metodología RUP de ingeniería del software [en línea].2009. [fecha de consulta: 15 de septiembre de 2017]

Disponible en: [www.eumed.net/libros/2009c/587/](http://www.eumed.net/libros/2009c/587/)

ISBN: 84 7829 037 0

FOWLER, Martin. UML, gota a gota. México: Addison Wesley Longman, 1999, 244 pp.

ISBN: 968 444 364 1

GRAU, Xavier y SANCHEZ, María, Desarrollo Orientado a Objetos con UML [en línea], Facultad de informática-UPM, Venezuela [fecha de consulta 14 de setiembre de 2017], Disponible en: <http://www.uv.mx/personal/maymendez/files/2011/05/umlTotal.pdf>

JIMENEZ DE PARGA, Carlos. UML Aplicaciones en Java y C++. Madrid: RA-MA Editorial, enero de 2015, 412 pp.

ISBN: 978 84 9964 516 2

JOYANES, Luis y ZAHONERO, Ignacio. Programación en C, C++, Java y UML. 2 ª ed. Mexico D. F.: Mcgraw Hill educación. 2014. 930 pp.

ISBN: 9786071512123