Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática

Desarrollo de Sistemas Informáticos Usando UML y RUP
Una Visión General

Álvaro Rendón Gallón

Popayán, agosto de 2004

Tabla de Contenido

Pág.

Presentación

El paradigma de orientación a objetos tiene sus orígenes a principios de los años 70, como una respuesta a la necesidad de manejar la complejidad de los sistemas. La idea de los "objetos", que aparecieron como el mecanismo más eficaz para representar los módulos en los cuales aquellos se descomponen, surgió de manera casi simultánea en varios dominios de la informática [1]:

- Avances en la arquitectura de los computadores, incluyendo los sistemas "basados en capacidades" y el apoyo en dispositivos físicos para los conceptos de sistemas operativos.
- Avances en lenguajes de programación, como se mostró en Simula, Smalltalk, CLU y Ada.
- Avances en las metodologías de programación, incluyendo la modularización y la ocultación de la información.

Hoy por hoy, la programación orientada a objetos parece haberse ganado un espacio en la comunidad académica y el mundo de los negocios, al ser reconocida como elemento esencial para el desarrollo de programas complejos y escalables, y como la clave para la construcción de aplicaciones distribuidas. Y es este último aspecto el que quizás más haya contribuido a ese reconocimiento; si bien algunos autores mantienen posiciones muy críticas con respecto a las "promesas" de lo que ellos consideran una nueva "moda tecnológica", parece ser que el gran auge de las plataformas distribuidas orientadas a objetos como CORBA y DCOM le han dado su bendición al paradigma, permitiéndole ingresar al exclusivo club de las innovaciones que superan las buenas intenciones y se incorporan definitivamente al acervo tecnológico mundial.

La expresión "*programación orientada a objetos*" se refiere no sólo a la elaboración de programas usando lenguajes orientados a objetos, como por desgracia se entiende comúnmente, sino también, y sobre todo, al uso del marco de razonamiento de la orientación a objetos a lo largo de todo el ciclo de desarrollo de los programas, desde la captura de los requerimientos hasta las pruebas de aceptación.

El concepto de objeto se ha vuelto omnipresente; tanto, que en cada campo donde ha encontrado aplicación han debido definir un *modelo de objetos* con el fin de establecer la conceptualización y terminología que utilizan. Solamente en el dominio de los sistemas de telecomunicaciones, que son inherentemente distribuidos y por consiguiente usuarios en primera línea del paradigma, se encuentran distintos modelos de objetos para TMN (Telecommunications Management Network), TINA (Telecommunications Integrated Network Architecture), y CORBA (Common Object Request Broker Architecture), para citar sólo unos ejemplos.

Todo esto parece conducir a la clásica pregunta: "¿Cómo era posible concebir los sistemas sin la orientación a objetos?". Sin embargo, a pesar de la "guerra de los métodos" orientados a objetos de los comienzos, o quizás debido a ella, muchos

programadores aún limitan su incursión en el mundo de los objetos al uso de sus lenguajes de programación, ignorando su verdadera esencia y origen: el manejo de la complejidad en el proceso de desarrollo.

Entre los muchos investigadores de la orientación a objetos hay tres autores que se han destacado por sus contribuciones al uso del paradigma en todo el proceso de desarrollo: Ivar Jacobson, Grady Booch y James Rumbaugh, los famosos "Three Amigos". Luego de muchos años de trabajo individual, desarrollado y difundiendo sus propios métodos, han unido sus teorías y su experiencia, y se han puesto a la cabeza de un formidable grupo de investigadores para construir dos herramientas con las cuales buscan estandarizar y por ende facilitar el uso de los objetos en la programación: el Lenguaje Unificado de Modelado (UML, *Unified Modeling Language*) y el Proceso Unificado de Rational para el Desarrollo de Programas (RUP, *Rational Unified Process*). Mientras que UML es ya un lenguaje maduro que ha logrado la aceptación de amplios sectores de la industria y la academia, RUP, lanzado a finales de 1998 [2], sigue siendo aún una propuesta que deberá depurarse y templarse al calor de la experiencia de su aplicación en el campo y los aportes de los casos de estudio.

RUP y UML están estrechamente relacionados entre sí, pues mientras el primero establece las actividades y los criterios para conducir un sistema desde su máximo nivel de abstracción --una idea en la cabeza del cliente-- hasta su nivel más concreto --un programa ejecutándose en las instalaciones del cliente, el segundo ofrece la notación gráfica necesaria para representar los sucesivos modelos que se obtienen en el proceso de refinamiento. A pesar de que UML fue lanzado como una notación de propósito general, capaz de soportar distintos enfoques en el proceso de desarrollo de los programas, sus propios autores han confesado que durante su definición tuvieron siempre en mente una conceptualización sobre el proceso de desarrollo de programas, que finalmente han plasmado en el RUP. Por supuesto, esto no significa que estas dos herramientas no puedan ser aplicadas de manera independiente, como lo demuestra la abundante literatura que existe sobre la aplicación de UML siguiendo diversos métodos.

El presente documento expone los conceptos básicos de UML sin olvidarse de que la notación está al servicio del proceso de desarrollo. Por esta razón, los ejemplos que apoyan la presentación de la notaciones reflejan el seguimiento de RUP, sin que sea necesario entrar en detalles para explicar el proceso; de todas maneras, al final se incluye una breve sección sobre el tema. Al centrarse en los aspectos básicos que soportan el desarrollo de aplicaciones, no se mencionarán otros aspectos del lenguaje muy útiles para el modelado a alto nivel y el desarrollo de herramientas como los perfiles, el intercambio de modelos usando XMI (XML[1] *Metadata Interchange*) y el lenguaje de restricciones OCL (*Object Constraint Language*).

Cabe advertir que el tratamiento del tema está muy orientado a la práctica, como corresponde a las asignaturas a las que está dirigido. Se supone que el lector ya conoce los principios fundamentales del paradigma de orientación a objetos, lo mismo que un lenguaje de programación orientado a objetos, preferiblemente C++ o Java, y es muy

[1] *eXtensible Markup Language.*

deseable que posea nociones de los diferentes conceptos relacionados con el ciclo de vida de un sistema, adquiridos normalmente en los cursos de Ingeniería de Programación.

Este documento está basado en las charlas ofrecidas por el autor en la cuarta asignatura del énfasis en Ingeniería de Sistemas Telemáticos (Proyecto de Investigación) y la electiva Sistemas Distribuidos Orientados a Objetos, ambos dentro del programa de Ingeniería Electrónica y Telecomunicaciones de la FIET, desde hace varios semestres. Habrá nuevas versiones, producto de una elaboración más cuidadosa, un mayor estudio y experiencia en el uso de estas herramientas, y de las observaciones que los amables lectores tengan a bien realizar.

El documento está estructurado de la siguiente manera: El Capítulo 1 presenta los antecedentes y la historia de UML; en el Capítulo 2 se describe el primer concepto de la notación UML que son las vistas; el Capítulo 3 es el más extenso y el más importante, pues explica los diagramas utilizados por la notación para describir los conceptos de la programación orientada a objetos; los mecanismos de extensión definidos para usar la notación en contextos específicos se exponen en el Capítulo 4; el Capítulo 5 presenta una visión general de RUP; y finalmente el Capítulo 6 ofrece algunas URL de interés.

1. Introducción

UML es el resultado de un esfuerzo dirigido a obtener una notación gráfica unificada para representar los modelos de sistemas desarrollados con el paradigma de orientación a objetos.

En términos de los impulsores iniciales de UML, Jacobson, Booch y Rumbaugh, es posible identificar tres generaciones de notaciones propuestas para el desarrollo de programas orientados a objetos.

En la **Primera Generación**, denominada la "guerra de los métodos" [3] aparecieron una gran cantidad de notaciones diferentes, propuestas desde los entornos académicos e industriales donde iba tomando fuerza el uso de los conceptos y lenguajes orientados a objetos en el desarrollo de aplicaciones. Aunque el concepto de "objeto" ya había sido propuesto en los años 70 y ya en ese entonces la orientación a objetos había sido incorporada en el lenguaje Simula, no fue hasta la popularización de lenguajes como Smalltalk y C++, a finales de los 80, que empezó a necesitarse una metodología para el desarrollo de programas orientados a objetos.

Entre 1989 y 1994, el número de métodos y notaciones pasó de 10 a 50, entre los que se destacaron OMT (Object Modeling Technique) [4], propuesto por Rumbaugh con énfasis en el análisis; el método de Booch [5], con grandes fortalezas en el diseño; OOSE/Objectory (Object-Oriented Software Engineering) [6], propuesto por Jacobson con una excelente herramienta para el análisis de comportamiento como son los Casos de Uso; Coad/Yourdon (OOA/OOD), uno de los primeros métodos de análisis y diseño orientados a objetos; Shlaer/Mellor; HOOD (Hierarchical Object-Oriented Design), un método jerárquico de diseño desarrollado para la Agencia Espacial Europea y que tiene una versión para sistemas de tiempo real críticos (HRT-HOOD) [7]; y ROOM (Real-Time Object-Oriented Modeling) [8], propuesta también para sistemas de tiempo real por el grupo que ahora trabaja con Rational en UML para tiempo real.

La mayoría de los métodos y notaciones sólo cubría una de las actividades del desarrollo, por ejemplo el diseño, y estaba enfocado hacia un dominio de aplicación específico, como en el caso de los sistemas de tiempo real. Esta situación planteó grandes dificultades a todos los sectores interesados en los métodos orientados a objetos: para las empresas que desarrollaban programas, implicaba un proceso de toma de decisiones delicado y plagado de riesgos para elegir, entre el vasto menú de métodos y notaciones, el (o generalmente los) que mejor se ajustaran a sus necesidades y líneas de productos, y entrenar a su personal para utilizarlos; para las empresas que producían herramientas, exigía un gran esfuerzo de desarrollo para ofrecer productos capaces de soportar múltiples notaciones, lo cual, por otra parte, limitaba enormemente la potencia de estas herramientas; y para la comunidad académica, significaba una fuerte restricción para el intercambio de ideas y experiencias por no existir un lenguaje común. Al final, esta gran variedad entorpecía, en lugar de promover, la difusión y el avance de estas técnicas.

En la **Segunda Generación**, los métodos empiezan a converger cuando diversos autores continúan adelante con sus propuestas retomando elementos del trabajo de sus colegas, y reúnen los conceptos de varias de las técnicas propuestas en la generación anterior con el fin de ofrecer un soporte integral para todo el ciclo de vida. Entre las más importantes están Fusion [9], una segunda versión de OOSE, OMT-2 y Booch'93. Estos métodos son completos en el sentido que cubren las distintas fases de desarrollo, pero mantienen un énfasis en ciertos dominios de aplicación (ingeniería de negocios, sistemas de información, etc.).

El esfuerzo de convergencia culmina con la **Tercera Generación**, en la que UML define la notación, cubriendo todo el ciclo de desarrollo y diversos dominios de aplicación, y RUP, que será retomado en el capítulo sobre el proceso de desarrollo.

La historia de UML (Figura 1 [10]) empieza en 1994 cuando Booch y Rumbaugh, trabajando para una empresa productora de herramientas de programación llamada Rational, deciden juntar sus trabajos Booch'93 y OMT-2 para proponer el "Unified Method", presentando la primera versión pública (Unified Method version 0.8) en octubre de 1995.

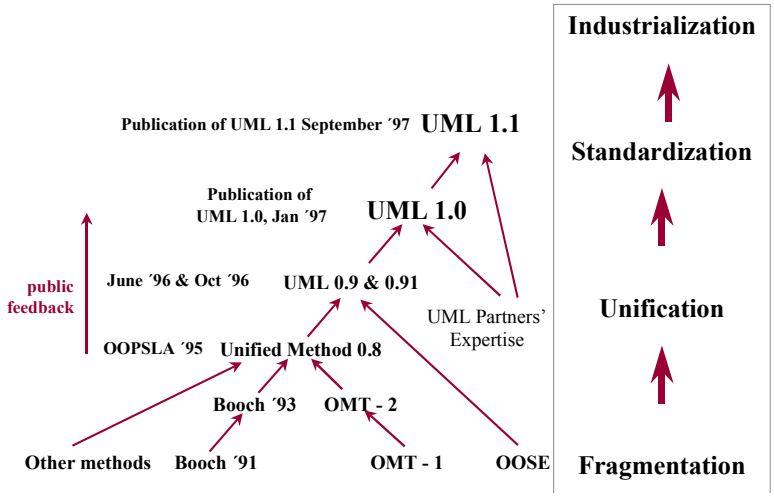


Figura 1. Trayectoria de UML

Poco más tarde se les une Jacobson, quien aporta su trabajo en OOSE y su larga experiencia en el desarrollo de programas en Ericsson (para las centrales AXE) y luego en su propia empresa, Objective Systems. De esta manera se configura el equipo de trabajo que es conocido como "the three amigos", quienes adoptan para su obra el nombre UML y entregan las versiones 0.9 y 0.91 en julio y octubre respectivamente de 1996, al igual que su propia herramienta de soporte: Rational Rose [11].

Conscientes de que el éxito de la labor unificadora no dependía únicamente de las virtudes técnicas de su propuesta, a lo largo del año 1996 los tres gurús utilizan su capacidad de convocatoria para lograr que un grupo de empresas se una a Rational en la

integración del Consorcio UML. El propósito de este consorcio es el de contribuir a la definición del lenguaje, pero también ofrecerle el respaldo de las grandes empresas del mundo de la informática. En enero de 1997, cuando se publica la versión 1.0, forman parte de él Digital Equipment Corporation, HP, IBM, Microsoft, Oracle, Unisys, Texas Instruments, i-Logix, Intellicorp, ICON Computing, MCI, y Rational.

En julio de 1997 el Consorcio UML presenta la versión 1.0 al proceso de adopción de normas de la OMG (Object Management Group), la organización internacional integrada por más de 800 empresas, organismos gubernamentales y universidades, que promueve la adopción de estándares para el desarrollo de aplicaciones distribuidas, basada en el uso del paradigma de orientación a objetos. Al ser adoptado como estándar de la OMG, UML recibe el impulso definitivo para convertirse en la notación universal de los modelos orientados a objetos.

La primera versión oficial fue la 1.1, publicada en septiembre de 1997 y adoptada por la OMG en noviembre del mismo año. La versión actual es la 1.5 [12], de marzo de 2003, y se espera una revisión mayor, UML 2.0 [13].

La definición de UML fue establecida desde su primera versión pública [14]: UML es un lenguaje usado para especificar, visualizar y documentar los componentes de un sistema en desarrollo orientado a objetos. Él representa la unificación de las notaciones de Booch, OMT y Objectory, al igual que las mejores ideas de otros metodologistas como se muestra en la Figura 2. Mediante la unificación de las notaciones usadas por estos métodos orientados a objetos, el Lenguaje Unificado de Modelado establece la base para un estándar *de facto* en el dominio del análisis y el diseño orientados a objetos, fundado en una amplia base de experiencia de los usuarios.

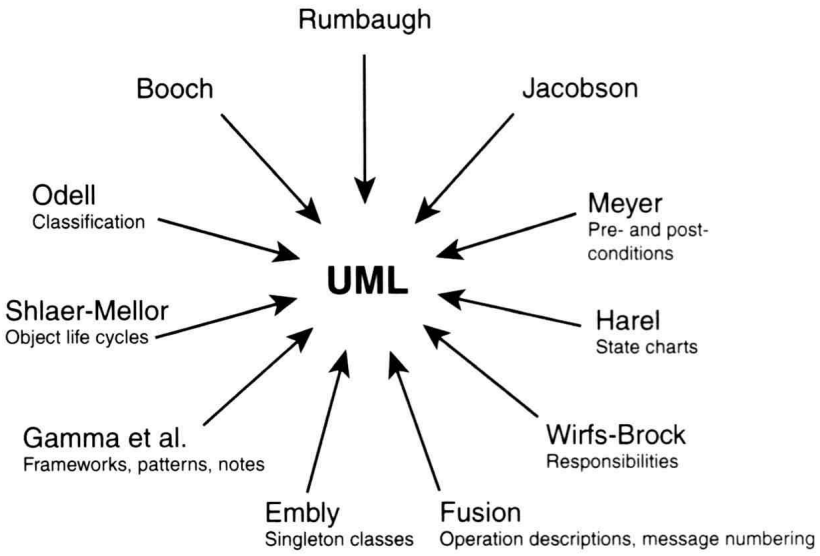


Figura 2. Las fuentes de UML

UML ha sido desarrollado con el propósito de ser útil para modelar diferentes sistemas: de información, técnicos (telecomunicaciones, industria, etc.), empotrados de tiempo

real, distribuidos; y no sólo es útil para la programación sino también para modelar negocios, es decir, los procesos y procedimientos que establecen el funcionamiento de una empresa.

En lo que corresponde al desarrollo de programas, posee elementos gráficos para soportar la captura de requisitos, el análisis, el diseño, la implementación, y las pruebas. Sin embargo no hay que olvidar que UML es una notación y no un proceso/método, es decir, es una herramienta útil para representar los modelos del sistema en desarrollo, mas no ofrece ningún tipo de guía o criterios acerca de cómo obtener esos modelos.

2. Vistas UML

La descripción de los sistemas se realiza en UML a través de **Vistas**, las cuales a su vez están integradas por **Diagramas**.

Esta estrategia parte del hecho de que un solo diagrama no puede expresar toda la información que se requiere para describir un sistema. Si se hace un símil con una edificación, no es posible elaborar un sólo plano que contenga todos los detalles de su construcción; en lugar de ello, se dibujan planos que presentan diferentes aspectos del edificio: la estructura, las instalaciones eléctricas, las instalaciones hidráulicas, el diseño exterior, etc. Así pues, es necesario utilizar conjuntos separados de diagramas, las vistas, para representar proyecciones del sistema relacionadas con aspectos particulares funcionales y no funcionales. La Figura 3 muestra las diferentes vistas consideradas en UML.

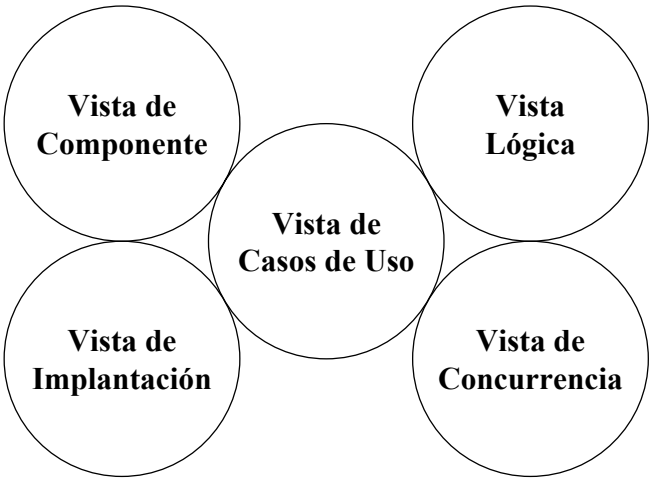


Figura 3. Vistas de UML

2.1 Vista de Casos de Uso

No es casual que en la figura, la Vista de Casos de Uso se represente en el centro de todas, haciendo el papel de enlace, pues ésta constituye efectivamente el hilo conductor de todo el proceso de desarrollo, pese a que es la única que no describe aspectos de la *construcción* del sistema sino de su *comportamiento*. La Vista de Casos de uso muestra la funcionalidad del sistema, tal como es percibida por actores externos.

La Vista de Casos de Uso es utilizada por todos los participantes en el proceso de desarrollo: los clientes, pues a través de ella se definen y expresan los requerimientos del sistema; y los equipos de diseño, desarrollo, y pruebas, pues tal como se mencionó arriba, conduce todo el proceso de desarrollo y verificación.

Utiliza los siguientes diagramas:
- Diagramas de Casos de Uso
- Diagramas de Actividad (opcional)

2.2 Vista Lógica

Muestra el diseño de la funcionalidad del sistema en sus dos aspectos esenciales: su estructura, es decir, los componentes que lo integran, y su comportamiento, expresado en términos de la dinámica de interacción de dichos componentes.

Es utilizada fundamentalmente por los equipos de diseño y desarrollo, y consta de los siguientes diagramas.

Para la descripción de estructura:
- Diagramas de Clases y de Objetos

Para la descripción del comportamiento:
- Diagramas de Estado, Secuencia, Colaboración y Actividad.

2.3 Vista de Componentes

UML no se limita a ofrecer una notación para representar los *modelos* obtenidos en el proceso de desarrollo de los programas, que al fin y al cabo constituyen una abstracción de los mismos, sino que también ofrece elementos para representar las entidades concretas en las que finalmente reside el resultado de todo el trabajo de desarrollo: los *archivos*. Mediante la Vista de Componentes se muestra la organización del código y demás archivos que hacen parte del sistema, tanto los que han sido desarrollados (programas fuente, ejecutables, etc.) como los que han sido adquiridos (bibliotecas de funciones o de servicios, componentes reutilizados, etc.); además, muestra también las relaciones de dependencia que existen entre ellos.

Es utilizado por el grupo de desarrollo y consiste en el Diagrama de Componentes.

2.4 Vista de Implantación

Muestra la implantación del sistema en la arquitectura física, indicando dónde se localizan los ejecutables del sistema y cómo se comunican entre sí. Para ello, se utiliza una descripción de los *nodos* del sistema, que son los computadores donde éste se ejecuta, y los dispositivos periféricos relevantes.

Es utilizado por los grupos de desarrollo, integración y pruebas, y consiste en el Diagrama de Implantación. Éste es el único diagrama de UML que permite representar los dispositivos físicos utilizados por la aplicación desarrollada.

2.5 Vista de Concurrencia

Es una combinación de las vista Lógica, de Componentes y de Implantación, en la que se muestra el manejo de los aspectos de concurrencia en el sistema, especialmente los de comunicación y sincronización. Para ello, el sistema se divide en procesos, que manejan su propio flujo de control al interior de un procesador, y procesadores, como unidades independientes de ejecución; y se presentan tanto los aspectos estáticos de la asignación de los componentes a la arquitectura física, como los aspectos dinámicos de su interacción.

Esta es una vista de gran importancia para los sistemas distribuidos y de tiempo real, y es utilizada principalmente por lo grupos de desarrollo e integración.

Consta de los siguientes diagramas.

Para la descripción de la implementación:
- Diagramas de Componente e Implantación.

Para la descripción dinámica:
- Diagramas de Estado, Secuencia, Colaboración y Actividad.

3. Diagramas UML

Los diagramas de UML se pueden clasificar de la siguiente manera:

- Diagrama de Casos de Uso.

- Diagramas de Clase y Diagramas de Objetos.

- Diagramas de Comportamiento.
  - Diagramas de Secuencia.
  - Diagramas de Colaboración.
  - Diagramas de Estados.
  - Diagramas de Actividad.

- Diagramas de Implementación.
  - Diagramas de Componentes.
  - Diagramas de Implantación.

3.1 Diagrama de Casos de Uso

Sirve para describir las interacciones del sistema con su entorno, identificando los **Actores**, que representan los diferentes roles desempeñados por los usuarios del sistema, y los **Casos de Uso**, que corresponden a la funcionalidad que el sistema ofrece a sus usuarios, explicada desde el punto de vista de éstos. Los actores no son solamente humanos, pudiendo ser también otros sistemas con los cuales el sistema en desarrollo interactúa de alguna manera.

> *Un **Actor** define un conjunto coherente de roles que los usuarios de una entidad pueden jugar cuando interactúan con ella. Se puede considerar que un Actor juega un rol diferente con respecto a cada Caso de Uso con el cual se comunica* [12].
>
> *Un **Caso de Uso** es un tipo de clasificador que representa una unidad coherente de funcionalidad suministrada por un sistema, un subsistema o una clase, tal como se manifiesta mediante secuencias de mensajes intercambiados entre el sistema (subsistema, clase) y uno o más interactores externos (llamados actores), junto con las acciones realizadas por el sistema (subsistema, clase)* [12].

La Figura 4 muestra un Diagrama de Casos de Uso que describe parcialmente un sistema para la gestión de una biblioteca. El sistema tiene cuatro actores, representados por muñecos: el Lector, el Monitor, el Director y el SI_Administrativo. El primero de ellos no interactúa directamente con el sistema en algunos casos de uso, pero se lo incluye para brindar mayor claridad a la descripción de su funcionalidad.

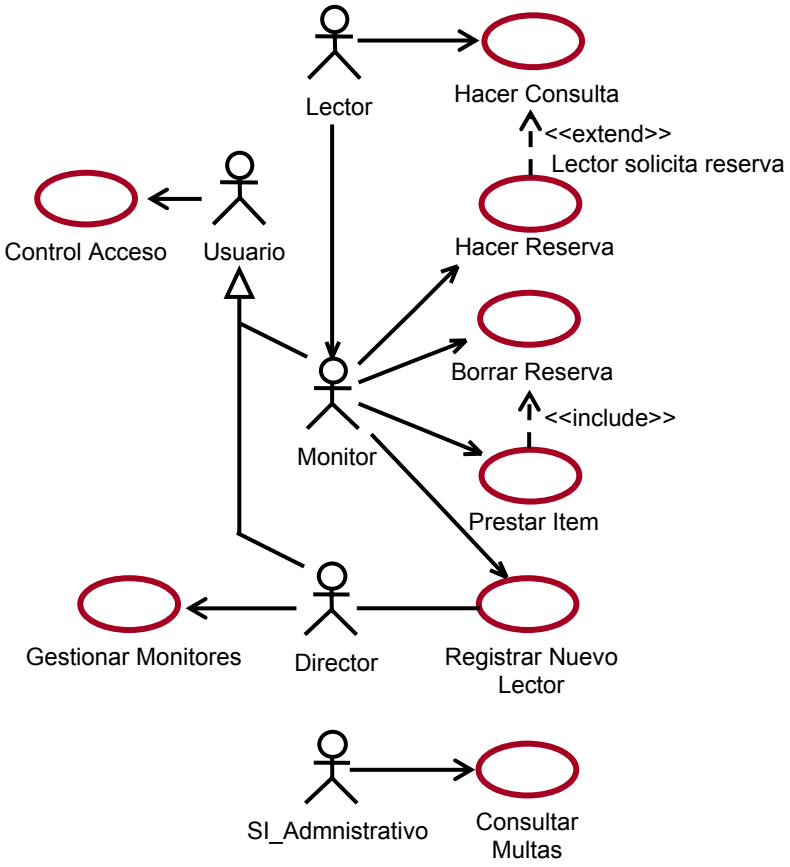


Figura 4. Diagrama de casos de uso

Los casos de uso se representan mediante óvalos, y describen lo que el sistema hace para sus usuarios. En el diagrama se muestran los que corresponden a:

Las operaciones más relacionadas con el Lector:
- Hacer Consulta: Buscar un recurso en la Biblioteca, con opción de reserva.
- Hacer Reserva: Asignar a un Lector un libro para que pueda retirarlo más tarde.
- Borrar Reserva: Eliminar la asignación un libro a un Lector.
- Prestar Ítem: Entregar un recurso a un Lector.
- Registrar Nuevo Lector: Registrar un nuevo Lector en el sistema y fijar sus atributos.

Una operación que corresponde al Director:
- Gestionar Monitores: Dar de alta y de baja a los Monitores y administrar sus atributos.

Una operación común a varios tipos de usuario:
- Control Acceso: Solicitar y verificar identificación (*login*) y clave (*password*).

Una operación que corresponde a un actor no humano:
- Consultar Multas: Obtener información sobre las multas de los Lectores, con el fin de tenerlas en cuenta en las actividades de la administración (contabilidad, certificados de paz y salvo, etc.).

Una descripción más completa incluiría otros casos de uso de índole administrativa como Catalogar Título, Retirar Recurso, Reponer Recurso, Hacer Inventario, etc. Todos los casos de uso se pueden encerrar en un rectángulo con el fin de delimitar el sistema.

Entre los actores y los casos de uso se establecen **asociaciones**, que se representan mediante una línea sólida e indican cuáles actores participan en un caso de uso. Todo caso de uso tiene siempre un actor (y sólo uno) que lo "dispara", denominado iniciador, siendo conveniente identificarlo en los casos de uso que tienen varios actores, ya sea etiquetando su asociación con la palabra "iniciador", o como en la figura, usando una flecha para representarla. En el caso de uso Registrar Nuevo Lector, está claro que el actor iniciador es el Monitor; el Director también participa en el caso de uso, pero sólo cuando recibe el nuevo carné del Lector, para su firma, luego de que el Monitor ha ingresado toda la información correspondiente.

Entre los casos de uso también se pueden establecer **relaciones**, las cuales son de tres tipos: inclusión, extensión y generalización. La relación de Inclusión se representa con una flecha de línea discontinua etiquetada con el estereotipo[2] «include». Una relación de Inclusión desde el caso de uso A hacia el caso de uso B, indica que el comportamiento descrito en el caso de uso B es incluido en el caso de uso A. Tal es la situación con el caso de uso Borrar Reserva del ejemplo, que se "ejecuta" cuando un Monitor cancela la reserva porque el Lector ya no requiere un título, y como parte del caso de uso Prestar Ítem, cuando el Monitor entrega un título reservado y debe cambiar su estado de "Reservado" a "Prestado".

La relación de Extensión es representada también por una flecha discontinua, etiquetada con el estereotipo «extend». Una relación de Extensión desde un caso de uso C hacia un caso de uso D, indica que el caso de uso D puede incluir (condicionado al cumplimiento de condiciones específicas establecidas en la extensión) el comportamiento del caso de uso C. Esta es la situación para el caso de uso Hacer Consulta, que es extendido por el caso de uso Hacer Reserva cuando el Lector ha encontrado un libro disponible y desea reservarlo: cuando se cumple la condición "Lector solicita reserva", Hacer Reserva extiende a Hacer Consulta.

La relación de Generalización desde un caso de uso E hacia un caso de uso F indica que E es una especialización de F. Se representa mediante una flecha con la línea sólida y la cabeza cerrada y vacía (un triángulo), que es la notación de generalización.

Una vez identificados los actores y los casos de uso en el diagrama, se detallan estos últimos, normalmente utilizando una descripción textual aunque para casos de uso más complejos puede usarse un Diagrama de Actividad.

La descripción de los casos de uso de un sistema no es homogénea ni en el tiempo ni en el espacio. Su nivel de detalle se incrementa a medida que se avanza en el proceso de desarrollo, y en un momento dado es posible tener un mayor nivel de detalle para ciertos

[2] Ver Sección 4.3.

casos de uso, los más críticos, mientras que otros menos importantes se dejan para más tarde.

En [15] se proponen dos clasificaciones para los casos de uso, según su nivel de detalle y el nivel de abstracción de su descripción.

De acuerdo al nivel de detalle se tienen dos categorías:

a) Casos de Uso de Alto Nivel

Describen las interacciones entre los actores y el sistema de manera muy breve, usando dos o tres frases. Son utilizados durante las fases iniciales de captura de requerimientos con el fin de obtener rápidamente una visión de la funcionalidad y el grado de complejidad del sistema.

El siguiente ejemplo ilustra el formato para la descripción de los casos de uso de alto nivel:

Caso de uso:	**Hacer Reserva**
Actores:	Monitor (iniciador)
Tipo:	Primario
Descripción:	El Monitor solicita al sistema reservar un ítem solicitado por un Lector. El Sistema verifica la información del ítem y del Lector. El Sistema verifica e informa la disponibilidad del ítem solicitado. El Sistema modifica el estado del ítem, asignándolo al Lector.

En el campo Actores deben incluirse todos los actores que están asociados al caso de uso, señalando cuál de ellos es el iniciador.

El Tipo corresponde a una categoría asignada a los casos de uso dependiendo de su importancia, y que es la base para establecer un orden de prioridad en el momento de planificar su implementación. Los tipos son:

- **Primario**. Representa una interacción principal y común en el sistema.
- **Secundario**. Representa una interacción menor o de rara ocurrencia.
- **Opcional**. Representa una interacción que puede no ser abordada.

b) Casos de Uso Extendidos

Describen las interacciones con mayor detalle que los de alto nivel, enumerando paso a paso los eventos que se presentan durante una ocurrencia típica del caso de uso.

El siguiente ejemplo ilustra el formato para la descripción de los casos de uso extendidos:

Información General

Caso de Uso:	**Hacer Reserva**
Actores:	Monitor (iniciador)
Propósito:	Asignar a un Lector un libro para que pueda retirarlo más tarde.
Resumen:	El Monitor ingresa la información del Lector y del libro solicitado. El Sistema verifica la información del ítem y del Lector, verifica e informa la disponibilidad del libro solicitado, y modifica el estado del ítem, asignándolo al Lector.
Tipo:	Primario y abstracto.
Referencias cruzadas:	Funciones: R1.1, R1.6. Casos de Uso: Control de Acceso, Hacer Consulta.

Precondiciones

El sistema debe contar con la siguiente información:
- Información del Lector: Código, nombre, estado (Activo, Suspendido, etc.).
- Información de los libros: Código, título, estado (Disponible, Reservado, etc.).

El Monitor debe ejecutar el caso de uso Control de Acceso.
Puede iniciarse con el caso de uso Hacer Consulta.

Flujo Principal

- Este caso de uso empieza cuando el Monitor elige en el menú principal del Sistema la opción Reserva.
- El Sistema presenta al Monitor el Formulario de Reserva de la Figura 5, que solicita el código del Lector y el código del libro a reservar.
- El Monitor captura total o parcialmente la información de reserva, y selecciona una de las opciones del final del formulario: **Aceptar** y **Cancelar**.
- Si elige la opción **Aceptar**, Subflujo S1: Verificar Lector y Estado de un Libro.
- Si elige la opción **Cancelar**, Subflujo S2: Cancelar la Reserva de un Libro.

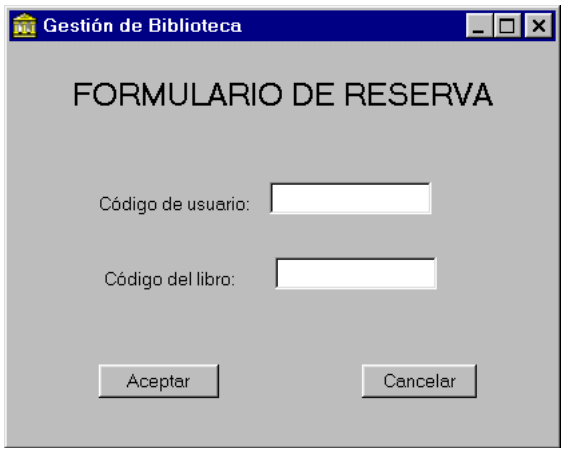


Figura 5. Formulario de Reserva

SubFlujos

S1: Verificar Lector y Estado de un Libro.

- El Sistema verifica el código del Lector (E1).
- El Sistema verifica el código del libro solicitado (E2)
- El Sistema consulta el estado del libro solicitado.
- Si el libro solicitado está disponible, el Sistema modifica su estado a Reservado, lo asigna al Lector indicado al comienzo, y presenta al Monitor el cuadro de diálogo de la Figura X para informarle del éxito de la operación. Cuando el Monitor pincha Aceptar el Sistema regresa al Menú Principal.
- Si el libro solicitado está reservado o prestado, el Sistema presenta al Monitor el cuadro de diálogo de la Figura Y donde le informa la fecha en la que el libro estará disponible. Cuando el Monitor pincha Aceptar el Sistema regresa al Menú Principal.
- Si el libro solicitado está en proceso por parte del personal de la biblioteca (mantenimiento, clasificación, etc.), el Sistema presenta al Monitor el cuadro de diálogo de la Figura Z donde le informa que el libro está fuera de servicio. Cuando el Monitor pincha Aceptar el Sistema regresa al Menú Principal.

S2: Cancelar la Reserva de un Libro.

- El Sistema despliega el mensaje de la Figura S solicitando confirmar la cancelación.
- El Monitor presiona el botón Aceptar.
- El Sistema regresa al Menú Principal.

Flujos de Excepción

E1: Mensaje de error: código del Lector no es válido.

- El Sistema despliega el mensaje de error de la Figura W informando que el código del Lector no es válido
- El Monitor presiona el botón Aceptar.
- El Sistema regresa al Menú Principal.

E2: Mensaje de error: código del libro no es válido.

- El Sistema despliega el mensaje de error de la Figura Z informando que el código del Libro no es válido
- El Monitor presiona el botón Aceptar.
- El Sistema regresa al Menú Principal.

Algunas observaciones sobre la descripción de los casos de uso extendidos:

- En la información general, cuando se declaran los actores, debe señalarse cuál es el actor iniciador del caso de uso.
- En el resumen puede usarse la descripción del caso de uso de alto nivel correspondiente.
- Las referencias cruzadas indican con cuáles funciones del sistema y otros casos de uso existe relación. Las funciones se nombran utilizando la etiqueta que se les asigna

en la Tabla de Funciones del Sistema, elaborada durante la captura de requerimientos.
- El flujo principal debe comenzar con la frase "Este caso de uso empieza cuando el actor ...", mencionando el iniciador.
- Los Subflujos corresponden a diferentes caminos que sigue la interacción de manera normal.
- Los flujos de excepción describen situaciones que se salen del funcionamiento normal del sistema.
- Es conveniente acompañar la descripción con figuras mostrando las interfaces de usuario (GUI, listados, etc.), ya sea como maquetas, en los casos de uso abstractos, o con gráficos capturados en la pantalla y formatos finales de impresión, en los casos de uso reales.

Los casos de uso extendidos pueden a su vez clasificarse de acuerdo a su nivel de abstracción, en dos categorías:

a) Casos de Uso Abstractos

Describen las interacciones de manera ideal, abstrayendo los detalles de tecnología e implementación, especialmente aquellos relacionados con las interfaces de usuario.

b) Casos de Uso Reales

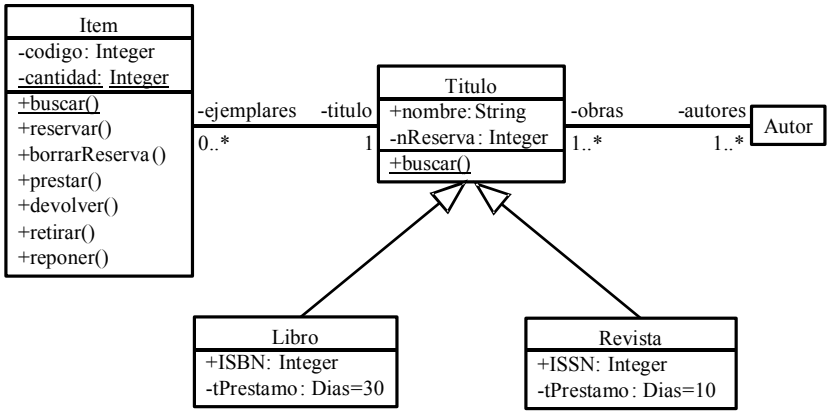
Describen las interacciones en términos de su diseño real, incluyendo los detalles de las tecnologías empleadas en las entradas y salidas. Cuando se utilizan interfaces de usuario, se muestran imágenes capturadas de la pantalla y se discute el manejo de los elementos gráficos.

3.2 Diagrama de Clases

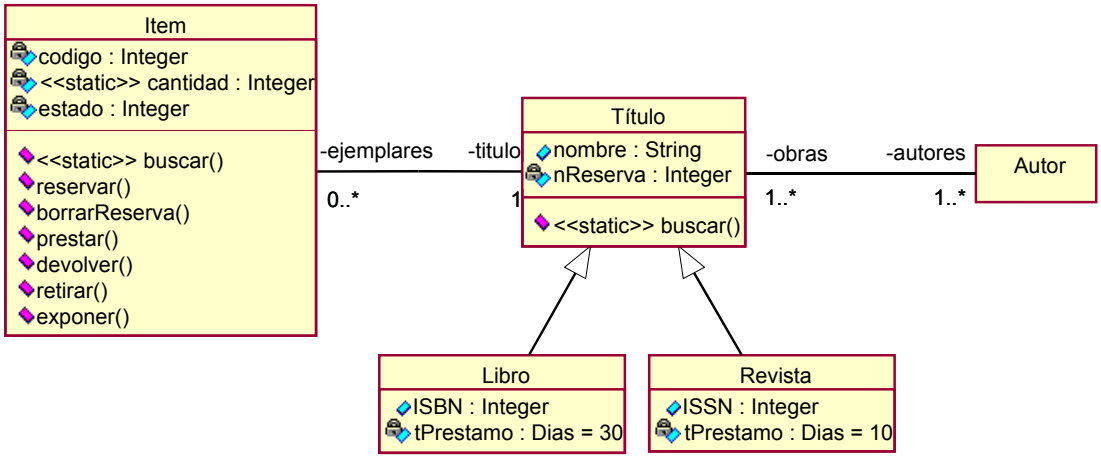
> *Un diagrama de clases es una colección de elementos de un modelo estático declarativo, tales como clases, interfaces, y sus relaciones, conectados como un grafo entre sí y con sus contenidos* [12].

El diagrama de clases representa la estructura de un modelo estático, y no muestra información temporal; sin embargo, puede incluir diagramas de objetos cuyas instancias deben ser compatibles con un diagrama de clases particular. La Figura 6 presenta un diagrama de clases que contiene cinco clases y las relaciones que existen entre ellas.

Las clases son representadas mediante un rectángulo con tres campos, como en el caso de las clases Item y Titulo en la Figura 6. El primer campo contiene el nombre de la clase; el segundo los atributos, indicando nombre y tipo; y el tercer campo contiene las operaciones o métodos de la clase. La representación puede simplificarse mostrando sólo dos campos, que corresponden al nombre de la clase y sus atributos (clases Libro y Revista en ejemplo), o un sólo campo, correspondiente al nombre (clase Autor).



a) Notación estándar



b) Notación de Rational Rose

Figura 6. Diagrama de clases

La visibilidad de los atributos y operaciones se puede indicar mediante los símbolos '+' para el caso de los públicos, '-' si son privados, y '#' si son protegidos (nótese que la herramienta Rational Rose utiliza una notación "gráfica" para el efecto).

Los atributos y operaciones con *alcance de clase*, es decir, que son únicos para todas las instancias de la clase ("estáticos", en C++), pueden señalarse subrayándolos. Tal es el caso en la clase Item del atributo cantidad, que indica cuántos ítems existen, y la operación buscar(), que permite localizar un ítem por su código.

En los atributos se puede mostrar si tienen definidos valores iniciales, como en el caso de tPrestamo en las clases Libro y Revista.

Existen además unas "guías de estilo" que recomiendan, por ejemplo, usar itálicas para las operaciones virtuales (que no son implementadas en la clase actual sino en clases derivadas) o usar negrilla para atributos especiales como las claves candidatas en el diseño de bases de datos.

Las relaciones entre clases pueden ser de cuatro tipos: asociación, generalización, dependencia y refinamiento. A continuación de describe cada una de ellas, más la representación de plantillas.

3.2.1 Asociación

La asociación es una relación que indica que existen enlaces entre los objetos de las clases relacionadas. Por ejemplo, los objetos de una clase contienen referencias a los objetos de otra clase para acceder a sus atributos o para utilizar las operaciones que ellos ofrecen.

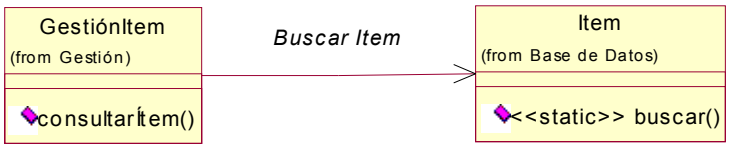
Cuando se trata de **asociaciones binarias**, se representan mediante una línea sólida entre las clases relacionadas; cuando la asociación es terciaria o de mayor orden, se representa mediante un diamante al que se conectan con líneas las clases relacionadas.

En la Figura 6 se muestran dos asociaciones binarias. La asociación entre las clases Ítem y Título es bidireccional, es decir, recíproca: Ítem tiene una referencia a Título, que le permite saber a qué título (obra) corresponde, y a su vez Título posee una referencia a Ítem, que le permite conocer cuántos ejemplares existen de esa obra. En la clase Ítem, la referencia a la clase Título se encuentra en su atributo título, a través del cual puede acceder, por ejemplo, al atributo público nombre de Título. El atributo privado título de Ítem aparece representado en el rol que desempeña la clase Título en la asociación; los roles de las clases en las asociaciones se representan con nombres ubicados en el extremo de las clases correspondientes.

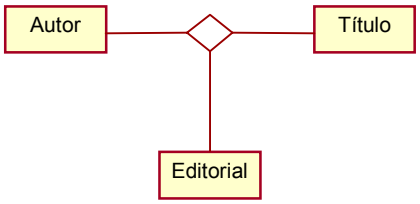
En lenguaje Java, la implementación de la relación que va de Ítem hacia Título sería la siguiente:

```
class Item {
      private Integer codigo;
      private static Integer cantidad;
      private Integer estado;
      private Título titulo;
      public static Item buscar(String nombre);
      ...
}
```

En la Figura 7a se muestra otro ejemplo de asociación binaria. En este caso se trata de una asociación que navega en un solo sentido, por lo que se agrega una flecha en el extremo de llegada. La clase GestiónÍtem utiliza la asociación, denominada *Buscar Ítem*, en la localización de un ítem solicitado, para lo cual invoca la operación estática buscar() de la clase Ítem. La referencia no se encuentra en un atributo de GestiónÍtem, como en el ejemplo anterior, sino en una variable local de la operación consultarÍtem().



a) Asociación binaria



b) Asociación ternaria

Figura 7. Relaciones de asociación

En este caso, la implementación de la relación sería la siguiente:

```
class GestiónItem {
      ...
      public void consultarItem(String nombre) {
            Item itemBuscado;
            ...
            itemBuscado = Item::buscar(nombre);
            ...
      }
      ...
}
```

Una asociación puede incluir a más de dos clases. La Figura 7b muestra una **asociación ternaria**, representada mediante un diamante grande, entre las clases Autor, Título y Editorial.

Con el fin de permitir a sus usuarios expresar con la mayor claridad la semántica de las asociaciones representadas en un diagrama de clases, UML ofrece un conjunto de "adornos" para aquellas, como el sentido, la cardinalidad y los roles.

El sentido se puede representar con una flecha, como ya se mostró, o mediante un triángulo lleno junto al nombre de la asociación. Cuando la asociación se representa mediante una línea simple, se asume que se navega en ambos sentidos, como las de la Figura 6.

La cardinalidad de la relación, es decir, cuántos objetos pueden participar en la asociación, se indica mediante rangos localizados en sus extremos; el símbolo '*' denota un número entero no negativo. En la Figura 6, cada Ítem sólo puede tener un Título, pero cada Título puede tener cero (no se han adquirido o se han dado de baja) o más ejemplares; en la otra asociación, un Título puede tener uno (no puede ser cero) o más autores, y a su vez un Autor puede tener una o más obras.

Los roles, como ya se anotó, permiten expresar el papel que cumplen las clases en la asociación, y se indican mediante nombres ubicados en el extremo de la clase correspondiente.

La **agregación** es un tipo de asociación que expresa la relación "parte de", esto es, una relación entre el todo y sus partes, y se representa agregando un diamante vacío en el extremo que corresponde al todo. En el ejemplo de la Figura 8a, la clase Estante está integrada por objetos de la clase Ïtem, cuyas referencias se tienen en el atributo contenido.

Un caso especial de la agregación es la **composición**, que representa una relación más fuerte entre el todo y sus partes, en la cual las partes sólo tienen sentido como parte del todo, y son construidas y destruidas junto con el todo. La notación utilizada es un diamante lleno en el extremo que corresponde al todo, tal como muestra la Figura 8b. En el ejemplo de la figura, cuando se crea la interfaz gráfica para el inicio de sesión (IU_Inicio), se crean también todos sus componentes: el mensaje, la imagen de la llave, las etiquetas y campos para el nombre de usuario y su clave, y los botones para Aceptar y Cancelar; estos componentes no tienen sentido individualmente, pues son parte integral de la interfaz gráfica. La implementación de esta relación se hace normalmente incluyendo las partes como atributos *por valor* [5] del todo:

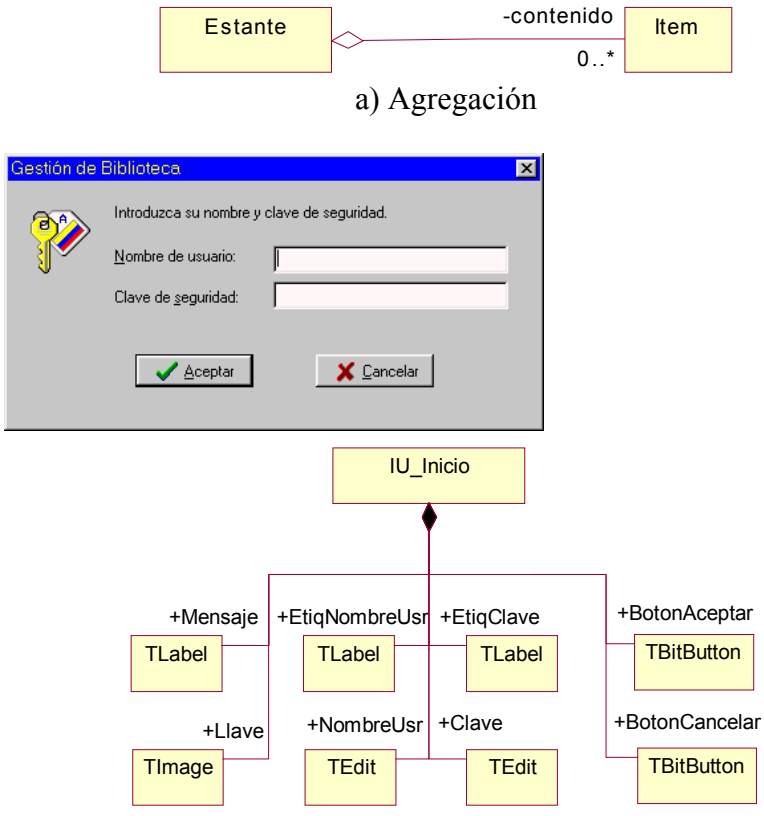


Figura 8. Relaciones de agregación

```
class IU_Inicio extends TForm {
      TLabel Mensaje = new TLabel();
      TImage Llave = new TImage();
      TLabel EtiqNombreUsr = new TLabel();
      TEdit NombreUsr = new TEdit();
      TLabel EtiqClave = new TLabel();
      TEdit Clave = new TEdit();
      TBitButton BotonAceptar = new TBitButton();
      TBitButton BotonCancelar = new TBitButton();
      ...
}
```

Por el contrario, en el ejemplo de la Figura 8a hay gran independencia entre los objetos de las clases Estante e Ítem: un Estante puede estar vacío, y los Ítems pueden cambiar de Estante o incluso no estar en ninguno (está prestado, en reparación, etc.); la implementación de esta agregación "débil" se hace mediante atributos *por referencia* [5]:

```
class Estante {
      private Item contenido[CAP_MAX];
      ...
}
```

3.2.2 Generalización

La generalización es una relación entre un elemento más general y un elemento más específico y sirve para representar la *herencia* entre clases. El símbolo utilizado es una línea sólida entre las clases relacionadas, con un triángulo vacío en el extremo que corresponde a la clase más general.

Esta es llamada también una relación "es-un(a)", pues siempre debe poderse usar esta expresión entre las clases relacionadas. En el ejemplo de la Figura 6 se utiliza la herencia para modelar el hecho de que los Títulos de una biblioteca, que tienen todos atributos y operaciones comunes (nombre, número de reserva, etc.), se clasifican en dos categorías, Libros y Revistas, las cuales se diferencian por sus características (ISBN vs. ISSN) y por su manejo (los libros se prestan por 30 días pero las revistas sólo por 10); en el diagrama, Libro *es-un* Título al igual que Revista, con lo cual ambos heredan los atributos y operaciones de su padre, pero agregan atributos diferentes. También puede suceder que las clases hijas agreguen nuevas operaciones o incluso redefinan las de su padre. En la Figura 9 se muestra un caso de herencia múltiple, donde Oso Panda hereda de Oso y también de Herbívoro.

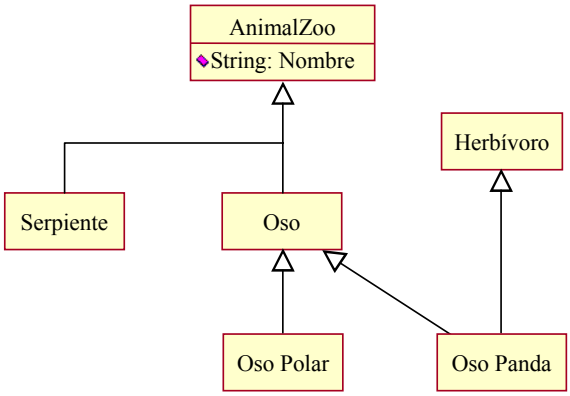


Figura 9. Relaciones de generalización

3.2.3 Dependencia

La dependencia es una relación en la cual un elemento depende del otro, de tal manera que un cambio en el elemento independiente afecta al elemento dependiente. Se representa mediante una línea discontinua con una flecha en el extremo correspondiente al elemento independiente, y se puede etiquetar con un estereotipo para identificar el tipo de dependencia.

En el ejemplo de la Figura 10a, se muestran las relaciones de dependencia entre los paquetes (ver Sección 3.4) con los que se ha estructurado una aplicación: el paquete de Ventanas, que agrupa todas las clases que manejan las interfaces gráficas de usuario, el paquete Gestión, que contiene todas las clases responsables de la gestión de información, y el paquete Base de Datos, donde están la clases que contienen la información. El paquete Gestión depende del paquete Base de Datos puesto que sus clases acceden a atributos o invocan operaciones de las clases de este último. Una situación similar se presenta entre el paquete Ventanas y el paquete Gestión.

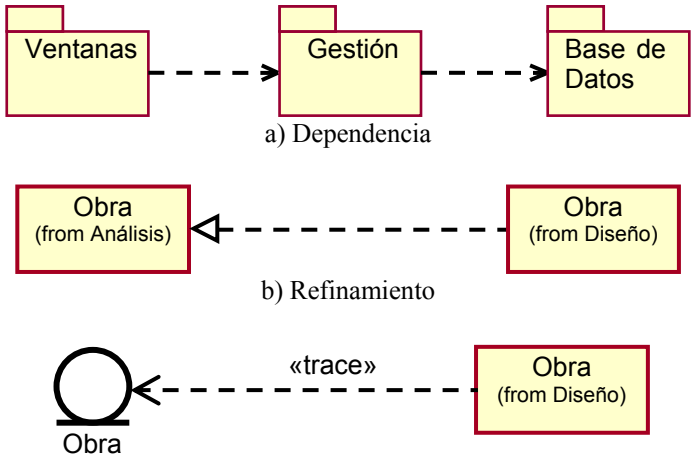


Figura 10. Relaciones de dependencia y refinamiento

3.2.4 Refinamiento

El refinamiento, también llamado *realización*, es una relación entre dos descripciones de un mismo elemento, ubicadas en diferentes niveles de abstracción; la descripción más refinada *realiza* la descripción más abstracta. Se representa mediante una línea discontinua con un triángulo vacío en el extremo de la clase más general o abstracta.

La Figura 10b muestra un ejemplo de relación de refinamiento entre una Clase Obra, obtenida durante el proceso de análisis del sistema y por tanto parte integrante del modelo de análisis, y la Clase Obra, como parte del modelo de diseño. Este tipo de representación es muy útil para soportar el rastreo de los modelos del sistema.

Una representación alternativa del mecanismo de rastreo se muestra en la Figura 10c, donde se utiliza una relación de dependencia etiquetada con el estereotipo «trace». En este ejemplo, la Clase Obra en e modelo de análisis ha sido representada mediante el estereotipo de clase entidad introducido por RUP.

3.2.5 Plantillas

Las plantillas[3] es un concepto que es soportado por algunos lenguajes, como C++, y que consiste básicamente en una clase *parametrizada*, a través de la cual se puede especificar un grupo de clases con una cierta funcionalidad, y que sólo da lugar a clases reales cuando se definen los parámetros.

Los parámetros son representados en un rectángulo de líneas discontinuas localizado en la esquina superior derecha de la clase.

En el ejemplo de la Figura 11, la clase parametrizada Array tiene los parámetros T (una clase) y n (un entero); con esta plantilla se pueden crear arreglos de clases cuyas instancias pueden tener hasta *n* elementos.

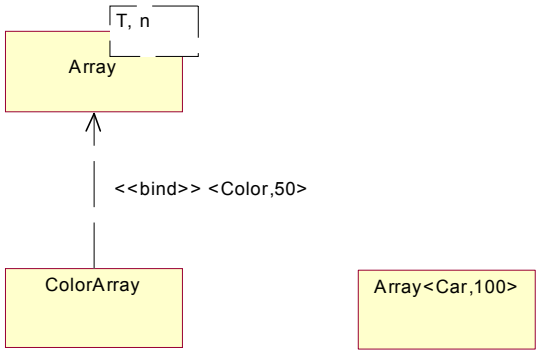


Figura 11. Plantilla

[3] Templates

En la misma figura se han representado de distinta manera dos clases creadas a partir de la plantilla. La clase Array<Car,100> indica directamente cuál es la plantilla con la que ha sido creada y con cuáles parámetros. Por su parte, para la clase ColorArray se muestra una relación de dependencia etiquetada con el estereotipo «bind» seguido por los parámetros utilizados.

3.3 Diagrama de Objetos

Si bien la estructura estática de los modelos está integrada por clases y no por objetos, frecuentemente es necesario utilizar diagramas de objetos con el fin de ilustrar cómo se instancia en un momento dado un diagrama de clases. La notación utilizada es básicamente la misma que en los diagramas de clases, con algunas diferencias para identificar los objetos, como se muestra en la Figura 12, donde la parte superior representa una relación entre las clases Autor y Título, y la parte inferior una instancia de esa relación en la que participan un objeto de la primera clase (gabo) y dos de la segunda (novela100 y cuento2).

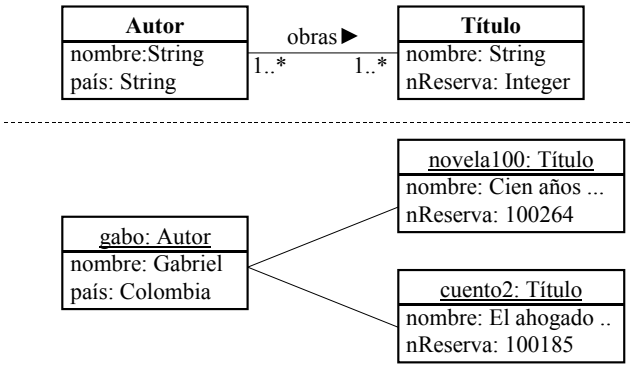


Figura 12. Diagrama de objetos

Tal como se indica en el ejemplo, los objetos son mostrados como una clase, pero con el nombre subrayado e indicando la clase a la que pertenecen separando clase y nombre con dos puntos. También es posible omitir el nombre, dejando sólo los dos puntos y la clase subrayados. Otra diferencia reside en los atributos, pues en los objetos se indica su nombre y su valor actual.

3.4 Paquetes

Los paquetes son un mecanismo de estructuración, utilizados para agrupar cualquier tipo de elementos de los modelos que tienen alguna relación semántica, incluyendo otros paquetes. Entre ellos se pueden establecer relaciones de dependencia (ver Figura 10a), refinamiento y generalización, y también pueden ofrecer interfaces. En el ejemplo de la Figura 13, las clases GestiónÍtem, GestiónLector y GestiónMonitor están contenidas en el paquete Gestión.

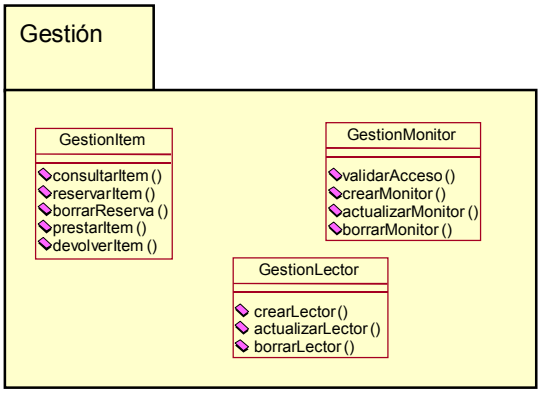


Figura 13. Diagrama de paquetes

3.5 Interfaces

Las interfaces son un elemento de la mayor importancia para la construcción de sistemas bien estructurados, en los cuales juegan el papel de contratos entre los usuarios de los servicios ofrecidos por una clase o un subsistema, y sus implementaciones. Su representación consiste en un pequeño círculo, unido con una línea sólida al elemento que la soporta u ofrece (clase, paquete o componente), junto con el nombre de la interfaz. Por su parte, los elementos clientes de la interfaz se conectan a ella mediante una relación de dependencia.

En el diagrama de la Figura 14, el subsistema Gestión (representado por un paquete) ofrece la interfaz GestiónÍtem, con las cinco operaciones representadas debajo de la interfaz; esta interfaz es utilizada por el subsistema Ventanas, donde están las clases que manejan los menús y demás elementos gráficos de interacción con los usuarios. La interfaz puede ser implementada en el subsistema Gestión mediante una única clase (e.g. GestiónÍtem de la Figura 13) o mediante varias clases como se muestra en el ejemplo, todo lo cual es transparente a sus clientes.

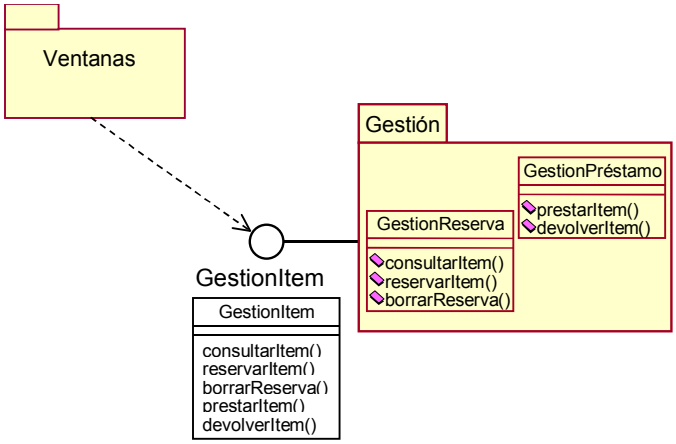


Figura 14. Interfaz

3.6 Diagrama de Secuencias

Mientras que los diagramas anteriores permiten modelar la estructura de un sistema, representando su configuración estática, el comportamiento de éstos, es decir, su dinámica, se modela utilizando Diagramas de Secuencia, Diagramas de Colaboración, Diagramas de Estados y Diagramas de Actividad.

Un Diagrama de Secuencias contribuye a la descripción de la dinámica del sistema en términos de la interacción entre sus *objetos*. Esta interacción se lleva a cabo a través de *mensajes*, que en el mundo de la orientación a objetos no significan lo mismo que en los protocolos de comunicación; un mensaje generalmente se implementa mediante la invocación de una operación desde el objeto "fuente" al objeto "destino".

En el Diagrama de Secuencias aparecen desplegados de manera horizontal los objetos que participan en la interacción, y cada uno de ellos tiene un eje vertical que corresponde al tiempo. Los mensajes entre los objetos se representan mediante flechas etiquetadas con el nombre de la operación, la señal o la acción de interacción correspondiente. El formato de la flecha permite diferenciar el tipo de mensaje, que puede ser (Figura 15):

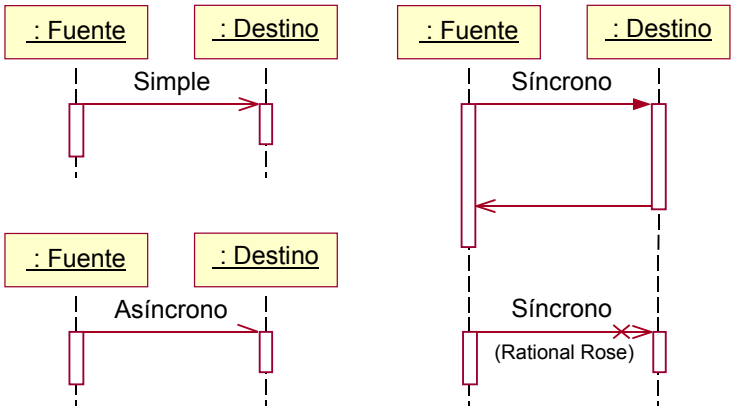


Figura 15. Tipos de mensaje

- Simple: Usado cuando no se conocen los detalles del tipo de comunicación o no son relevantes en el diagrama. También para representar el retorno de un mensaje síncrono.
- Síncrono: Representa la invocación de una operación en la cual el objeto invocante se queda bloqueado esperando la terminación de la misma. Opcionalmente se puede representar el retorno de la operación con un mensaje simple. (Nótese que el símbolo soportado por la herramienta Rational Rose es distinto).
- Asíncrono: Representa la invocación no bloqueante, cuando el objeto que invoca la operación continúa de inmediato su hilo de ejecución, sin esperar respuesta ni que la operación sea ejecutada por el objeto invocado.

Existen dos formas de Diagramas de Secuencia: la forma genérica y la forma de instancia. En la forma genérica, se describen todas las alternativas que pueden presentarse en la interacción entre un conjunto de objetos; eso implica que el diagrama puede contener ramificaciones, condiciones y bucles; las condiciones, representadas por expresiones booleanas entre corchetes, determinan si un mensaje puede ser enviado. En la forma de instancia, por el contrario, sólo se describe un posible escenario; el flujo de control fluye de manera determinista; por consiguiente, para describir diferentes escenarios de interacción entre un grupo de objetos se requieren varios diagramas en forma de instancia.

El Diagrama de Secuencias de la Figura 16 representa en forma de instancia un escenario exitoso de control de acceso de un monitor al sistema de gestión de la biblioteca. En él participan objetos de las clases Monitor (los actores también son clases), la interfaz gráfica para control de acceso (IU_Inicio), la interfaz gráfica del menú principal (IU_MenúPpal), la gestión del monitor (GestiónMonitor), y la tarjeta con la información del monitor (TarjetaMonitor). Obsérvese que los identificadores de los objetos están subrayados y sólo constan de los nombres de sus clases, que van después de los dos puntos. Los objetos pueden llevar sus propios nombres, sobre todo cuando hay más de un objeto de la misma clase en el diagrama.

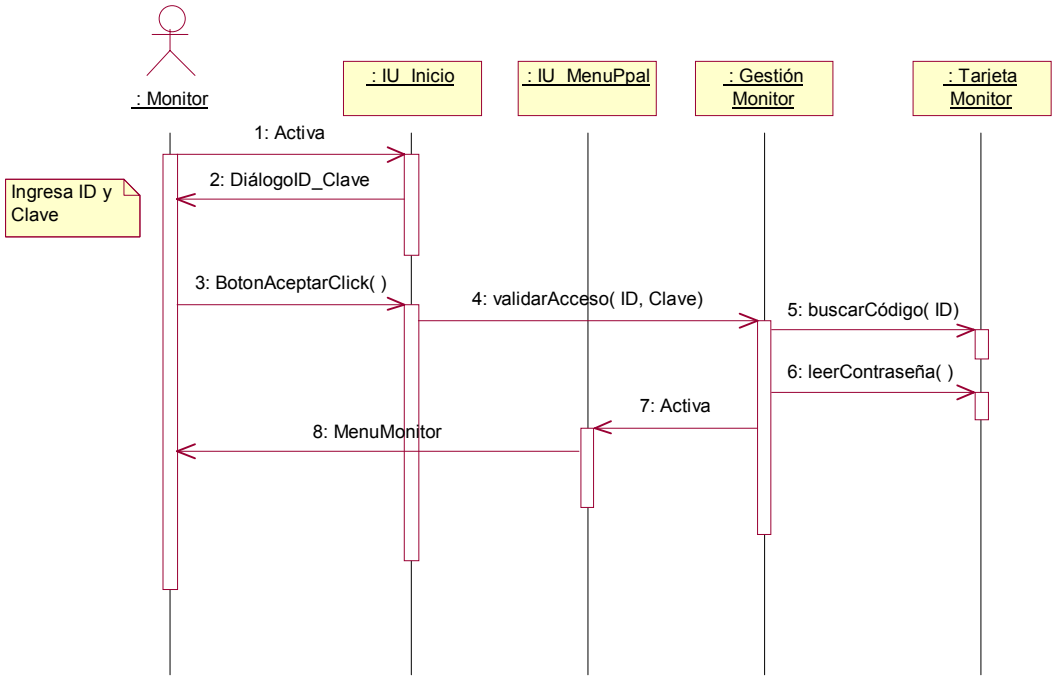


Figura 16. Diagrama de secuencias

La secuencia se inicia cuando el Monitor, al ordenar la ejecución de la aplicación, activa el objeto de la clase IU_Inicio, que le presenta la interfaz de la Figura 8 solicitándole su identificación (*login*) y su clave. El Monitor ingresa la información solicitada y presiona el botón Aceptar, el cual genera un evento que es atendido por IU_Inicio. Éste recoge la

identificación y la clave, e invoca la operación validarAcceso() ofrecida por GestiónMonitor. Ésta operación por su parte utiliza la operación estática buscarCódigo() en TarjetaMonitor para localizar el objeto que corresponde a la identificación entregada por el Monitor, y a continuación obtiene en este objeto la contraseña que tiene programada, invocando la operación leerContraeña(), con el fin de compararla con la que dio el Monitor. Como el escenario que se describe es exitoso, la información es correcta y por tanto GestiónMonitor activa el menú principal (IU_MenúPpal), que despliega en la pantalla el menú del monitor.

La Figura 18a muestra un Diagrama de Secuencias en forma genérica, describiendo la interacción entre objetos que participan en la impresión de archivos. Las condiciones printer fee y printer busy determinan dos secuencias alternativas: la invocación de la operación de impresión en la impresora, o de la operación de almacenamiento en la cola.

3.7 Diagrama de Colaboración

Los Diagramas de Colaboración muestran no sólo los mensajes a través de los cuales se produce la interacción entre los objetos, como en los Diagramas de Secuencia, sino también los enlaces entre los objetos; se trata pues de una mezcla de Diagrama de Objetos y Diagrama de Secuencia. Mientras que en el Diagrama de Secuencia se hace énfasis en el tiempo, en el de Colaboración el énfasis está puesto en la estructura (objetos y sus enlaces).

Al igual que los Diagramas de Secuencia, los de Colaboración pueden asumir las formas genérica y de instancia. Puesto que en la forma genérica se muestra en un solo diagrama las distintas posibilidades de interacción de un conjunto de objetos, las etiquetas de los mensajes tienen una sintaxis compleja, que les permite incluir expresiones de sincronización con otros mensajes, condiciones, iteraciones, y ramificaciones. Esta sintaxis tiene la forma:

```
predecesor '[' condición ']' secuencia ':' expresión
```

Predecesor es una lista de números de secuencia que especifica cuáles mensajes deben haberse enviado y atendido antes del envío del mensaje actual. La *condición* es una expresión booleana que debe ser cierta para habilitar el envío del mensaje. *Secuencia* es una expresión que en principio permite numerar los mensajes, pero también especificar concurrencia, iteración y condiciones (excluyentes) para ramificaciones. En *expresión* aparece el nombre del mensaje que se envía, con sus argumentos y, si es el caso, con el valor que retorna como resultado de la operación invocada.

También es posible indicar cuáles objetos se crean o se destruyen durante la interacción.

En la Figura 17 se presenta el Diagrama de Colaboración correspondiente al Diagrama de Secuencias de la Figura 16.

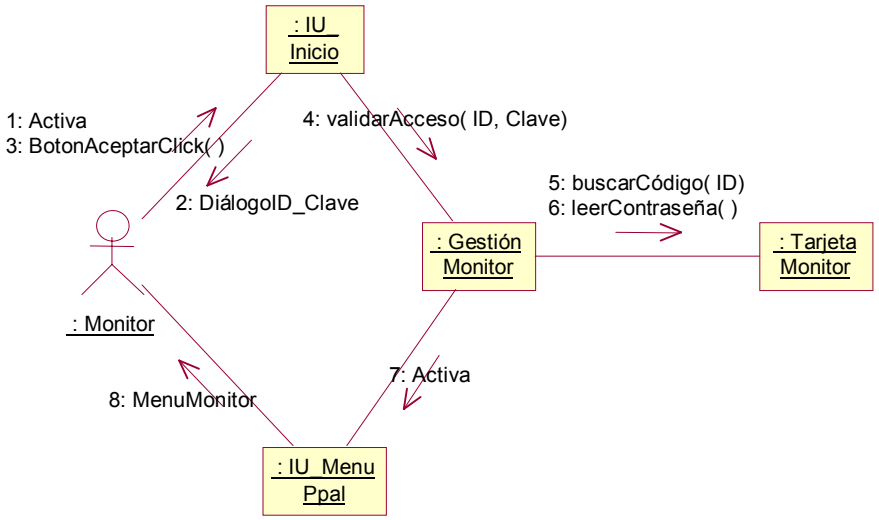
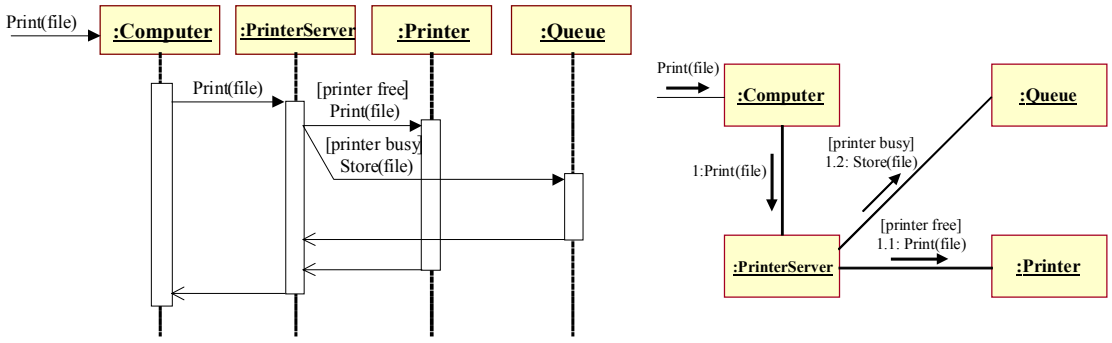


Figura 17. Diagrama de colaboración

Por su parte la Figura 18b muestra el Diagrama de Colaboración en forma genérica correspondiente al Diagrama de Secuencias que la acompaña. En él puede observarse las condiciones y el uso de los números de secuencia para especificar el orden y anidamiento de los mensajes: los mensajes 1.1 y 1.2 están anidados con respecto al mensaje 1, es decir, hacen parte de las operaciones requeridas para ejecutarlo.



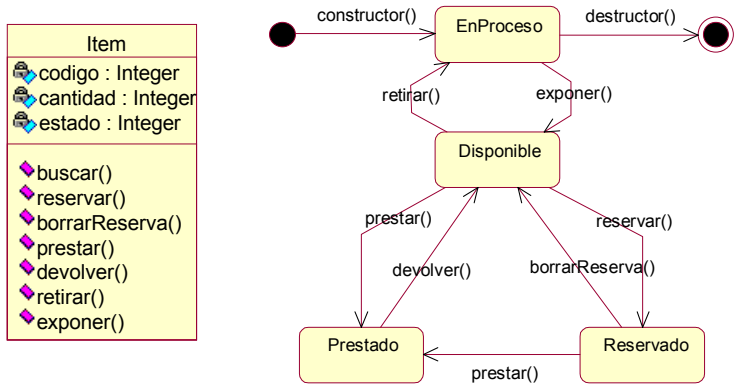
a) Diagrama de Secuencias b) Diagrama de Colaboración

Figura 18. Diagramas en forma genérica

3.8 Diagrama de Estados

Mientras que un Diagrama de Secuencias describe parte de la dinámica de un sistema en términos de la interacción entre *varios objetos* del sistema, generalmente de distintas clases, el Diagrama de Estados permite describirla en términos del ciclo de vida de *un objeto de una clase*, mostrando los estados que éste puede tener y los estímulos que dan lugar a los cambios de estado.

El diagrama de la Figura 19b se representa el comportamiento de la clase Ítem mostrada en la Figura 19a. Cuando la biblioteca adquiere un nuevo ítem se crea un objeto Ítem (usando la operación de construcción de la clase) que lo representa al interior del sistema, el cual queda inicialmente en el estado EnProceso en tanto el ítem real es preparado para entrar en servicio. Cuando el ítem está listo se lo lleva al estante que le corresponde, y en el sistema se invoca la operación exponer() para dejarlo en estado Disponible, que indica que está al servicio de los lectores. En Disponible, el ítem puede ser reservado o prestado a un lector, o sacado de circulación para mantenimiento o baja definitiva; estas acciones y los cambios de estado correspondientes están soportados por las operaciones reservar(), prestar() y retirar(), respectivamente. Desde el estado Reservado se puede regresar a Disponible, a través de borrarReserva() cuando un lector cancela su reserva, o bien se puede avanzar al estado Prestado, cuando el lector hace efectiva su reserva y se lleva el ítem. Por su parte, el estado Prestado sólo tiene una transición de salida posible, hacia Disponible, cuando el lector devuelve el ítem prestado. Finalmente, el ciclo de vida del objeto termina cuando, estando en el estado EnProceso, se le da de baja definitiva y se invoca a su destructor. Nótese que la clase Ítem posee un atributo estado, donde se registra el estado del objeto.



a) Clase Ítem b) Diagrama de estados

Figura 19. Diagrama de estados simple

Como puede apreciarse en la figura, los estados son representados por rectángulos redondeados con su nombre dentro; hay además dos estados especiales: el estado inicial o de creación del objeto, representado con un círculo lleno, y el estado final o de destrucción del objeto, representado con un círculo lleno rodeado por otro círculo. Las transiciones se representan con flechas que van desde el estado de origen hasta el de destino, y que se etiquetan con el estímulo que las produce; en el caso del ejemplo, los estímulos corresponden a la invocación de operaciones del objeto.

En el ejemplo de la Figura 19 se ha modelado que un ítem se saca de circulación, invocando la operación extraer(), sólo cuando está en Disponible. Si se quisiera dar la posibilidad de invocar la operación extraer() en cualquier estado, para considerar por ejemplo la pérdida del ítem, en un diagrama de estados simple habría que crear

transiciones desde cada uno de los estados hasta el estado EnProceso, etiquetados con la operación extraer(); en un diagrama con un número importante de estados, esto afectaría de manera muy negativa su legibilidad. David Harel propuso en 1987 los *Statecharts* [16], un formalismo gráfico para representar máquinas de estados finitos que incluye además los conceptos de estados jerárquicos anidados y concurrencia de estados. Estos nuevos conceptos simplifican enormemente el modelado y constituyen una herramienta muy poderosa para representar el comportamiento de los sistemas. UML ha adoptado los *Statecharts* para los Diagramas de Estado, permitiendo entre otras cosas el uso de sub-estados y la creación de jerarquías de estados.

La Figura 20a representa el comportamiento de los objetos de la clase Ítem considerando la invocación de extraer() en cualquier estado. El modelo consta de dos estados principales, EnProceso y Circulando, que representan respectivamente cuándo el ítem está en manos de la administración de la biblioteca (o en situación de ser dado de baja) y cuándo está al servicio de los lectores. El estado Circulando consta, a su vez, de los sub-estados inicial (el círculo lleno), Disponible, Reservado y Prestado. La transición desde el sub-estado inicial a Disponible señala que éste es el sub-estado de destino cuando un ítem entra al estado Circulando por efecto de la invocación de la operación exponer(). Por su parte, la invocación de la operación retirar(), en cualquiera de los sub-estados de Circulando, genera una transición hacia el estado EnProceso.

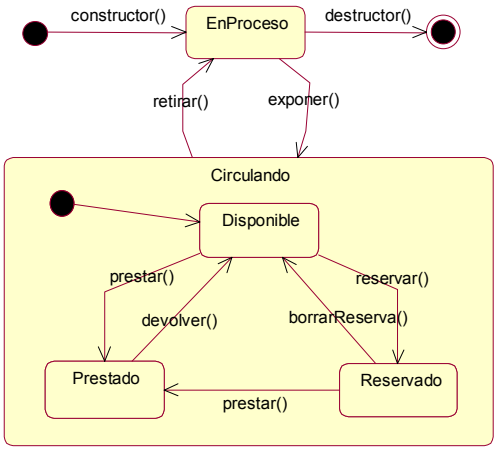
La Figura 20b presenta un Diagrama de Estados más completo. El sub-estado Libre, dentro del estado Activo, muestra los tres campos que se pueden representar en un estado: el campo superior corresponde al nombre, el campo central a las variables del estado, y el campo inferior a las actividades que ocurren dentro del estado. Las variables del estado se pueden listar y asignarles valores iniciales, mientras que en el campo de actividades se pueden especificar eventos entre los cuales se han definido tres estándar: *entry*, *exit* y *do*. El evento *entry* especifica acciones que ocurren cuando se entra al estado, como por ejemplo enviar un mensaje; el evento *exit* especifica acciones que ocurren a la salida del estado, como por ejemplo detener un temporizador; y el evento *do* especifica acciones realizadas durante la permanencia en el estado, como por ejemplo hacer cálculos, las cuales pueden ser interrumpidas por un evento que causa la salida del estado. Las actividades tienen el siguiente formato:

```
nombre-evento lista-argumentos '/' acción
```

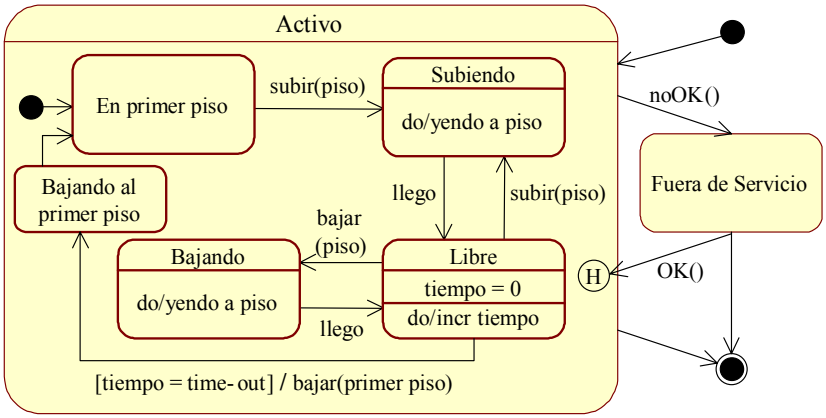
De manera similar a las clases, los estados pueden representarse con sólo dos campos (generalmente nombre y actividades), o con uno (nombre).

La figura muestra también una transición asociada con una condición, que además especifica una acción. Se trata de la transición de Libre a "Bajando al primer piso", que establece que cuando la variable tiempo llega al valor time-out se abandona el estado Libre y se ejecuta la operación bajar(primer piso). En general, las transiciones tienen el siguiente formato:

```
evento '[' condición ']' '/' acción '^' cláusula-envío
```



a) Clase Ítem



b) Clase Ascensor

Figura 20. Diagrama de estados jerárquicos

La condición es una expresión booleana que debe ser cierta para habilitar la ocurrencia de la transición; si se combina con un evento, *ambos* deben ocurrir para disparar la transición. La cláusula de envío es un tipo especial de acción que sirve para enviar mensajes durante la transición.

Cuando un estado tiene una transición de salida sin un evento asociado a ella, esta transición ocurre cuando todas las actividades especificadas dentro del estado han sido realizadas.

Finalmente, en la Figura 20b se representa también el indicador de la historia de un estado, que consiste en la letra H encerrada en un círculo. Cuando una transición de entrada a un estado no llega hasta el borde del mismo, sino que entra hasta el indicador de la historia, significa que el destino de la transición no es el sub-estado inicial sino el sub-estado en el cual se encontraba el objeto dentro del estado en cuestión, antes de abandonarlo. Así pues, el indicador de la historia "recuerda" el último sub-estado ejecutado dentro de un estado. En el ejemplo de la clase Ascensor, la transición

etiquetada con OK() lleva desde el estado "Fuera de Servicio" al sub-estado del estado Activo en el que se encontraba el objeto cuando se invocó la operación NoOK(), por ejemplo, Libre en el tercer piso.

3.9 Diagrama de Actividad

Es utilizado para describir una secuencia de acciones, las cuales pueden corresponder a distintos niveles de abstracción de un sistema: el algoritmo de una operación en una clase, la interacción de un grupo de objetos, la especificación de un caso de uso, las actividades que integran un procedimiento en una empresa, etc.

Aunque sintácticamente los Diagramas de Actividad se definen como una variante de los Diagramas de Estado, pues sus símbolos son en principio los mismos, su semántica es bastante diferente. Aquellos están más orientados a mostrar las acciones, mientras que éstos están centrados en los estados; y los primeros pueden involucrar a objetos de varias clases, mientras que los segundos describen siempre el comportamiento de los objetos de una clase específica.

Los Diagramas de Actividad son en esencia diagramas de flujo, con algunos elementos adicionales que les permiten expresar conceptos como la concurrencia y la división del trabajo. Tal como se muestra en la Figura 21, utilizan los símbolos de estados, denominados *estados de acción*, para describir las actividades, y también usan los símbolos para el estado inicial y el estado final. Tienen condiciones para habilitar las transiciones entre una acción y otra, y además un símbolo para los puntos de decisión, que consiste en un diamante grande con una o más transiciones de entrada y dos o más transiciones de salida etiquetadas con condiciones.

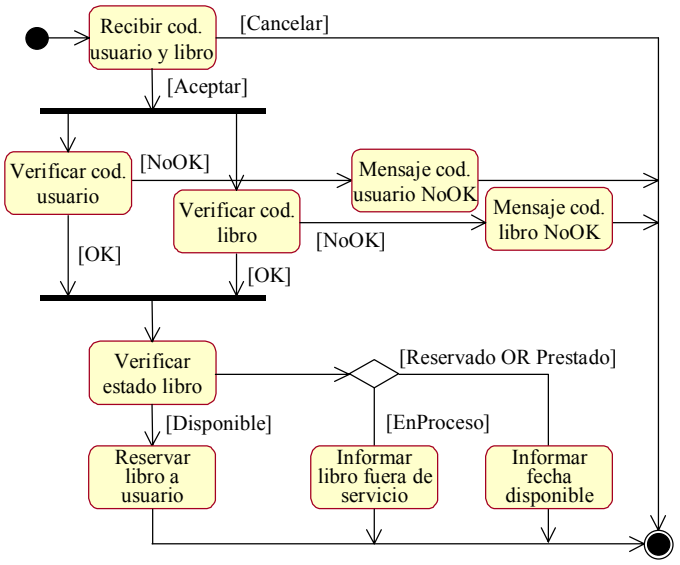


Figura 21. Diagrama de actividad

La Figura 21 especifica el caso de uso Hacer Reserva, descrito en forma textual en la página 15. Aparte de los estados de acción, el punto de decisión y las condiciones, la figura muestra uno de los elementos particulares de los Diagramas de Actividad: las acciones ejecutadas de manera concurrente. La línea gruesa horizontal dibujada en la transición de la condición Aceptar que sale de la acción "Recibir código de usuario y libro", es usada para indicar que la transición resulta en varias acciones paralelas, como son "Verificar código usuario" y "Verificar código libro", las cuales pueden ejecutarse en cualquier orden o en forma simultánea. La misma barra horizontal sirve también para mostrar la unificación de varias transiciones que se sincronizan para dar lugar a una nueva acción, como en el caso de las transiciones de condición OK que salen de estas dos últimas acciones; sólo si *ambas* transiciones se ejecutan tiene lugar la ocurrencia de la acción "Verificar estado libro".

En los Diagramas de Actividad también se puede especificar la división de trabajo o de responsabilidades entre objetos de un sistema o secciones de una organización. Para ello se utilizan los carriles[4], que consisten en divisiones verticales del diagrama etiquetados con el nombre del objeto o sección correspondiente, en los cuales se colocan las acciones que son realizadas por él.

Otros elementos que pueden hacer parte de un Diagrama de Actividad son los objetos, incluidos como insumos o resultados de una actividad o simplemente como afectados por ella, y las señales, que se envían y reciben en las transiciones.

3.10 Diagrama de Componentes

En tanto todos los diagramas descritos hasta ahora muestran los elementos intangibles de una aplicación de programación (clases, objetos, relaciones, estados, actividades, etc.), el Diagrama de Componentes presenta sus elementos tangibles: los archivos. Se lo utiliza, entonces, para describir la estructura física del código de la aplicación en términos de sus componentes (código fuente, binario o ejecutable) y sus dependencias.

Tal como muestra la Figura 22, los componentes son representados mediante un rectángulo adornado con una elipse y dos rectángulos pequeños. En el ejemplo de la figura, el componente ejecutable client.exe depende de los componentes binarios Client.o y grid.o, y de las librerías libc.a y Orbix.a. Por su parte Client.o depende del componente de código fuente Client.cc, éste del archivo de cabecera grid.hh, y finalmente éste del archivo de descripción de interfaces grid.idl. La dependencia de un componente A respecto a un componente B indica que un cambio en B implica que debe modificarse A, normalmente a través de una de las herramientas del entorno de desarrollo como compiladores, encadenadores, etc.

[4] Swimlanes.

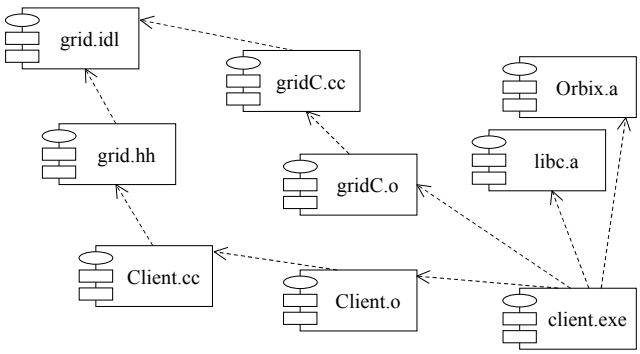


Figura 22. Diagrama de componentes

3.11 Diagrama de Implantación

Como se ha comentado al principio, UML es una notación orientada a sistemas intensivos en programación, por lo que no está dotada de elementos para describir en detalle los componentes físicos de un sistema (circuitos, módulos físicos, tarjetas, etc.). Sin embargo, dado que los componentes de programación requieren de componentes físicos para ser ejecutados, es necesario disponer de un diagrama que describa la arquitectura física del sistema, sobre todo si es distribuido, y cómo se ejecutan sobre ella los componentes de programación.

En un Diagrama de Implantación se muestran nodos, conexiones, componentes y objetos. Los nodos representan objetos físicos con recursos computacionales como procesadores y periféricos; pueden mostrarse como una clase (e.g. una familia de procesadores) o una instancia, por lo que su nombre sigue la misma sintaxis establecida para clases y objetos. Las conexiones son asociaciones de comunicación entre los nodos, y se etiquetan con un estereotipo que identifica el protocolo de comunicación o la red utilizada. Los componentes son archivos de código ejecutable, que residen y se ejecutan dentro de un nodo; se pueden representar relaciones de dependencia entre los componentes que, de manera similar a las dependencias entre paquetes, corresponden al uso de servicios. Los objetos pueden incluirse en el diagrama contenidos en otro objeto, en un paquete o simplemente en un nodo.

En el Diagrama de Implantación de la Figura 23 se muestran cuatro nodos: Servidor Web, Servidor Base de Datos, PC Lector y PC Monitor. La conexión del Servidor Web con el Servidor Base de Datos utiliza el protocolo JDBC, y con el PC Lector se realiza a través de la Web mediante el protocolo HTTP. Por su parte, el PC Monitor accede al Servidor Base de Datos a través de una LAN con TCP/IP. Dentro de cada nodo se han ubicado los componentes de la aplicación que se residen o se ejecutan en cada uno, y se muestran también las dependencias entre estos componentes.

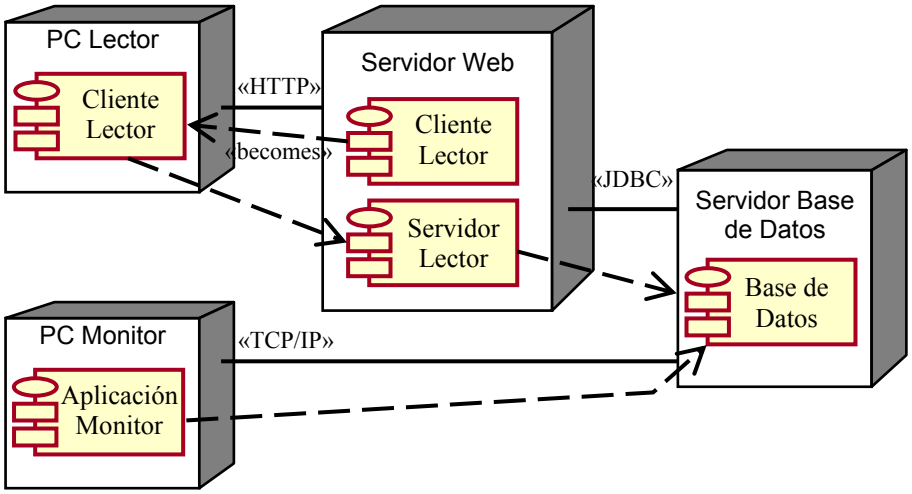


Figura 23. Diagrama de implantación

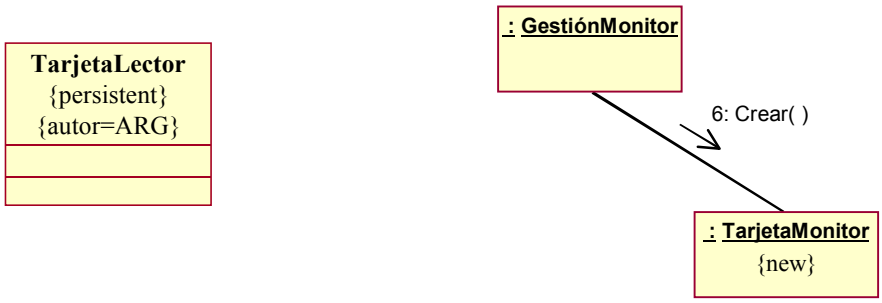
El componente Cliente Lector aparece en Servidor Web y PC Lector porque se descarga desde el primero al segundo (puede ser una página web cliente o una *applet* Java), lo cual se muestra mediante una relación de dependencia con el estereotipo «becomes».

4. Mecanismos de Extensión de UML

Uno de los grandes propósitos de UML ha sido el de convertirse en el lenguaje universal para la representación de los diferentes modelos elaborados durante el desarrollo de una aplicación de programación. Para lograrlo, ha incorporado diversos elementos sintácticos que le permiten expresar una gran variedad de conceptos utilizados en múltiples dominios de aplicación. Este esfuerzo generalista fue sin embargo llevado sólo hasta cierto punto, con el fin de evitar que el lenguaje se tornara demasiado complejo. En consecuencia, con el fin de permitir a sus usuarios adaptarlo y extenderlo para ajustarse a las necesidades o estándares de una organización o un proyecto, sus diseñadores le han incluido tres mecanismos de extensión: valores etiquetados, restricciones y estereotipos.

4.1 Valores etiquetados

Son propiedades asignadas a un elemento, en forma de pares nombre-valor, y se representan usando la sintaxis `{nombre = valor}`, o `{nombre}` cuando es de tipo booleano. En el ejemplo de la Figura 24a, {persistent} es un valor etiquetado booleano predefinido por UML, que se asocia a una clase para indicar que su valor se mantiene entre diferentes ejecuciones del programa, y {autor=ARG} es un valor etiquetado definido por el usuario.



a) Valores etiquetados b) Restricciones

Figura 24. Mecanismos de extensión

4.2 Restricciones

Son reglas que limitan la semántica de uno o más elementos del lenguaje y son representadas entre llaves. En la Figura 24b se muestra un fragmento de un Diagrama de Colaboración, en el que la restricción {new} ha sido aplicada a un objeto de la clase TarjetaMonitor para indicar que éste es creado durante la ejecución de la interacción.

4.3 Estereotipos

Son semánticas asignadas por el usuario a elementos que ya existen en el lenguaje. Algunos estereotipos tienen su sintaxis gráfica propia, como es el caso del estereotipo Actor asociado a las clases que representan usuarios del sistema, cuya representación gráfica es un muñeco. Todos tienen una sintaxis textual que consiste en el nombre del estereotipo encerrado entre los caracteres «».

La Figura 25 muestra cuatro estereotipos de clase, donde las tres primeras tienen también su propia representación gráfica. El nombre del estereotipo se coloca encima del nombre de la clase, con lo cual se asigna a ésta una semántica muy específica, que para el caso es la que ha retomado RUP para las clases de análisis, a saber:

- «Boundary» (Interfaz). Proveen la interfaz con el usuario o con otros sistemas. Dependen del entorno del sistema y se obtienen examinando las relaciones actor-escenario en los casos de uso; más adelante se refinan durante el diseño para considerar los protocolos de comunicación.
- «Entity» (Entidad). Sirven para el almacenamiento de información. Reflejan el mundo real o se crean para tareas internas; no dependen del entorno del sistema y pueden ser independientes de la aplicación. Se obtienen examinando las responsabilidades del sistema en los casos de uso.
- «Control». Coordinan los eventos necesarios para implementar el comportamiento especificado en el caso de uso. Son dependientes de la aplicación. Al comienzo, existe una clase de control para cada par actor-caso de uso.
- «Exception». Gestionan las excepciones que se presentan en la aplicación.

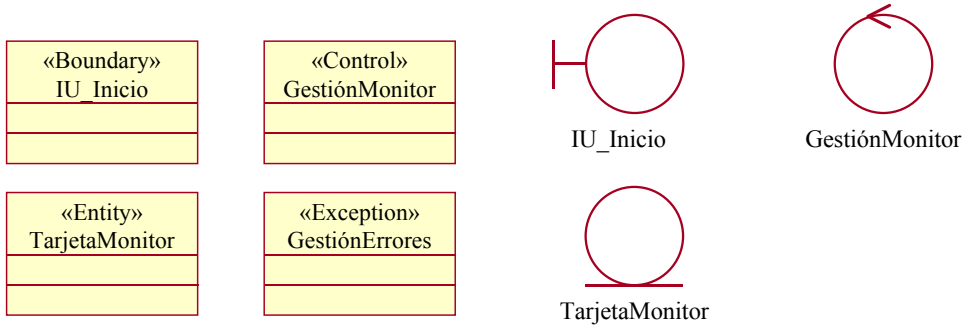


Figura 25. Estereotipos de clase

5. El Proceso Unificado de Rational (RUP)

5.1 Introducción

El Proceso Unificado de Rational (RUP, *Rational Unified Process*) [2], de manera similar a UML, es fruto de los aportes de un gran número de investigadores y empresas de desarrollo de programas. Entre los métodos más importantes que constituyen la base de RUP figuran los siguientes, que como puede verse, cubren diversos aspectos del ciclo de vida de desarrollo:

- Objectory: Método de desarrollo propuesto originalmente por Jacobson [6], caracterizado por ser un método orientado a objetos centrado alrededor de *Casos de Uso.*
- Rational Approach: Método de desarrollo resultante de la unificación de los conceptos desarrollados por Kruchten [17], Booch [5] y Royce [18], entre los que se destacan los de proceso iterativo y desarrollo centrado en la arquitectura del programa.
- SQA Process: Método de pruebas.
- Requirements College: Guías para la gestión de requisitos.

RUP es un proceso de ingeniería de programación que busca asegurar la producción de software de alta calidad, satisfaciendo las necesidades del cliente, y con arreglo a un plan y presupuesto predecibles.

Sus características más importantes son:

- Es un proceso iterativo, basado en el refinamiento sucesivo del sistema.
- Es un proceso controlado, donde juegan un papel de primordial importancia la gestión de requisitos y el control de los cambios.
- Basado en la construcción de modelos visuales del sistema.
- Centrado en el desarrollo de la arquitectura, por lo que maneja el concepto de desarrollo basado en componentes.
- Conducido por los Casos de Uso.
- Soporta técnicas orientadas a objetos y en particular el uso de UML.
- Configurable.
- Fomenta el control de calidad.
- Soportado por herramientas.

5.2 Organización

Indiscutiblemente, en el desarrollo de una aplicación se sigue un proceso en el cual se avanza paulatinamente en la comprensión de la funcionalidad requerida y cómo realizarla, hasta llegar a su construcción. Esto requiere la ejecución de un conjunto de

actividades que se manejan como un proyecto, es decir, con un objetivo final, un plazo y un presupuesto. Como en todo proyecto, es importante contar con puntos intermedios de control a lo largo de su ejecución, denominados hitos, que se establecen cuando se elabora el plan de trabajo y sirven de faro para verificar que el proyecto marcha adecuadamente.

En el modelo en cascada, el proceso de desarrollo avanza en forma secuencial a través de cinco actividades fundamentales: captura de requisitos, análisis, diseño, implementación y pruebas. El modelo plantea que cada actividad debe completarse antes de proceder a la siguiente, por lo cual ellas mismas se convierten en referentes para el avance del proyecto en el tiempo y reciben la denominación de *fases*. Así pues, un proyecto se planifica colocando como hitos la finalización de las distintas fases, donde normalmente se entregan uno o varios productos asociados al desarrollo del sistema.

RUP rompe la secuencialidad de las actividades fundamentales del modelo en cascada al plantear un desarrollo incremental e iterativo, en el cual no es necesario agotar completamente una actividad para iniciar la siguiente. En lugar de ello, se avanza a través de la construcción de prototipos, cada uno de los cuales exige la ejecución parcial de las actividades fundamentales. Puede verse entonces el desarrollo incremental como una serie de iteraciones, cada una de las cuales se realiza siguiendo el modelo en cascada.

Esta estrategia conlleva a que no pueda seguirse utilizando la terminación de las actividades fundamentales para establecer los hitos del proyecto, pues esto sucede hacia el final de su ejecución. Se hace necesario entonces establecer nuevos criterios para definir los puntos de control del proyecto; criterios que estarán determinados por los productos obtenidos en las sucesivas iteraciones.

Por esta razón RUP organiza las actividades de desarrollo siguiendo dos criterios ortogonales ilustrados en la Figura 26. En el eje vertical, se describen lo que hemos venido llamando actividades fundamentales y que en términos de RUP se denominan *componentes*, los cuales establecen cómo avanzar en la conceptualización y construcción del sistema. En la figura, el nivel de intensidad del trabajo en cada componente se representa mediante la amplitud de la gráfica asociada. Corresponden a la estructura estática del proceso de desarrollo, pues definen *qué* acciones se deben realizar.

En el eje horizontal, se describen los criterios para la planeación y control en el *tiempo*. Corresponden a la dinámica del proceso de desarrollo pues establecen *cuándo* se deber realizar las acciones definidas por los componentes.

5.2.1 Organización por Componentes

Los componentes del proceso de desarrollo agrupan las actividades de acuerdo al nivel de abstracción en el que están localizadas y su naturaleza, y establecen *qué* hay que hacer, *quién* debe hacerlo y *cómo* hacerlo. Cada componente se describe en los siguientes términos:

- Artefacto (*artifacts*), que representan cualquier tipo de información generada, modificada o utilizada en el desarrollo del sistema. Por ejemplo: en el componente de Análisis se elaboran las Clases de Análisis.
- Trabajadores (*workers*), que corresponden a los roles (una misma persona puede desempeñar varios roles) que intervienen en el componente. Por ejemplo: En el componente Análisis el Arquitecto participa en la elaboración de las Clases de Análisis.
- Flujos de trabajo (*workflows*) y actividades, que deben ser adelantadas por los trabajadores para obtener los artefactos del componente.

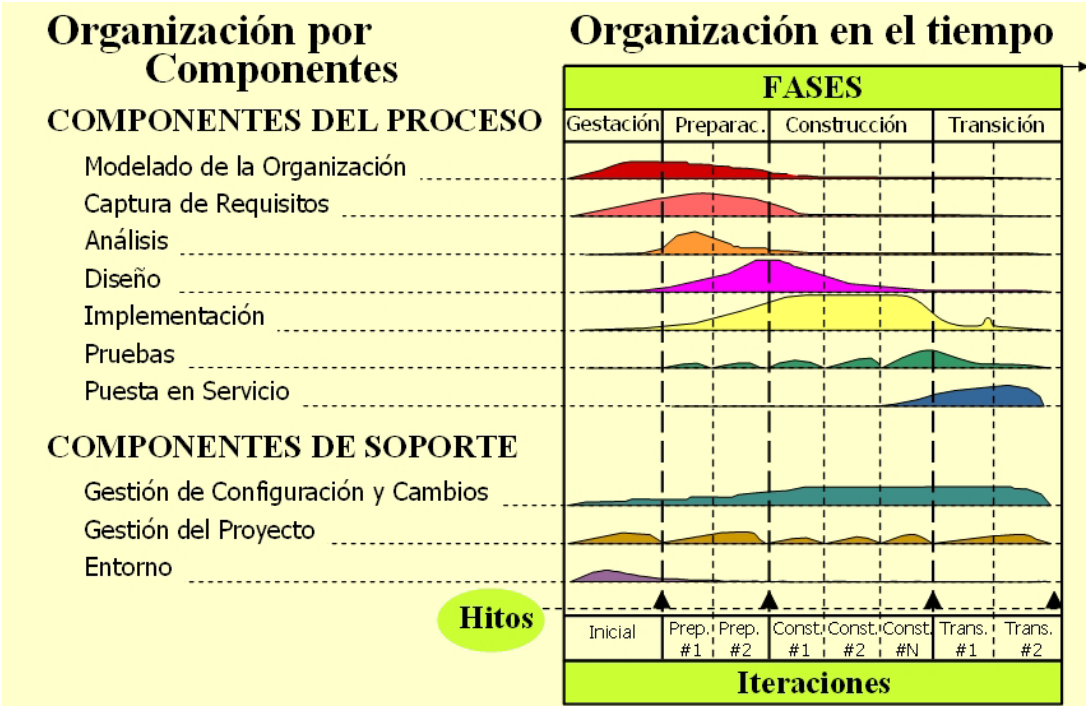


Figura 26. Organización del Proceso Unificado

Existen dos tipos de componentes: los del proceso de ingeniería, que se refieren a las actividades relacionadas en forma directa con la obtención del producto, y los de soporte, que se refieren a las actividades administrativas del proceso.

Los componentes del proceso de ingeniería son siete, a saber:

- **Modelado de la Organización**. Consiste en la identificación y documentación de la estructura y funcionamiento de la organización en la cual operará la aplicación a desarrollar. Su objetivo es brindar un entendimiento a clientes y desarrolladores sobre *cuál es el problema* de la organización, identificar mejoras potenciales y establecer el impacto que la aplicación a desarrollar tendría sobre la organización.
- **Captura de Requisitos**. Su propósito es obtener la descripción de *para qué sirve el sistema*, y lograr un acuerdo entre el equipo de desarrollo y el cliente en este aspecto.

- **Análisis**. En este componente se define la estructura (clases, paquetes, etc.) y comportamiento del sistema. Su propósito es obtener una descripción de *cómo funciona* el sistema.
- **Diseño**. Mientras que Análisis se ha centrado en establecer la funcionalidad del sistema, el componente de Diseño se enfoca a lograr que esa funcionalidad se haga posible sobre una arquitectura física (computadores, redes, etc.) y un entorno de implementación (sistemas operativos, lenguajes de programación, etc.) dados. Su propósito es obtener una descripción de *cómo se construye* el sistema.
- **Implementación**. Construcción del sistema obteniendo los *archivos* ejecutables, de configuración, librerías, etc.
- **Pruebas**. Se verifican los modelos, prototipos y demás artefactos ejecutables del sistema bajo desarrollo.
- **Puesta en Servicio**. En este componente se realizan las actividades requeridas para poner en funcionamiento el producto en las instalaciones del cliente.

La Figura 27 presenta la relación entre los componentes del proceso de ingeniería y los modelos obtenidos. Se destaca el papel central que desempeña el modelo de casos de uso.

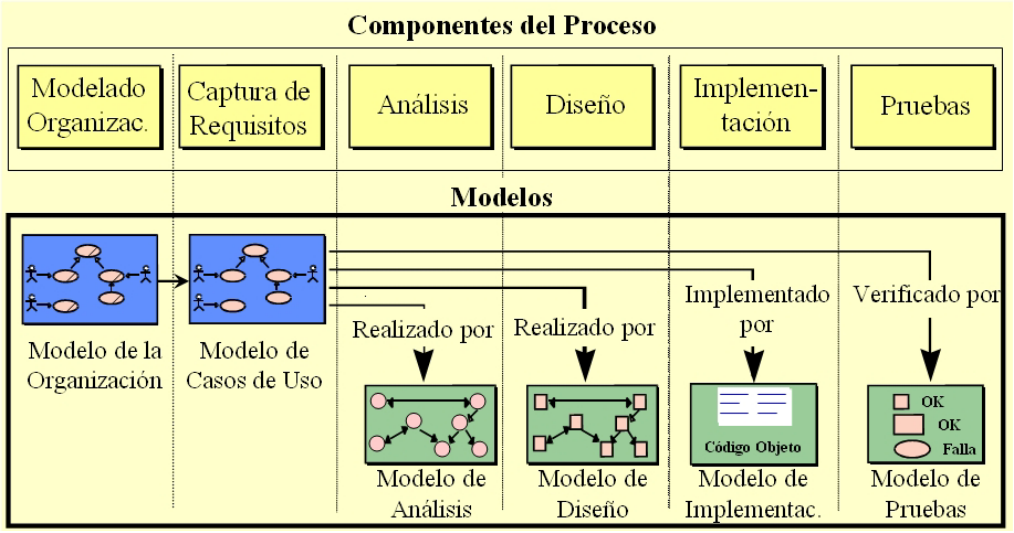


Figura 27. Componentes del proceso y modelos

Por su parte, los componentes de soporte son tres, a saber:

- **Gestión de configuración y cambios**. Lleva control sobre la evolución iterativa del sistema, registrando las modificaciones de sus partes y las configuraciones que dan lugar a los prototipos operacionales.
- **Gestión del proyecto**. Define los aspectos específicos de un proceso de desarrollo iterativo. Para ello brinda un marco de razonamiento para la gerencia de proyectos intensivos en programación, junto con guías prácticas para la planificación,

constitución de equipos de trabajo, ejecución y supervisión de proyectos, y criterios para el manejo de riesgos.

- **Entorno**. Su propósito es establecer la organización del entorno de desarrollo de programación (procesos y herramientas) requerida por el equipo de desarrollo.

5.2.2 Organización en el tiempo

Establece la dinámica del proceso de desarrollo, definiendo los criterios de planeación y control de su ejecución en el tiempo. Está expresada en términos de Ciclos, Fases, Iteraciones e Hitos:

- **Ciclo**: Desarrollo de una nueva versión del producto. Corresponde al ámbito de un proyecto.
- **Fases**: Etapas en el desarrollo de una versión (proyecto). Cada fase termina en un Hito y puede estar dividida en Iteraciones. Las Fases y los Hitos asociados son:

Fase	Hito
Gestación (*Inception*)	Definición de objetivos y factibilidad
Preparación (*Elaboration*)	Elaboración de la Arquitectura
Construcción (*Construction*)	Elaboración del producto
Transición (*Transition*)	Satisfacción del usuario

- **Hito**: Punto de control, donde generalmente se revisan los resultados del proceso y se decide si se avanza a la siguiente Fase o Iteración.
- **Iteración**: Unidad de desarrollo del producto, en la que se obtienen o refinan uno o más artefactos del sistema.

La división principal del proceso de desarrollo en el tiempo son las fases. A continuación se describe brevemente el propósito de cada una [19]:

- **Gestación**. Delimitar el alcance del proyecto y elaborar el estudio de factibilidad (*business case*). Para ello, se deben identificar todas las entidades externas con las cuales el producto va a interactuar (actores) y definir la naturaleza de estas interacciones a alto nivel. Esto implica la identificación de todos los casos de uso y la descripción de los más significativos. El estudio de factibilidad incluye factores de éxito, evaluación de riesgos y estimación de los recursos requeridos, y la planeación de las fases estableciendo las fechas de los hitos principales.

  Para cumplir con el propósito de esta fase se debe avanzar sustancialmente en los componentes de Modelado de la Organización y Captura de Requisitos.

  El hito de esta fase es la definición de los objetivos y alcance del sistema, y el estudio de factibilidad. Con base en este estudio, la dirección de la empresa a cargo del desarrollo deberá tomar la decisión de si el proyecto es viable y puede realizarse, o debe ser cancelado o re-planteado a fondo.

- **Preparación**. Analizar el dominio del problema, establecer una sólida base arquitectónica, desarrollar el plan del proyecto y eliminar los elementos de más alto riesgo del proyecto. Para alcanzar estos objetivos se debe tener una visión de "100 Km de amplitud y 1 cm de profundidad" acerca del sistema. Las decisiones sobre la arquitectura tienen que tomarse con base en el entendimiento de todo el sistema: su alcance, funcionalidad más importante y requisitos no funcionales tales como los de rendimiento.

  Puede afirmarse que la fase de preparación es la más crítica de las cuatro. Al final de esta fase, la ingeniería "dura" se considera completa y el proyecto enfrenta su más importante día del juicio: la decisión de si se realizan o no las fases de construcción y transición. Para la mayoría de los proyectos, este momento también corresponde a la transición entre una operación trasladable, ligera y de bajo riesgo, a una operación de alto costo, gran riesgo y muchísima inercia. En tanto que el proceso debe siempre dar lugar a cambios, las actividades de la fase de elaboración aseguran que la arquitectura, los requisitos y el plan son suficientemente estables, y los riesgos están suficientemente mitigados, de manera que se pueda predecir el costo y el calendario para completar el desarrollo. Conceptualmente, este nivel de precisión debería corresponder al que necesita una organización para comprometerse con una fase de construcción de precio fijo.

  En la fase de preparación, se construye un prototipo ejecutable de la arquitectura en una o más iteraciones, dependiendo del alcance, tamaño, riesgo y novedad del proyecto. Este esfuerzo debe dirigirse al menos a los casos de uso críticos identificados en la fase de gestación, los cuales conllevan típicamente los mayores riesgos técnicos del proyecto. Si bien el objetivo es siempre un prototipo incremental con componentes de calidad de producción, no se excluye el desarrollo de uno o más prototipos exploratorios y de descarte, para mitigar riesgos específicos tales como decisiones de diseño versus requisitos, estudiar la factibilidad de un componente, o realizar demostraciones a inversores, clientes y usuarios finales.

  Todo esto implica recorrer en varias iteraciones prácticamente todos los componentes del proceso, dejando poco por hacer en los componentes de Modelado de la Organización, Captura de Requisitos y Análisis, y habiendo avanzado lo suficiente en las actividades de Diseño, Implementación y Pruebas como para definir la arquitectura.

  El hito de esta fase es la definición de la arquitectura del sistema, junto con una revisión detallada de sus objetivos y alcance, y la resolución de los mayores riesgos. El proyecto aún puede ser abortado o re-planteado a fondo, pues la inversión no ha sido tan alta hasta este momento.

- **Construcción**. Desarrollar todos los componentes y funcionalidades restantes del sistema, e integrarlos en el producto, así como probar completamente todas las características. La fase de construcción es, en cierto sentido, un proceso de manufactura donde el énfasis está puesto en la gestión de los recursos y el control de las operaciones para optimizar costos, calendarios y calidad. En este sentido, el foco

de atención de la administración sufre una transición desde el desarrollo de propiedad intelectual en las fases de concepción y preparación, hacia el desarrollo de productos instalables en las fases de construcción y transición.

Muchos proyectos son tan grandes que pueden adoptar una estrategia de desarrollo incremental con construcciones paralelas. Estas actividades paralelas pueden acelerar de manera significativa la disponibilidad de entregas instalables, pero pueden así mismo incrementar la complejidad de la gestión de recursos y la sincronización de los flujos de trabajo. Hay una alta correlación entre una arquitectura robusta y un plan entendible. En otras palabras, una de las cualidades críticas de la arquitectura es su facilidad de construcción. Esta es la razón por la cual se enfatiza tanto durante la fase de preparación en el desarrollo balanceado de la arquitectura y el plan.

El hito de esta fase es la obtención de un producto operacional, listo para ponerlo en manos de sus usuarios finales, a la vez que se verifica que su sitio de instalación y usuarios estén listos para empezar la operación. A menudo, a esta entrega del producto se le da el nombre de "beta".

- **Transición**. Transferir el producto a la comunidad de usuarios. Una vez el producto se entrega al usuario final, surgen usualmente asuntos que requieren desarrollar nuevas entregas, corregir algunos problemas o terminar algunas características que habían quedado pospuestas.

  La fase de transición se inicia cuando una línea de base del producto está suficientemente madura para ser implantada en el dominio del usuario. Esto normalmente requiere que se haya completado un subconjunto utilizable del sistema con un nivel de calidad aceptable y que la documentación de usuario esté disponible, de manera que la transferencia al usuario produzca resultados positivos para todas las partes. Esto incluye:
  - La realización de "pruebas beta" para validar el nuevo sistema con respecto a las expectativas de los usuarios.
  - La operación en paralelo con el sistema que se va a reemplazar.
  - La conversión de las bases de datos en operación.
  - El entrenamiento de los usuarios y el personal de mantenimiento.
  - La presentación del sistema a los equipos de mercadeo, distribución y ventas.

  La fase de transición se enfoca en las actividades que se requieren para poner la aplicación en manos de los usuarios. Típicamente, en esta fase se tienen varias iteraciones que incluyen entregas beta y entregas de disponibilidad general, así como entregas con errores corregidos y nuevas características. Se dirige un esfuerzo considerable hacia los usuarios en el desarrollo de su documentación, entrenamiento, soporte en el uso inicial del producto, y recibiendo su realimentación. Sin embargo, en este punto del ciclo de vida, la realimentación de los usuarios debe centrarse prioritariamente en la sintonización, configuración, instalación y usabilidad del producto.

El hito de esta fase es la obtención de la satisfacción de los usuarios. En este punto se establece si los objetivos fueron logrados y si se debería iniciar un nuevo desarrollo del producto. En algunos casos, este hito coincide con el final de la fase de gestación de un nuevo ciclo de desarrollo.

5.3 Fase de Gestación

En la fase de gestación, los esfuerzos del equipo de desarrollo están enfocados en determinar el alcance del proyecto y contribuir a la elaboración de su estudio de factibilidad. Para ello se proponen cinco ciclos del modelo de desarrollo en espiral; estos ciclos no son necesariamente secuenciales, pues pueden ser realizados por varios equipos de trabajo que se realimentan sus resultados. Los ciclos son:

Ciclo 0: Modelado de la Organización. Consiste en la construcción de una versión inicial del Modelo de la Organización, lo suficientemente completa para obtener una caracterización del sistema a desarrollar. En algunos proyectos es suficiente con obtener sólo un Modelo del Dominio y en otros no se requiere este ciclo.

Ciclo 1: Definición del Modelo de Referencia. Consiste en la selección del modelo de desarrollo específico que va a ser empleado en la construcción del sistema, es decir, en la elección de las herramientas metodológicas y operacionales que van a ser empleadas, y su instanciación a las condiciones concretas de la compañía desarrolladora, el proyecto y el equipo humano. En estrecha relación con esta actividad, se define también la organización del equipo humano, en términos de las personas y los roles que van a participar en el proyecto.

Ciclo 2: Desarrollo del Producto. Consiste en la ejecución de una o más iteraciones (en cuyo caso en lugar de un solo ciclo en espiral se tendrían varios) a través de los cinco componentes fundamentales del desarrollo: Captura de Requisitos, Análisis, Diseño, Implementación y Pruebas. En la Captura de Requisitos se busca identificar todos los casos de uso del sistema, así como describir los más significativos en términos de riesgos y de impacto sobre la arquitectura del sistema. Los componentes de Análisis y Diseño están dirigidos a obtener una primera aproximación a la arquitectura del sistema, de manera que pueda identificarse una estructura inicial del sistema y sus posibles interfaces, así como su arquitectura física (dónde se va a ejecutar) y los mecanismos genéricos que se utilizarían en el entorno de implementación (sistemas operativos, manejadores de base de datos, plataformas de distribución, etc.); todo ello, con el fin de evaluar los mayores riesgos que puedan surgir para la construcción del producto. Finalmente, la Implementación y Pruebas serían requeridos en la medida que la coordinación del proyecto decida que es necesario construir un prototipo de descarte para evaluar la arquitectura propuesta.

Ciclo 3: Elaboración del Plan de Desarrollo. Consiste en la definición de la estrategia global que se va a seguir en el desarrollo del sistema, incluyendo el tiempo asignado al proyecto y cada una de sus fases, los hitos más importantes, y las iteraciones que se

proponen para cada fase. El plan se complementa con una estimación de los recursos requeridos para el desarrollo del proyecto, en términos de personal, equipos, materiales, etc., que conducen finalmente a una propuesta de presupuesto. La estrategia de desarrollo está fuertemente influenciada por los riesgos identificados, la forma como se anticipa que van a ser mitigados, y las acciones de contingencia que se tomarían en caso de que finalmente ocurran. Por esta razón, antes de elaborar el plan debe haberse hecho un cuidadoso análisis de riesgos.

Ciclo 4: Estudio de Factibilidad. Consiste en la realización de un análisis económico a partir del cual se toma la decisión de dar luz verde al proyecto. Para ello, la compañía desarrolladora tiene en cuenta los recursos requeridos, los costos de la inversión, las proyecciones de retorno y los estudios de mercado.

5.4 Modelado de la Organización

Su objetivo fundamental es ofrecer a los participantes en el proyecto (clientes, usuarios y desarrolladores) una visión común sobre la estructura y el funcionamiento de la organización en la cual se implantará el sistema el sistema a desarrollar.

Este es un componente de desarrollo opcional para muchos proyectos, pero que es casi indispensable en el caso de sistemas de gestión de información y en muchas aplicaciones web, para las cuales uno de los factores de éxito consiste en la comprensión del entorno organizacional en el cual va a operar el sistema a desarrollar, además del impacto que éste va a tener sobre la propia organización.

En el contexto que aquí se utiliza, el término "organización" tiene una acepción muy amplia, pudiendo cubrir desde un departamento de una entidad oficial hasta un conglomerado de empresas; no se refiere sólo a entidades con ánimo de lucro, razón por la cual el término *business*, que es el que utiliza toda la literatura en inglés al respecto, no se ha traducido por "negocio", que sugiere la búsqueda de lucro. Los roles, artefactos y actividades definidos por RUP para este componente del proceso de desarrollo (*Business Modeling*) están siendo utilizados no sólo en proyectos para la construcción de aplicaciones informáticas, sino también como una metodología para el mejoramiento de los procesos organizacionales (BPI, *Business Process Improvement*) y la reingeniería de procesos organizacionales (BPR, *Business Process Re-engineering*) [20].

Es muy común que un cliente requiera los servicios de un equipo de desarrollo, presentando directamente las características de la solución para lo que él cree que son los problemas de su organización. A través de la ejecución de las actividades de este componente, los participantes entienden los problemas reales de la organización, identifican mejoras potenciales, y llegan a un acuerdo sobre los procesos de la organización que se requiere y desea soportar con el sistema a desarrollar, y las características generales de este sistema.

El producto principal de este componente es el Modelo de la Organización, que juega un papel muy importante en el entendimiento de la organización, la captura de los requisitos del sistema que se va a poner a su servicio, y la identificación de los cambios que deben producirse en la organización con la puesta en marcha del sistema.

El Modelo de la Organización consta de un Modelo de Casos de Uso de la Organización y un Modelo de Objetos de la Organización. El Modelo de Casos de Uso de la Organización consiste en un conjunto de diagramas de casos de uso construidos con estereotipos de UML para el modelado de la organización. El Modelo de Objetos de la Organización consiste en un conjunto de diagramas de objetos, opcionalmente explicados mediante diagramas de secuencia, construidos ambos con estereotipos de UML para el modelado de la organización.

La especificación de UML producida por el OMG [12] contiene un capítulo donde presenta ejemplos de perfiles UML, y uno de ellos es un perfil para el modelado de organizaciones donde presenta varios estereotipos. Por su parte, Philippe Kruchten propone en su libro sobre RUP [21] otros estereotipos que complementan los anteriores.

Los estereotipos con notación gráfica propuestos por el OMG son (Figura 28):

Trabajador (worker): Es una clase que representa una abstracción de un humano que actúa dentro del sistema (referido a la organización modelada). Un trabajador interactúa con otros trabajadores y manipula entidades (clases entity) mientras participa en la realización de casos de uso (de la organización).

Trabajador de caso de uso (caseWorker): Es un caso especial de trabajador que interactúa de manera directa con actores que se encuentran fuera del sistema.

Trabajador interno (internalWorker): Es un caso especial de trabajador que interactúa con otros trabajadores y entidades dentro del sistema.

Entidad (entity): Es una clase pasiva, esto es, sus objetos no inician interacciones por sí mismos. Representan recursos de la organización (insumos, documentos, equipos, productos, etc.).

El OMG también ha propuesto otros estereotipos sin representación gráfica propia, como es el caso de Unidad Organizacional (organizationUnit) y Unidad de Trabajo (workUnit) cuya notación gráfica es un paquete con la etiqueta del estereotipo.

Los estereotipos propuestos por Kruchten son (Figura 29):

Actor de la Organización (*Business Actor*): Representa una entidad externa con quien interactúa la organización.

Caso de Uso de la Organización (*Business Use Case*): Contiene un conjunto de actividades relacionadas que realiza la organización cuando interactúa con el entorno, una vez recibe un estímulo por parte de un actor de la organización.

Realización de Caso de Uso de la Organización (*Business Use-Case Realization*): Es una caja blanca que representa la descripción de cómo interactúan actores, trabajadores y entidades de la organización para realizar un caso de uso de la organización.

Unidad de la Organización (*Organization Unit*): Es una colección de roles, responsabilidades y recursos que comparten un propósito común. Son utilizadas para dividir un modelo de organización complejo en elementos independientes que interactúan entre sí a un nivel superior que los trabajadores y entidades de la organización.

Trabajador de la Organización (*Business Worker*): Representa un rol activo que alguien o algo asume en la realización de un caso de uso de la organización. Este rol puede ser desempeñado por una persona, un equipo de trabajo o una aplicación informática [20].

Entidad de la Organización (*Business Entity*): Representa un recurso de la organización.

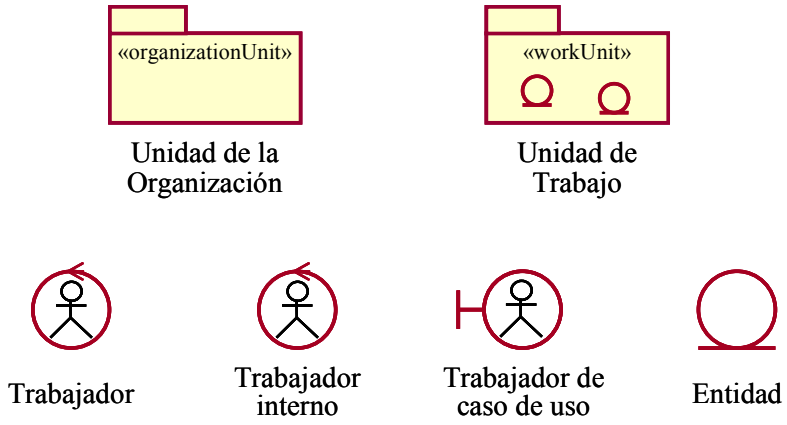


Figura 28. Estereotipos para el Modelo de la Organización (OMG)

Figura 29. Estereotipos para el Modelo de la Organización (Kruchten)

A continuación se presenta un pequeño ejemplo de cómo pueden utilizarse estos estereotipos para modelar una organización, y cómo este modelo contribuye al desarrollo de una aplicación informática que será utilizada por esa organización. La organización elegida es una biblioteca universitaria.

La Figura 30 muestra una parte del Modelo de Casos de Uso de la biblioteca, con dos actores y tres casos de uso de la organización, a saber:

- Lector: Es un usuario de la biblioteca.
- Sistema Nacional de Intercambio: Es una organización en la cual participan las bibliotecas del país, creada para facilitar el préstamo de material bibliográfico por parte de cualquier institución que lo tenga disponible para la institución donde es requerido.
- Reservar Libro: Registrar un libro de la biblioteca a nombre de un lector, que más adelante pasará a retirarlo.
- Prestar Libro: Entregar un libro a un lector, asignándole un plazo para devolverlo.
- Suministrar Artículo: Entregar a un usuario la copia de un artículo solicitado. Si el artículo no se encuentra en la biblioteca, se recurre al Sistema Nacional de Intercambio para obtener una copia para el solicitante.

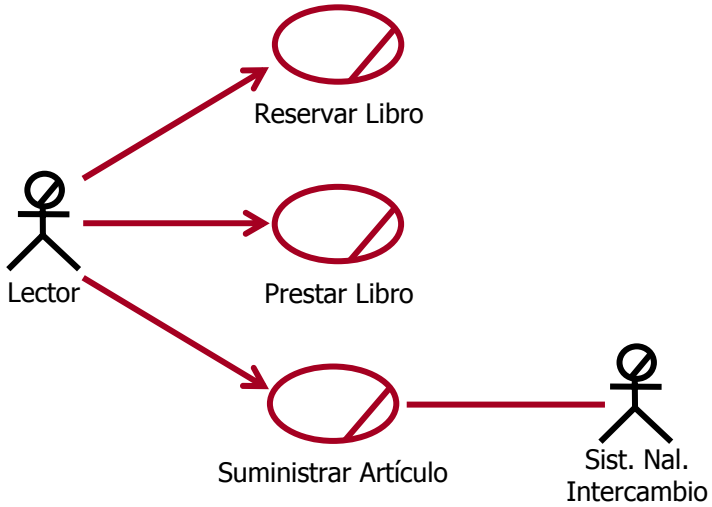


Figura 30. Modelo de Casos de Uso de la Organización

En la Figura 31 se representa la realización del caso de uso Prestar Libro, en el cual, además del actor Lector, participan dos trabajadores y tres entidades de la organización, a saber:

- Monitor: Representa a un estudiante que trabaja en la biblioteca atendiendo los usuarios.
- Director: Representa al director de la biblioteca.
- TarjetaLector: Entidad que contiene la información de un lector. Hay un objeto por cada lector.

- Catálogo: Entidad que contiene la información de los títulos que posee la biblioteca.
- Libro: Entidad que contiene la información de un libro. Hay un objeto por cada libro.

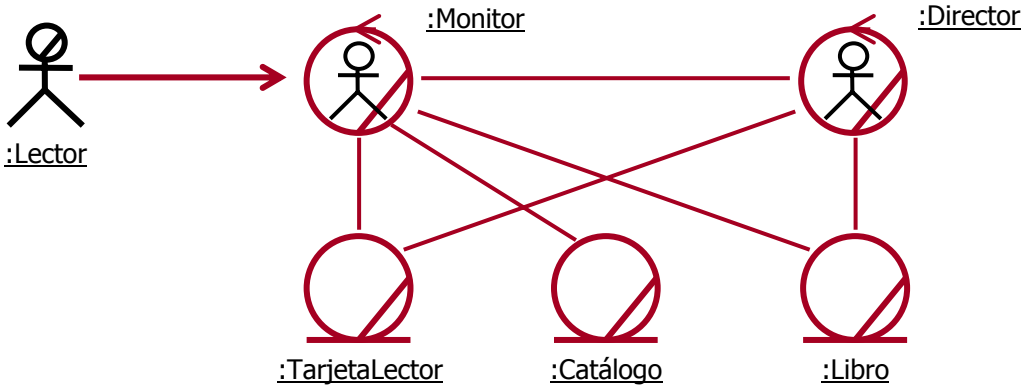


Figura 31. Modelo de Objetos de la Organización

La dinámica de la realización del caso de uso puede describirse de manera textual, para escenarios simples, o mediante uno o varios diagramas de secuencia. La siguiente descripción textual del caso de uso Prestar Libro se representa también en el diagrama de la Figura 32.

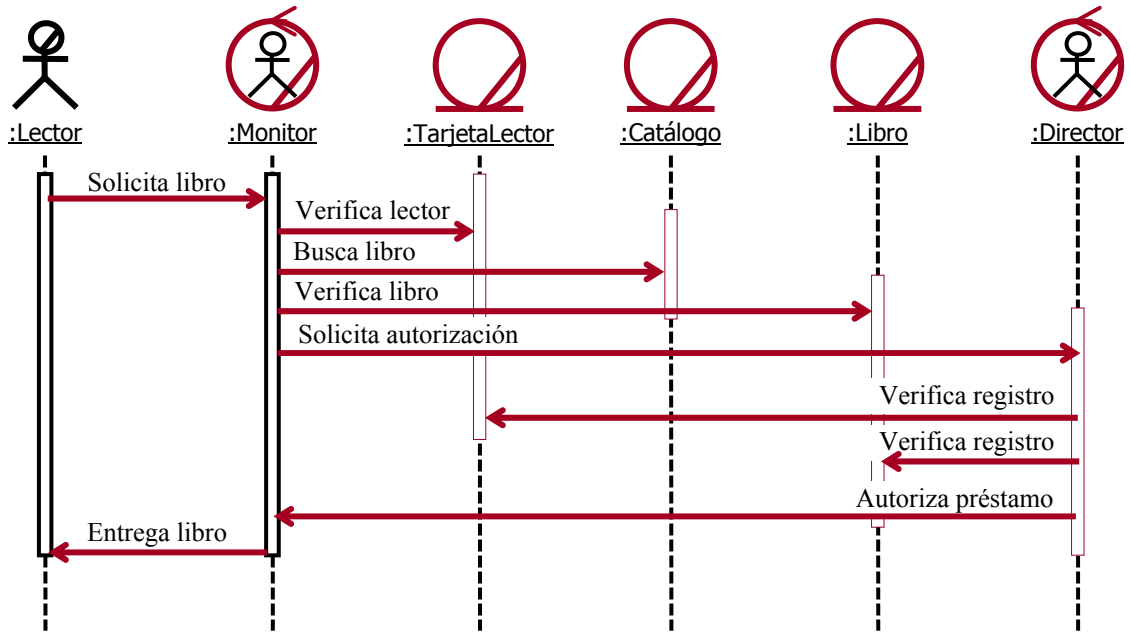


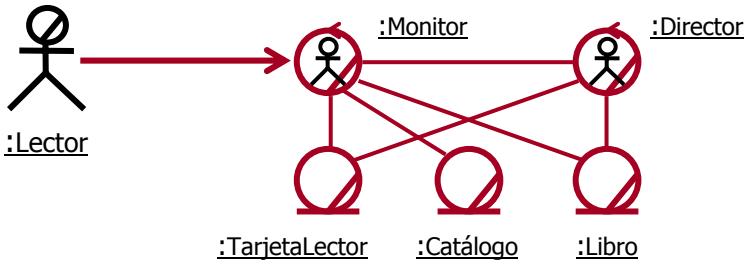
Figura 32. Diagrama de Secuencias

- El caso de uso se inicia cuando el Lector solicita un libro al Monitor.
- El Monitor consulta TarjetaLector para verificar que el Lector no esté impedido para realizar préstamos.
- El Monitor consulta Catálogo para verificar que la biblioteca posee el Libro solicitado.
- El Monitor consulta Libro para verificar que está disponible para préstamo.

- El Monitor registra la información del préstamo en TarjetaLector y Libro y solicita la autorización del Director para realizar el préstamo.
- El Director verifica la información registrada en TarjetaLector y Libro, y autoriza el préstamo.
- El Monitor entrega el libro al Lector.

El Modelo de la Organización no sólo contribuye a identificar requisitos del sistema a desarrollar, sino que también aporta elementos para la construcción de los modelos de éste como se ilustra en la Figura 33. En el Modelo de Casos de Uso de la aplicación que prestará servicio en la biblioteca se muestra el caso de uso Prestar Libro (Figura 33c); para este caso de uso de la aplicación, los actores surgen de un actor y un trabajador del Modelo de Objetos de la Organización (Lector y Director, respectivamente). Por su parte, las tres entidades que se muestran del Modelo de Análisis de la aplicación (Figura 33d) surgen también de las entidades del Modelo de Objetos de la Organización.

a) Modelo de Casos de Uso de la Organización (caso de uso PrestarLibro)



b) Modelo de Objetos de la Organización

c) Modelo de Casos de Uso de la aplicación (caso de uso PrestarLibro)

d) Modelo de Análisis (clases de Entidad)

Figura 33. Evolución de los modelos

6. URLs y Herramientas

Con el fin de complementar la información brindada en este documento, se incluyen en este capítulo algunas direcciones de páginas Web que tratan sobre UML y RUP.

- OMG UML Resource Page. http://www.omg.org/technology/uml/index.htm
- IBM Rational UML Resource Center. http://www-306.ibm.com/software/rational/uml/
- Enlaces de Cetus sobre UML. http://www.cetus-links.org/oo_uml.html
- Listados de herramientas UML:
  - Objects by Design. http://www.objectsbydesign.com/tools/umltools_byCompany.html
  - Mario Jeckle. http://www.jeckle.de/umltools.htm
  - Bernd Oestereich. http://www.oose.de/uml_tools.htm
- Herramientas UML:
  - Poseidon for UML. http://www.gentleware.com
  - Delphia Object Modeler (D.OM). http://www.si.fr.atosorigin.com/dom/english/index.html
  - ArgoUML*. http://argouml.tigris.org
  - Umbrello UML Modeller*. http://uml.sourceforge.net/index.php
  - UMLGraph*. http://www.spinellis.gr/sw/umlgraph/
  - Jude*. http://objectclub.esm.co.jp/Jude/jude-e.html
  - Objecteering/UML. http://www.objecteering.com/products.php
  - Fujaba Tool Suite. http://www.fujaba.de
  - EclipseUML. http://www.omondo.com/index.jsp
  - IBM Rational Rose Developer Family. http://www-306.ibm.com/software/awdtools/developer/rose/
  - Object Domain de Object Domain Systems. http://www.objectdomain.com.
  - Borland® Together® ControlCenter. http://www.togethersoft.com/products/controlcenter/index.jsp.
- Enlaces de RUP:
  - RUP Information. http://www.programming-x.com/programming/rup.html
  - IBM Rational Unified Process. http://www-306.ibm.com/software/awdtools/rup/
  - Literatura de IBM sobre RUP, UML y otras herramientas. http://www-306.ibm.com/software/rational/info/literature/whitepapers.jsp

* Programas de código abierto (*Open Source Software*).

Referencias

[1] H. Levy. "Capability-Based Computer Systems". Digital Press. 1984.

[2] I. Jacobson, G. Booch, and J. Rumbaugh. "The Unified Software Development Process". Addison-Wesley. 1999.

[3] H.-E. Eriksson and M. Penker. "UML Toolkit". John Wiley and Sons. 1998.

[4] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and F. Lorensen. "Object-Oriented Modeling and Design". Prentice-Hall. 1991.

[5] G. Booch. "Object-Oriented Analysis and Design with Applications". Benjamin Cummings, 1994.

[6] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. "Object-Oriented Software Engineering. A Use Case Driven Approach". Addison-Wesley. 1992.

[7] A. Burns and A.J. Wellings. "HRT-HOOD: A Structured Design Method for Hard Real-Time Systems" Real-Time Systems, No. 6, Vol. 1, January 1994.

[8] B. Selic, G. Gullekson, J. McGee, and I. Engelberg. "ROOM: An Object-Oriented Methodology for Developing Real-Time Systems". In Fifth International Workshop on Computer-Aided Software Engineering (CASE'92), Montreal, Canada. July 1992.

[9] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremes. "Object-Oriented Development: The Fusion Method". Prentice-Hall. 1994.

[10] OMG. "UML Summary. v1.1". UML Specification Document ad/97-08-03. http://www.org.omg. September 1997.

[11] T. Quantrani. "Visual Modeling with Rational Rose and UML". Addison-Wesley. 1998.

[12] Object Management Group. “OMG Unified Modeling Language Specification”. Versión. 1.5. Documento formal/03-03-01. Marzo 2003.

[13] OMG. "UML 2.0 Specifications nearing completion". Marzo 2004. http://www.omg.org/technology/uml/index.htm

[14] Rational Software Corporation. "The Unified Method. Draft Edition (0.8)". October 1995.

[15] C. Larman. "Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design". Prentice-Hall. 1998.

[16] D. Harel et al. "STATEMATE: A Working Environment for the Development of Complex Reactive Systems". IEEE Transactions on Software Engineering, 16(4):403-414, April 1990.

[17] Philippe Kruchten. "The 4+1 View Model of Architecture". IEEE Software, 12 (6):42-50, November 1995.

[18] Walker Royce. "Software Process Management: A Unified Framework". Reading, MA: Addison Wesley. 1998.

[19] Rational. "Rational Unified Process: Best Practices for Software Development Teams". 1999. http://www-306.ibm.com/software/rational/info/literature/whitepapers.jsp.

[20] D.J. de Villiers. "The New Business Modeling Discipline". Empulsys White Paper. 12 May 2003. http://www.empulsys.com (background & philosophy, downloads).

[21] Philippe Kruchten. “The Rational Unified Process. An Introduction”. Second Edition. Addison Wesley. 2000.