Sistemas de información para la dirección:

Un enfoque guiado por un caso de estudio

Simona Bernardi
Lacramioara Dranca

1ª edicíon, 2015.

Edita:
Centro Universitario de la Defensa
Ctra. de Huesca s/n
50090 Zaragoza (España)
url: http://www.cud.unizar.es

Impresión:
Edelvives Talleres Gráficos

Impreso en España
Printed in Spain

Deposito legal: Z 1835-2014
ISBN: 978-84-940583-5-6

Prólogo

Las tecnologías de la información y las comunicaciones se han convertido en el gran dinamizador social, económico y de las organizaciones en el final del pasado siglo e inicios del XXI. Refiriéndonos al caso de las organizaciones, las TIC han cambiado el modo de relación entre sus miembros, han contribuido a cambiar su estructura interna y su control, y también les plantean retos actuales como el manejo de las ingentes cantidades de datos que pueden recopilar acerca de su entorno.

Hablando de organizaciones, los términos *procesos de dirección* y *sistemas de información* pueden considerarse absolutamente intricados. No hay organización pública ni privada que aspire a altos niveles de eficacia, eficiencia y calidad que no considere su sistema o sistemas de información un elemento clave para el soporte de la planificación, la toma de decisiones y el control de la organización. Esta es la razón de ser de una asignatura como Sistemas de Información para la Dirección en carreras en las que sus titulados aspiran a tomar parte en la gestión y dirección de empresas y organizaciones, como es el caso de la Ingeniería de Organización Industrial que se imparte en el Centro Universitario de la Defensa.

Las profesoras Bernardi y Dranca han configurado una propuesta original de asignatura, bien contextualizada para la formación de los Oficiales del Ejército de Tierra, y en la que emplean una metodología que permite formar a los alumnos en competencias que no son sólo las propias de la disciplina, pues el trabajo en equipo y el método del caso se aplican para obtener resultados rigurosos. Aludir a la cuestión metodológica es relevante, pues el manual que ellas han elaborado no se limita a abordar una serie de contenidos de la disciplina, sino que está marcado por una metodología docente de primer nivel, de manera que constituya la mejor herramienta para el alumno de la asignatura.

Es un motivo de satisfacción dar la bienvenida a este nuevo número de la colección de textos docentes del CUD que sirve de soporte a la asignatura Sistemas de Información para la Dirección. Pero este manual no va a ser solo una ayuda para los alumnos del centro, puede ser también una adecuada referencia para profesores y alumnos de asignaturas de muchas otras titulaciones, precisamente por el interés de su propuesta. La implantación de sistemas de información en las organizaciones, las herramientas de explotación de bases de datos, y las herramientas de inteligencia del entorno son en muy grandes rasgos tres de los aspectos que están cubiertos en el libro.

Estoy seguro de que el libro va a resultar del máximo interés para el lector y el usuario.

Zaragoza, 12 de enero de 2015

Jorge Rosell
Profesor Titular de Organización de Empresas
Subdirector del Centro Universitario de la Defensa

Prefacio

Este libro se ha escrito para los estudiantes de *Sistemas de Información para la Dirección (SID)*, asignatura impartida por el Centro Universitario de la Defensa (CUD) en el tercer curso del grado en Ingeniería de Organización Industrial - perfil defensa.

La asignatura introduce los sistemas de información y su papel en la gestión de las actividades de una organización o empresa. Además de presentar los conceptos básicos que conforman los sistemas de información y el entorno tecnológico que les da soporte en la actualidad, muestra al alumnado los fundamentos metodológicos para la captura y representación de la información, el desarrollo, la implantación y el mantenimiento de los sistemas de información.

En particular, se pone énfasis en las actividades relacionadas con las etapas tempranas de desarrollo de un sistema de información -es decir la recogida y el análisis de requisitos y el modelado de negocio-, con el diseño conceptual y lógico de una base de datos y en las buenas prácticas utilizadas hoy en día para llevar a cabo estas actividades: técnicas de *brainstorming*, técnicas de modelado y entrevistas con las partes interesadas, uso de herramientas software CASE (*Computer Aided Software Engineering*), DBMS (*Data Base Management System*) y colaborativas en la nube (por ejemplo, *Google Drive*).

Analizar un sistema de información es una tarea compleja que requiere buenas dotes de abstracción y de sentido crítico. Por ello se pretende durante el curso potenciar ambas capacidades a través de la realización de proyectos en *equipos auto-organizados* en que se aplican técnicas de modelado y entrevistas con las partes interesadas -es decir los potenciales usuarios de un sistema de información. Durante los primeros dos cursos de impartición de la asignatura, se propusieron al alumnado dos casos de estudios muy próximos: un Sistema de Información para el Centro Universitario de la Defensa (SICUD) - año académico 2012/2013- y un Sistema de Información para la Academia General Militar (SIAGM) -año académico 2013/2014. Para la realización de las entrevistas a las partes interesadas, por parte del alumnado, se ha contado con la imprescindible colaboración del personal del CUD, y del personal civil y militar de la Academia General Militar (AGM).

El enfoque práctico de la asignatura se ha plasmado por medio de dos proyectos de innovación docente. En el primer proyecto (Dranca et al, 2013) se ha implantado la actividad de entrevista para el caso de estudio SICUD, mientras en el segundo proyecto (Dranca et al, 2014) se ha consolidado la experiencia del proyecto anterior a través de un nuevo caso de estudio (SIAGM), y se han explotado varias herramientas colaborativas en la nube tanto para la planificación docente como para la realización del proyecto por parte del alumnado.

El libro es una recopilación de los apuntes escritos por las autoras, para los alumnos de los dos cursos pasados, consultando las fuentes citadas en las referencias. En particular

se ha utilizado el caso de estudio *CarShare* -organización ficticia cuyo objetivo de negocio es promover el uso compartido de coches- como ejemplo a lo largo del libro para ilustrar los conceptos teóricos. El libro reúne además una colección de ejercicios y casos de estudio propuestos a los estudiantes en los primeros dos cursos de la asignatura.

Organización del libro

El libro está organizado en cuatro partes. La primera parte es una introducción a los sistemas de información, ingeniería de sistemas y del software. La segunda parte considera las etapas temprana del proceso de desarrollo de un sistema de información. En particular, se consideran las técnicas utilizadas para la recogida y análisis de requisitos, y el modelado de negocio. La tercera parte se centra en las bases de datos, haciendo hincapié en las técnicas de análisis y diseño de las mismas. La cuarta parte se centra en las herramientas de apoyo a la toma de decisiones. En particular describe el lenguaje de referencia para manipulación de datos en bases de datos relacionales, SQL-DML (*Structured Query Language - Data Manipulation Language*), y los componentes principales de las herramientas de inteligencia de negocio (*Business Intelligence*, en inglés): los almacenes de datos y herramientas OLAP (*On-Line Analytical Processing*). Finalmente, el libro incluye tres apéndices: el Apéndice A describe el caso *CarShare* y el Apéndice B introduce el caso de estudio *UPS*. El Apéndice C presenta ejemplos de aplicación de los tres tipos de diagramas UML, utilizados a lo largo del libro, al caso *CarShare*.

Parte I. Introducción a los sistemas de información.

El Capítulo 1 introduce los conceptos básicos de datos-información-conocimiento y de sistema, define los sistemas de información e identifica sus principales componentes.

El Capítulo 2 describe la ingeniería de sistemas como un enfoque interdisciplinario para el desarrollo de sistemas complejos. Se analizan los procesos principales como soporte de la ingeniería de sistemas, según el estándar ISO-IEC15288 (2008). Se introduce también la ingeniería del software como la aplicación de los principios y procesos de la ingeniería de sistemas al desarrollo de sistemas software. Se examinan las peculiaridades del software, las etapas de desarrollo de un sistema software, los principales modelos de ciclo de vida y el Lenguaje Unificado de Modelado (*Unified Modeling Language - UML*) para la especificación de sistemas software.

Parte II. Desarrollo de un sistema de información: etapas tempranas.

El Capítulo 3 introduce el concepto de requisito, la tradicional clasificación de requisitos en *funcionales* y *no funcionales*, y las técnicas utilizadas para la recogida y especificación de requisitos. En particular, describe una técnica en concreto para la captura y especificación sistemática de los requisitos funcionales: los casos de uso.

El modelado de negocio tiene como propósito principal comprender y especificar el funcionamiento del negocio de la organización en la que un sistema de información se va a utilizar. El Capítulo 4 introduce el modelado de negocio como una técnica que permite proporcionar

información, sobre los conceptos del dominio de interés y los procesos de negocio, al equipo de trabajo del proyecto de desarrollo de un sistema software para delimitar su alcance.

Parte III. Desarrollo de bases de datos: análisis y diseño.

El Capítulo 5 introduce las bases de datos con sus principales características, y los sistemas de gestión de bases de datos, sus funciones y las ventajas que aportan. Se presentan además los modelos de datos y las diferentes etapas que incluye el diseño de una base de datos.

El Capítulo 6 se centra en las técnicas de modelado que se aplican durante el diseño conceptual de una base de datos. Concretamente se presenta la especificación con el modelo Entidad-Relación (ER), utilizando para ello la notación de Martin. Se repasan los principales elementos del modelo ER y se ejemplifican utilizando el caso de estudio *CarShare*.

El Capítulo 7 se centra en la fase de diseño lógico de una base de datos. Se presenta el modelo relacional y la forma de construirlo a partir del modelo conceptual ER. Como particularidad del modelo relacional se explican las restricciones de integridad intra-relacionales e inter-relacionales. Se introduce también el lenguaje SQL (*Structured Query Language*), incidiendo sobre su condición de lenguaje de definición de datos (SQL-DDL - *Data Definition Language*).

Parte IV. Uso de sistemas de información: herramientas de apoyo a la toma de decisiones.

El Capítulo 8 introduce el lenguaje de referencia para manipulación de datos en bases de datos relacionales, SQL-DML, que permite realizar consultas sobre una base de datos para extraer información de una o varias tablas.

El Capítulo 9 revisa los tipos de sistemas de información en las organizaciones, haciendo hincapié en los sistemas informacionales. Se introducen las herramientas de inteligencia de negocios con sus principales componentes: los almacenes de datos y las herramientas OLAP. Finalmente se mencionan a otras herramientas de BI, como las técnicas de minería de datos utilizadas también en el ámbito de la seguridad y ciberseguridad.

Agradecimientos

Queremos agradecer a todas las personas e instituciones que hicieron posible la implantación de la asignatura tal como se está impartiendo a día de hoy y la realización de este trabajo. En particular, agradecemos al Prof. Dr. Jorge Rosell -subdirector del CUD- por su apoyo en la implantación de las actividades de entrevistas del alumnado con las partes interesadas del SICUD, y a la AGM por la autorización y soporte proporcionado en la actividad de entrevistas dentro del proyecto SIAGM.

Un agradecimiento especial va al Cte. José Jaime Rodríguez Travieso y al Cte. José Félix Calviño González por la identificación de los temas de interés en la AGM a proponer al alumnado y de las partes interesadas del SIAGM. Agradecemos también al personal del CUD y al personal civil y militar de la AGM que han sido entrevistados por el alumnado: sin ellos esta actividad no hubiera sido viable.

Finalmente, agradecemos al TCol. Joaquín R. Guerrero Benavent por proporcionarnos referencias sobre los sistemas de información de mando y control militares, a nuestra com-

pañera la Dra. María Teresa Lozano Albalate por su colaboración en los dos proyectos de innovación docente, y la inapreciable ayuda del Dr. Fco. Javier Vidal Bordes -responsable de la biblioteca CUD- en la revisión y corrección de este libro.

Zaragoza, 10 de enero de 2015

Simona Bernardi
Lacramioara Dranca

Índice general

Acrónimos

ACID	Atomicity, Consistency, Isolation, Durability
BD	Base de Datos
BI	Business Intelligence
BOM	Business Object Model
BSC	Balanced ScoreCard
CASE	Computer Aided Software Engineering
CCAA	Comunidad Autónoma
CMI	Cuadro de Mando Integral
COTS	Commercial Off-The-Shelf
CRM	Customer Relationship Management
CRUD	Create, Retrieve, Update, Delete
CSV	Comma Separated Values
DBMS	Data Base Management System
DDL	Data Definition Language
DFD	Data Flow Diagram
DML	Data Manipulation Language
DSS	Decision Support System
DW	Data Warehouse
EBP	Elementary Business Process
ER	Endidad-Relación
ERD	Entity Relationship Diagram
ERP	Enterprise Resource Planning
ESS	Executive Support System
ETL	Extraction Transformation Loading
FK	Foreign Key
GB	Giga Byte
ICT	Information and Communication Technology
INE	Instituto Nacional de Estadística
INEM	Instituto Nacional de Empleo
MB	Mega Byte
MIS	Management Information System
NEC	Network Enabled Capability
OMG	Object Management Group
OLAP	On-Line Analytical Processing
OLTP	On-Line Transaction Processing

PAPS	Phased Armament Programming System
PB	Peta Byte
PDA	Personal Digital Assistant
PK	Primary Key
SCM	Supply Chain Management
SI	Sistema de Información
SIEM	Security Information and Event Management
SQL	Structured Query Language
SysML	Systems Modeling Language
TIC	Tecnologías de la Información y Comunicaciones
TPS	Transaction Processing System
UML	Unified Modeling Language
UP	Unified Process

Parte I
Introducción a los sistemas de información

Capítulo 1
Introducción

Resumen Este capitulo introduce los sistemas de información. Se repasan conceptos básicos de dato, información, conocimiento y sistema y se definen los sistemas de información e identifican sus principales componentes.

1.1. Conceptos básicos

Datos, información, conocimiento

Datos e información son términos que se suelen utilizar a veces de manera indiscriminada, sin percibir las diferencias entre uno y otro. En el diccionario de la RAE (Real Academia Española) la definición de dato aparece de la siguiente manera:

> **Dato** : del latín datum, lo que se da:
>
> 1. m. Antecedente necesario para llegar al conocimiento exacto de algo o para deducir las consecuencias legítimas de un hecho.
> 2. m. Documento, testimonio, fundamento.
> 3. m. *inform.* Información dispuesta de manera adecuada para su tratamiento por un ordenador.

Se entiende que el dato no tiene sentido en sí mismo, sino que se utiliza en la toma de decisiones o en la realización de cálculos a partir de un procesamiento adecuado y teniendo en cuenta su contexto.

La **información** se sitúa en un nivel superior (véase Figura 1.1), siendo constituida por datos que han sido moldeados en una forma que es *significativa* y *útil* al receptor. Dado que la información se define en términos de relevancia con respecto al receptor, se puede dar la circunstancia de que lo que es información para una persona pueda ser simplemente un dato para otra.

La información permite resolver problemas y tomar decisiones y su aprovechamiento racional es la base del *conocimiento*.

El **conocimiento** es información combinada con experiencia, contexto, interpretación y reflexión. Es una forma de información con alto valor que está lista para ser aplicada a las decisiones y a las acciones (Davenport et al, 1997).

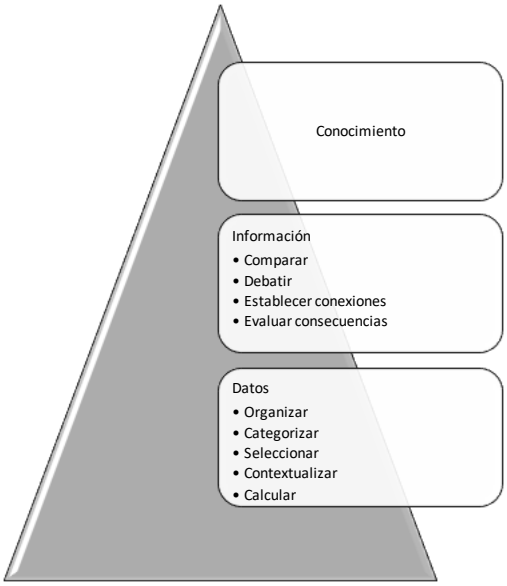


Figura 1.1 Pirámide del conocimiento.

Por ejemplo el número cero es un dato que no modifica nuestro nivel de conocimiento. Sin embargo si se nos dice que hay una temperatura de 0°C fuera (contextualizar), el dato se transforma en información. Al evaluar sus consecuencias aumenta nuestro nivel de conocimiento y se ven afectadas nuestras decisiones (abrigándonos al salir fuera).

Propiedades de la información

La información de calidad puede ayudar a las personas y sus organizaciones a realizar tareas de manera más eficiente y eficaz. La información es uno de los *recursos* más valiosos de una organización. Puede considerarse incluso estratégico, por el hecho de ser una poderosa arma en la toma de decisiones a cualquier nivel.

Cuando se habla de información de calidad se tienen en cuenta ciertas propiedades de la información consideradas deseables. El Cuadro 1.1 agrupa varias propiedades deseables de la información (Stair and Reynolds, 2012).

El concepto de sistema

Existen numerosas definiciones de sistemas de información, cada una con sus matices. Para una mejor comprensión se procede primero a examinar el concepto de *sistema*, para cuya definición existe un mayor consenso.

Un sistema:

- está formado por varios elementos que *interaccionan*
- funciona persiguiendo un *objetivo*
- tiene *entradas*, las procesa y genera una o varias *salidas* (resultados)
- está provisto por un mecanismo de control (re-alimentación) por el cual una cierta proporción de la salida se redirige a la entrada, y así regula su comportamiento.

Propiedad	Descripción
Accesible	La información debe ser fácilmente accesible por los usuarios autorizados para que la puedan obtener en el formato adecuado y en el momento adecuado para satisfacer sus necesidades.
Completa	La información debe contener todos los elementos necesarios para que permita la toma de decisiones. Por ejemplo, un informe de inversión tiene que contener todos los costes.
Económica	Debe ser relativamente económico producir la información. El valor de la información debe estar en equilibrio con el coste de producirla.
Fiable	La información tiene que gozar de la confianza de los usuarios. La fiabilidad de la información depende a veces de la fiabilidad del método de recogida de datos y, en otras ocasiones, de la fuente de la información.
Flexible	Útil en diferentes escenarios.
Oportuna	La información debe encontrarse en el lugar y momento oportunos. Si se desea comprar un producto, no sirve saber que hay una oferta mejor, una vez realizada la compra.
Precisa	La información debe ser fiel reflejo de la realidad.
Relevante	La información debe tener relación directa con la decisión que se quiere tomar.
Segura	El acceso a la información debe ser restringido únicamente a los usuarios autorizados.
Simple	La información debe ser sencilla, reducida a lo realmente importante, sin producir sobrecarga.
Verificable	Existe la posibilidad de comprobar para asegurarse de que la información es correcta (por ejemplo mediante la comprobación de varias fuentes para la misma información).

Cuadro 1.1 Propiedades deseables de la información.

Estamos rodeados por una gran variedad de sistemas. Ejemplos: el sistema solar, la columna vertebral, un ordenador, una central nuclear, etc, etc. Las organizaciones en general, y las empresas en particular también pueden ser consideradas como sistemas que, a partir de entradas (datos, dinero, trabajo, materiales, etc), mediante un proceso de transformación, generan salidas (productos, servicios, dividendos, impuestos, información, etc.).

1.2. Sistema de información

Partiendo de la definición de sistema, se puede definir de forma general a un sistema de información como conjunto de componentes que interaccionan entre sí para lograr un objetivo común: satisfacer las necesidades de información de una organización.

En el caso de sistemas de información, se identifica la entrada al sistema con las actividades de recolección y captura de **datos** en bruto y la salida con **información** (véase Figura 1.2), normalmente en forma de documentos, informes. La información se obtiene mediante un procesamiento de los datos de entrada. Parte de la información del sistema se puede utilizar para realizar cambios en las actividades de entrada o de procesamiento (re-alimentación).

Hoy en día, cuando se habla de sistemas de información se les suele identificar con los sistemas basados en ordenadores. Sin embargo los sistemas de información existen desde mucho antes que los ordenadores.

Existen sistemas de información *manuales*, como por ejemplo los informes, formularios en papel, escritos a mano (Fernández-Alarcón, 2006). Las conversaciones de trabajo en la maquina de café también pueden considerarse como sistema de información. Este ejemplo

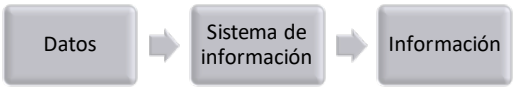


Figura 1.2 Representación de un sistema de información.

entra además en la categoría de los sistemas de información *informales*, puesto que no opera en conformidad con reglas predefinidas.

Por otra parte, los sistemas *formales* de información son aquellos que se apoyan en definiciones fijas y aceptadas de datos y procedimientos y que operan en conformidad con reglas predefinidas.

El estudio de los sistemas de información se ocupa de cuestiones y puntos de vista aportados por diferentes disciplinas como: la informática, la gestión de empresas, investigación operativa, psicología, economía o sociología. Este libro se centra en los **sistemas formales de información basados en ordenadores**, por lo que de aquí en adelante cuando se hable de sistemas de información se hará referencia expresamente a sistemas formales de información basados en ordenadores.

Una de las definiciones más aceptadas para sistemas de información viene dada por (Laudon and Laudon, 2012):

> "Un conjunto de componentes interrelacionados que reúne (u obtiene), procesa, almacena y distribuye información para apoyar la toma de decisiones y el control en una organización. Además de apoyar la toma de decisiones, la coordinación y el control, los sistemas de información también pueden ayudar a los gerentes y trabajadores a analizar problemas, a visualizar asuntos complejos y crear productos nuevos."

De esta definición se entiende que un sistema de información está formado por *componentes interrelacionados*. La definición no detalla de qué componentes se trata; se detallarán en la Sección 1.3.

Por otro lado, la definición identifica las actividades que realizan los sistemas de información. Se trata de: 1) la recolección de datos, 2) su procesado y almacenamiento y 3) distribución de información.

Los principales objetivos de un sistema de información (basado en ordenadores) son proporcionar información que sirva:

- en el proceso de toma de decisiones
- para el control de una organización.

Los sistemas de información automatizan los procesos operativos de la organización. La implantación en empresas se realiza con la pretensión de obtener ventajas competitivas ya que los sistemas de información contribuyen en alcanzar la excelencia operativa, crear nuevos productos, servicios o incluso nuevos modelos de negocio ((Amazon Inc., 2014) es un buen ejemplo en ese sentido), conocer mejor a los clientes y a los proveedores y mejorar la toma de decisiones.

1.3. Componentes de un sistema de información

Un sistema de información trabaja con datos. Los datos se almacenan de forma estructurada, agrupados en ficheros y en bases de datos (BD). Resulta lógico que **los datos** sean, dentro del sistema, un componente especial. A partir de este componente, se pueden identificar los demás componentes del sistema de información con los que interactúa.

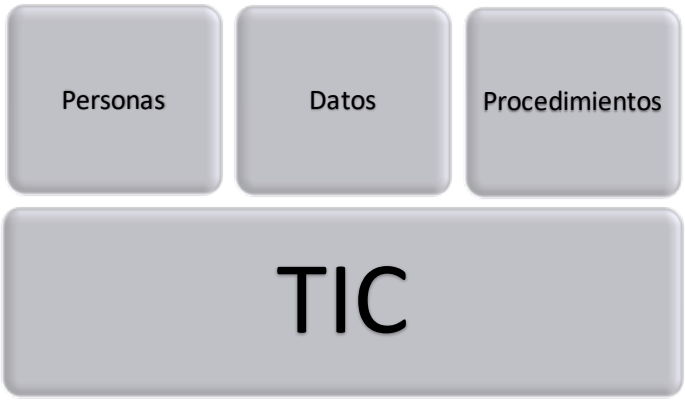


Figura 1.3 Componentes de un sistema de información.

Un sistema de información basado en ordenadores se articula sobre la tecnología *informática* (hardware y software) y la tecnología de las *comunicaciones* (redes, sean de voz o datos). El conjunto de estas tecnologías se engloba bajo el término en inglés de *Information and Communication Tecnology* (ICT). En castellano está más arraigado el término de Tecnologías de la Información y Comunicaciones: **TIC**.

En la mayoría de los sistemas de información las **personas** son el componente más importante, ya que son las que marcan la diferencia entre el éxito y el fracaso en las organizaciones. Se incluye tanto a las personas que gestionan, ejecutan, programan y mantienen el sistema, como a los usuarios del sistema. Concretamente, se identifican los siguientes grupos de personas (Fernández-Alarcón, 2006):

- **Propietarios de sistemas.** Son aquellas personas que patrocinan y promueven los sistemas de información. Entre sus funciones entran: fijar el presupuesto y los plazos para el desarrollo y el mantenimiento de los sistemas de información, y dar el visto bueno al sistema de información final.
- **Usuarios de sistemas.** Son las personas que utilizan los sistemas de información de una forma regular para capturar, introducir, validar, transformar y almacenar datos e información. Entre todos los grupos de individuos que participan en el desarrollo de un sistema de información, los usuarios es el más cuantioso. Dentro de este grupo se identifican además diferentes clases de usuarios: internos (personal administrativo, técnico, gestores y directivos) y externos (clientes, proveedores, aliados o partners, trabajadores externos).
- **Diseñadores de sistemas.** Son expertos en tecnología que resuelven las necesidades y las restricciones manifestadas por los usuarios de la empresa mediante recursos tecnológicos. Dada la divergencia de perspectivas entre usuarios y diseñadores, en el medio se introduce la figura del analista de sistemas.

- **Analistas de sistemas.** Son las personas que estudian los problemas y las necesidades en la organización para determinar como podrían combinarse personas, procedimientos, datos y TICs para obtener mejoras en la organización.
- **Constructores de sistemas.** Son los especialistas en tecnología encargados de fabricar el sistema de información, a partir de las especificaciones de diseño obtenidas de los diseñadores de sistemas.
- **Gestor de proyecto.** Es la persona que se encarga de planificar, supervisar y controlar proyectos en lo que concierne al calendario, el presupuesto, la satisfacción del cliente, las normas técnicas y la calidad del sistema.

Los **procedimientos** son también considerados como componente de un sistema de información. Incluyen las estrategias, políticas, métodos y reglas para el uso del sistema de información. Algunos procedimientos hacen referencia a la documentación o al procesamiento de flujos de información, otros a la protocolización, normalización de conductas. Por ejemplo, algunos procedimientos describen el orden en el que cada tarea debe ser ejecutada. Otros describen quién puede acceder a ciertos hechos en las bases de datos o qué hacer en caso de un desastre.

1.4. Cuestiones

1. ¿Qué relación hay entre sistemas informáticos y sistemas de información?
2. ¿Qué sistemas de información (basados en ordenadores) conoce?
3. Enumere algunos ejemplos de empresas cuyo modelo de negocio se basa expresamente en las TIC.
4. ¿Cómo mejoran los sistemas de información (basados en ordenadores) la toma de decisiones?

Capítulo 2
Ingeniería de sistemas y del software

Resumen Este capítulo describe la ingeniería de sistemas como un enfoque interdisciplinario para el desarrollo de sistemas complejos. Se analizan los procesos principales de soporte de la ingeniería de sistemas, según el estándar ISO-IEC15288 (2008). Se introduce también la ingeniería del software como la aplicación de los principios y procesos de la ingeniería de sistemas al desarrollo de sistemas software. Se examinan las peculiaridades del software, las etapas de desarrollo de un sistema software, los principales modelos de ciclo de vida y el Lenguaje Unificado de Modelado (*Unified Modeling Language - UML*) para la especificación de sistemas software.

2.1. Ingeniería de sistemas

La ingeniería de sistemas es un enfoque interdisciplinario para el diseño, la realización, la gestión técnica, el uso y el mantenimiento de sistemas complejos (Monforte Moreno et al, 2010).

El término *Ingeniería de Sistemas* fue acuñado la primera vez en el 1940 por los laboratorios Bell, en el marco del desarrollo de proyectos complejos para el Departamento de Defensa de los EEUU. Sin embargo la práctica en ingeniería de sistemas proliferó a partir de la segunda Guerra Mundial, porque resultaba un método eficiente para resolver problemas complejos, en particular para el desarrollo de misiles nucleares y satélites espaciales en órbita.

Desde entonces se han definido muchos estándares. El primero fue el estándar militar MIL-STD-499 (1969) publicado por la Fuerza Aérea de los EEUU, para su aplicación en la industria militar. Su versión mejorada, la versión bravo MIL-STD-499B (1993), sigue siendo un referente en este ámbito.

Las principales organizaciones americanas de estandarización han derivado sus estándares basados en el MIL-STD-499B (1993): IEEE-1220 (1998), ISO-IEC15288 (2008), INCOSE (2006), etc. Con el paso del tiempo y la experiencia, tanto en el ámbito civil como militar la tendencia está variando hacia la armonización de los estándares internacionales.

El objetivo que persigue la ingeniería de sistemas es obtener un sistema que satisfaga de forma eficaz y eficiente las necesidades del cliente, incrementando las posibilidades de éxito del proyecto, reduciendo los riesgos y los costes durante el desarrollo del sistema.

2.1.1. Procesos del ciclo de vida

Los estándares definen las tareas interdisciplinarias involucradas en el ciclo de vida del sistema con el objetivo de transformar las necesidades del cliente en una solución válida.

A continuación se considera el estándar ISO-IEC15288 (2008) que es una referencia clave para el desarrollo de sistemas complejos en diferentes dominios de aplicación (aeroespacial, TIC, transporte, financiero, administración, militar, naval, etc.).

El estándar ISO-IEC15288 (2008) organiza las tareas relacionadas entre sí en *procesos* e identifica cuatro principales grupos de procesos de soporte de la ingeniería de sistemas (véase Figura 2.1).

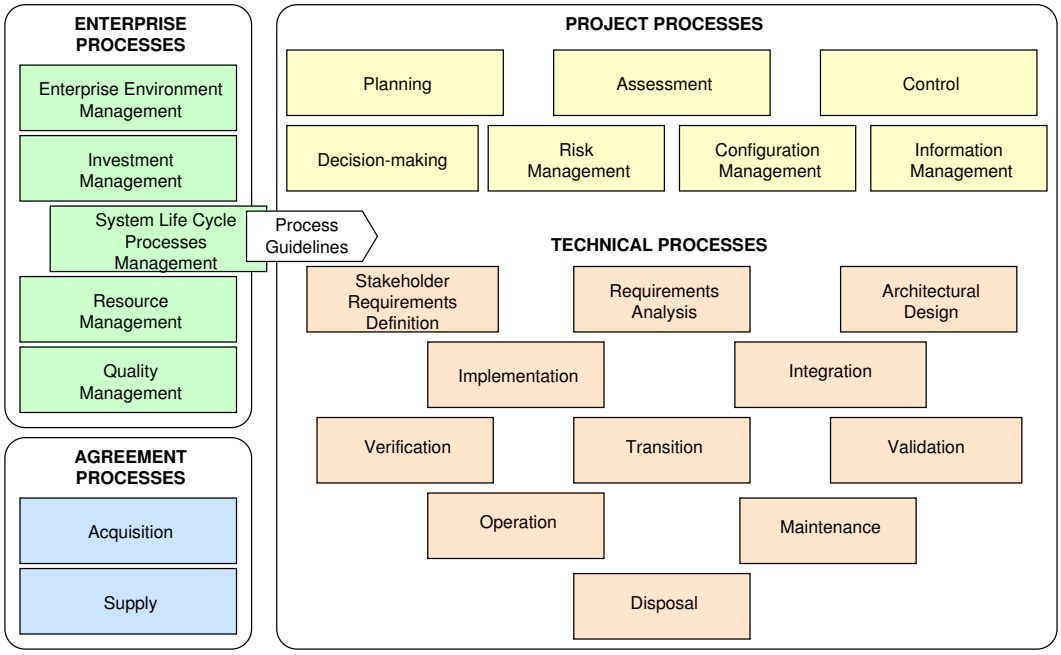


Figura 2.1 Procesos del ciclo de vida del sistema según ISO-IEC15288 (2008).

Los procesos técnicos (*technical processes*) incluyen las etapas fundamentales del desarrollo de un sistema:

- Definición de requisitos: recoger, negociar y documentar los requisitos de todas las partes interesadas (*stakeholders*), estudiar la viabilidad del sistema.
- Análisis de requisitos: revisar, evaluar, clasificar los requisitos por orden de prioridad, especificar los requisitos desde un punto de vista funcional y técnico (es decir, la especificación de los requisitos tiene que ser comprensible por las personas involucradas en los procesos técnicos de desarrollo).
- Diseño de la arquitectura: encontrar una solución que satisfaga los requisitos. En la fase de diseño se estudian posibles soluciones alternativas y se decide la estructura general que tendrá el sistema (es decir, su diseño arquitectónico).
- Implementación: concretar la solución. En el ámbito del desarrollo de un sistema de información, una vez que se sabe qué funciones debe desempeñar el sistema de información

(análisis) y se ha decidido cómo se organizan sus distintos componentes (diseño), es el momento de pasar a la implementación. Antes de escribir una sola línea de código (o de crear una tabla en la base de datos) es fundamental haber comprendido bien el problema que se pretende resolver y haber aplicado los principios básicos de diseño que permitan construir un sistema de información de calidad.

- Integración: combinar las partes del sistema conforme a la arquitectura diseñada. Las actividades de integración se llevan a cabo concurrentemente a las actividades de verificación y validación del sistema.
- Verificación: analizar la solución para averiguar si todos los requisitos tanto funcionales como no funcionales (por ejemplo, requisitos de rendimiento, de fiabilidad) están satisfechos, es decir verificar si el sistema se ha construido bien. Este proceso incluye varias actividades, como pruebas de unidad para comprobar el correcto funcionamiento de un componente del sistema, pruebas de integración que sirven para detectar errores en las interfaces de componentes, aplicación de modelos matemáticos para evaluar métricas de rendimiento, fiabilidad, etc. (por ejemplo, redes de Petri, redes de colas,...).
- Transición: entregar el sistema a la organización, poner en funcionamiento el sistema (su instalación o despliegue), adiestrar a los usuarios que lo utilizarán.
- Validación: confirmar que el sistema cumple con los requisitos de todas las partes interesadas. A diferencia de la verificación, el proceso de validación está sujeto a la aprobación de evaluadores/consultores externos.
- Operación: usar el sistema, hacer informes sobre el rendimiento, el mal funcionamiento del sistema, proveer recomendaciones para la mejora.
- Mantenimiento: eliminar los defectos que se detecten durante la vida útil del sistema, adaptar el sistema a nuevas necesidades de tipo tecnológico (por ejemplo, en el caso de un sistema software, la actualización del sistema operativo, uso de nuevo hardware, etc), añadir nuevas funcionalidades cuando se proponen características deseables que supondrían una mejora del sistema ya existente.
- Retirada del servicio: retirar un componente del sistema, manejar material peligrosos o tóxicos según reglamentación.

El desarrollo de un sistema se gestiona a través de proyectos. Los procesos de proyecto (*project processes*) se aplican conforme al riesgo y la complejidad de un proyecto. Incluyen tanto aquellas actividades de soporte a un proyecto específico (planificación, evaluación y control) como aquellas actividades más generales de gestión (toma de decisiones, gestión de riesgo, gestión de la configuración y gestión de la información).

- Planificación: establecer las direcciones e infra-estructura necesarias para evaluar y controlar el avance del proyecto, identificar en detalle las tareas, las personas, los recursos, definir un calendario de las actividades.
- Evaluación: recoger y evaluar el estado del proyecto, comparar los resultados obtenidos con los planes para establecer la madurez del proyecto. Las actividades de evaluación se programan periódicamente.
- Control: dirigir los esfuerzos del proyecto, establecer cambios cuando se verifican desviaciones al plan original o cuando el proyecto no refleja la madurez prevista.
- Toma de decisiones: analizar alternativas y tomar decisiones a lo largo del ciclo de vida de un sistema, documentar las decisiones tomadas.
- Gestión del riesgo: tratar con la incertidumbre presente durante el ciclo de vida, identificar los riesgos (de negocio, de proyecto, técnicos, etc.), estimar los riesgos, clasificarlos por orden de prioridad, definir estrategias para atenuarlos, conseguir un balance entre los posible riesgos y las oportunidades de negocio.

- Gestión de la configuración: identificar y definir los elementos de configuración del sistema (por ejemplo, en el ámbito de desarrollo de un sistema de información: la documentación, ejecutables y código fuente de aplicaciones software, pruebas, etc.); controlar los cambios en la configuración del sistema, modificar y definir versiones del sistema.
- Gestión de la información: identificar la información, recurso imprescindible del sistema a gestionar, definir su representación, garantizar la seguridad de la información (integridad, disponibilidad, confidencialidad).

Los procesos de negocio (*enterprise processes*) proveen las pautas para la ejecución de los procesos mencionados previamente:

- Gestión del proyecto (*enterprise environment management*): establecer y mantener políticas y procedimientos a nivel de organización (o empresa) que apoyan la capacidad de la organización para la adquisición y el suministro de productos o servicios.
- Gestión de la inversión: iniciar y mantener las inversiones en los proyectos que cumplen con los objetivos de la organización y retirar o cambiar las inversiones en los proyectos que no cumplen.
- Gestión de los procesos del ciclo de vida del sistema: establecer un modelo de desarrollo eficiente para el ciclo de vida del sistema, adaptar las directrices generales a las necesidades de la organización, establecer medidas de evaluación de los procesos, monitorizar la ejecución de los procesos. En la Subsección 2.2.2 se consideran los principales modelos de desarrollo de sistemas software.
- Gestión de los recursos: crear y mantener el conjunto de recursos necesarios para la organización. Entre los recursos se incluye el personal de la organización.
- Gestión de la calidad: establecer las pautas para la gestión de la calidad (políticas, estándares, procedimientos), establecer los objetivos de calidad a nivel de organización, evaluar la satisfacción del cliente, evaluar los proyectos, identificar y introducir posible mejoras. Este proceso ha sido tratado en detalle en la asignatura de calidad (Acero et al, 2012).

Finalmente los procesos de negociación (*agreement processes*) incluyen la adquisición y el suministro de productos o servicios.

Sistemas de sistemas

El incremento en la complejidad de los sistemas ha dado lugar a la definición de *sistemas de sistemas*, es decir, sistemas integrados a gran escala, heterogéneos y operables independientemente unos de otros, conectados por red para alcanzar un objetivo común.

Un ejemplo de sistema de sistemas en el entorno militar, es un NEC (*Network Enabled Capability*) donde los componentes de las capacidades militares (equipamiento, entrenamiento, personal, infraestructura, doctrina, organización, información y logística) no sólo se comunican por red sino que se benefician de una visión y entendimiento compartido para conseguir una mejor toma de decisiones (Monforte Moreno, 2012).

2.2. Ingeniería del software

Los procesos definidos por la ingeniería de sistemas, son directrices generales que tienen que ser adaptadas al tipo de sistema. Por ejemplo, el Ministerio de Hacienda y Adminis-

traciones Públicas aplica la metodología METRICA (2012) para la sistematización de las actividades que dan soporte al ciclo de vida de sus sistemas de información, donde los procesos considerados son un sub-conjunto de los mencionados en la subsección 2.1.1.

La ingeniería del software es una disciplina que se ocupa de estudiar y aplicar los principios y métodos de la ingeniería al desarrollo, uso y mantenimiento del software (Abran et al, 2004). Se puede ver como la ingeniería de sistemas aplicada a sistemas software.

El software tiene características peculiares que hay que tener en cuenta para comprender la dificultad de su proceso de desarrollo, como la complejidad, la variabilidad y la invisibilidad (Monforte Moreno et al, 2010).

- **Complejidad**. El número de estados de un sistema software supera en varios órdenes de magnitud al de un dispositivo digital. Además, el aumento de la escala de producción de una entidad software no se resuelve con elementos de mayor tamaño, sino con el aumento del número de elementos diferentes, que en la mayoría de los casos interactúan entre si de forma no lineal, aumentando la complejidad. La complejidad del software se refleja en la dificultad de especificar el software de manera completa y coherente, de extender el software con nuevas funcionalidades, de gestionar un proyecto de desarrollo del software, etc.
- **Variabilidad**. A diferencia de otros procesos productivos (por ejemplo coches, edificios, etc.), el software está sujeto a cambios continuos tanto durante el proceso de su desarrollo como durante su funcionamiento.
- **Invisibilidad**. La construcción de edificios o piezas industriales utiliza abstracciones geométricas de la realidad que se desea construir, permitiendo al diseñador y contratista detectar contradicciones y omisiones fácilmente. El software es *intangible* por eso no es posible realizar una representación geométrica del mismo. La especificación del software se realiza típicamente a través del modelado, es decir, su estructura se define a través de un conjunto de diagramas relacionados, donde cada diagrama representa una abstracción del software según un determinado punto de vista. El punto de vista estático hace hincapié sobre los componentes del software y las relaciones entre ellos, el punto de vista dinámico hace hincapié sobre el *comportamiento* del software durante su ejecución e incluye la especificación del flujo de control, el flujo de los datos, las secuencias temporales de estados, etc.

2.2.1. Etapas del proceso de desarrollo del software

Los procesos de la ingeniería de sistemas, descritos en la Subseccción 2.1.1, se aplican también para el desarrollo del software. Cualquier sistema software va pasando por una serie de etapas a lo largo de su vida. En cada etapa se realizan una serie de tareas relacionadas a los procesos mencionados anteriormente. Sin embargo, las tareas concretas que se realicen (su grado de rigor y el orden en que se lleven a cabo) dependerán de la naturaleza del proyecto al que nos enfrentemos.

A continuación se analizan, en detalle, los procesos técnicos y de gestión de un proyecto software. Los procesos de negocio y de negociación están fuera del ámbito de estudio de este libro. Sin embargo, en el Capítulo 4, se volverán a mencionar los procesos de negocio para contextualizar la disciplina de modelado de negocio.

2.2.1.1. Procesos técnicos

Requisitos

La definición y el análisis de requisitos son las primeras tareas a realizar en el proceso de desarrollo de un sistema software. Los requisitos son el conjunto de características que debe poseer y restricciones que debe cumplir el sistema para satisfacer las necesidades de las partes interesadas (clientes, usuarios, etc.). En general, un requisito debe ser:

- **Alcanzable:** debe ser un objetivo realista, posible de alcanzar con los recursos disponibles y dentro de los plazos establecidos.
- **Completo:** debe contener toda la información necesaria, y no remitir a otras fuentes externas que lo expliquen con más detalle.
- **Consistente:** no debe entrar en conflicto con otros requisitos.
- **No ambiguo:** debe ser claro, preciso y tener una única interpretación posible.
- **Verificable:** se debe poder verificar mediante inspección, análisis, demostración o pruebas.
- **Trazable:** se debe poder identificar a lo largo del ciclo de vida y relacionar con otros requisitos.

La etapa de establecimiento de requisitos es esencial y crítica al mismo tiempo: esencial porque si no se sabe con precisión qué es lo que se necesita, ningún proceso de desarrollo permitirá obtenerlo; crítica porque, en ocasiones, las partes interesadas desconocen con exactitud sus necesidades o no pueden expresarlas con claridad. Las técnicas utilizadas para la obtención y análisis de requisitos son varias, en el Capítulo 3 se ilustrarán técnicas de modelado basadas en la definición de casos de uso.

Diseño

Una vez especificados los requisitos, la siguiente etapa es diseñar el sistema. Tampoco esta es una tarea sencilla, ya que se trata de resolver un problema que puede tener muchas soluciones. En la fase de diseño se analizan las posibles soluciones alternativas y se define la *arquitectura* del sistema, es decir cómo se descompone y organiza el software en sub-sistemas o componentes. A partir de la arquitectura del sistema, se procede con el *diseño detallado*, en el que se describe el comportamiento de cada sub-sistemas o componentes.

Implementación

La etapa de implementación incluye la codificación del diseño, la verificación mediante pruebas, la depuración del código. En esta etapa hay que seleccionar las herramientas adecuadas, un entorno de desarrollo que facilite las tareas y un lenguaje de programación apropiado para el tipo de sistema a desarrollar. La elección de estas herramientas dependerá en gran parte de las decisiones de diseño tomadas y del entorno en el que el sistema deberá funcionar. También se desarrollan casos de pruebas que permitan ir comprobando el funcionamiento del sistema conforme se vaya construyendo.

Pruebas

La etapa de pruebas tiene como objetivo detectar los errores/fallos que se hayan podido cometer en las etapas anteriores del proyecto (y, finalmente, corregirlos). Cuanto antes se descubra un error mejor, ya que el descubrimiento de fallos en las etapas tempranas implica un coste menor para su corrección. La actitud del personal involucrado en las pruebas debe ser colaborativa, de forma que el descubrimiento de errores se enfoque como algo positivo que aporta un valor añadido al producto, contribuyendo al aseguramiento de la calidad del sistema.

La búsqueda de errores puede tener distintas formas, en función del contexto y de la etapa del proyecto:

- Las **pruebas de unidad** sirven para comprobar el correcto funcionamiento de un componente concreto de nuestro sistema. En este tipo de pruebas se deben buscar situaciones límite que expongan las limitaciones de la implementación del componente, ya sea tratando a éste como una caja negra ("pruebas de caja negra") o fijándose en su estructura interna ("pruebas de caja blanca").
- Las **pruebas de integración** verifican la interacción entre componentes/elementos software.
- Las **pruebas de sistema** se centran en su comportamiento general. Este nivel comprueba además el cumplimiento de requisitos no funcionales como la facilidad de uso, la fiabilidad, el rendimiento, etc.

Mantenimiento

La última etapa de desarrollo es el mantenimiento. Una vez instalado el sistema software es necesario darle soporte para garantizar su funcionamiento hasta la finalización del ciclo de vida (es decir, hasta cuando se decide migrar o retirar el sistema software). Hay diferentes tareas de mantenimiento que se clasifican a continuación.

- **Correctivas:** se modifica el sistema para eliminar defectos (*residual bugs*, en inglés).
  Un defecto en el sistema (p. ej. agujero de seguridad) puede ser causa de fallos software, es decir el sistema software no funciona como fue diseñado o como especificado en los requisitos. Hay varios tipos de defectos: de diseño, lógicos, de implementación. Los defectos de diseño se producen cuando, por ejemplo, los cambios realizados en el software son incorrectos, incompletos, mal comunicados por las partes interesadas o mal interpretados por los analistas. Los defectos lógicos son el resultado de test no válidos o incompletos, de la implementación incorrecta del diseño. Los defectos de implementación son causados por la codificación errónea del diseño detallado. Las acciones correctivas se concretan a menudo en soluciones de emergencia conocidas como *parches* (*patching*, en inglés) que suelen dar lugar a un incremento de la complejidad del software y provocar un efecto dominó, es decir a consecuencia de la corrección de un error hay que realizar cambios adicionales en el sistema software.
- **Adaptativas:** se modifica el sistema para que pueda funcionar en un entorno de operación distinto.
  Cualquier acción que se comienza como consecuencia de los cambios en el entorno en el que un sistema software debe operar se denomina *cambio adaptativo*. Un cambio adaptativo es un cambio impulsado por la necesidad de acomodar las modificaciones en el entorno de operación del sistema software, sin el cual el sistema se haría cada vez menos útil hasta convertirse en obsoleto. El término *entorno de operación* se refiere a todas las condiciones

que actúan desde fuera sobre el sistema: las reglas de negocio (p. ej. en el contexto de un sistema software de gestión de las ventas de productos en una tienda, el porcentaje de descuento aplicado en período de rebajas), las políticas gubernamentales (p. ej., el cálculo de los impuestos sobre las ventas), las plataformas hardware (p. ej., pantallas táctiles, lectores de código de barra) y software (p. ej., sistema operativo). Un cambio a la totalidad o parte de este entorno de operación impulsará cambios adaptativos en el sistema software. Con este tipo de mantenimiento, los usuarios del sistema software no suelen ver un cambio directo en el funcionamiento del sistema, pero los responsables del mantenimiento tendrán que gastar recursos para efectuar el cambio.

- **Perfectivas:** se modifica el sistema para añadir nuevas funcionalidades o mejorar su calidad.
  Las acciones de mantenimiento perfectivo incluyen todos los cambios, ampliaciones y mejoras para satisfacer las nuevas necesidades de los usuarios (p. ej., la firma digital en una banca en línea). Estas acciones se basan en la premisa de que ya que el sistema software llega a ser útil, los usuarios tienden a experimentar con nuevos casos más allá del alcance para el que fue desarrollado inicialmente. Los requisitos pueden mejorar las funcionalidades del sistema existente o su calidad, en particular su rendimiento y mantenibilidad.
- **Preventivas:** se modifica el sistema para detectar y corregir defectos latentes antes de que produzcan fallos y degrade el rendimiento.
  El mantenimiento preventivo es similar al perfectivo, pero ha de involucrar también modificaciones para prevenir el *envejecimiento del software* (*software aging*, en inglés), es decir fallos y la degradación con el tiempo del rendimiento del sistema software (Yurcik and Doss, 2001). Hay muchos ejemplos de software utilizado masivamente que ha sufrido el fenómeno de envejecimiento, como por ejemplo el navegador Netscape, el sistema operativo Windows95. Para prevenir el envejecimiento del software se utilizan técnicas de *rejuvenecimiento del software* (*software rejuvination*). El rejuvenecimiento es un enfoque pro-activo que implica detener la ejecución del sistema software periódicamente, efectuar la limpieza del estado, y luego reiniciar el software. Puede involucrar a todas o algunas de las siguientes acciones: colección de basura (*garbage collection*), des-fragmentación de la memoria, reinicialización de las estructuras de datos internas de los programas.

2.2.1.2. Gestión del proyecto

La gestión de un proyecto software, igual que otro proyecto, requiere una gestión que permita cumplir con los requisitos en los plazos y costes establecidos. Las actividades iniciales de gestión comprenden la determinación del ámbito del proyecto, la realización de un estudio de viabilidad, el análisis de los riesgos asociados al proyecto, una estimación del coste del proyecto, su planificación temporal y la asignación de recursos a las distintas etapas del proyecto. Durante el desarrollo del sistema, las actividades de gestión consisten en la monitorización y control de los procesos técnicos que se van ejecutando. Según la planificación adoptada, habrá determinados puntos del proyectos o hitos (*milestones*), donde se debe verificar la satisfacción de los requisitos y resolver los problemas que surjan. En todo el proyecto de desarrollo resulta fundamental una adecuada gestión de la configuración del software (*software configuration management*), más conocida vulgarmente por uno de sus aspectos, el control de versiones. Independientemente de su importancia, el valor del control de versiones es incalculable para evitar pérdidas irreparables y también para volver a una versión anterior de los artefactos (documentación, código, etc.) si las últimas modificaciones no resultaron del todo acertadas. Las actividades finales de gestión comprenden la com-

probación de que se han completado todas las tareas y las mediciones para identificar la necesidades de información a lo largo del proyecto y proveer resultados para la mejora de proyectos futuros.

2.2.2. Modelos de ciclo de vida del software

Existen distintos modelos de desarrollo, es decir distintas formas de organizar el orden concreto en el que se acometerán las etapas del ciclo de vida del software y también distintas maneras de referirse a estas etapas. A continuación se analizan los principales modelos de desarrollo del software utilizados (Inteco, 2009; Brown and Wallnau, 1996).

2.2.2.1. Modelo en cascada

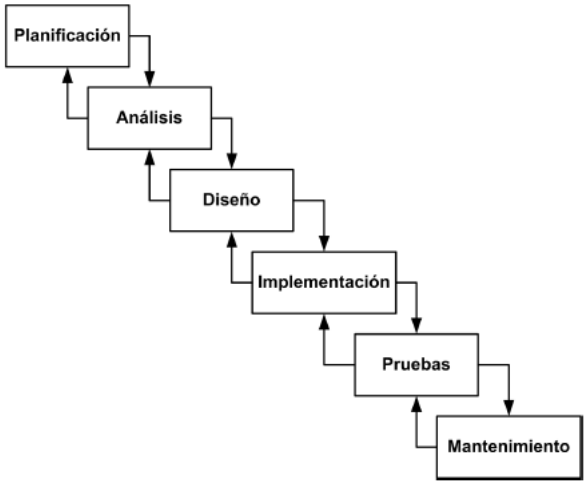


Figura 2.2 Modelo en cascada.

El modelo *en cascada* es el ciclo de vida clásico (véase Figura 2.2): se pasa, en orden, de una etapa a la siguiente sólo tras finalizar con éxito las tareas de verificación y validación propias de la etapa. Si resulta necesario, únicamente se da marcha atrás hasta la fase inmediatamente anterior. Este modelo tradicional de ciclo de vida exige una aproximación secuencial al proceso de desarrollo que, desgraciadamente, presenta una serie de graves inconvenientes cuando se trata de desarrollar software:

- Los proyectos reales raramente siguen el flujo secuencial de actividades que propone este modelo.
- Normalmente, es difícil para el cliente establecer explícitamente todos los requisitos al comienzo del proyecto y, además, los requisitos del software suelen cambiar durante el desarrollo.
- No habrá disponible una versión operativa del sistema hasta llegar a las etapas finales del proyecto, por lo que la rectificación de cualquier decisión tomada erróneamente en las

etapas iniciales del proyecto supondrá un coste adicional significativo, tanto económico como temporal.

2.2.2.2. Modelo iterativo

Los modelos iterativos consisten en descomponer un proyecto de desarrollo de software en una serie de subproyectos de menor envergadura. Cada subproyecto incluye todas las etapas del ciclo de vida pero se centra en aportar nuevas funcionalidades para el sistema desde el punto de vista del usuario final del mismo. Los analistas establecen prioridades entre los requisitos iniciales del sistema para decidir qué parte del mismo se construirá primero. La asignación de las prioridades a los requisitos se basa en varios criterios: el riesgo (ej., complejidad técnica, incertidumbre del esfuerzo o de la facilidad de uso), la cobertura de las partes principales del sistema, la criticidad de las funcionalidades según el cliente y los usuarios finales.

Los modelos iterativos permiten adelantar el momento en el que se determina si un proyecto es técnicamente viable o no (con lo que se eliminan costes innecesarios si, finalmente, el proyecto hubiese de cancelarse). También promueven una mejor comunicación con el usuario/cliente, ya que se dispondrá antes de una versión operativa del sistema, aunque sea de funcionalidad reducida. Estas versiones intermedias del producto ayudan a la eliminación de malentendidos que pueden surgir en la etapa de recogida de requisitos. Además, ayudan a que el usuario se forme una idea más clara de lo que realmente necesita.

Para planificar un proyecto que siga un modelo iterativo, primero se prepara una descomposición a grandes rasgos del proyecto en una serie de iteraciones, cada una de las cuales se considerará como un proyecto independiente. En vez de realizar una planificación detallada de todo el proyecto, sólo se detallará el plan correspondiente a la primera iteración. Sí se establecerán las fechas de inicio y finalización de las distintas iteraciones y se definirán los objetivos principales de cada una de ellas. Estos objetivos se pueden redefinir conforme avance el proyecto.

A lo largo de los años se han propuesto varios modelos iterativos de desarrollo de software, entre ellos destacamos el modelo en espiral y el modelo iterativo-incremental.

Modelo en espiral

El modelo *en espiral* de Barry Boehm hace especial hincapié en la prevención de riesgos. Este modelo define cuatro actividades principales: planificación (determinar los objetivos, alternativas y restricciones del proyecto), análisis de riesgos (análisis de alternativas e identificación/resolución de riesgos), ingeniería (desarrollo del producto) y evaluación (revisión por parte del cliente y valoración de los resultados obtenidos de cara a la siguiente iteración). En cada iteración alrededor de la espiral se construyen versiones cada vez más completas del software.

Modelo iterativo-incremental

El modelo *iterativo-incremental* se caracteriza por realizar entregas por etapas del sistema. Los métodos ágiles de desarrollo (entre los que se encuentra la programación extrema XP, Scrum) y el Proceso Unificado (UP) se basan en este modelo. Usualmente, el proyecto se

Figura 2.3 Modelo en espiral.

descompone en iteraciones de longitud fija (*timeboxed*), de 1 a 6 semanas, y cada iteración ha de proporcionar algún aspecto completo de la funcionalidad del sistema. Cada ciclo se focaliza en las funciones de mayor valor añadido. De esta forma, si se cancelase el proyecto en cualquier momento, el usuario siempre tendrá lo máximo que se puede conseguir con los recursos invertidos hasta el momento. Igualmente, se puede prorrogar el proyecto si se considera interesante seguir añadiéndole funcionalidades al sistema.

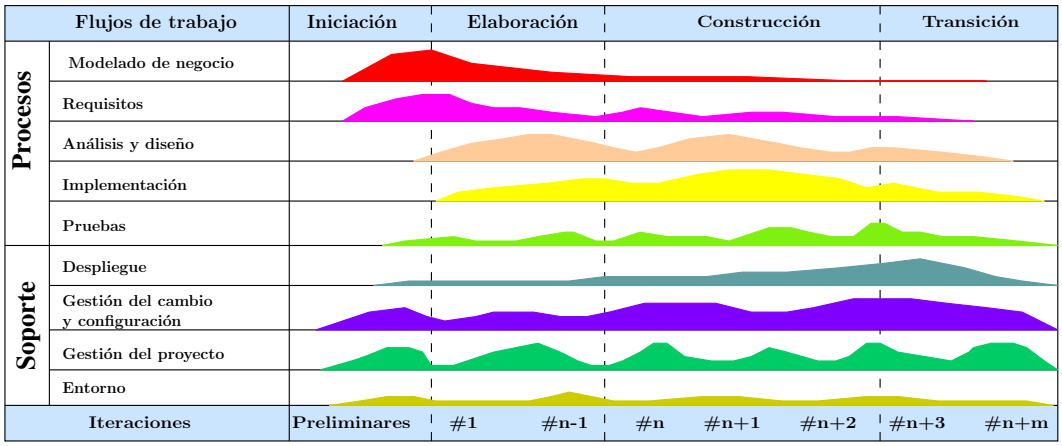


Figura 2.4 Modelo iterativo e incremental: el Proceso Unificado.

En particular, en el Proceso Unificado - esquematizado en Figura 2.4 - el ciclo de vida de un proyecto se descompone en cuatro fases principales (iniciación, elaboración, construcción y transición). Cada una de estas fases, a su vez, está caracterizada por una serie de objetivos (*milestones*) y se puede descomponer en un número de iteraciones de duración fija: cada iteración corresponde al desarrollo de un subproyecto. Los procesos de trabajo, descritos en la subsección 2.1.1 en el ámbito de la ingeniería de sistemas, se concretan en actividades

llamadas *disciplinas*. El Proceso Unificado es un proceso de desarrollo guiado por el modelado (*model driven development*) y usa el lenguaje Unificado de Modelado (*Unified Modeling Language*) para especificar el software a lo largo del ciclo de vida. Véase la subsección 2.2.3 para una introducción a UML.

2.2.2.3. Modelo en V

Es una variante del modelo en cascada que trata de resolver algunos de los inconvenientes de este último (Inteco, 2009). En particular, unos de los problemas del modelo en cascada es que los errores o defectos siempre se encuentran demasiado tarde en el ciclo de vida, ya que las pruebas no se realizan hasta el final del proyecto. En el *modelo en V*, representado en Figura 2.5, las pruebas se empiezan lo más pronto posible en el ciclo de vida. Las pruebas no son sólo una actividad basada en la ejecución del software; hay una variedad de actividades que se han de realizar antes del fin de la fase de codificación (o implementación). Estas actividades de verificación y validación se llevan a cabo en paralelo con las actividades de definición y análisis de requisitos, diseño y codificación. Los técnicos de pruebas necesitan trabajar con los desarrolladores y analistas de negocio de tal forma que puedan realizar estas actividades y producir una serie de pruebas. Dentro del modelo en V, las pruebas de validación tienen lugar durante las etapas tempranas, por ejemplo, revisando los requisitos de usuario y, después, durante las pruebas de aceptación de usuario. La parte izquierda de la V representa la descomposición de los requisitos y la creación de las especificaciones del sistema. El lado derecho de la V representa la integración de partes y su verificación. La V significa *Validación y Verificación*.

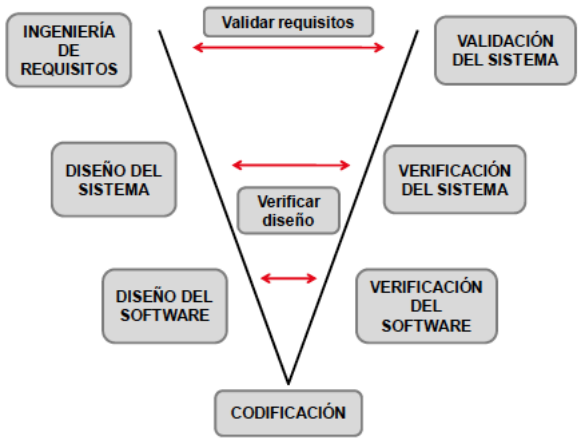


Figura 2.5 Modelo en V.

Entre las críticas que se le hacen a este modelo se destacan las siguientes:

- Es un modelo muy rígido, como el modelo en cascada.
- Tiene poca flexibilidad y ajustar el alcance es difícil y caro.
- El software se desarrolla durante la fase de codificación, por lo que no se producen prototipos del software.

- El modelo no proporciona caminos claros para problemas encontrados durante las fases de pruebas.

La contratación y el desarrollo de sistemas de información para el mando y control deben seguir los procedimientos establecidos en la adquisición de programas del Ministerio de Defensa. En particular, la contratación es estricta en cuanto a costes y plazos, por lo que se utiliza un sistema, *PAPS* (*NATO Phased Armament Programming System*), basado en el modelo en V que integra medidas de aseguramiento de la calidad como la verificación y validación (Monforte Moreno et al, 2010). Sin embargo, los proyectos desarrollados según el acercamiento PAPS han tenidos dificultades en términos de retrasos en las entregas, sobre-costes y baja aceptación por parte del usuario final (es decir, el ejército). El modelo iterativo-incremental es, seguramente, más apropiado por que aporta una mayor flexibilidad, alcanzando resultados antes que en el modelo en V y permitiendo así una evaluación por parte de los usuarios finales en una etapa temprana, de forma que los cambios en los requisitos se pueden realizar a un coste más reducido.

2.2.2.4. Modelo basado en componentes

Tradicionalmente los ingenieros del software han seguido un enfoque de desarrollo descendente (*top-down*) basado en las etapas de análisis-diseño-implementación para la construcción de sus sistemas, donde se establecen los requisitos y la arquitectura software y luego se va desarrollando, diseñando e implementando cada parte de software hasta obtener la aplicación completa implementada. En cambio, el modelo basado en componentes es un modelo de desarrollo ascendente (*bottom-up*), donde se ensamblan e integran componentes software ya existentes.

Un componente software es una unidad de composición de aplicaciones software, que posee un conjunto de requisitos, ofrece servicios predefinidos y es capaz de comunicarse con otros componentes por medio de interfaces. Los requisitos determinan las necesidades del componente en cuanto a recursos, como por ejemplo las plataformas de ejecución que necesita para funcionar. Las interfaces de un componente determinan tanto las operaciones que el componente implementa, es decir los servicios que ofrece, como las que precisa utilizar de otros componentes durante su ejecución.

A partir de los componentes reutilizables dentro de un mercado global de software nace el concepto fundamental en estos entornos, el de componente COTS (*Commercial Off-The-Shelf*). Este concepto cambia la idea tradicional del desarrollo de software donde todo es propietario - o la reutilización se reduce a un ámbito local (el de la empresa que desarrolla el software), hacia nuevas metodologías de desarrollo de software donde es posible contar con elementos externos.

El proceso de desarrollo de sistemas basados en componentes está compuesto por diferentes etapas, aunque en la literatura podemos encontrar distintas categorías y formas a la hora de referirse a estas etapas. La Figura 2.6 esboza el modelo de Brown and Wallnau (1996) que incluye cuatro fases: (a) la selección de componentes; (b) la adaptación de componentes; (c) el ensamblaje de los componentes al sistema; y (d) la evolución del sistema.

Durante la fase de selección se identifican los componentes de posible interés, que se pueden encontrar localmente o remotamente (desarrollados por otras empresas). Se analizan en detalle para descubrir sus interfaces y el resultado de la selección es un conjunto de componentes cualificados. Durante la siguiente fase de adaptación se corrigen posibles conflictos entre los servicios ofrecidos por los componentes cualificados obteniendo componentes adaptados. La composición de los componentes adaptados se realiza durante el ensamblaje,

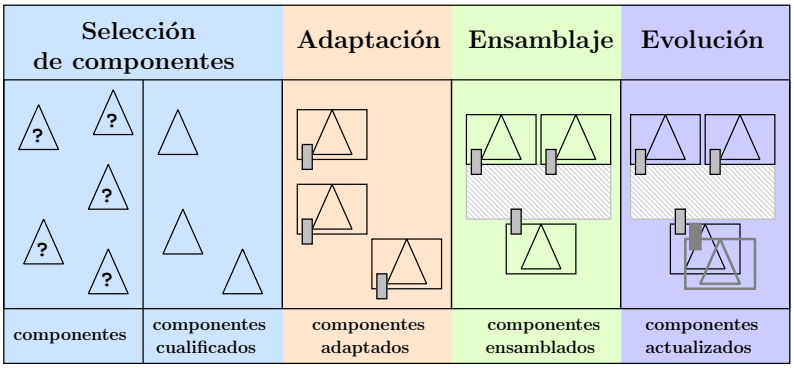


Figura 2.6 Modelo basado en componentes de (Brown and Wallnau, 1996).

según un estilo de arquitectura (por ejemplo, CORBA (OMG, 2011)). Finalmente, la fase de evolución corresponde con la etapa de mantenimiento del sistema, donde los componentes se reemplazan con nuevas versiones o con diferentes componentes que proveen servicios similares.

El modelo basado en componentes se usa principalmente para el desarrollo de sistemas software distribuidos y *abiertos*. Los sistemas abiertos son sistemas concurrentes, reactivos, extensibles y permiten a componentes heterogéneos ingresar o abandonar el sistema de forma dinámica.

2.2.3. Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado - *Unified Modeling Language* (UML), en inglés - es un lenguaje visual para la especificación, el desarrollo y la documentación de un sistema software. UML (2010) es un estándar OMG[1], desde el 1997, que incluye un conjunto de diagramas para modelar un sistema desde diferentes puntos de vista (véase Cuadro 2.1). Los diagramas estáticos describen la estructura de un sistema, los diagramas dinámicos describen el comportamiento del sistema, es decir sus funcionalidades.

Diagramas estáticos	Diagramas dinámicos
Diagrama de clases	Diagrama de casos de uso
Diagrama de componentes	Diagrama de actividades
Diagrama de objetos	Diagrama de máquinas de estados
Diagrama de paquetes	Diagrama de secuencias
Diagrama de estructuras compuestas	Diagrama de comunicación
Diagrama de despliegue	Diagrama de tiempo (cronogramas)
	Diagrama global de las interacciones

Cuadro 2.1 Diagramas UML.

Todos los diagramas de UML se suelen utilizar a lo largo del proceso de desarrollo de un sistema, y cada diagrama se puede usar para diferentes propósitos. Por ejemplo, los

[1] Enlace: `http://www.omg.org/`

diagramas de clases se pueden usar: 1) en la etapa temprana de análisis de un sistema para capturar los conceptos y definir un glosario de términos en el marco de un dominio de aplicación y 2) en la etapa de diseño de un sistema software para definir los objetos software con sus atributos y métodos.

Hay que observar que **UML no es una metodología de desarrollo de sistemas**, es simplemente un conjunto de notaciones que se usan en el contexto de una metodología. Se pueden distinguir tres maneras de aplicar UML:

- *UML como borrador.* Diagramas informales e incompletos (a menudo esbozados a mano en una pizarra), que se crean para explorar partes complejas de un problema o de una solución, explotando la expresividad de los lenguajes visuales. Esta manera de aplicar UML es típica de las metodologías *ágiles* de desarrollo.
- *UML como proyecto.* Diagramas de proyecto detallados utilizados para 1) la generación de código (*forward engineering*, en inglés) o para 2) la ingeniería inversa (*reverse engineering*, en inglés) y comprender el código existente. Antes de programar, unos diagramas detallados pueden ser una guía para la generación del código, obtenido automáticamente con una herramienta CASE (*Computer Aided Software Engineering*). Normalmente los diagramas se usan para especificar parte del código, durante la implementación el desarrollador completa el resto del código.
- *UML como lenguaje de programación.* La especificación completa del software con UML. El código se genera automáticamente y no es modificado por los desarrolladores. Este uso de UML está aún en fase de desarrollo, en términos tanto teóricos como de facilidad de uso de herramientas CASE de soporte.

UML es un lenguaje de modelado de propósito general, sin embargo es posible definir *perfiles* UML para adaptar el lenguaje a un dominio específico de aplicación. Por ejemplo, el perfil DoDAF-MODAF (2012) -que es también un estándar OMG- adapta la notación UML[2] para el modelado de sistemas de información usados por la Defensa de EEUU y Británica.

En este libro se utilizan tres tipos de diagramas UML: los diagramas de casos de uso para la especificación de los requisitos de un sistema software, los diagramas de clase para la definición del modelo de dominio de un sistema software y los diagramas de actividades para la especificación de los procesos de negocio. En el Apéndice C se presentan ejemplos de aplicación de los tres tipos de diagramas UML, haciendo hincapié sobre los elementos de modelados más importantes. Para un estudio más profundo se aconseja la consultación de las referencias Debrauwer and VanDerHeyde (2011); Fowler, M. (2004).

[2] También se basa sobre otro lenguaje de modelado, SysML, utilizado en la ingeniería de sistemas.

2.3. Cuestiones

1. ¿Qué es la ingeniería de sistemas?
2. ¿Cuáles son los procesos a soporte de la ingeniería de sistemas según el estándar ISO-IEC15288?
3. ¿Qué es un *sistema de sistemas*? Proveer un ejemplo en el ámbito militar.
4. ¿Qué es la ingeniería del software y qué relación tiene con la ingeniería de sistemas?
5. ¿Cuáles son las características del software que lo hacen diferente de otros productos?
6. Describir las etapas del proceso de desarrollo del software.
7. ¿En qué consiste la gestión de un proyecto software?
8. Mencionar los principales modelos de desarrollo del software, indicando para cada uno las principales ventajas/desventajas.
9. ¿Qué es el Lenguaje Unificado de Modelado (UML) y para qué se usa?
10. ¿Qué tipos de diagramas incluye UML?
11. ¿Para qué se usan los diagramas de casos de uso, los diagramas de clase y los diagramas de actividades?

Parte II
Desarrollo de un sistema de información: etapas tempranas

Capítulo 3
Definición y análisis de requisitos con casos de uso

The indispensable first step to getting the things you want out of life: decide what you want. — Ben Stein

Resumen La definición y análisis de requisitos son las primeras tareas a llevar a cabo en el desarrollo de un sistema de información o sistema software. Este capítulo, introduce el concepto de requisito, la tradicional clasificación de requisitos en *funcionales / no funcionales* y las técnicas utilizadas para la recogida y especificación de requisitos. Posteriormente, se describe una técnica en concreto para la captura y especificación sistemática de los requisitos funcionales: los casos de uso.

3.1. Introducción

Las primeras etapas de desarrollo de un sistema de información o sistema software son la definición y el análisis de requisitos Larman (2004). Los requisitos son el conjunto de *características* que debe poseer y *restricciones* que debe cumplir el sistema para satisfacer a las necesidades de las *partes interesadas*.

Los requisitos de un sistema se clasifican tradicionalmente en requisitos funcionales (o de comportamiento) y requisitos no funcionales (o de calidad). Los primeros indican *qué tendrá que hacer* el sistema mientras los segundos indican *cuán bien tendrá que funcionar* el sistema. Por ejemplo:

- *El sistema tiene que calcular el precio del viaje en base a los siguientes parámetros: número de km a recorrer, precio actual de la gasolina y número de pasajeros.* (requisito funcional)
- *El tiempo de respuesta del sistema tiene que ser menor que/igual a 30 segundos, para el 90 % de las peticiones* (requisito de rendimiento)
- *El sistema tiene que proveer un servicio por lo menos 360 días/año, es decir la disponibilidad de servicio tiene que ser mayor o igual al 98 %* (requisito de fiabilidad)
- *La interfaz de usuario tiene que ser de tipo táctil y el texto tiene que ser visible a una distancia de un metro* (requisito de facilidad de uso).

El primer requisito es funcional porque especifica lo que tiene que hacer el sistema, los demás son requisitos no funcionales porque expresan calidades del sistema o de los servicios ofrecidos por el sistema.

En el método de desarrollo del Proceso Unificado Jacobson et al (1999), los requisitos se clasifican según el modelo FURPS+ (acrónimo de Functional, Usability, Reliability, Performance, Supportability):

(F) Funcionales: características de comportamiento del sistema, capacidades, seguridad.
(U) Facilidad de uso: factores humanos, ayuda, documentación.
(R) Fiabilidad: frecuencia de fallos, capacidad de reparación.
(P) Rendimiento: tiempo de respuesta, productividad, uso de recursos.
(S) Capacidad de soporte: adaptación, mantenimiento, internacionalización, configurabilidad.
(+) Otros: de implementación (por ejemplo, limitación de recursos/lenguajes/herramientas/hardware...), de interfaz (restricciones relacionadas a la necesidad de comunicación con sistemas externos), de operación (gestión del sistema en el contexto de operación), legales (por ejemplo, uso de licencias software).

Se hace constar que actualmente no hay un consenso, entre los estándares, en la clasificación de los requisitos. Por ejemplo, algunos estándares consideran los requisitos de seguridad como funcionales Jacobson et al (1999) otros como no funcionales IEEE-830 (1998). Por otro lado, dependiendo de cómo se especifica, un requisito puede expresar una propiedad funcional o no funcional. Considérese las siguientes especificaciones del mismo requisito (de seguridad):

Caso1. *El sistema tiene que proporcionar un servicio de autenticación para el acceso a los datos únicamente a los usuarios autorizados.*
Caso2. *El sistema tiene que garantizar la accesibilidad de los datos únicamente a los usuarios autorizados.*

El primer caso expresa un requisito funcional: *el servicio de autenticación* es una funcionalidad del sistema. El segundo caso expresa un requisito no funcional porque no especifica una funcionalidad del sistema sino propiedades de seguridad, en concreto la disponibilidad (*garantizar la accesibilidad...*) y la confidencialidad (*... únicamente a los usuarios autorizados*) de los datos.

Existen muchas técnicas para recoger los requisitos: por ejemplo, técnicas de definición de casos de uso, *brainstorming* en reuniones donde participan los desarrolladores y las partes interesadas, grupos de trabajo que entrevistan a los usuarios, demostraciones a los clientes para obtener una re-alimentación, etc. Los modelos de desarrollo de tipo iterativo-incremental (por ejemplo, el Proceso Unificado y los métodos ágiles) fomentan todas aquellas técnicas que involucran las partes interesadas, en particular los usuarios que utilizarán el sistema. Los requisitos se especifican en el lenguaje del cliente (no técnico).

A continuación se describe la técnica de definición de los casos de uso, utilizada para la captura y especificación sistemática de los requisitos funcionales de un sistema. Para facilitar la comprensión de la técnica, se utiliza el caso *CarShare* (véase el Apéndice A) que ofrece un servicio en línea para el uso compartido de coches.

3.2. Casos de uso

Los casos de uso son historias *escritas* que describen un **actor** que usa el sistema para conseguir un objetivo: por ejemplo, el conductor de *CarShare* que quiere encontrar compañeros de viaje. Los actores son *roles* que tienen un comportamiento, como: una persona, un ordenador u otro sistema que interacciona con el sistema que se está analizando (por ejemplo, el sistema que provee el servicio de autorización al pago en línea). Un **escenario** es una secuencia de acciones e interacciones entre el sistema y un actor. Un escenario describe

una historia de uso del sistema por parte de un actor. Por ejemplo, un escenario (de éxito) del sistema CarShare es el siguiente:

Registrar viaje. El Conductor quiere registrar un nuevo viaje. La Aplicación CarShare pide los datos del viaje (lugar de origen y destino, fecha y hora de salida, etapas intermedias, la marca del coche, el modelo, la matrícula y el número de plazas, el número de plazas ofertadas, la declaración de estar en posesión de un permiso de conducir y de un seguro vigente, comentarios opcionales). El Conductor introduce los datos. La Aplicación CarShare valida y guarda los datos. La Aplicación CarShare pide fijar un precio de la plaza. El Conductor introduce el precio. La Aplicación CarShare valida y guarda el precio. La Aplicación CarShare pide la forma de pago de las plazas por los Pasajeros. El Conductor elige la forma de pago. La Aplicación CarShare registra la opción elegida y publica el nuevo viaje.

Un **caso de uso** es una colección de escenarios relacionados (de éxito y de fracaso) que describe un *actor* que usa el sistema para alcanzar un *objetivo*. Por ejemplo:

Gestionar cancelación viaje.
Escenario principal de éxito: El Conductor decide cancelar un viaje. La Aplicación CarShare recupera los datos del viaje, chequea la fecha de salida (es posterior a la actual) y el número de plazas reservadas (no hay reservas). La Aplicación CarShare pide la confirmación de la cancelación. El Conductor confirma la cancelación. La Aplicación CarShare actualiza el estado del viaje en "cancelado".
Escenarios alternativos:
- Si la fecha de salida no es posterior a la actual, la Aplicación CarShare avisa al Conductor que no es posible la cancelación del viaje.
- Si hay plazas reservadas y la forma de pago elegida por el Conductor es por adelantado, la Aplicación CarShare envía una notificación a los Pasajeros que habían reservado una plaza y envía los datos del Conductor y de los Pasajeros al Sistema de autorización al crédito para proceder al reembolso del dinero pagado por adelantado a los Pasajeros.

El **modelo de casos de uso** incluye los casos de uso identificados y descritos; es la especificación de las funcionalidades del sistema y de su contexto de operación. Los casos de uso **no** son diagramas UML, son historias escritas sobre el uso del sistema por parte de actores. El modelo de casos de uso puede incluir un diagrama de casos de uso UML que sirve como *modelo de contexto* del sistema y como índice de los nombres de casos de uso. Los casos de uso definen *contratos* en relación al comportamiento del sistema: sirven para negociar el precio del producto software con el cliente.

3.2.1. Tipos de actores

Un actor es un elemento externo que interacciona con el sistema que se está analizando. Un actor no representa un individuo en concreto, sino representa un **rol**: un individuo puede tener uno o varios roles. Se pueden distinguir tres tipos de actores, como se indica a continuación.

Actor principal: es un individuo (o sistema) que consigue algún beneficio por la ejecución del caso de uso, recibiendo alguna cosa/servicio de valor medible u observable. El actor principal consigue sus objetivos utilizando los servicios proporcionados por el sistema que se está analizando. Es útil identificar los actores principales para encontrar los objetivos de los usuarios del sistema.

Actor de soporte: es un individuo (o sistema) que ofrece servicios al sistema que se está analizando. Por ejemplo, en el caso CarShare, el sistema que provee el servicio de autorización al pago en línea (*Sistema de autorización al crédito*) es un actor de soporte. Es útil identificar los actores de soporte para aclarar las interfaces externas y las reglas de intercambio de la información.

Actor fuera de escena: es un individuo (o sistema) que tiene interés en los servicios ofrecidos por el sistema que se está analizando. Por ejemplo, en el caso CarShare, el consejo de administración que ha pedido el desarrollo de la aplicación o las empresas interesadas en comprar espacios de publicidad en el sitio web de CarShare.

3.2.2. Tipos de formato para describir los casos de uso

Los casos de uso se pueden escribir usando diferentes formatos.

Formato breve: es un resumen conciso de un sólo párrafo que describe el escenario principal de éxito. La descripción del caso de uso *Registrar viaje* presentada anteriormente es un ejemplo de formato breve. Este formato se usa durante el análisis inicial de los requisitos.

Formato informal: incluye unos párrafos que describen varios escenarios (el principal y unos alternativos). La descripción del caso de uso *Gestionar cancelación viaje* presentada anteriormente es un ejemplo de formato informal.

Formato detallado: es una descripción más detallada de los escenarios. Para ello hay plantillas, como por ejemplo, la plantilla de Cockburn (véase Cuadro 3.1). En un modelo de desarrollo iterativo e incremental, después de que muchos casos de uso hayan sido identificados y descritos en formato breve, un porcentaje de los casos de uso de los más críticos se refina usando el formato detallado. Por ejemplo, el Proceso Unificado de desarrollo aconseja describir aproximadamente el 10 % de los casos de uso durante la fase inicial (fase de ideación). Hay varios criterios para establecer la criticidad de los casos de uso. Típicamente se consideran las funcionalidades del sistema consideradas de elevado valor de negocio por las partes interesadas, la complejidad técnica (del diseño, de implementación, ...), la cobertura (se prefiere cubrir las partes principales del sistema, es decir adoptar un desarrollo en amplitud más que en profundidad).

A continuación se analizan las secciones de la plantilla de Cockburn (Cuadro 3.1).

Nombre del caso de uso

El nombre del caso de uso tiene que ser explicativo del objetivo a alcanzar y empezar con un verbo. Por ejemplo: *Registrar viaje, Gestionar cancelación viaje.*

Alcance

El alcance de un caso de uso define los límites del sistema que se está analizando. Un caso de uso que describe el uso de un sistema software (como la aplicación *CarShare*) o un sistema que incluye hardware y software se llama caso de uso de **sistema**. Un caso de uso que describe los procesos de negocio de una empresa u organización se llama caso de uso de **negocio**. Los casos de uso tratados en este capítulo son casos de uso de sistema.

Sección	Comentario
Nombre del caso de uso	Empieza con un verbo
Alcance	El sistema a desarrollar
Nivel	Objetivo usuario/sub-función
Actor principal	Solicita al sistema un servicio
Partes interesadas	Quién está interesado en el caso de uso y qué necesita
Pre-condiciones	Qué debe ser cierto antes de comenzar este caso de uso (y merece la pena ser mencionado)
Garantías de éxito	Qué debe ser cierto si el caso de uso se ejecuta con éxito (y merece la pena ser mencionado)
Escenario principal	Escenario común de *éxito* (sin condiciones)
Extensiones	Escenarios alternativos (de éxito y fracaso)
Requisitos especiales	Requisitos no funcionales relacionados
Otra info	Ejemplo: problemas abiertos

Cuadro 3.1 Plantilla de Cockburn para la descripción detallada de los casos de uso.

Nivel

Hay dos tipos de niveles: el nivel objetivo *usuario* y el nivel *sub-función*. El primero describe los escenarios de interacción entre un actor principal y el sistema. Un caso de uso de nivel sub-función es un caso de uso de soporte para alcanzar el objetivo usuario. En un diagrama de casos de uso UML, los casos de uso de nivel sub-función están relacionados con los casos de nivel objetivo usuario (con relaciones «*include*» o «*extend*»). Por ejemplo, en Figura 3.1, el caso de uso *Calcular precio plaza* es de nivel sub-función.

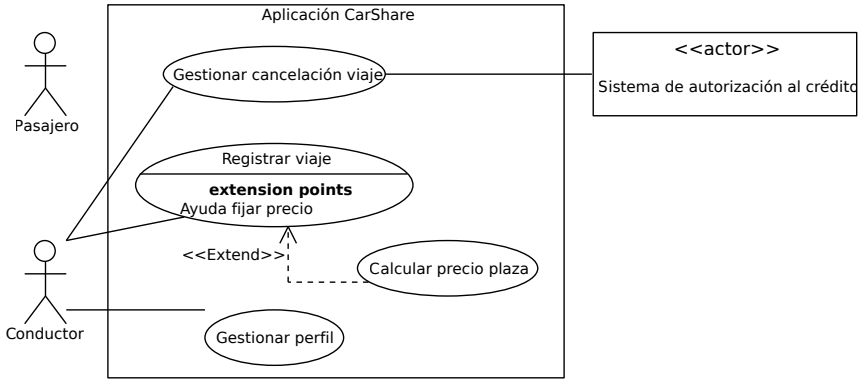


Figura 3.1 Diagrama de casos de uso de la aplicación *CarShare* (no completo).

Actor principal

Es el actor que solicita un servicio al sistema que se está analizando para alcanzar un objetivo. Por ejemplo, en Figura 3.1 el actor principal del caso de uso *Registrar viaje* es el *Conductor* (el caso de uso se une al actor principal con una *asociación*).

Partes interesadas

Es una lista de las partes interesadas (actores de soporte o fuera de escena). Para cada parte interesada se describe su objetivo. Por ejemplo (caso de uso *Gestionar cancelación viaje*):

- Sistema de autorización al crédito: quiere recibir los pedidos para efectuar las transacciones en el formato y según el protocolo correcto.
- Pasajero: quiere recibir una notificación de la cancelación. Si ha pagado por adelantado, quiere recibir el reembolso del precio de la plaza.

Pre-condiciones y garantías de éxito

Las pre-condiciones describen qué tiene que ser siempre cierto antes de empezar un escenario del caso de uso. Las garantías de éxito (o post-condiciones) describen qué tiene que ser siempre cierto cuando un escenario de éxito (el principal o uno alternativo) del caso de uso ha sido ejecutado.

Tanto las pre-condiciones como las garantías de éxito tienen que ser condiciones sobre el estado del sistema software, por ejemplo (caso de uso *Gestionar cancelación viaje*):

Pre-condiciones El Conductor está identificado y autenticado.

Garantías de éxito El estado del viaje ha sido actualizado (estado "cancelado"). La notificación de cancelación del viaje a los Pasajero ha sido enviada. Las peticiones de reembolso a los Pasajeros que han pagado por adelantado han sido enviadas al Sistema de autorización al crédito.

Escenario principal

El escenario principal se llama también "camino feliz" porque describe una posible interacción entre el actor principal y el sistema, interacción que satisface los intereses de las partes interesadas. El escenario principal no incluye condiciones o alternativas: éstas se consideran en la sección *Extensiones*. Un escenario está compuesto por una secuencia de pasos que pueden ser:

- una interacción entre actores
- una verificación de datos o validación
- un cambio de estado del sistema

El primer paso de un escenario puede describir un evento que provoca la ejecución del escenario. Los nombres de los actores se escriben con la inicial mayúscula para identificarlos fácilmente en el texto. Un ejemplo (caso de uso *Gestión cancelación viaje*):

1. El Conductor decide cancelar un viaje.
2. La Aplicación CarShare recupera los datos del viaje.
3. La Aplicación CarShare chequea la fecha de salida (es posterior a la actual)
4. La Aplicación CarShare pide la confirmación de la cancelación.
5. El Conductor confirma la cancelación.

6. La Aplicación CarShare chequea el número de plazas reservadas (no hay reservas).
7. La Aplicación CarShare actualiza el estado del viaje a "cancelado".

Extensiones

Las extensiones son muy importantes y normalmente incluyen la mayoría del texto de un caso de uso. Cada extensión describe una posible alternativa (de éxito o de fracaso) al escenario principal y se indica con una etiqueta que hace referencia a un paso del escenario principal. Primero se describe la condición y después la reacción. Por ejemplo, una alternativa al paso 6 del escenario principal (caso de uso *Gestionar cancelación viaje*) es cuando haya reservas de plazas. El escenario alternativo se indica con la etiqueta 6*a* y se ejemplifica a continuación:

6a. Hay reservas:

1. La Aplicación CarShare envía la notificación de cancelación viaje a los Pasajeros que han reservado la plaza.
2. La Aplicación CarShare chequea la forma de pago elegida por el Conductor:
   2a. La forma de pago es por adelantado:
      1. La Aplicación CarShare envía los datos (del Conductor y de los Pasajeros) al Sistema de autorización al crédito para proceder al reembolso del dinero a los Pasajeros.
      2. El Sistema de autorización al crédito confirma la recepción correcta de los datos.

Obsérvese en Figura 3.1, que el caso de uso *Gestionar cancelación viaje* está asociado no sólo al actor principal (*Conductor*) sino también al *Sistema de autorización al crédito*, siendo éste un actor de soporte porque tiene una interacción con la Aplicación CarShare (paso 6*a*.2*a*). Sin embargo, con respecto a este caso de uso, el *Pasajero* es un actor fuera de escena: simplemente recibe una notificación por la Aplicación CarShare (paso 6*a*.1). Por razones de legibilidad de los diagramas de casos de uso no se añade la asociación entre el caso de uso y el Pasajero.

Si hubieran otras alternativas al paso 6, se indicarían con etiquetas 6*b*, 6*c*, etc. Es importante escribir la condición como algo que pueda ser detectado por el sistema o el actor principal.

Una extensión puede reemplazar una secuencias de pasos del escenario principal. Por ejemplo si la fecha de salida no es posterior a la actual, no hay que realizar todos los pasos siguientes al paso 3 del escenario principal. En este caso la etiqueta del escenario alternativo tiene que indicar el primer paso y el último paso afectados:

3-7. Fecha de salida no es posterior a la actual:

1. La Aplicación CarShare avisa al Conductor que no es posible la cancelación.

Cuando acaba la ejecución de una extensión, por defecto, el escenario se une con el escenario principal. En el ejemplo, una vez acabada la alternativa $6a$, se vuelve al siguiente paso del escenario principal (el paso 7). Por otro lado, la alternativa 3-7 no se une con el escenario principal porque no hay pasos ulteriores en éste último. Si hubiera otros pasos $8, 9, \ldots$ en el escenario principal, una vez acabada la alternativa 3-7 se volvería al paso 8.

Si se quiere describir una extensión que puede ocurrir en cualquier paso del escenario principal, se usa la notación: $*a, *b, ...$:

*a. En cualquier momento, la Aplicación CarShare falla:

1. El Administrador reinicia la Aplicación CarShare, se identifica y pide la restablecimiento del estado previo.
2. La Aplicación reconstruye el estado previo (*roll-back*).....

Cuando una extensión es compleja (requiere la descripción de muchos pasos) es mejor considerarla en un caso de uso separado que se define a nivel de sub-función.

Por ejemplo, el caso de uso *Calcular precio plaza* es una extensión del caso de uso *Registrar viaje* que permite al Conductor obtener el precio de la plaza por la Aplicación CarShare en base a varios criterios. Considérese la siguiente descripción del escenario principal del caso de uso *Registrar viaje*:

1. El Conductor quiere registrar un nuevo viaje.
2. La Aplicación CarShare pide los datos del viaje (lugar de origen y destino, fecha y hora de salida, etapas intermedias, la marca del coche, el modelo, la matrícula y el número de plazas, el número de plazas ofertadas, la declaración de estar en posesión de un permiso de conducir y de un seguro vigente, comentarios opcionales).
3. El Conductor introduce los datos.
4. La Aplicación CarShare valida y guarda los datos.
5. La Aplicación CarShare pide fijar un precio de la plaza.
6. El Conductor introduce el precio.
7. La Aplicación CarShare valida y guarda el precio.
8. La Aplicación CarShare pide la forma de pago (en efectivo o por adelantado).
9. El Conductor elige la forma de pago.
10. La Aplicación CarShare registra la opción elegida y publica el nuevo viaje.

La extensión (escenario alternativo al paso 6) será del siguiente tipo:

6a. El Conductor quiere una ayuda para el cálculo del precio de la plaza:

1. La Aplicación CarShare ejecuta Calcular precio plaza.

Si se utilizan herramientas UML CASE, que incluyen editores de texto hipertextuales, el nombre del caso de uso subrayado suele ser un enlace a la descripción detallada del mismo. Obsérvese que en el diagrama de casos de uso UML (Figura 3.1), los casos de uso *Registrar*

viaje y *Calcular precio plaza* están relacionado con una relación `extend`. En el Apéndice C (sección C.1) se describen las principales relaciones entre casos de uso.

Requisitos especiales

Normalmente los requisitos no funcionales consideran el sistema como entidad y abarcan varios casos de uso. Sin embargo, si un requisito no funcional (o una restricción) está relacionado con un caso de uso especifico, entonces es aconsejable describirlo en el caso de uso.

Otra info

Esta última sección incluye otra información útil que no haya sido incluida en las secciones anteriores. Por ejemplo, restricciones de tipo tecnológico o relacionadas con el formato de los datos, problemas abiertos que necesitan más investigación.

3.2.3. ¿Cómo escribir los casos de uso?

El estilo usado para describir los casos de uso tiene que ser **esencial** y **conciso**. Hay que ignorar la interfaz de usuario (cómo el usuario introduce los datos en el sistema y cómo el sistema muestra los resultados al usuario) y centrarse en el objetivo del usuario.

A continuación se ejemplifican dos descripciones del escenario principal del caso de uso *Gestionar perfil*:

Ejemplo 1:

1. El Conductor se identifica.
2. La Aplicación CarShare convalida la identidad.
3. ...

Ejemplo 2:

1. El Conductor inserta su ID y password en la ventana.
2. La Aplicación CarShare convalida la identidad.
3. ...

En el ejemplo 1 el estilo usado es esencial: durante el diseño de la aplicación CarShare se decidirá qué solución es la más apropiada para identificar al conductor (interfaces gráficas de usuarios, etc...). El estilo usado en el ejemplo 2 es concreto, implícitamente se asume que la identificación se realizará a través de una interfaz gráfica de usuario. Usar un estilo conciso significa evitar detalles superfluos y no adelantar decisiones de diseño en las tareas de definición y análisis de requisitos.

Otra guía es usar un estilo **caja negra**, es decir centrarse en qué tiene que hacer el sistema y no cómo funciona internamente. Dos ejemplos para el caso de uso *Registrar viaje*:

Ejemplo 1 (estilo caja negra):

- La Aplicación CarShare registra el viaje.

Ejemplo 2 (estilo no caja negra):

- La Aplicación CarShare registra el viaje en una base de datos.
  ... o aún peor:
- La Aplicación CarShare ejecuta una instrucción SQL INSERT para el viaje.

La recomendación es centrarse en el resultado de interés que el caso de uso produce para el actor principal.

3.2.4. ¿Cómo encontrar los casos de uso?

Los casos de uso cumplen con los objetivos de los actores principales. Un método para encontrarlos es:

1. Identificar los actores del sistema a desarrollar.
2. Identificar los objetivos de cada actor.
3. Crear un modelo de contexto con UML, es decir un diagrama de casos de uso UML, donde:
   - definir el límite del sistema (¿es una aplicación software? ¿es un sistema que incluye hardware y software? ¿es una entera organización?). Los actores son siempre externos al sistema;
   - definir los casos de uso relacionados con los actores principales. La identificación de los actores principales facilita la definición del límite del sistema. El nombre del caso de uso (siempre empieza con un verbo) se elige en base al objetivo.
4. Describir cada caso de uso encontrado en formato breve.

En general, cada objetivo corresponde a un caso de uso de nivel objetivo usuario, pero hay excepciones. Por ejemplo, se suele definir un único caso de uso para objetivos separados del tipo *CRUD* (*Create, Retrieve, Update, Delete*) es decir para crear, leer, actualizar y eliminar datos.

En la aplicación CarShare, el caso de uso *Gestionar perfil* incluye añadir un nuevo perfil, modificar los datos del perfil y eliminar el perfil (véase Figura 3.1). Para encontrar los actores

Actor	Objetivo
Conductor	Registrar viaje Gestionar cancelación viaje Crear perfil Modificar perfil Eliminar perfil ...
Sistema de autorización al crédito	...
Pasajero	...

Cuadro 3.2 Lista actores-objetivos de la aplicación CarShare (no completa).

y sus objetivos se puede usar una lista actores-objetivos (véase Cuadro 3.2).

El *tiempo* es un actor principal cuando un sistema tiene que ejecutar tareas en respuesta a eventos periódicos o que ocurren en fechas establecidas.

3.2.5. Preguntas a hacer/hacerse para encontrar actores y objetivos

Además de los actores primarios y objetivos obvios, a continuación se presentan algunas preguntas que ha de hacerse para identificar actores y casos de uso que pueden haberse omitido en una primera aproximación:

- ¿Es el tiempo un actor? Es decir: ¿hace algo el sistema en respuesta a un evento de tiempo? ¿existe algún evento que se ejecute de forma automática o periódica?
- ¿A quien se notifica los errores y fallos?
- ¿Quién o qué proporciona entradas al sistema?
- ¿Quién o qué recibe las salidas del sistema?
- ¿Cuáles son las principales tareas realizadas por cada actor?
- ¿El actor actualizará alguna información en el sistema?
- ¿El actor leerá alguna información en el sistema?
- ¿Qué información necesita cada actor del sistema?
- ¿Qué información introduce cada actor en el sistema?
- ¿Tiene el actor que ser informado sobre los cambios inesperados?

3.2.6. ¿Cómo verificar si un caso de uso es útil?

Las siguientes reglas sirven como guías para verificar si un caso de uso tiene un nivel apropiado para describir requisitos de un sistema software a desarrollar: la utilidad de un caso de uso siempre depende del límite del sistema.

Test del jefe

Si un jefe pregunta a su equipo de analistas *¿Qué habéis hecho todo el día?* Y los analistas responden: *el login!*, es muy probable que el jefe no estará contento. Si no lo es, el caso de uso no pasa el test del jefe, es decir significa que no tiene como objetivo obtener un resultado medible (nivel de objetivo usuario).

Test EBP (Elementary Business Process)

> *Un EBP es una actividad básica llevada a cabo por una persona en un determinado lugar y tiempo, en respuesta a un evento de negocio, que añade un valor de negocio medible y deja los datos del sistema en un estado consistente (Larman, 2004).*

No hay que tomar al pie de la letra la definición: no quiere decir que un caso de uso no pasa este test si requiere dos personas para llevarlo a cabo. Sino que hay que tener sospecha de un caso de uso si es necesario tener sesiones múltiples para ejecutarlo. Por ejemplo *Ponerse de acuerdo para un viaje* no es un EBP, más bien es un caso de uso de negocio (es decir, de alcance más amplio): involucra varios actores -el conductor, el pasajero, el sistema de autorización al crédito, etc.- que interaccionan con la aplicación *CarShare* en momentos diferentes. Por otro lado, hay que tener sospecha también si el escenario principal del caso de uso está compuesto por un solo paso. Por ejemplo *Introducir precio de la plaza* no es un EBP porque no añade un valor de negocio medible (tampoco pasaría el test del jefe mencionado anteriormente).

Test de la dimensión

La longitud del texto asociado al caso de uso puede proveer indicaciones sobre su nivel de detalle: normalmente, la descripción del escenario principal de un caso de uso está compuesto por un conjunto de pasos. Por ejemplo, el caso de uso *Insertar el lugar de origen y de destino del viaje* no pasa el test de la dimensión porque se puede describir con un solo paso.

3.3. Cuestiones

1. ¿Qué se entiende por requisito funcional y por requisito no funcional?
2. Proveer ejemplos de requisitos no funcionales.
3. ¿Qué es un caso de uso y qué relación tiene con el diagrama de casos de uso UML?
4. ¿Qué representa un actor y qué tipos de actores hay?
5. ¿Cómo se identifican los casos de uso?
6. ¿Cómo se escribe un caso de uso? ¿Qué tipos de formato hay para describir casos de uso?
7. ¿Qué significa proceso básico de negocio (EBP)?
8. ¿Quién inicia la interacción en un caso de uso de sistema?

3.4. Casos prácticos

Ej. 1 — ¿Cuáles de los siguientes casos de uso son útiles para describir los requisitos de la aplicación *CarShare*?

- *Ponerse de acuerdo para un viaje.*
- *Gestionar cancelación viaje.*
- *Efectuar el login.*

Se apliquen las reglas descritas en la subsección 3.2.6 motivando las respuestas.

Ej. 2 — Considerando el enunciado del caso de estudio *CarShare* y la información obtenida con la entrevista al consejero delegado (Apéndice A), se proceda con los siguientes pasos:

- Identificar los casos de uso de la aplicación *CarShare*.
- Describir cada caso de uso en formato breve.
- Refinar el diagrama de contexto de Figura 3.1.

- Clasificar los casos de uso en base a la criticidad (alta, baja).
- Describir los casos de uso más críticos en formato detallado según la plantilla de Cockburn.

Ej. 3 — La empresa *HorseBetting S.A.* es una casa de apuestas deportivas a través de la cual es posible apostar a las carreras de caballos que se celebran en el territorio nacional. HorseBetting S.A., a través de las gacetas que publica, pone a disposición de aquel que se persone en su casa de apuestas toda la información relacionada con las carreras hípicas celebradas, así como el seguimiento de los eventos en directo, consulta de pronósticos y estadísticas, y obviamente, la posibilidad de realizar apuestas en taquilla.
La casa de apuestas quiere mejorar su servicio y dar la posibilidad a sus clientes de realizar apuestas por medio de Internet. Con este propósito, los que quieran utilizar el servicio tendrán que abrir una cuenta en línea, subscribir un contrato de apertura cuenta con *HorseBetting* y efectuar un primer depósito en la cuenta antes de jugar. Las modalidades para realizar abonos en la cuenta subscrita pueden ser diferentes, por ejemplo utilizando una tarjeta de crédito, una transferencia bancaria etc. Una vez abierta la cuenta, el cliente de *HorseBetting* puede acceder a los servicios de consulta de los eventos hípicos con los pronósticos actualizados en tiempo real (cada 3 minutos) y realizar apuestas de manera inmediata. En una carrera, las apuestas directas (un sólo caballo) son la forma más sencilla de apostar a los caballos. Hay también apuestas múltiples que permiten apostar, en una carrera, sobre más de un caballo según diferentes reglas. De todos modos, para apostar, el jugador tiene que indicar el número (dorsal) o nombre del caballo elegido (o de los caballos elegidos), el tipo de apuesta y la cantidad de dinero que desea apostar. Finalmente, el sistema tendrá que producir un recibo electrónico.
El pronóstico para un caballo en una carrera es diferente según el tipo de apuesta y los pronósticos son proporcionados a *HorseBetting*, en tiempo real, por el sistema de información de la Sociedad Estatal Loterías y Apuestas del Estado (SI_SELAE). Por otro lado, *HorseBetting* proporciona los datos relacionados con las apuestas de sus clientes a SI_SELAE para que éste pueda calcular las estadísticas y pronósticos a nivel nacional. Una vez finalizada una carrera, el importe a cobrar por cada apuesta acertada en las carreras será abonado automáticamente por *HorseBetting* en la cuenta del jugador.

- Identificar los casos de uso del sistema en línea de *HorseBetting S.A.*
- Proveer una descripción breve de los casos de uso.
- Definir un diagrama de contexto con actores y casos de uso.
- Identificar un caso de uso crítico, justificando su elección.
- Proveer una descripción detallada del mismo utilizando la plantilla de Cockburn.

Capítulo 4
Modelado de negocio

Resumen El modelado de negocio es una disciplina que tiene como propósito principal comprender y especificar el funcionamiento del negocio de la organización en la que un sistema de información se va a utilizar. Este capítulo introduce el modelado de negocio como una técnica que permite proporcionar información, sobre los conceptos del dominio de interés y los procesos de negocio, al equipo de trabajo del proyecto de desarrollo de un sistema software para delimitar su alcance. En particular, se hace hincapié en la creación de dos modelos: el modelo de dominio y el modelo de proceso.

4.1. Introducción

La disciplina de modelado de negocio (*business modeling*, en inglés) se relaciona con los procesos de negocio de la ingeniería de sistemas (véase subsección 2.1.1 - Figura 2.1) y se emplea principalmente para comprender y especificar el funcionamiento del negocio de una organización. Se puede también emplear para identificar aquellas áreas del negocio que pueden ser mejoradas a través de la automatización de los procesos, o por la reingeniería de procesos de negocio (*business process reingeneering*), y oportunidades de nuevos negocios.

En particular, la reingeniería de procesos de negocio es un acercamiento para dar una nueva orientación a los procesos claves de una organización. En el ámbito de la reingeniería de procesos de negocio, el rol de los analistas de sistema radica en el uso de TIC novedosas, como consecuencia de los cambios requeridos. Por ejemplo, una metodología puntera para el análisis y re-diseño de los procesos de negocios es la *minería de procesos* (van der Aalst, 2011) que se basa en el uso combinado de técnicas de minería de datos y de modelado.

El artefacto producido por el modelado de negocio es el "modelo de objeto de negocio"(BOM -*Business Object Model*). El BOM es un modelo de alto nivel de abstracción: representa cómo tienen que estar relacionadas las partes interesadas en el negocio y las principales entidades del negocio y cómo necesitan colaborar para realizar el negocio de la organización. Al mismo tiempo especifica un *glosario de negocio* de la organización, es decir un vocabulario común que debe ser compartido por las partes interesadas. Puede incluir varios modelos creados con diferentes tipos de diagramas UML (por ejemplo, diagramas de clase, de actividad, y de secuencias).

El Proceso Unificado (Jacobson et al, 1999), mencionado en la subsección 2.2.2, es una metodología de desarrollo de un sistema software que incluye el modelado de negocio entre sus disciplinas. El Proceso Unificado hace hincapié en la parte del BOM que proporciona

información al equipo de trabajo del proyecto de desarrollo de un sistema software para delimitar su alcance, es decir el modelo de dominio y el modelo de proceso. A continuación se detalla el modelo de dominio, en la Sección 4.3 se describe el modelo de proceso.

4.2. Modelo del dominio

El modelo de dominio es uno de los artefactos que se puede crear en la disciplina de modelado de negocio. Se especifica, utilizando la notación UML, con un diagrama de clases (véase Apéndice C.2) donde las clases representan objetos del mundo real en un dominio de interés.

A través del modelo de dominio se identifican los conceptos importantes de un dominio concreto de aplicación (Larman, 2004). La información que proporciona el modelo de dominio podría, de manera alternativa, expresarse en prosa, mediante sentencias. Sin embargo es más fácil entender los distintos conceptos y sus relaciones mediante un lenguaje visual (un porcentaje significativo del cerebro participa en el procesamiento visual - o como se dice: "una imagen vale más que mil palabras").

En realidad, el modelo de dominio es un *diccionario visual* de las abstracciones relevantes, vocabulario del dominio e información del dominio. En particular, un modelo de dominio incluye:

- clases conceptuales,
- asociaciones entre clases,
- atributos de clases conceptuales.

4.2.1. Clases conceptuales

Una clase conceptual es una idea, cosa u objeto. Una clase conceptual se puede considerar en términos de su símbolo, significado y extensión:

- *símbolo*: palabra -o imagen- que representa una clase conceptual
- *significado*: la definición de una clase conceptual
- *extensión*: el conjunto de ejemplos (instancias, objetos) a los que se aplica la clase conceptual.

Se puede considerar, por ejemplo, el concepto de dominio "carro de combate"(véase Figura 4.1 - parte superior). La abstracción -o símbolo- de este concepto de dominio es la clase conceptual *CarroDeCombate*, que refleja en el modelo de dominio únicamente los detalles de interés para el analista (es decir, el *modelo*).

El *CarroDeCombate* se puede definir como (significado):

> *Vehículo blindado de ataque con tracción de orugas o ruedas, diseñado principalmente para enfrentarse a fuerzas enemigas utilizando fuego directo.*

Finalmente, la extensión del *CarroDeCombate* la forman todos los ejemplos de carros de combate, representados por objetos de la clase *CarroDeCombate*.

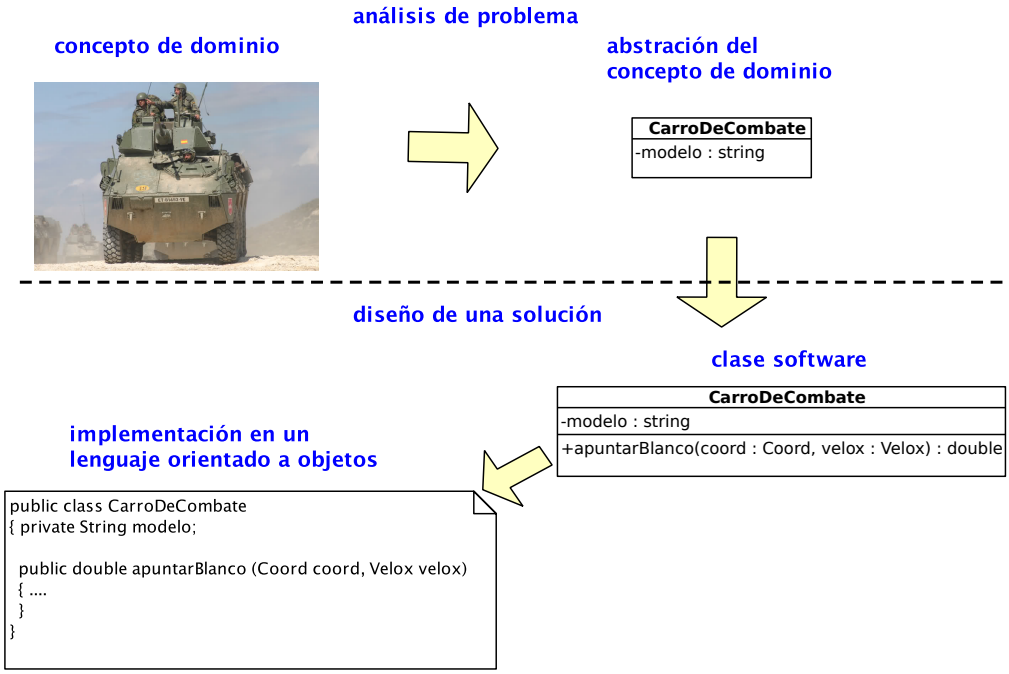


Figura 4.1 Análisis y diseño orientado a objetos.

4.2.2. Modelación orientada a objetos

El desarrollo de un sistema software puede ser compleja y la descomposición -divide y vencerás- es una estrategia común para tratar esta complejidad mediante la división del espacio del problema en unidades fáciles de comprender. En el *análisis estructurado*, la dimensión de la descomposición es por procedimientos o funciones. Sin embargo, en el *análisis orientado a objetos*, la dimensión de la descomposición es fundamentalmente por entidades del dominio. Una diferencia esencial entre el análisis orientado a objetos y el estructurado es la división por clases conceptuales (objetos) en lugar de la división por funciones. Por lo tanto, la principal tarea del análisis es identificar diferentes conceptos en el dominio del problema y documentar el resultado en un modelo de dominio.

Sin embargo el modelo de dominio (como parte del análisis del problema) es una representación de cosas del mundo real del dominio de interés, y no de componentes software (ya parte del diseño de una solución), como una clase Java o C++ u objetos software con responsabilidades. Por lo tanto las clases conceptuales pueden incluir atributos pero no operaciones o métodos.

El modelo de dominio es fuente de inspiración para el diseño e implementación de una solución al problema. Las clases conceptuales (que capturan la esencia del problema) pueden inspirar nombres y objetos en la solución lógica (diseño) e implementación donde se usan clases software.

Considérese, por ejemplo, el problema de desarrollar un simulador de combate terrestre. El carro de combate es obviamente unos de los posibles recursos y la clase conceptual *CarroDeCombate* en la Figura 4.1 (parte superior) representa todos los carros de combate a disposición de los atacantes y defensores. La clase está caracterizada por un atributo

-*modelo*- que permite especificar el modelo de cada carro de combate concreto. Durante la etapa de diseño se encuentra una posible solución y, en particular, se utiliza la clase conceptual *CarroDeCombate* para definir una clase software con el mismo nombre y atributo. El atributo *modelo* es una variable y como tal se le asigna un tipo de dato (*string*). Además, la clase software posee una operación (*apuntarBlanco*) que calcula el ángulo de tiro dadas las coordenadas del blanco (*coord*) y la velocidad de disparo (*velox*). Finalmente, en la etapa de implementación, es posible generar automáticamente parte del código con una herramienta CASE, en un lenguaje de programación orientado a objetos (por ejemplo, Java).

4.2.3. ¿Cómo crear un modelo de dominio?

En un modelo de desarrollo iterativo-incremental, como el Proceso Unificado, el modelo de dominio se acota contemplando los requisitos definidos en la iteración actual y anteriores. Es normal omitir clases conceptuales en una iteración y descubrirlas en las siguientes iteraciones. Cuando se encuentren, se pueden añadir al modelo de dominio. Por lo tanto el modelo de dominio se construye de manera incremental a lo largo de varias iteraciones.

Los pasos a realizar para construir el modelo de dominio son:

1. Encontrar las clases conceptuales.
2. Añadir las asociaciones entre clases.
3. Añadir los atributos a las clases.

4.2.4. Identificación de las clases conceptuales

Una técnica para el modelado del dominio es el uso de patrones de análisis, es decir modelos de dominio parciales ya existentes, creados por expertos.

Cuando no se dispone de un modelo ya existente, se puede utilizar una *lista de categorías de clases conceptuales*. El Cuadro 4.1 contiene categorías habituales que, normalmente merece la pena tener en cuenta, sobre todo en el caso de los sistemas de información empresariales. Los ejemplos se han extraído del dominio de las tiendas y de las ventas de viajes en línea (véase el caso *CarShare*), y están ordenados por importancia con respeto al dominio estudiado. Por ejemplo, las transacciones comerciales son consideradas críticas porque mueven dinero. Las siguientes entradas en la lista son categorías directamente relacionadas con las transacciones: partes de una transacción (línea), actores relacionados con las transacciones, lugares donde se efectúan las transacciones.

Otra técnica útil, debido a sus simplicidad, es el análisis lingüístico: identificar los nombres y las frases nominales en las descripciones textuales detalladas de los casos de uso (véase Capítulo 3).

Se debe tener cuidado con este método ya que no es posible realizar una correspondencia automática de nombres a clases porque las palabras en lenguaje natural son ambiguas. Frases nominales diferentes podrían representar la misma clase conceptual.

En cualquier caso, es otra fuente de inspiración. Por ejemplo, para crear un modelo de dominio del caso *CarShare* se puede empezar utilizando la descripción detallada de los casos de uso *Registrar viaje* y *Gestionar cancelación viaje* del Capítulo 3 -subsección 3.2.2.

Categoría de clases conceptuales	Ejemplos
Transacciones comerciales	Venta, Reserva, PagoPorAdelantado
Líneas de la transacción	LíneaDeVenta
Roles de personas/sistemas relacionados con las transacciones, actores de casos de uso	Cliente, Conductor, Pasajero
Lugares de transacciones o servicios	Concesionario
Eventos relevantes	Venta, Viaje, PagoPorAdelantado
Objetos tangibles o físicos	Coche, Lugar
Descripción de objetos	DescripciónCoche
Catálogos que contienen descripciones de objetos	CatálogoCoche
Contenedores de objetos	Concesionario
Objetos en un contenedor	Coche
Otros sistemas que colaboran	SistemaAutorizaciónAlCrédito
Registro de finanzas, trabajo, contratos, cuestiones legales	LibroDeContabilidad
Instrumentos/ servicios financieros	LíneaDeCredito, Caja
Manuales, documentos de referencia para llevar a cabo un trabajo	ListaDeCambioDePrecios, ListaPreciosGasolina, ContratoDeViaje
Conceptos abstractos	Tiempo

Cuadro 4.1 Lista de categorías de clases conceptuales.

Registrar viaje (escenario principal)

1. El **Conductor** quiere registrar un nuevo **viaje**.
2. La Aplicación CarShare pide los datos del **viaje** (**lugar de origen y destino**, **fecha y hora de salida**, **etapas intermedias**, la **marca del coche**, el **modelo**, la **matrícula** y el **número de plazas**, el **número de plazas ofertadas**, la **declaración de estar en posesión de un permiso de conducir** y de un **seguro vigente**, **comentarios opcionales**).
3. El **Conductor** introduce los datos.
4. La Aplicación CarShare valida y guarda los datos.
5. La Aplicación CarShare pide fijar un **precio de la plaza**.
6. El **Conductor** introduce el **precio**.
7. La Aplicación CarShare valida y guarda el **precio**.
8. La Aplicación CarShare pide la **forma de pago (en efectivo o por adelantado)**.
9. El Conductor elige la **forma de pago**.
10. La Aplicación CarShare registra la opción elegida y publica el **nuevo viaje**.

Gestionar cancelación viaje (escenario principal)

1. El **Conductor** decide cancelar un **viaje**.
2. La Aplicación CarShare recupera los datos del **viaje**.
3. La Aplicación CarShare chequea la **fecha de salida** (es posterior a la actual)
4. La Aplicación CarShare pide la confirmación de la cancelación.
5. El Conductor confirma la cancelación.
6. La Aplicación CarShare chequea el **número de plazas reservadas** (no hay **reservas**).
7. La Aplicación CarShare actualiza el **estado del viaje** a “cancelado”.

Escenarios alternativos

6a. Hay **reservas**:

1. La Aplicación CarShare envía la notificación de cancelación viaje a los **Pasajeros** que han reservado la **plaza**.
2. La Aplicación CarShare chequea la **forma de pago** elegida por el **Conductor**:
   2a. La **forma de pago** es **por adelantado**:
      1. La Aplicación CarShare envía los datos (del **Conductor** y de los **Pasajeros**) al **Sistema de autorización al crédito** para proceder al **reembolso del dinero** a los **Pasajeros**.
      2. El **Sistema de autorización al crédito** confirma la recepción correcta de los datos.

3-7. **Fecha de salida** no es posterior a la actual:

1. La Aplicación CarShare avisa al **Conductor** que no es posible la cancelación.

Obsérvese con el ejemplo que algunas de las frases nominales sugieren clases conceptuales y algunas otras sugieren atributos de las clases conceptuales. Además hay frases nominales diferentes que expresan los mismos conceptos. Para un análisis exhaustivo, se recomienda combinar esta técnica con el uso de la lista de categorías de clases conceptuales (Cuadro 4.1).

4.2.4.1. Clases conceptuales candidatas para el caso *CarShare*

A partir del análisis lingüístico y de la lista de categorías de clases conceptuales se genera una lista de clases conceptuales del dominio.

No existe una lista "correcta" o completa. La lista generada es una colección de conceptos del dominio que el analista considera relevante en una iteración concreta del proceso de desarrollo del sistema software. En cualquier caso, siguiendo la estrategia de identificación, diferentes analistas producirán listas similares. Modelos diferentes pueden ser igual de válidos, siendo resultado de diferentes motivaciones, casos de uso y escenarios analizados, iteraciones en el desarrollo del sistema software.

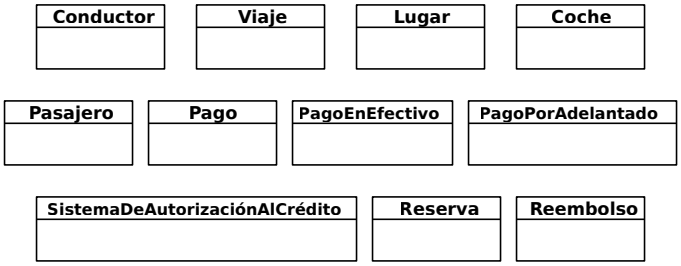


Figura 4.2 Ejemplo de clases conceptuales - caso *CarShare*.

En la Figura 4.2 se puede observar un ejemplo de clases conceptuales para el caso *CarShare*, restringido a los casos de uso *Registrar viaje* (escenario principal) y *Gestionar cancelación*

viaje (escenarios principal y alternativos). No todas las frases nominales identificadas durante el análisis lingüístico tienen correspondencia con una clase conceptual: como se verá más adelante, algunas tendrán correspondencia con atributos de clases conceptuales o con asociaciones entre clases conceptuales.

Por ejemplo la "plaza" no aparece como clase conceptual. ¿Por qué?

Si en la reserva se requiere indicar una plaza concreta en el coche para un viaje, sí que el concepto de "plaza" puede ser relevante y merece la pena introducir una clase conceptual correspondiente. Sin embargo, si en la reserva es suficiente indicar el número de plazas que se quiere reservar en un viaje no hace falta crear una clase conceptual para la "plaza" y el número de plazas se puede indicar como atributo de la clase *Reserva*. Después de haber realizado una entrevista con el consejero delegado de la empresa CarShare, se ha elegido la segunda opción para la aplicación *CarShare*.

4.2.4.2. ¿Atributo o clase?

Uno de los errores más comunes al realizar el modelo de dominio es confundir una clase con un atributo. La regla general es:

> *Si en el mundo real no se piensa en X como un número o texto, entonces con mucha probabilidad X es una clase conceptual, no un atributo.*

Por ejemplo un viaje se realiza con un coche. Se podría considerar el coche como un atributo de la clase conceptual *Viaje*. Sin embargo el coche es algo más que una cadena de caracteres. En el mundo real es un medio de transporte, algo que ocupa un espacio. Por lo tanto *Coche* debería ser representado por una clase conceptual relacionada con el *Viaje*: en la Figura 4.3, es mejor la representación de la derecha.

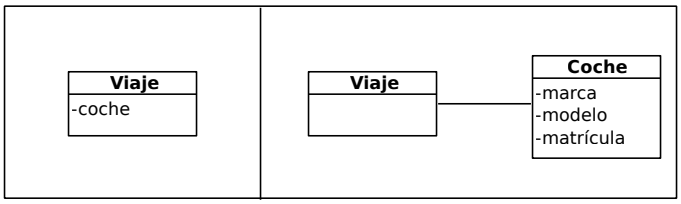


Figura 4.3 ¿Atributo o clase?

4.2.4.3. Clases conceptuales de descripción

Una clase de descripción contiene información que describe otros objetos. Por ejemplo, en el dominio de concesionarios de coches se puede introducir una clase *DescripciónCoche* (caracterizada por una marca, un modelo, etc.) relacionada con una segunda clase *Coche*, de modo que una instancia de la clase *DescripciónCoche* describe a varias instancias de una segunda clase *Coche*.

Es interesante añadirlas cuando se necesita la descripción de un artículo o servicio, independiente de la existencia de algún ejemplo de esos artículos o servicios, o la eliminación de instancias de un artículo (o servicio) provoca una pérdida de información que se necesita mantener. Se recomienda utilizarlas también si su introducción reduce información redundante o duplicada.

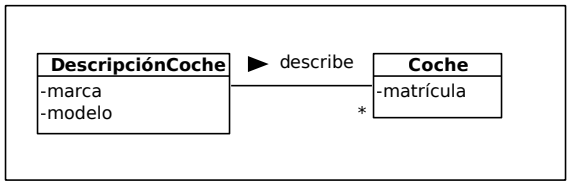


Figura 4.4 Ejemplo de clase conceptual de descripción.

La Figura 4.4 muestra el uso de la clase conceptual de descripción en el dominio de concesionarios de coches, donde una instancia de la clase *DescripciónCoche* describe a muchas instancias de la clase *Coche*. La marca y el modelo son atributos de la clase descripción porque son características que pueden ser comunes a muchos coches; por otro lado, la matrícula es un atributo de la clase *Coche* porque cada coche tiene su número de matrícula.

4.2.5. Identificación de las asociaciones

Una asociación entre dos clases indica una relación (significativa e interesante) entre instancias de las clases consideradas.

Resulta útil identificar aquellas asociaciones entre clases conceptuales que son necesarias para satisfacer los requisitos e información de los escenarios actuales que se están considerando y que ayudan a entender el modelo de dominio.

Se recomienda evitar demasiadas asociaciones, ya que relacionar las clases conceptuales todas con todas resta valor, entendimiento al modelo de dominio.

Las asociaciones que merece la pena registrar son aquellas que implican conocimiento de una relación que **se necesita recordar** (o conservar) durante algún tiempo. Por ejemplo, ¿necesitamos recordar qué instancias de una *Reserva* están asociadas con una instancia de *Viaje*? Por supuesto, de otra forma no sería posible averiguar los detalles del viaje reservado, enviar una notificación de la nueva reserva al conductor, calcular el precio de la reserva.

Por otra parte el *Conductor* puede en algún momento cancelar un *Viaje* previamente registrado y la aplicación *CarShare* pide la confirmación de los datos introducidos. Sin embargo la confirmación por parte del *Conductor* no se considera algo que "se-necesita-recordar" (es decir, no se necesita almacenar la información relacionada al evento "confirma cancelación del viaje").

El Cuadro 4.2 recoge una lista de asociaciones comunes a tener en cuenta para la inclusión en un modelo de dominio: siempre los ejemplos son del dominio de las tiendas, de los concesionarios de coches y de las ventas de viajes en línea.

En el modelado de dominio una asociación es una manifestación de que una relación es significativa en un sentido puramente conceptual. Se deben evitar las asociaciones redundantes o derivadas.

Categoría de asociaciones comunes	Ejemplos
A es una transacción relacionada con otra B	Reserva–Pago
A es una línea de transacción o informe de B	LíneaDeVenta–Venta
A es una parte física o lógica de B	Reserva–Viaje
A es una descripción de B	DescripciónCoche–Coche
A es un producto/ servicio para una transacción	Viaje–Reserva
A es un rol relacionado con una transacción	Pasajero–Reserva, Conductor–Viaje
A está contenido físicamente/ lógicamente en B	Reserva–Viaje
A es miembro de B	Cajero–Tienda
A utiliza o gestiona B	Conductor–Viaje, Pasajero–Reserva
A se conoce/registra/recoge/captura en B	Coche–Viaje
A es una subunidad organizativa de B	Departamento–Tienda

Cuadro 4.2 Listas de asociaciones comunes.

4.2.5.1. Elementos de una asociación

Una asociación debe tener un nombre. El nombre de una asociación es una frase verbal que aclara cómo una clase está relacionada con otra, por lo que se recomienda utilizar un nombre significativo, de forma que la secuencia *NombreClase - FraseVerbal - NombreClase* sea legible y tenga sentido en el contexto del modelo (véase el ejemplo de Figura 4.5).

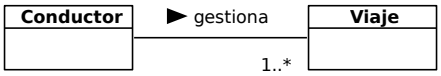


Figura 4.5 Ejemplo asociación - caso *CarShare*.

Para que la secuencia resulte más fácil de leer se puede añadir de forma opcional una flecha para indicar el sentido de lectura.

Los números que se observan en la Figura 4.5 indican la *multiplicidad* de la asociación. La multiplicidad representa cuantas instancias de una clase B se pueden asociar con una instancia de la clase A. La multiplicidad se indica en ambos extremos de una asociación. Allí donde no aparezca especificada se entiende que tiene el valor por defecto, uno. En la Figura 4.5 las multiplicidades de la asociación especifican que *un* viaje es gestionado por *un* conductor (y sólo uno), mientras que un conductor gestiona uno o más viajes. Por lo tanto, la multiplicidad pone además de manifiesto una restricción del dominio que se reflejará más tarde en el software.

Algunos otros ejemplos de multiplicidad se muestran en la Figura 4.6, donde una instancia de B se asocia exactamente con una instancia de A mientras que una instancia de A se asocia con: cero o más instancias de B (en el primer ejemplo), uno o más instancias de B (en el segundo ejemplo), de uno a 40 instancias de B (en el tercer ejemplo), exactamente 5 instancias de B (en el cuarto ejemplo) y exactamente 3, 5 u 8 instancias de B (en el último ejemplo).

Dos clases pueden tener múltiples asociaciones, para representar relaciones *diferentes*. Por ejemplo, las asociaciones entre las clases *Viaje* y *Lugar* en la Figura 4.7 *sale-de* y *llega-a* representan diferentes relaciones, por lo tanto se deben mostrar por separado.

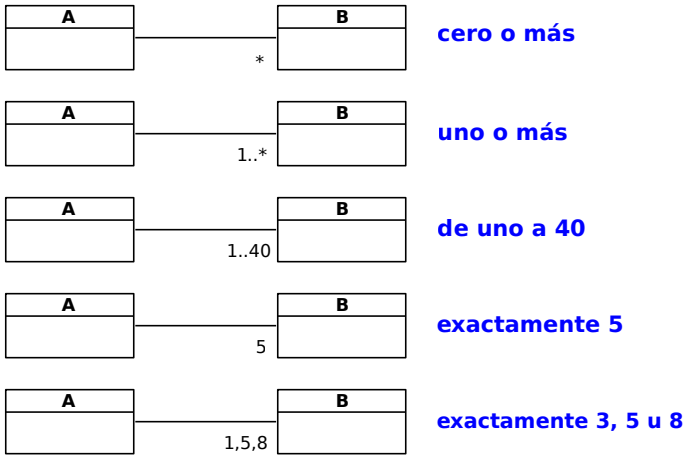


Figura 4.6 Valores de multiplicidad.

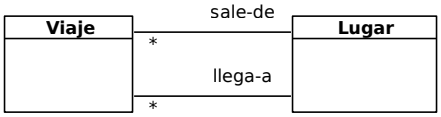


Figura 4.7 Múltiples asociaciones - caso *CarShare*.

4.2.5.2. Generalización

En el paradigma de la orientación a objetos, la herencia es el mecanismo por el que se comparten atributos y operaciones entre clases usando la relación de *generalización*. El uso de la *generalización* facilita la re-utilización de las clases -y, por consiguiente, del código que implementa sus funcionalidades- porque permite que los objetos de una clase hereden las características (atributos) y las operaciones dentro de una estructura jerárquica de clases.

En el contexto del modelo de dominio, y también del diseño conceptual de bases de datos (véase Capítulo 6), la generalización es una técnica para construir clasificaciones taxonómicas entre conceptos y representarlas como jerarquías de clases conceptuales. Por ejemplo, los conceptos de *Pago*, *PagoPorAdelantado* y *PagoEnEfectivo* son similares: es posible organizarlos en una jerarquía de clases conceptuales donde el *Pago* representa el concepto más general (super-clase) y *PagoPorAdelantado* y *PagoEnEfectivo* representan conceptos especializados (sub-clases) que heredan características de la super-clase.

La Figura 4.8 muestra el modelo de dominio parcial (clases, asociaciones y generalizaciones) del caso *CarShare*, limitado a los requisitos capturados con el análisis lingüístico de la descripción detallada de los casos de uso *Registrar viaje* (escenario principal) y *Gestionar cancelación viaje* (escenarios principal y extensiones). En particular, las asociaciones se han identificado utilizando el criterio "se-necesita-recordar" y la lista de asociaciones comunes:

- Transacciones relacionadas a otras transacciones: Reserva - conlleva - Pago, Reembolso - se refiere a - PagoPorAdelantado
- A es una parte lógica de B: Reserva - se refiere a - Viaje
- A es un rol relacionado con una transacción: Pasajero - gestiona - Reserva, Conductor - gestiona - Viaje, SistemaDeAutorizaciónAlCrédito - procede al - Reembolso

- A gestiona (registra, cancela) B: Conductor - gestiona - Viaje, Pasajero - gestiona - Reserva
- A utiliza B: Viaje - se efectúa en - Coche, Conductor - tiene - Coche
- A se captura en B: Viaje (B) - sale de - Lugar (A), Viaje (B) - llega a - Lugar (A)

El modelo de dominio de la Figura 4.8 incluye también dos generalizaciones que especifican las dos formas de pago contempladas en la aplicación *CarShare*, es decir el pago por adelantado y el pago en efectivo. La notación UML para la generalización es una flecha que conecta la super-clase con sus sub-clases, apuntando a la super-clase.

La super-clase *Pago* es un concepto abstracto, es decir no tiene instancias propias: un pago se puede realizar sólo por adelantado (sub-clase *PagoPorAdelantado*) o en efectivo (sub-clase *PagoEnEfectivo*).

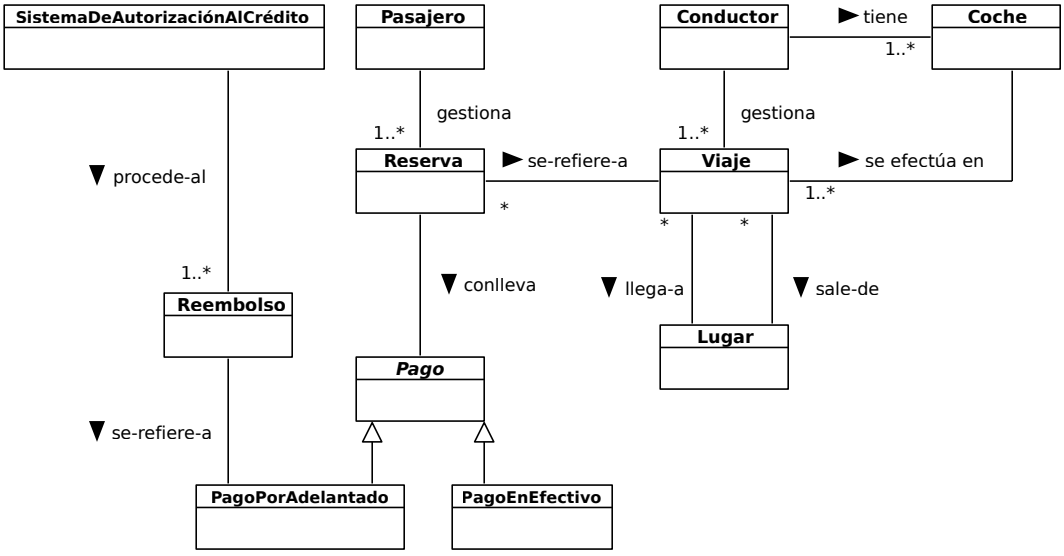


Figura 4.8 Modelo de dominio del caso *CarShare* - clases, asociaciones y generalizaciones

4.2.6. Identificación de atributos

Los atributos representan características, propiedades de una clase conceptual. Es importante identificar aquellos atributos necesarios para satisfacer requisitos de información para los escenarios actuales que estamos analizando. Se deben incluir en un modelo del dominio aquellos atributos para los que los requisitos sugieren o implican una necesidad de registrar la información.

Por ejemplo, en el caso *CarShare* un viaje tiene una fecha y hora de salida. En consecuencia, la clase conceptual *Viaje* necesita los atributos *fechaSalida* y *horaSalida*.

Como se comentó en el apartado 4.2.4.2, hay algunos conceptos que no deberían representarse como atributos, sino como asociaciones. El tipo de un atributo **no** tiene que ser un concepto de dominio complejo, como *Coche*.

Generalmente, los atributos en un modelo de dominio tienen asociado:

- Tipos de datos *primitivos*
  - booleanos (p. ej., estar en posesión de un permiso de conducir), números (p. ej., número de plazas ofertadas), caracteres, cadenas (p. ej. matrícula del coche), etc.
- Tipos enumerados
  - p. ej., tipo de servicio en la reserva de un vuelo (business, economony), estado del viaje (registrado, cancelado, efectuado).
- Otros *tipos de datos*
  - p. ej., dirección, precio.

Se recuerda que en los lenguajes de programación un tipo enumerado es un tipo de datos definido por el programador que solo puede tener como valores (*enumeration literals*, en inglés) los definidos en una lista. Los tipos enumerados se representan en UML como clases estereotipadas con «*enumeration*» y la lista de valores del tipo enumerado se añaden en la clase (véase Figura 4.9-a).

En UML se pueden definir nuevos tipos de datos, como tipo de datos compuestos por tipos de datos primitivos o enumerados. Un tipo de dato compuesto representa un conjunto de valores para los cuales no es significativa una identidad única (en el contexto del sistema analizado).

Los tipos de datos se representan en UML como clases estereotipadas con «*datatype*» que contienen atributos como las clases. Sin embargo a diferencia de una clase, cuyas instancias pueden tener valores iguales para todos los atributos, las instancias de un «*datatype*» tienen por lo menos un atributo con valor diferente.

En la Figura 4.9-b, la *Dirección* es un tipo de dato, por lo tanto no pueden existir dos instancias diferentes con mismos valores para *ciudad*, *calle*, *número* y *códigoPostal* (se trataría de la misma dirección). Al contrario, es significativo distinguir entre dos instancias de *Conductor* cuyos nombres son idénticos, ya que pueden representar individuos diferentes.

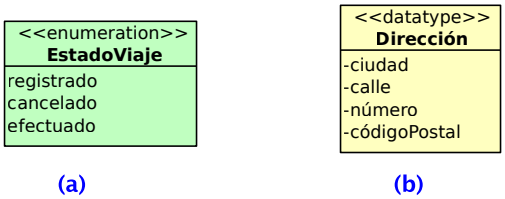


Figura 4.9 Ejemplo de: a) tipo enumerado y b) tipo de dato.

Se recomienda introducir un nuevo «*datatype*» cuando:

- El dato está compuestos por secciones separadas
  - p. ej., una dirección (véase la Figura 4.9-b).
- Hay operaciones asociadas a los datos, como la validación
  - p. ej., el DNI del conductor o del pasajero.
- El dato tiene otros atributos

  - p. ej., en el dominio de las tiendas, el precio de una promoción de un artículo puede tener una fecha de comienzo y fin.
- Es una cantidad con unidad
  - p. ej., el precio del viaje está caracterizado por una unidad de moneda. Las cantidades normalmente se especifican con un número y una unidad de medida.

No se debe utilizar atributos para relacionar clases conceptuales en el modelo del dominio. La violación más típica de este principio es añadir un tipo de atributo de clave externa, como se hace normalmente en el diseño de bases de datos (véase Capítulos 6 y 7), para asociar dos clases.

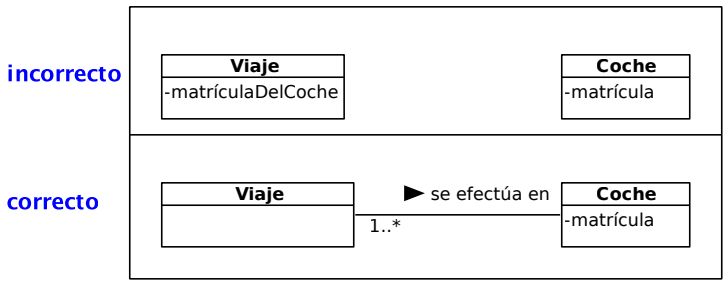


Figura 4.10 Ejemplo: no relacionar clases conceptuales utilizando atributos.

Por ejemplo, en la Figura 4.10, no es deseable definir el atributo *matrículaDelCoche* en la clase *Viaje* con el propósito de relacionar implícitamente un viaje con un coche. La mejor manera de expresar que un viaje se efectúa en un coche es con una asociación entre las dos clases conceptuales, no con un atributo.

La Figura 4.11 muestra un ejemplo de modelo de dominio (versión completa) para el caso *CarShare*. Los atributos elegidos reflejan los requisitos capturados en una iteración del proceso de desarrollo según el Proceso Unificado y restringidos a los casos de uso *Registrar viaje* (escenario principal) y *Gestionar cancelación viaje* (escenarios principal y alternativos). En el modelo de dominio, no es necesario definir el tipo para los atributos de tipo primitivo. Sin embargo, si un atributo está caracterizado por un tipo definido por el analista es buena práctica asignar explícitamente el tipo al atributo. Por ejemplo, el *estado* del viaje es de tipo enumerado *EstadoViaje*, el *precio* del viaje y el *precio* pagado con la reserva de las plazas son atributos de tipo *Precio* y la *dirección* del lugar es un atributo de tipo *Dirección*.

4.3. Modelo de proceso

Un *proceso* es un conjunto de actividades coordinadas (secuenciales y/o concurrentes) que se llevan a cabo para alcanzar un objetivo común. Las actividades pueden ser manuales o automáticas.

En particular, un *proceso de negocio* es un tipo de proceso característico de una organización o empresa que se lleva a cabo para alcanzar un objetivo de negocio.

En el caso *CarShare*, un ejemplo de proceso de negocio es el conjunto de actividades relacionadas con varios casos de uso para el acuerdo de un viaje entre un pasajero y un con-

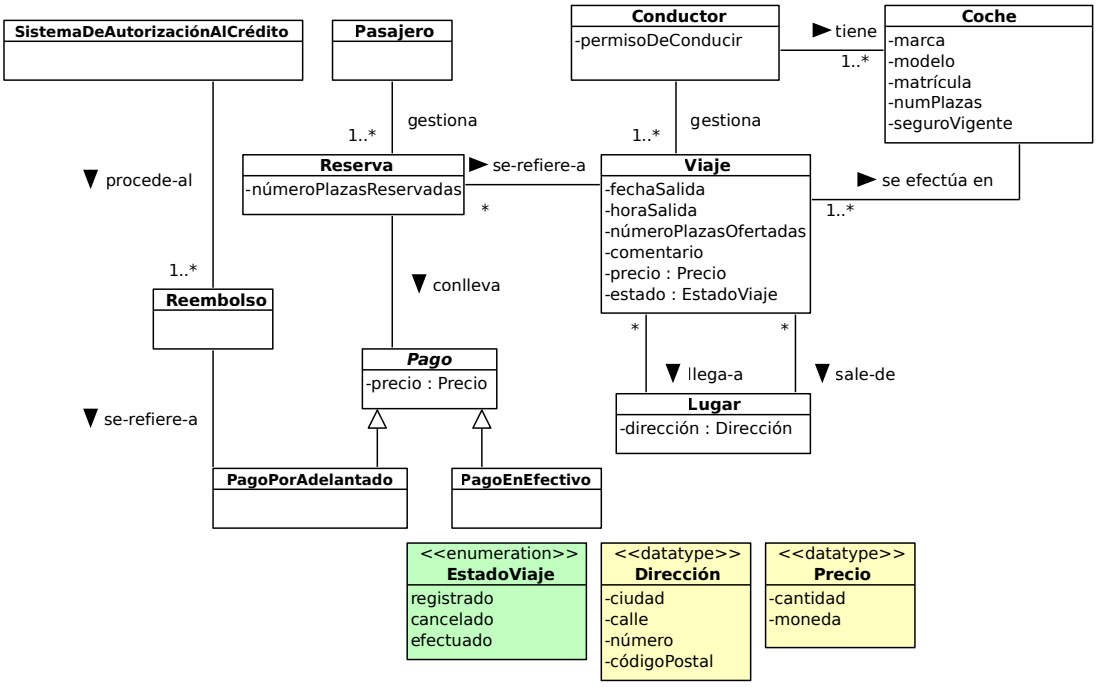


Figura 4.11 Ejemplo de modelo de dominio (versión completa) del caso *CarShare* - clases, asociaciones, generalizaciones, atributos y tipos.

ductor. Hay varios actores involucrados en este proceso: el pasajero, la aplicación *CarShare*, el conductor, el sistema externo de autorización al crédito.

En el dominio de las tiendas, un ejemplo de proceso de negocio es el conjunto de actividades que se llevan a cabo para premiar la fidelidad de los clientes con el objetivo (de negocio) de incrementar las ventas. La premiación puede implicar el seguimiento de las compras y la asignación de bonos de compra (u ofertas) periódica a los clientes. También en este proceso existen muchas partes involucradas como, por ejemplo, el cliente, el cajero, el sistema software utilizado para el registro de las ventas, el departamento de marketing, etc.

El modelo de negocio, a parte del modelo de dominio que es una vista estática del sistema que se está analizando, puede incluir también modelos de procesos que representan los procesos de negocio de una organización. En el Proceso Unificado, se usan los diagramas UML de actividades para crear modelos de procesos (véase Apéndice C.3).

Un diagrama UML de actividades permite representar:

- Las acciones secuenciales y concurrentes de un proceso (es decir, el flujo de control).
- El flujo de datos producidos/requeridos por las acciones.
- Los participantes que están involucrados en el proceso. Normalmente los participantes son los actores principales, de soporte y fuera escena. Obsérvese que el sistema o aplicación software que se está analizando es también un actor en el modelo de procesos.

4.3.1. ¿Cuándo crear un modelo de proceso?

Por lo general, el texto (en formato detallado) de un caso de uso es suficiente para describir las interacciones entre el actor principal y el sistema que se está analizando.

Es útil crear un modelo de proceso con un diagrama UML de actividades cuando el proceso es complejo, es decir hay muchas partes involucradas que ejecutan actividades relacionadas con el proceso. En particular, el modelo de proceso permite representar, de manera explícita, las relaciones "causa-efecto" entre las acciones que pertenecen a diferentes casos de uso.

La Figura 4.12 muestra, por ejemplo, el modelo de proceso para el acuerdo de un viaje entre un pasajero y un conductor. El modelo contempla interacciones -entre actores y la aplicación *CarShare*- que pertenecen a casos de uso diferentes y que se realizan en momentos diferentes (es decir, en diferentes sesiones de trabajo).

El proceso empieza con el *Pasajero* que realiza los pasos correspondientes a los casos de uso *Buscar viaje* (acciones *Introduce los criterios de búsqueda*, *Busca viaje según los criterios de búsqueda* y *Muestra lista viaje disponibles* o, en alternativa, *Avisa al viajero*) y *Reservar plaza* (acciones *Selecciona viaje* y *Notifica nueva reserva*).

A continuación el *Conductor* realiza los pasos correspondientes al caso de uso *Confirmar reserva* (acciones *Evalúa la nueva reserva* y *Actualiza estado del viaje* o, como alternativa, *Notifica rechazo del Conductor*).

Finalmente el *Pasajero* realiza los pasos del caso de uso *Pagar reserva plaza*, que incluye también una interacción entre la aplicación *CarShare* y el actor de soporte *Sistema de autorización al crédito* (las acciones restantes).

4.4. Cuestiones

1. ¿Qué es el modelado de negocio?
2. ¿Cuáles artefactos del *modelo de objeto negocio* (BOM) se producen para especificar el alcance de un proyecto software?
3. ¿Cuál diagrama UML se usa para representar un modelo de dominio?
4. ¿Qué diferencia hay entre un modelo de dominio y un modelo de clases software?
5. Explique la multiplicidad de las asociaciones mostradas en la Figura 4.7.
6. ¿Qué es un tipo enumerado («*enumeration*»)?
7. ¿Qué es un tipo de dato («*datatype*»)?
8. ¿Qué diferencia hay entre un tipo enumerado y un tipo de dato?
9. ¿Qué es un modelo de procesos?
10. ¿Cuál diagrama UML se usa para representar un modelo de procesos?
11. ¿Es necesario introducir una clase conceptual de descripción de los coches (p. ej. *DescripciónCoche* de Figura 4.4) en un modelo de dominio de la aplicación *CarShare*? Justifíquese la respuesta.

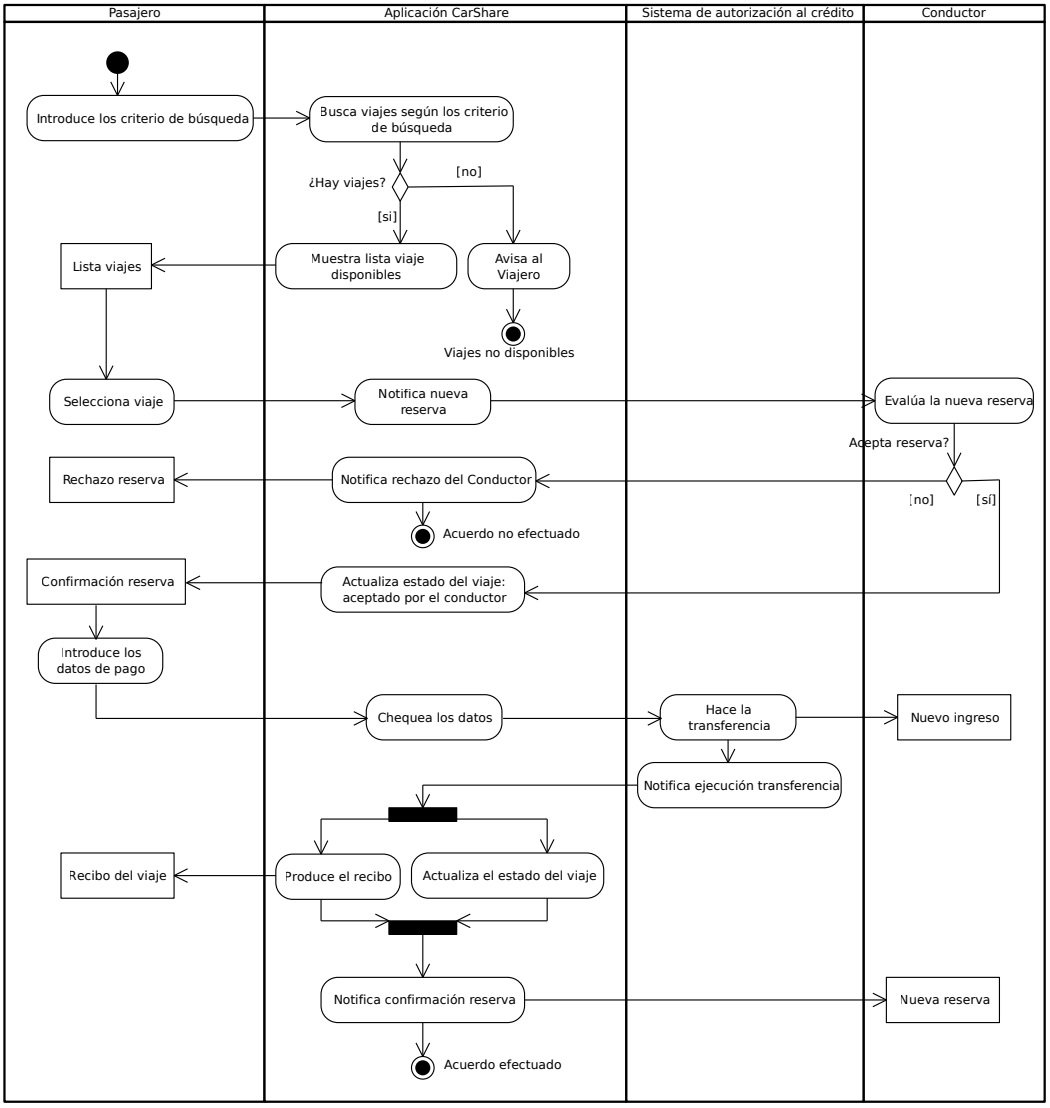


Figura 4.12 Modelo de proceso para el acuerdo de un viaje entre un pasajero y el conductor.

4.5. Casos prácticos

Ej. 1 — Modifique el modelo de dominio de Figura 4.11 para que un viaje pueda incluir etapas intermedias. Posteriormente, se extienda el modelo de dominio conforme a la descripción detallada de los casos de uso más críticos identificados en el caso práctico Ej. 2 del Capítulo 3.

Ej. 2 — Considérese el caso práctico Ej. 3 del Capítulo 3. Se defina un modelo de dominio utilizando la descripción detallada del caso de uso crítico. Se defina también un modelo de proceso que represente las actividades relacionadas con la apuesta en una carrera de caballos por parte de un jugador y el posible cobro de la ganancia (en caso de apuesta acertada).

Parte III
Desarrollo de bases de datos: análisis y diseño

Capítulo 5
Introducción a las bases de datos

Resumen Este capítulo introduce las bases de datos, con sus principales características, y los sistemas de gestión de bases de datos, sus funciones y las ventajas que aportan. Se presentan además los modelos de datos y las diferentes etapas que incluye el diseño de una base de datos.

5.1. Características de las bases de datos

Los datos son un componente especial en cualquier sistema de información tal como se ha visto en el Capítulo 1. Las organizaciones, mediante sus sistemas de información, memorizan y mantienen gran cantidad de datos relacionados a su ámbito de actuación: pueden ser datos relacionados con cuentas bancarias, estudiantes inscritos en un curso, guías telefónicas etc.

En los sistemas de información basados en ordenadores todos esos datos se almacenan de forma estructurada, agrupados en ficheros y en bases de datos (BD).

> Las bases de datos son estructuras de **datos** gestionadas por un conjunto de **programas** que permiten almacenar grandes cantidades de datos.

Si bien las bases de datos son técnicamente colecciones de ficheros, garantizan ciertas "buenas propiedades" que se repasan a lo largo de este capítulo, que las hace en muchas ocasiones superiores al uso de ficheros.

Algunas características generales de las bases de datos son:

- **Datos persistentes.** Tal como se venía adelantando, las bases de datos son técnicamente colecciones de ficheros que se guardan en la memoria secundaria (p. ej. en el disco duro). Por lo tanto puede haber diferentes aplicaciones haciendo uso de la misma base de datos. Los datos son disponibles siempre, sin estar encapsulados dentro de una única aplicación. A modo de ejemplo, en el enfoque de las bases de datos, los datos de contactos de una agenda electrónica pueden ser utilizados tanto por el gestor del correo electrónico como por aplicaciones de aviso de cumpleaños.
- **Datos compartidos.** En el enfoque de programación con ficheros, cada aplicación gestiona sus propios ficheros. En el ejemplo anterior, cada una de las dos aplicaciones mencionadas (gestor del correo electrónico, aviso de cumpleaños) podría mantener de forma individual datos parciales de contactos, según las necesidades de la aplicación. A pesar

de que todas estas aplicaciones gestionan datos sobre contactos (datos comunes), cada una requiere datos que no puede obtener del fichero gestionado por otra aplicación. Se genera redundancia en definir y almacenar los datos, lo que implica aumento del espacio de almacenamiento y esfuerzos adicionales en mantener actualizados datos duplicados (p. ej. cambios o correcciones en los apellidos de un contacto se deberían realizar en todos los ficheros donde aparezca). Este enfoque puede dar lugar a problemas de *inconsistencia* en los datos guardados si la actualización no se realiza en todos los ficheros. Incluso si la actualización se llegase a realizar en todos los ficheros, aun se pueden tener problemas de inconsistencia debido a que cada aplicación efectúa las actualizaciones de manera independiente (p. ej. se puede llegar a actualizar el nombre de un contacto de diferentes maneras en cada uno de los ficheros afectados). En el enfoque de las bases de datos, los datos se guardan en una única base de datos y son compartidos por diferentes aplicaciones y/o usuarios. Cualquier actualización afecta a un único dato y se tienden a evitar las *redundancias* en los datos.

Por otra parte, a raíz del enfoque *datos compartidos*, diferentes aplicaciones y/o usuarios pueden acceder a datos comunes a la vez. En todos estos tipos de accesos se tienen que evitar las *inconsistencias* debidas a accesos concurrentes al mismo dato (p. ej. si varios viajeros intentan reservar a la vez el mismo asiento en un viaje, finalmente el asiento tiene que quedar asignado a un único viajero).

- **Datos relacionados.** Las bases de datos almacenan datos relacionados entre sí. No es correcto referirse a un surtido aleatorio de datos como base de datos.
- **Gran volumen de datos.** Las bases de datos pueden tener cualquier tamaño y complejidad. Se puede hablar de bases de datos con unos cuantos cientos de registros con estructura sencilla (p. ej. la lista de contactos mencionada anteriormente), bases de datos con una estructura mucho más compleja y tamaño del orden de los gigabytes/terabytes (p. ej. la base de datos comercial de (Amazon Inc., 2014) cuyo tamaño fue estimado hace unos cuantos años (Elmasri and Navathe, 2007) en torno a los 2 terabytes) hasta bases de datos con tamaño del orden de los petabytes (p. ej. *World Data Climate Center* - centro mundial para datos del clima, una de las bases de datos más grandes del mundo).
- **Naturaleza autodescriptiva.** Las bases de datos almacenan los datos en sí (p. ej. en el caso de la agenda electrónica, nombres concretos, números de teléfono, etc.) pero también la manera que están estructurados y las restricciones que deben cumplir. La estructura de los datos (también llamada **esquema**) hace referencia a la estructura de cada fichero, el tipo y formato de almacenamiento de cada elemento. Las restricciones que deben cumplir los datos (p. ej. que una dirección de e-mail tiene que contener el símbolo @ para ser considerada válida) se tratarán en detalle en el Capítulo 7. El esquema de la base de datos así como las restricciones se suelen denominar **meta-datos** y se almacenan en la misma base de datos. De allí la naturaleza autodescriptiva de las bases de datos, ya que junto a los datos se almacena también su descripción (meta-datos).
- **Independencia entre programas y datos.** El enfoque de las bases de datos proporciona la capacidad de que los datos permanezcan intactos y accesibles independientemente de las modificaciones a la base de datos que contiene los datos. Esta independencia permite, por ejemplo, re-diseñar una base de datos para satisfacer nuevas necesidades de información de sus usuarios sin preocuparse de que los usuarios/aplicaciones que sigan haciendo el mismo uso de los datos de pronto no sean capaces de acceder a ellos. A modo de ejemplo, en el caso de la lista de contactos mencionada anteriormente, si se necesita incorporarle datos de los números de teléfono de cada contacto para ser utilizados por una aplicación que gestione las llamadas telefónicas, en el enfoque de las bases de datos, los cambios realizados en la lista de contactos no requieren modificaciones/actualizacio-

nes en las demás aplicaciones que utilizaban ya la lista de contactos (gestor del correo electrónico, aviso de cumpleaños).

5.2. Sistemas de gestión de bases de datos

Una base de datos se crea y mantiene mediante un conjunto de programas que bien pueden ser diseñados específicamente para dichas tareas o bien se puede tratar de software de propósito general, es decir un **sistema de gestión de bases de datos** (**DBMS** - *Database Management System*).

> Un DBMS es una colección de programas que permiten a los usuarios **definir**, **construir** y **manipular** una base de datos para distintas aplicaciones (Elmasri and Navathe, 2007).

La definición resalta las funciones esenciales de un DBMS: descripción (definición), construcción y manipulación de los datos.

Definir una base de datos hace referencia a concretar sus meta-datos. Los DBMS suelen proveer para ello **lenguajes de definición de datos** (DDL - *Data Definition Language*).

La construcción de la base de datos es el proceso de almacenar los datos concretos sobre algún medio de almacenamiento controlado por el DBMS. La manipulación de la base de datos incluye tareas como consultar o actualizar los datos en una base de datos. Los DBMS suelen proveer para ello **lenguajes de manipulación de datos** (DML - *Data Manipulation Language*). El Capítulo 8 explica las consultas en bases de datos mediante el subconjunto DML de un lenguaje ampliamente utilizado: SQL (*Structured Query Language*).

Aparte de las funciones esenciales, los DBMS pueden ofrecer otras capacidades. A continuación se enumeran algunas de las funciones complementarias más destacadas de un DBMS:

- **Restricción de accesos no autorizados.** Como se ha visto anteriormente en la Sección 5.1, una de las características de las bases de datos es que los datos pueden ser compartidos entre varios usuarios. Sin embargo, es probable que no todos los usuarios tengan permiso acceder a todo el contenido de la base de datos. Algunos datos almacenados en la base de datos pueden ser *confidenciales*, por lo que el acceso se debe restringir solo a ciertos usuarios autorizados. Es más, los usuarios autorizados pueden tener permitido el acceso solo para la lectura de datos o pueden poseer también permisos de modificación/actualización de los datos. El DBMS debe contar con un subsistema de control de acceso que permita a los administradores de la bases de datos crear cuentas y especificar restricciones para ellas. El DBMS debe además garantizar el cumplimiento de dichas restricciones.
- **Control de concurrencia.** En el caso de que diferentes aplicaciones y/o usuarios autorizados intenten actualizar el mismo dato, dicha actualización se debe realizar de manera controlada. El DBMS debe incluir software para control de concurrencia, de manera que los datos queden en un estado consistente después de accesos simultáneos.
- **Garantizar la integridad de los datos.** Los meta-datos de una base de datos incluyen restricciones que deben cumplir los datos para ser considerados válidos (p. ej. una dirección de e-mail tiene que contener el símbolo @ para ser considerada válida). Las restricciones de integridad se identifican durante el diseño de la base de datos y el DBMS se

encarga de garantizar su cumplimiento por los datos almacenados (p. ej. impedir que se almacene como dirección de e-mail un dato que no contenga el símbolo @), no permitiendo la ejecución de operaciones que dejen los datos incompletos o incorrectos.
- **Suministro de copias de seguridad y recuperación.** Ante fallos de hardware (p. ej. un corte de suministro de electricidad) o de software (p. ej. un error -*bug*- en una aplicación que accede a la base de datos), el DBMS se debe encargar de restaurar la base de datos a un estado consistente, sea el estado inmediatamente antes del fallo, o incluso reanudar la ejecución de alguna tarea que estaba en ejecución desde el punto que fue interrumpida.
- **Soporte de múltiples vistas de los datos.** Diferentes usuarios de una misma base de datos pueden tener necesidades de información y permisos de acceso diferentes y por lo tanto requerir perspectivas (también denominadas **vistas**) diferentes de la base de datos. Una vista puede ser un subconjunto de la base de datos o puede contener datos virtuales derivados de los datos almacenados. Un DBMS puede proporcionar mecanismos para definir múltiples vistas.

En términos ideales, un DBMS debería contar con todas estas funciones aunque no todos los DBMS las implementan. Sin embargo, es conveniente que un DBMS cuente con todas estas funciones si es utilizado para manejar una base de datos multiusuario de una organización/empresa, donde requerimientos como los mencionados a continuación suelen ser más exigentes:

- **fiabilidad**: capacidad para restaurar los datos (al menos parcialmente) en caso de fallos -de hardware o de software;
- **privacidad**: capacidad para garantizar el acceso a los datos únicamente a los usuarios autorizados;
- **eficiencia**: capacidad para garantizar tiempo de respuesta de las operaciones y ocupación de espacio aceptables. La eficiencia depende en gran medida de la técnica de almacenamiento de datos;
- **eficacia**: capacidad para facilitar las actividades de la organización.

A la hora de elegir un DBMS, existen diferentes alternativas. A continuación se ofrece un listado de los DBMS más populares, tanto libres como de pago:

Tipo de DBMS	Ejemplos
libre	PostgreSQL, SQLite, MySQL, HSQLDB (utilizado además de manera embebida por OpenOffice.org, LibreOffice, ApacheOpenOffice)
de pago	Oracle, IBM DB2, FileMaker, Microsoft SQL Server, Microsoft Access

Cuadro 5.1 Ejemplos de DBMS disponibles en el mercado.

5.3. Modelos de datos

Un DBMS ofrece a los usuarios una representación conceptual de los datos que no incluye muchos detalles sobre el almacenamiento de los mismos ni cómo implementa las funcionalidades que proporciona (recordar la independencia programas/datos). Esta *abstracción* se obtiene mediante el uso de diferentes modelos de datos.

Un modelo de datos es una colección de conceptos que sirven para describir la estructura de una base de datos.

La estructura de una base de datos incluye tipos de datos y restricciones que deben cumplir sus datos. Los modelos de datos pueden incluir además un conjunto de operaciones básicas (para especificar lecturas y actualizaciones de la base de datos) y cada vez más a menudo operaciones definidas por el usuario (p. ej. en las bases de datos orientadas a objetos).

Existen tres tipos de modelos de datos:

1. **Modelo conceptual:** representa el dominio del problema tal y como el usuario lo concibe.
2. **Modelo físico:** es un modelo muy detallado que describe técnicamente cómo se almacenan los datos en el ordenador. Suele permanecer oculto al usuario final.
3. **Modelo de representación (lógico o de implementación):** es un modelo intermedio entre el modelo conceptual y el modelo físico. Sus conceptos pueden ser entendidos por usuarios finales aunque no están demasiado alejados de la forma en la que los datos se almacenan en el ordenador. La aplicación de un modelo lógico a un DBMS concreto tiene como consecuencia la obtención de un modelo físico.

5.4. Diseño de bases de datos

Igual que las aplicaciones software, las bases de datos están caracterizadas por un ciclo de vida que incluye las etapas descritas en el Capítulo 2.

El diseño de una base de datos define la estructura y las característica de los datos (**meta-datos**). Tradicionalmente se descompone en tres fases:

- *Diseño conceptual*: descripción concisa de la estructura de los datos a partir de los requisitos, en términos de entidades, atributos y relaciones. Como parte del diseño de una base de datos, el Capítulo 6 describe el **Modelo Entidad-Relación**, un modelo de datos de alto nivel muy popular.
- *Diseño lógico*: descripción de la estructura de los datos con un modelo lógico (p. ej. el modelo relacional). Como parte del diseño de una base de datos, el Capítulo 7 detalla el **Modelo Relacional**, un modelo ampliamente utilizado en el pasado que sigue siendo muy utilizado. Los modelos de datos orientados a objetos son una nueva familia de modelos de implementación aun más próximos a los modelos conceptuales.
- *Diseño físico*: descripción de la estructura de los datos a nivel interno, de acuerdo con las características del sistema gestor de bases de datos que se decide utilizar (véase la Figura 5.1).

Hoy en día el enfoque en auge es el desarrollo de aplicaciones software y BD guiado por el modelado (*model-driven development*, en inglés) (Frankel, 2003; Stahl and Vólter, 2006; Kelly and Tolvanen, 2008). Según este enfoque, tanto el código de una aplicación software como de una BD se pueden generar automáticamente de los modelos de diseño y hay siempre más herramientas CASE que soportan este proceso de generación automática. En muchas herramientas CASE, la separación entre modelo conceptual-lógico-físico ya no está tan clara porque los modelos son a menudo vistas, cada una con un nivel de detalle de los meta-datos diferente, que se crean conforme a un acercamiento iterativo-incremental utilizando

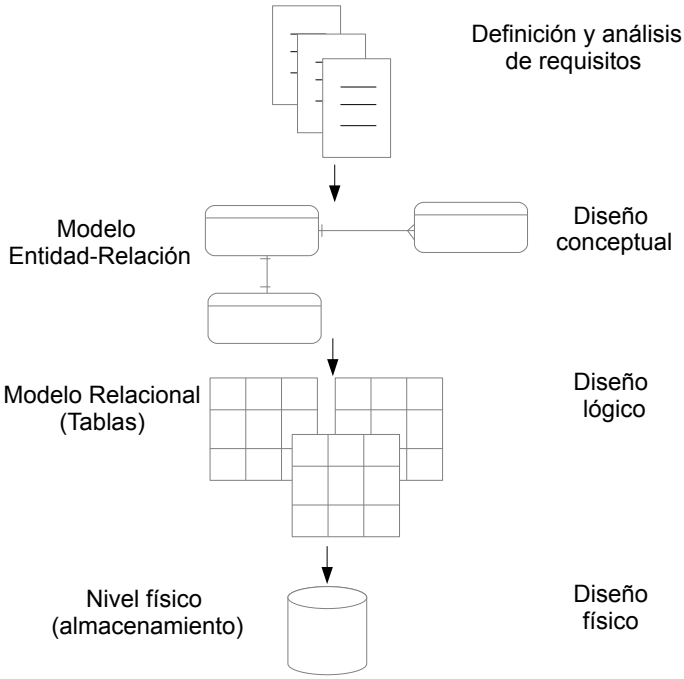


Figura 5.1 Etapas de desarrollo de una base de datos.

un mismo tipo de modelo, como por ejemplo en Visual Paradigm (2015) un modelo Entidad-Relación o un diagrama de clase UML.

5.5. Cuestiones

1. Indique algunos ejemplos de meta-datos que podría contener una base de datos utilizada por una agenda electrónica.
2. Encontrar en la literatura, algoritmos de control de concurrencia para los accesos simultáneos.
3. ¿Se puede considerar a una hoja Excel como base de datos? ¿Cuales son las diferencias entre utilizar una hoja Excel para gestionar datos frente a utilizar una base de datos gestionada por un DBMS?

Capítulo 6
Diseño conceptual de bases de datos

Resumen Este capítulo se centra en las técnicas de modelado que se aplican durante el diseño conceptual de una base de datos. Concretamente se presenta la especificación con el modelo Entidad-Relación (ER), utilizando para ello la notación de Martin. Se repasan los principales elementos del modelo ER y se ejemplifican utilizando el caso de estudio *CarShare*.

6.1. Modelo Entidad-Relación

Un modelo entidad-relación (ER) describe las relaciones que existen entre las diferentes categorías de datos utilizados en un sistema de información que se requieran almacenar en una base de datos. Este modelo plasma el diseño conceptual de la base de datos mediante diagramas entidad-relación (ERD).

Cada metodología de modelado puede utilizar una notación diferente ya que no existe un estándar para representar los elementos de un diagrama ER. La notación tradicional propuesta por Chen (véase Figura 6.1, parte derecha) es ampliamente utilizada en libros académicos (Elmasri and Navathe, 2007), pero no todas las herramientas CASE la implementan. Existen diferentes notaciones alternativas: a lo largo de este libro se recurrirá a la notación de Martin[1] (Fernández-Alarcón, 2006) (véase Figura 6.1, parte izquierda). Esta notación está ampliamente soportada por herramientas CASE (p. ej. Visual Paradigm (2015)) debido a su legibilidad y el uso eficiente del espacio de representación.

A pesar de las diferentes notaciones, los elementos de modelado de un ERD son:

- Entidades.
- Atributos de entidades.
- Relaciones entre entidades.

6.1.1. Entidad

Una entidad representa un conjunto de datos con las mismas características o propiedades, que son significativos en el contexto de un dominio de aplicación: por ejemplo *Viaje* es una entidad significativa en el contexto de *CarShare* porque representa todos los viajes que

[1] Conocida también como notación *Crow's Foot*.

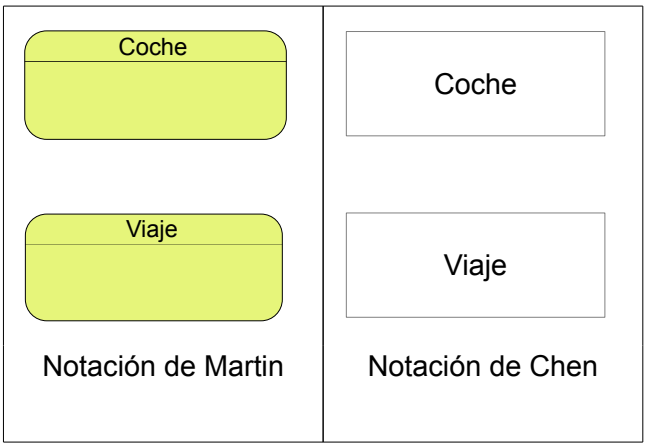


Figura 6.1 Notaciones para las entidades.

gestiona la empresa. Una instancia de entidad es un dato concreto: por ejemplo, dentro del mismo contexto de *CarShare*, un coche en concreto con la matricula *2010 XXX* es una instancia de la entidad *Coche*. Una entidad tiene un nombre que la identifica, se escribe con la primera letra en mayúscula y en singular. Una entidad es un concepto parecido (no igual) al concepto de clase de un diagrama UML de clases. Existen varias categorías de entidades:

- *Personas*. Las entidades que pertenecen a esta categoría representan individuos, grupos u organizaciones. Ejemplos: Conductor, Cliente, Empleado, Estudiante.
- *Lugares*. Ejemplos: Edificio, Tienda, Habitación, Aula.
- *Objetos*. Cualquier objeto puede ser descrito a través de datos. Ejemplos: Coche, Libro, Máquina.
- *Eventos*. Suelen provocar transacciones, activar procesos. Ejemplos: Viaje, Venta, Pedido, Evaluación.
- *Conceptos abstractos*. Ejemplos: Cuenta, Asignatura, Curso.

6.1.2. Atributo

Una entidad es un concepto del cual se quiere almacenar información, por lo que es necesario identificar qué datos se quieren/necesitan almacenar de cada instancia de una entidad determinada. Un atributo representa una característica común a todas las instancias de una entidad: cada instancia tendrá un valor asignado al atributo. Un atributo tiene un nombre (singular) que lo identifica: es un concepto similar al atributo de una clase en un diagrama de clase UML. Por ejemplo, los atributos de la entidad *Coche* (véase Figura 6.2) son: matrícula, VIN (*Vehicle Identification Number* - número de bastidor), la ficha (marca, modelo, número de plazas).

En los diagramas ER existen dos tipos de atributos: los **simples** y los **compuestos**. Un atributo simple corresponde a un tipo de dato básico, como un texto, un número, un booleano. Los atributos compuestos representan la agrupación de varios atributos simples: corresponden a los atributos de un «*datatype*» en un diagrama de clase UML. Por ejemplo un atributo *ficha* de la entidad *Coche* contiene los atributos simples *marca, modelo* y *numPlazas*

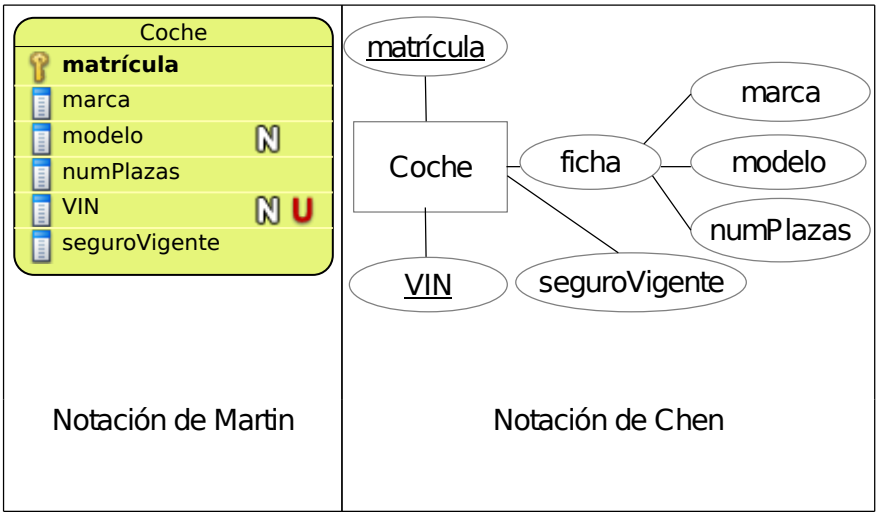


Figura 6.2 Notaciones para los atributos de una entidad.

(véase la notación de Chen en Figura 6.2). En el caso de herramientas CASE que sólo soporten la definición de atributos simples, los atributos compuestos no aparecen en el diagrama ER y se reemplazan por sus partes como atributos simples. En el ejemplo de la Figura 6.2, la *ficha* no aparece como atributo, sino que se han definido su partes como atributos simples (*marca, modelo* y *numPlazas*).

Una entidad contiene un conjunto de instancias: es necesario poder identificar cada instancia. Para ello, siempre tiene que existir un atributo (o un conjunto de atributos) cuyo valor permita identificar unívocamente a una y sólo una instancia de la entidad. Este atributo (o conjunto de atributos) se llama **clave**. Una entidad puede tener más que una clave: la que se elige utilizar como identificador de instancias se denomina **clave primaria**, las demás son **alternativas**. Por ejemplo, los atributos *matrícula* y *VIN* de la entidad *Coche* en Figura 6.2 son dos claves: de hecho cada coche tiene una matrícula y un número de bastidor diferente de los demás. Como clave primaria se ha definido la matrícula.

En general, siempre se puede encontrar una clave para cada entidad: por ejemplo, el conjunto de todos los atributos de una entidad (si se ha definido bien la entidad) es una clave. Un buen criterio de modelado es encontrar una clave mínima, es decir el subconjunto de atributos más pequeño que permita identificar unívocamente a las instancias de la entidad. Si una clave está compuesta por muchos atributos es aconsejable definir un nuevo atributo identificador a propósito.

Un atributo puede tener valores **nulos** (caracterizado por el símbolo $\mathbb{N}$ en Figura 6.2, notación de Martin): un valor nulo corresponde a la ausencia de información para una determinada instancia de una entidad. Por ejemplo, el atributo *modelo* (de coche) de la entidad *Coche* puede tener valores nulos para algún coche en concreto (p. ej. porque se desconoce este dato). Obviamente la clave principal de la entidad no puede tener valores nulos.

Finalmente, un atributo se puede definir como **único**: en este caso corresponde a una clave alternativa (caracterizado por el símbolo $\mathbb{U}$ en Figura 6.2, notación de Martin).

6.1.3. Relación

Las entidades no existen de forma independiente, sino están relacionadas. Una relación es como una asociación entre clases en un diagrama de clases UML y cada relación tiene un nombre que la identifica (normalmente se usa una frase verbal). Una relación puede representar un evento que vincula dos o más entidades, o una afinidad lógica entre dos o más entidades. Existen muchos tipos de relaciones, pero la más común es la relación binaria, es decir una relación entre dos entidades. La Figura 6.3 muestra las notaciones de Martin y de Chen para representar la relación binaria *utilizadoEn* entre la entidad *Coche* y la entidad *Viaje*.

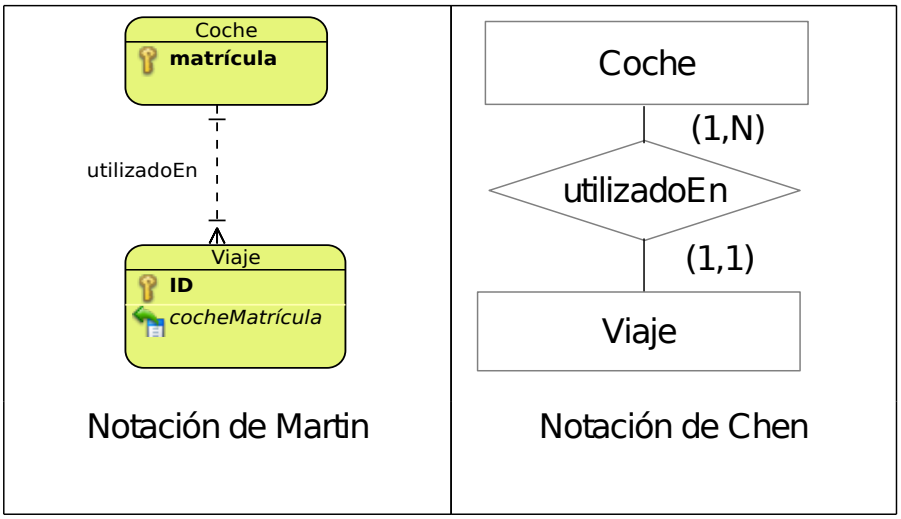


Figura 6.3 Notaciones para una relación entre dos entidades.

Una relación está caracterizada por su **cardinalidad**. La cardinalidad es el número mínimo y máximo de las instancias de una entidad que pueden estar relacionadas con una instancia de la otra entidad. En Figura 6.3, un coche es utilizado en uno o varios (muchos) viajes -cardinalidad $(1, N)$- y en un viaje se utiliza un único coche -cardinalidad $(1, 1)$. Obsérvese la notación de Martin para expresar las cardinalidades: el caso $(1, 1)$ se representa con una raya, el caso $(1, N)$ se representa con una raya y el símbolo del tridente. Obsérvese además que las cardinalidades están puestas en sentido opuesto en las dos notaciones.

El Cuadro 6.1 recoge los diferentes tipos de cardinalidades que se pueden especificar para las relaciones de un diagrama ER y sus correspondientes representaciones en la notación de Martin.

Tipo cardinalidad	Representación
Exactamente uno	– – ┼
Cero o uno	– – ○|
Uno o muchos	– – ┼<
Cero o muchos	– – ○<

Cuadro 6.1 Tipos de cardinalidades con la notación de Martin.

Un concepto asociado a una relación entre dos entidades es la clave externa (o foránea -*foreign key*, en inglés). En una relación binaria identificamos dos entidades: la entidad *padre* y la entidad *hija*. Una **clave externa** es un atributo de la entidad hija que toma valores de la clave primaria de la entidad padre: se usa para poder relacionar las instancias de la entidad hija con las instancias de la entidad padre.

Por ejemplo, en la Figura 6.3, notación de Martin, considérese la relación *uno-a-muchos utilizadoEn* entre *Coche* y *Viaje*. La entidad padre es *Coche* (tiene cardinalidad uno en la relación) y la entidad hija es *Viaje* (tiene cardinalidad *muchos* en la relación). La entidad hija tiene un atributo *Cochematrícula* que es la clave externa y se refiere a la clave primaria *matrícula* de la entidad *Coche*.

6.1.4. Entidad asociativa

Se puede tener una relación *muchos-a-muchos* entre dos entidades, es decir varias instancias de una entidad están asociadas a varias instancias de la otra entidad. Por ejemplo, en la Figura 6.4(A), una persona participa como pasajero en (cero o) varios viajes y en un viaje pueden participar como pasajeros (cero o) varias personas. En este caso se dice que

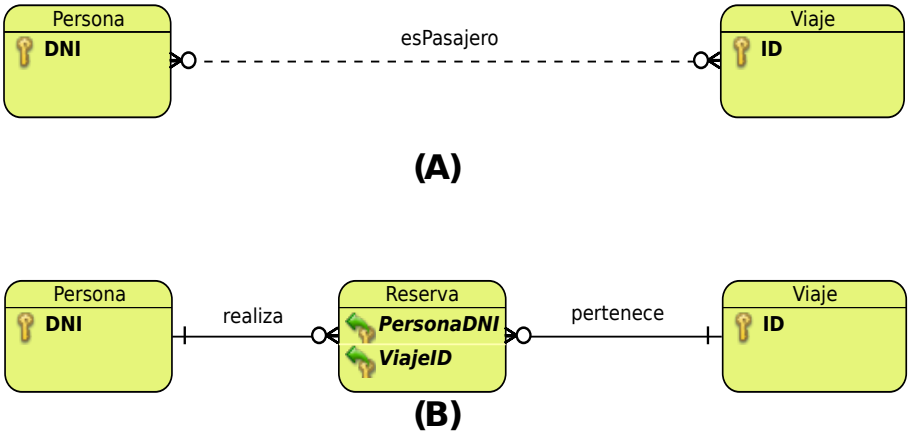


Figura 6.4 (A) Relación muchos-a-muchos. (B) Entidad asociativa.

la relación no es específica. Un diagrama ER no debería contener relaciones no específicas, porque dificulta la comprensión del modelo y además complica su traducción en el modelo lógico.

Una solución para simplificar una relación *muchos-a-muchos* es el uso de una *entidad asociativa*. El proceso es el siguiente:

- Se define una nueva entidad. En el ejemplo de Figura 6.4(B), la nueva entidad es *Reserva*.
- Se asocia la nueva entidad con las entidades iniciales a través de dos relaciones de tipo *uno-a-muchos*. Las entidades iniciales tendrán cardinalidad uno en la relación. Las cardinalidades de la entidad asociativa tendrán que corresponder con las cardinalidades de la relación *muchos-a-muchos* (A). En el ejemplo, la cardinalidad de la entidad asociativa en la relación entre *Persona* y *Reserva* es *cero o más* (B), es decir igual que la cardinalidad de *Viaje* en la relación *muchos-a-muchos* (A). La cardinalidad de la entidad asociativa

en la relación entre *Viaje* y *Reserva* es *cero o más* (B), es decir igual que la cardinalidad de *Viaje* en la relación *muchos-a-muchos* (A).
- La clave primaria de la entidad asociativa está formada por las claves externas que corresponden a las claves primarias de las dos entidades relacionadas.

6.1.5. Generalización

En ciertas ocasiones se puede observar la existencia de varias entidades que representan elementos muy similares y que comparten varios atributos, aunque otros sean distintos. Por ejemplo, un pasajero y un conductor en el caso de estudio *CarShare* comparten atributos como el DNI, el nombre y apellido. Sin embargo, también existen atributos que los diferencian. Por ejemplo, para un conductor se necesita almacenar la fecha a partir de la cual es valido su carnet de conducir. Ese dato no es necesario para un pasajero, por lo tanto no se debería recoger para una persona que fuera pasajero en algún viaje. Considérese que para el pasajero no se necesitan almacenar datos más allá de sus datos personales. Ante esta situación, se puede crear una entidad *supertipo*, que incorpora los atributos comunes, y entidad(es) *subtipo*, que heredan los atributos de datos de un supertipo y que añaden otros atributos de datos que son específicos de las instancias del subtipo. La Figura 6.5 muestra la entidad supertipo *Persona* que contiene los atributos comunes a las entidad subtipo. Una entidad subtipo está relacionada con la entidad supertipo con una relación *uno-a-uno* y, como consecuencia de esa relación, tiene una clave externa (*PersonaDNI* en el ejemplo presentado) que corresponde a la clave primaria (*DNI* en el ejemplo) del supertipo. La clave externa de la entidad subtipo es a la vez clave primaria o alternativa (según decisiones de diseño) de su entidad. En el ejemplo presentado, la clave externa *PersonaDNI* de la entidad *Conductor* es a la vez clave primaria de la entidad. En la Figura 6.5 no se ha incluido una

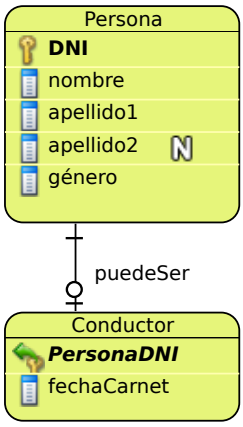


Figura 6.5 Generalización: *Persona* entidad supertipo, *Conductor* entidad subtipo

entidad *Pasajero*, al considerarse que no existían atributos específicos a un pasajero (que no fueran recogidos ya en la entidad *Persona*).

6.2. Cuestiones

1. Explique porqué la clave primaria de una entidad no puede admitir valores nulos.
2. Identifique las principales diferencias entre una entidad de un diagrama ER y una clase de un diagrama UML de clases.
3. Modifique el diagrama ER de la Figura 6.5 para permitir guardar la preferencia de un pasajero de viajar o no en el asiento delantero.
4. Complete el modelo ER para el caso de estudio *CarShare*. Compare el resultado con el modelo de dominio del Capítulo 4 (Figura 4.11)

6.3. Casos prácticos

Ej. 1 — Se quiere diseñar una base de datos para almacenar los datos relacionados con las asignaturas de una titulación universitaria, los docentes que imparten las asignaturas y los estudiantes matriculados. Cada asignatura tiene un código de identificación, un nombre y una descripción del temario. Además habrá que tener en cuenta que una asignatura puede o no estar activada. Los docentes tienen un código de identificación (NIP) y los estudiantes un número de matrícula (NIA). Tanto los docentes como los estudiantes pueden tener uno o más nombres, uno o dos apellidos y una dirección de contacto (calle/avenida, número, ciudad, código postal, número de teléfono/móvil). Un docente puede impartir una o más asignaturas, y una asignatura puede ser compartida entre uno o más docentes. Las asignaturas no activadas no tienen docentes asignados. Un estudiante puede estar matriculado en una o más asignaturas, y una asignatura pueden ser cursada por uno o más estudiantes. Las asignaturas no activadas no tienen estudiantes. Se tendrá que representar también la evaluación de los estudiantes, es decir la nota conseguida por un estudiante en el examen final de una asignatura.

Ej. 2 — Muchas personas han intentado ganar con las apuestas en las carreras de caballos buscando maneras para obtener ventajas respecto a los demás apostantes. Se ha publicado una gran variedad de libros sobre carreras de caballos, y cada autor afirma que su método lleva seguramente a beneficios. Hasta ahora Miguel no ha sido capaz de ganar mucho dinero siguiendo dichos métodos y ha decidido construir un sistema para hacer apuestas sofisticadas. La carrera de caballos es uno de los deportes más documentados: hay miles de datos publicados sobre las carreras que Miguel puede organizar y almacenar, una vez construida la base de datos. Cada caballo tiene un nombre y, para cada caballo, se puede encontrar información sobre el dueño, la raza, el peso y su progenitores. Con respecto a los jinetes, a parte del nombre, se puede encontrar información sobre el número de seguridad social, el peso, la estatura, el número de años de carreras y la edad. También hay información sobre los hipódromos: en particular, el nombre, la dirección, el tipo (tierra, césped, ...). Finalmente cada carrera tiene asignado un nombre, la fecha y hora, el estado (futura, en curso, pasada), el total apostado (el bote), el caballo ganador, la distancia del recorrido, la posición de cada participante. Un jinete monta exactamente a un caballo y un caballo está montado por un sólo jinete. Un caballo puede participar en una o más carreras en un hipódromo. Crear un modelo ER para almacenar todos los datos.

Ej. 3 — La empresa Martínez produce una amplia gama de repostería que reparte en toda España a diario. La empresa necesita almacenar la información relacionada con su empleados, clientes y productos. Cada empleado (o cliente) tiene un número de identificación, nombre

y dirección (calle, número, piso, puerta, ciudad, provincia y código postal). Además, para los empleados, se necesita guardar la información sobre el sexo, la fecha de nacimiento, su posición en la empresa (administrativo, comercial, producción, etc.), el sueldo cobrado por horas y por horas extras, la fecha de incorporación en la empresa. Los clientes están localizados a través de la longitud (X) y latitud (Y), y para cada cliente se guarda el tiempo requerido (en fracciones de hora) para la entrega de los pedidos. Finalmente, cada producto tiene un código de identificación, una descripción, el precio de venta y el número de unidades producidas a diario. Los productos pueden ser pedidos por uno o muchos clientes, y un cliente puede hacer un pedido de muchos productos a la vez. Cada empleado está relacionado a cero o muchos productos, un producto está producido por sólo un empleado. Crear un modelo ER de la base de datos que tenga en cuenta la descripción dada anteriormente.

Ej. 4 — Las principales compañías aéreas que proveen servicios en Taiwan son *UniAir, TransAsia AirWays, Far Eastern Transport* y *Great China Airlines*. La administración de la Aviación Federal de Taiwan necesita una base de datos con la información de todas las líneas aéreas. Esta información será accesible por todas las líneas aéreas de Taiwan con el objetivo de ayudar a las compañías en la evaluación de su posición competitiva en el mercado doméstico. Cada línea aérea se caracteriza por un número de identificación, nombre y dirección, una persona de contacto (nombre, número de teléfono). Los datos relacionados con los aviones son el número de identificación, la capacidad (número de personas máxima a transportar) y el modelo. Los empleados tienen un número de identificación, nombre, dirección, fecha de nacimiento, posición en la compañía y una cualificación. Cada ruta tiene un número de identificación, un origen y un destino, una clasificación (es decir, doméstico, internacional o intercontinental), una distancia en km y un precio básico por pasajero. Cada línea aérea tendrá que almacenar la información relacionada con sus transacciones: es decir, las ventas de vuelos a los pasajeros, los pagos del mantenimiento de los aviones. Cada transacción está caracterizada por un número de identificación, una fecha, una descripción, un importe cobrado/pagado. Cada empleado trabaja para una sóla línea aérea. Las líneas aéreas pueden asignar diferentes aviones a diferentes rutas dependiendo de la disponibilidad. Además, cada línea aérea puede hacer una o más transacciones: obviamente cada transacción está asociada sólo a una línea aérea. Crear un modelo ER de la base de datos que tenga en cuenta la descripción dada anteriormente.

Ej. 5 — Se requiere la construcción de una base de datos para monitorizar equipos, jugadores y partidos de baloncesto de una liga interacadémica. Para cada jugador se requiere registrar los siguientes datos: número de la seguridad social, nombre, dirección, fecha de nacimiento, posición en el equipo y el número de años que ha estado jugando con el equipo. Para cada equipo que participa en la liga se requiere registrar los siguientes datos: el nombre del equipo, el nombre de la academia a la que representa, la posición actual que ocupa el equipo, el número de partidos que ha ganado y que ha perdido hasta el momento. La base de datos guardará además datos sobre el entrenador del equipo. Estos datos incluyen: número de la seguridad social, nombre, edad, número de años que lleva entrenando al equipo actual, número total de años que lleva entrenado y el número de ligas que ha ganado como entrenador. Los siguientes datos se deben registrar sobre cada partido: su número de identificación, la fecha y lugar del evento, la hora del comienzo y fin, y el ganador. Un entrenador puede dirigir exactamente a un equipo, mientras un equipo puede tener exactamente un entrenador. Cada equipo puede jugar uno o más partidos, mientras un partido es jugado por un equipo invitado y un equipo local. Un equipo tiene varios jugadores y un jugador juega en un único equipo. Un equipo puede ganar más de un partido y un partido es ganado por exactamente

un equipo (los partidos no pueden acabar en empate). Definir un modelo ER de la base de datos que tenga en cuenta la descripción dada anteriormente.

Capítulo 7
Diseño lógico de bases de datos

Resumen Este capítulo se centra en la fase de diseño lógico de una base de datos. Se presenta el modelo relacional y la forma de construirlo a partir del modelo conceptual ER. Como particularidad del modelo relacional se explican las restricciones de integridad intra-relacionales e inter-relacionales. Se introduce también el lenguaje SQL, incidiendo sobre su condición de lenguaje de definición de bases de datos (DDL).

7.1. Modelo relacional

El **diseño lógico** de una base de datos se concreta en una especificación correcta y eficiente de sus **meta-datos**. El diseño lógico se define a partir del diseño conceptual. A lo largo de la historia han existido diferentes modelos lógicos de bases de datos (modelo jerárquico, modelo en red, modelo relacional) con diferencias sobre todo en la manera de estructurar los datos.

El modelo relacional es uno de los modelos más extendidos con diferencia. Fue propuesto por E.F. Codd en el año 1970 y se ha impuesto a lo largo del tiempo por ser un modelo simple y con sólidos fundamentos matemáticos. Concretamente el modelo relacional se basa en el concepto de relación y en la teoría de conjuntos.

El modelo relacional es independiente de la forma en la que se almacenan los datos. Por tanto la base de datos se puede implementar en cualquier DBMS que soporte el modelo relacional (la mayoría).

El elemento fundamental del modelo relacional es la **relación** que se puede representar en forma de **tabla**. Cada fila de la tabla es una colección de datos relacionados entre sí (**tupla**) que representan una instancia de una entidad del mundo real. Las columnas de la tabla son los **atributos** (propiedades) de los datos. Todos los valores de una columna tienen el mismo tipo de datos.

Una relación se especifica con un nombre y, entre paréntesis, con los nombres de los atributos que tiene:

> *Coche(matricula, marca, modelo, numPlazas, VIN, seguroVigente)*

La relación se puede representar en forma de tabla (como en la Figura 7.1), con las siguientes restricciones:

1. no puede haber filas duplicadas,
2. el valor de los atributos es atómico, es decir cada casilla de la tabla tiene un valor único a la vez, no una lista de valores.

Además, hace falta constar que el orden de las filas y columnas es irrelevante.

Data of Coche

matrícula (PK)	marca	modelo	numPlazas	VIN	seguroVigente
4952 DDD	VOLKSWAGEN	(NULL)	5	WVWZZZ3CZ9E334557	TRUE
5000 DXX	SKODA	Fabia	5	(NULL)	TRUE

Figura 7.1 Ejemplo de tabla en el modelo relacional.

7.2. Traducción del modelo ER al modelo relacional

El modelo conceptual Entidad Relación (ER) conjuga muy bien con el modelo lógico relacional.

Traducción de entidades

Hay una correspondencia directa entre una entidad del modelo ER con una tabla en el modelo relacional. Para cada entidad A del modelo ER, con atributos $(a_1, a_2, ..., a_n)$ se crea una tabla A con n columnas correspondientes a los atributos de la entidad A. La tabla tiene una clave primaria (*Primary Key* -**PK**- en inglés) formada por atributo(s) de la entidad A. Si no hay ningún atributo que pueda ser clave, se añade un atributo con este propósito.

Dominio de atributos

Cada atributo tiene un **dominio** que define los valores admisibles que puede tener. Los tipos de valores para los atributos son tipos de datos lógicos: el Cuadro 7.1 muestra una lista de los tipos de datos lógicos más utilizados.

Tipo	Descripción
bit	Un tipo de atributo que sólo puede tener como valor: verdadero (`TRUE`) o falso (`FALSE`)
varchar	Un conjunto limitado de caracteres (texto), incluidos los números
char	Un sólo carácter, incluidos números 0,. . . ,9
date	Una fecha (en cualquier formato)
double	Un tipo de número real (doble precisión)
float	Un tipo de número real (simple precisión)
integer	Un tipo de número entero
time	Una hora (en cualquier formato)

Cuadro 7.1 Tipos de datos lógicos.

La Figura 7.2 muestra los tipos de datos lógicos asociados a los atributos de la entidad *Coche*: los números en paréntesis especifican la longitud máxima. Por ejemplo, *varchar(10)* para el atributo *matrícula* indica que los valores pueden tener cómo máximo 10 caracteres.

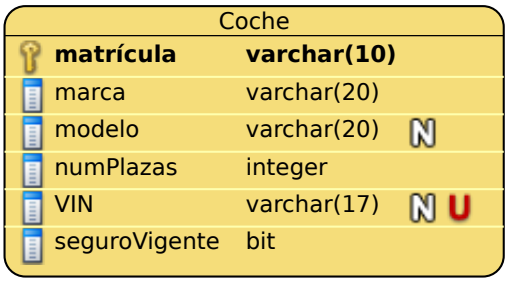


Figura 7.2 Dominios de atributos.

Además del tipo, los valores admisibles para un atributo pueden estar limitados a un cierto rango: la definición del rango debe tener un significado desde la perspectiva de los usuarios y del negocio. Por ejemplo los valores del atributo *numPlazas* (véase la Figura 7.2) se podrían limitar a un rango 1..7.

La traducción de la entidad representada en la Figura 7.2 a la correspondiente tabla en el modelo lógico relacional puede observarse en la Figura 7.1.

Traducción de relaciones

Las relaciones con cardinalidad uno-a-uno o uno-a-muchos en el modelo ER tienen una traducción directa al modelo lógico relacional mediante la introducción de claves externas. Concretamente, para una entidad que actúa de entidad hija en una relación (uno-a-uno o uno-a-muchos) en el modelo ER se introduce una(s) columna(s) especial(es) en la tabla correspondiente en el modelo relacional: las columnas clave externa/foránea (*Foreign Key* -**FK**- en inglés). En la Figura 7.3 se puede ver un ejemplo de traducción de un modelo ER al modelo relacional correspondiente.

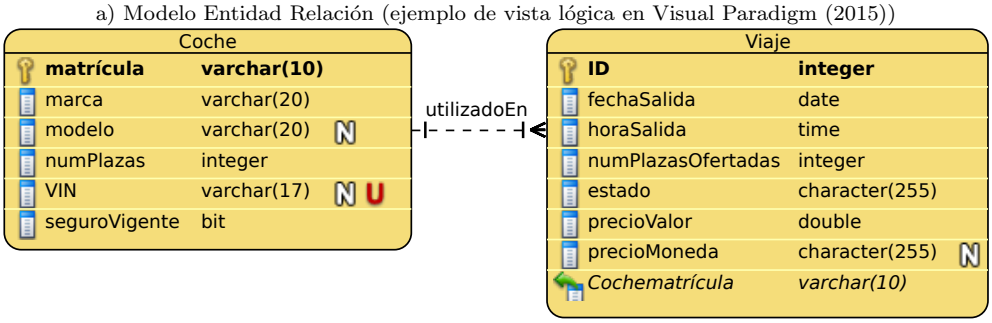
El atributo *Cochematrícula* (FK) de la tabla *Viaje* es una clave externa porque coge valores de una clave primaria de otra tabla (*Coche*). Las claves externas son muy importantes en el modelo relacional porque nos permiten asociar o enlazar las diferentes tablas. Una base de datos relacional es un conjunto de tablas que están interrelacionadas mediante claves externas.

Las relaciones muchos-a-muchos entre dos entidades en el modelo ER también tienen traducción a tablas en el modelo relacional. Se recomienda sin embargo la simplificación de las relaciones ya en el modelo ER introduciendo entidades asociativas, de modo que el modelo ER esté caracterizado sólo por relaciones uno-a-uno y/o uno-a-muchos.

La estructura de cada una de las tablas se define mediante **esquemas**, siendo el esquema de la base de datos el conjunto de los esquemas de todas sus tablas. El esquema de una base de datos define los **meta-datos** o meta información y representa los datos estables de una base de datos que no suelen cambiar (a menudo) en el tiempo.

Se identifica como **estado** de una tabla sus tuplas. El estado de una base de datos es el conjunto de las tuplas en las tablas (datos). El estado de una base de datos suele cambiar en el tiempo mediante operaciones de añadir, eliminar, modificar datos.

a) Modelo Entidad Relación (ejemplo de vista lógica en Visual Paradigm (2015))



b) Modelo Relacional

Data of Coche

matrícula (PK)	marca	modelo	numPlazas	VIN	seguroVigente
4952 DDD	VOLKSWAGEN	(NULL)	5	WVWZZZ3CZ9E334557	TRUE
5000 DXX	SKODA	Fabia	5	(NULL)	TRUE

Data of Viaje

ID (PK)	fechaSalida	horaSalida	numPlazasOfertadas	estado	precioValor	precioMoneda	Cochematrícula (FK)
1	19/06/2014	08:45:00	2	registrado	15	euro	✓ (NULL) / 4952 DDD / 5000 DXX

Figura 7.3 Traducción modelo ER a modelo relacional.

7.3. Restricciones de integridad

Existen reglas que tienen que ser satisfechas por las tuplas de cada tabla de una base de datos para considerarlas correctas. Estas reglas son denominadas **vínculos de integridad** o **restricciones de integridad**. Las restricciones de integridad se definen cuando se crea el esquema de una base de datos y se refieren a todas sus tuplas. Las restricciones de integridad especifican condiciones que deben cumplir los datos para su correcto almacenamiento.

Existen dos restricciones de integridad inherentes al modelo relacional:

1. **Restricción de clave**. Se refiere a las restricciones impuestas por la definición de la clave primaria. Una clave primaria es un atributo (o conjunto de atributos) cuyo valor identifica unívocamente a una y sólo una tupla de una tabla. Por lo tanto una clave **no puede tener valores repetidos** en diferentes tuplas, puesto que no permitiría saber de qué tupla se trata. La clave primaria **no puede tener tampoco valores nulos**. Si la clave primaria está formada por varios atributos, ninguno de los atributos que la forman puede tener valores nulos.
2. **Restricción de integridad referencial**. Se refiere a las restricciones impuestas por los atributos clave externa. Sirve para mantener la consistencias entre tuplas de dos tablas relacionadas mediante un atributo clave externa. Concretamente, establece que una tupla en una tabla T2 que haga referencia a otra tabla T1 deberá referirse a una tupla **existente** en esa tabla T1. Considérese el ejemplo de la Figura 7.3 b): el atributo *Cochematrícula*, que es clave externa en la tabla *Viaje* puede tener únicamente valores existentes en la tabla *Coche* para el atributo clave primaria *matrícula*. Un atributo clave externa puede tomar en ocasiones el valor nulo (NULL), si se ha especificado como atributo anulable.

Además de las restricciones inherentes al modelo relacional se pueden definir restricciones de integridad de usuario:

- **Restricción de valor único** (UNIQUE). Es una restricción que impide que un atributo tenga un valor repetido. Todos los atributos clave primaria cumplen esta restricción. No obstante es posible que un atributo no sea clave primaria y requiera una restricción de valor único (clave alternativa).
- **Restricción de valor nulo** (NULL). Un atributo es obligatorio si no admite el valor nulo o NULL. Si admite como valor el valor nulo o NULL, el atributo es opcional. Todos los atributos clave primaria son atributos NOT NULL.
- **Restricción de dominio**. Son restricciones sobre los valores de un atributo y se especifican con predicados lógicos. Una restricción de dominio exige que el valor que puede tener un atributo esté dentro del dominio definido. Considérese por ejemplo la tabla *Coche* (Figura 7.3 b)) que incluye, entre otros, una columna *numPlazas*. Se puede restringir el dominio de los posibles valores de la columna *numPlazas* en el intervalo $[1, 7]$ definiendo el siguiente predicado lógico:

$$(numPlazas >= 1)\ AND\ (numPlazas <= 7)$$

- **Restricción entre valores de atributos diferentes**. Son restricciones sobre los valores de un atributo que dependen de los valores de otro(s) atributo(s). De manera similar a las restricciones de dominio, se especifican con predicados lógicos. Por ejemplo, volviendo a considerar la tabla *Viaje*, se puede definir una restricción para asegurar que el valor del atributo *precioMoneda* se especifica siempre que el viaje no sea gratuito:

$$(precioValor = 0)\ OR\ (precioMoneda \text{ IS NOT } \texttt{NULL})$$

El predicado lógico no está satisfecho en caso de un precio (*precioValor*) diferente a 0 y el atributo *precioMoneda* con un valor `NULL`; está satisfecho en todo los demás casos.

7.4. Lenguaje de definición de datos

El modelo lógico es independiente de cualquier DBMS. La implantación definitiva de la base de datos en un sistema informático corresponde al diseño físico. Para ello se recurre a un sistema DBMS concreto y se realiza utilizando un lenguaje DDL (*Data Definition Language* o Lenguaje de Definición de Datos).

El lenguaje SQL (*Structured Query Language* o Lenguaje de Consulta Estructurado) se utiliza tanto para la manipulación de datos en un base de datos relacional como para definir su estructura. La principal función del sub-lenguaje DDL de SQL es crear las tablas y otros objetos de una base de datos. Para ello dispone de tres instrucciones básicas:

- **CREATE**. Crea un objeto de un determinado tipo (DATABASE; TABLE, etc.). Por ejemplo:

```
CREATE TABLE COCHE (MATRICULA varchar(10) NOT NULL,
        MARCA varchar(20) NOT NULL,
        MODELO varchar(20), NUM_PLAZAS integer NOT NULL,
```

```
        VIN varchar(17),
        SEGURO_VIGENTE bit,
        CONSTRAINT MATRICULA_PK PRIMARY KEY (MATRICULA)
       );
```

crea (en una base de datos HSQLDB (2014)) la tabla *Coche* con sus respectivas columnas, tal como aparecen en la Figura 7.1. Se observa que la sentencia sigue una estructura:

```
CREATE TABLE tabla (
        atributo1 tipo_de_dato NULL/NOT NULL,
        atributo2 tipo_de_dato NULL/NOT NULL,
        ...
        CONSTRAINT restriccion  PRIMARY KEY (atributox)
       );
```

Cada atributo puede ser `NULL` (anulable) o `NOT NULL` (no anulable). Si la opción no se rellena se entiende por defecto el valor NULL. La opción **CONSTRAINT** permite la definición de restricciones de integridad (en el ejemplo, la restricción de clave).

- **DROP**. Elimina un objeto de la base de datos. Por ejemplo:

```
DROP TABLE VIAJE;
```

borra de la base de datos la tabla *Viaje* junto con todos sus datos.

- **ALTER**. Modifica la definición de un objeto de la base de datos. Por ejemplo:

```
ALTER TABLE VIAJE
        ADD CONSTRAINT UTILIZADO_EN
        FOREIGN KEY (COCHE_MATRICULA)
        REFERENCES COCHE (MATRICULA);
```

modifica la tabla *Viaje*, concretamente la definición del atributo *Cochematrícula* que pasa a ser clave externa que enlaza la tabla *Viaje* con la tabla *Coche* a través del atributo clave primaria de esta última (*matrícula*). La sentencia *ALTER TABLE* permite no solo definir nuevas restricciones de integridad en una tabla (como se ve en el ejemplo anterior), sino también añadir, modificar, renombrar, borrar atributos en una tabla.

A continuación se muestran las principales sentencias SQL para añadir restricciones de integridad de usuario a una base de datos relacional:

```
ALTER TABLE tabla ADD UNIQUE (atributo1, atributo2);

ALTER TABLE tabla ADD CONSTRAINT restriccion
CHECK (condicion);

ALTER TABLE tabla ADD CONSTRAINT enum_restriccion
CHECK (columna IN ('op1', 'op2', 'op3'));
```

La primera sentencia se usa para añadir una restricción de valor único a un atributo o a un conjunto de atributos de una tabla. Por ejemplo, supóngase que la tabla *Coche* incluye un atributo *VIN* que representa el número de bastidor de un coche y se quiere añadir la restricción de valor único al atributo. La sentencia SQL será la siguiente:

```
ALTER TABLE COCHE ADD UNIQUE (VIN);
```

La segunda sentencia se usa para añadir restricciones de dominio y restricciones entre valores de atributos diferentes de una tabla. Por ejemplo la restricción de dominio del atributo

numPlazas de la tabla *Coche* (un número entre uno y siete) y la restricción entre valores del atributo *precioValor* y el atributo *precioMoneda* de la tabla *Viaje* comentados en la Sección 7.3 se expresan con las siguientes sentencias SQL:

```
ALTER TABLE COCHE ADD CONSTRAINT CHECK_NUM_PLAZAS
CHECK ((COCHE.NUM_PLAZAS >= 1) AND
       (COCHE.NUM_PLAZAS <= 7)
      );

ALTER TABLE VIAJE ADD CONSTRAINT CHECK_PRECIO
CHECK ((VIAJE.PRECIO_VALOR = 0) OR (PRECIO_MONEDA IS NOT NULL)
      );
```

La tercera sentencia se puede utilizar para añadir restricciones de dominio en caso de un atributo de tipo enumerado, es decir un atributo que puede tomar valores incluidos en una lista de valores permitidos. Por ejemplo, considerando la tabla *Persona* (traducción de la entidad *Persona*, véase la Figura 6.5) incluye un atributo *género* que representa el género de una persona y puede tener un valor V (varón) o M (mujer). La sentencia SQL a continuación expresa la restricción de dominio para el atributo *género*:

```
ALTER TABLE PERSONA ADD CONSTRAINT CHECK_GENERO
CHECK (PERSONA.GENERO IN ('V','M'));
```

Observar que para referirse a un atributo de una tabla se usa la sintaxis *nombreTabla.nombreAtributo*. Los literales de tipo carácter o cadena de caracteres (por ejemplo, 'V' y 'M') se ponen entre comillas simples.

Cada restricción añadida tiene un nombre (por ejemplo, *checkNumPlazas*, *checkFechas*, *checkGenero*) definido por el usuario, que se utiliza en caso de su posterior eliminación. Para la eliminación de una restricción de integridad se utiliza la sentencia:

```
ALTER TABLE tabla DROP CONSTRAINT restriccion;
```

Por ejemplo, si se quiere eliminar la restricción definida para el atributo *género* de la tabla *Persona* hay que ejecutar la siguiente sentencia SQL:

```
ALTER TABLE PERSONA DROP CONSTRAINT CHECK_GENERO;
```

Para obtener una lista de las restricciones de integridad definidas en una base de datos HSQLDB se puede definir la siguiente consulta:

```
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME
FROM INFORMATION_SCHEMA.SYSTEM_TABLE_CONSTRAINTS
```

La ejecución de la consulta permite obtener una lista de todos las restricciones de integridad de la base de datos (nombre de la restricción, tipo de la restricción y la tabla de referencia):

CONSTRAINT_NAME	CONSTRAINT_TYPE	TABLE_NAME
CHECK_NUM_PLAZAS	CHECK	COCHE
CHECK_PRECIO	CHECK	VIAJE
CHECK_GENERO	CHECK	PERSONA
SYS_CT_139	UNIQUE	COCHE
UTILIZADO_EN	FOREIGN KEY	VIAJE
...	...	...
DNI_PK	PRIMARY KEY	PERSONA
MATRICULA_PK	PRIMARY KEY	COCHE
VIAJE_ID_PK	PRIMARY KEY	VIAJE
...	...	...

En el ejemplo, las primeras tres restricciones se corresponden con las definidas previamente utilizando la sentencia SQL:

ALTER TABLE ... ADD CONSTRAINT ... CHECK ...,

la cuarta restricción corresponde a la restricción de valor único para el atributo *VIN* de la tabla *COCHE*. Las demás restricciones son restricciones de integridad referencial (clave externa) y de clave (clave primaria).

En una base de datos HSQLDB, la siguiente consulta SQL:

```
SELECT CONSTRAINT_NAME, CHECK_CLAUSE
FROM INFORMATION_SCHEMA.SYSTEM_CHECK_CONSTRAINTS
```

permite ver más en detalle la definición de las restricciones de integridad definidos con la sentencia SQL

ALTER TABLE ... ADD CONSTRAINT ... CHECK ...,

en particular los predicados lógicos que definen las restricciones. La ejecución de la consulta para el ejemplo considerado en esta sección produce la siguiente tabla:

CONSTRAINT_NAME	CHECK_CLAUSE
CHECK_NUM_PLAZAS	`(COCHE.NUM_PLAZAS>=1) AND (COCHE.NUM_PLAZAS<=7)`
CHECK_PRECIO	`(VIAJE.PRECIO_VALOR=0) OR (VIAJE.PRECIO_MONEDA IS NOT NULL)`
CHECK_GENERO	`PERSONA.GENERO IN ('V','M')`

Las sentencias SQL mostradas en este capítulo funcionan en el sistema de gestión de bases de datos HSQLDB. En otros DBMS estas sentencias pueden sufrir ligeras variaciones. Aparte de la utilización del lenguaje DDL, la definición de la base de datos se puede realizar mediante herramientas gráficas proporcionadas o compatibles con el DBMS utilizado.

7.5. Caso práctico

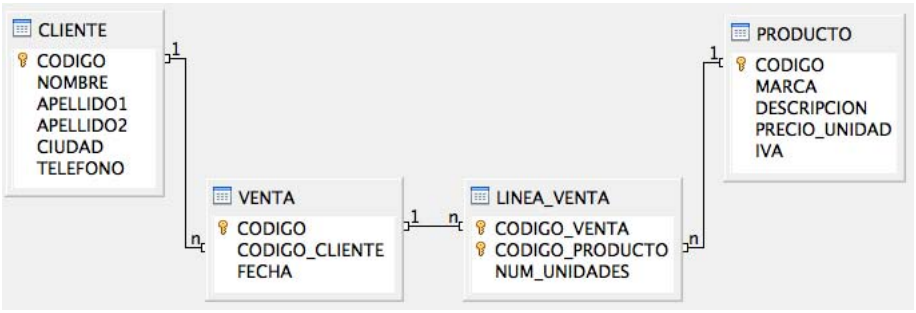


Figura 7.4 Estructura de base de datos *Gestión ventas*.

Considerar la estructura de base de datos de Figura 7.4 y los siguientes datos almacenados en sus tablas:

CLIENTE					
CODIGO	NOMBRE	APELLIDO1	APELLIDO2	CIUDAD	TELEFONO
1	José	Pérez	López	Zaragoza	976767676
2	Juan	García	Gómez	Zaragoza	666666666

VENTA		
CODIGO	CODIGO_CLIENTE	FECHA
1	2	12/12/12
1	1	12/12/13
2	4	12/12/12

PRODUCTO				
CODIGO	MARCA	DESCRIPCION	PRECIO_UNIDAD	IVA
A001	Bic	Bolígrafo negro	0,2	20
A002	Milan	Goma borrar	0,14	20

LINEA_VENTA		
CODIGO_VENTA	CODIGO_PRODUCTO	NUM_UNIDADES
1	A001	100
1	A002	100
2	A003	100

Verificar si los datos cumplen las restricciones de integridad (clave primaria y clave externa). En caso contrario, identificar las tuplas (filas) de las tablas que no cumplen las restricciones.

Parte IV
Uso de sistemas de información: herramientas de apoyo a la toma de decisiones

Capítulo 8
Consultas con SQL

Resumen El lenguaje de consulta estructurado SQL (Structured Query Language) es el lenguaje de referencia para la definición y manipulación de datos en bases de datos relacionales. Este Capítulo introduce la sintaxis de la sentencia SQL *SELECT* que permite realizar consultas sobre una base de datos para extraer información de una o varias tablas.

8.1. Introducción

El lenguaje de consulta estructurado SQL -*Structured Query Language* (López Montalbán et al, 2011)- es el lenguaje de referencia para la definición y manipulación de datos en bases de datos relacionales.

SQL es un lenguaje de programación declarativo, es decir un lenguaje cuyas sentencias describen los resultados que se quieren obtener en lugar de especificar cómo obtener dichos resultados a través de un procedimiento. Incluye un conjunto de sentencias para la definición del esquema de una base de datos (*Data Definition Language* - DDL) y un conjunto de sentencias para la manipulación de datos (*Data Manipulation Language* - DML).

En el Capítulo 7 (Sección 7.4) se ha introducido la parte DDL de SQL que permite crear, eliminar y modificar el esquema de una base de datos, es decir los meta-datos.

En este capítulo se hace hincapié en la parte DML de SQL que permite realizar consultas sobre una base de datos y actualizaciones (modificar, añadir, remover datos) de una base de datos. En particular, se introduce la sintaxis de la sentencia SQL *SELECT* que permite realizar consultas sobre una base de datos para extraer información de una o varias tablas. El resultado de una consulta se presenta en forma de nueva tabla (virtual).

8.2. Consultas sobre una tabla

Las consultas más sencillas involucran sólo una tabla de la base de datos y permiten seleccionar columnas de la tabla.

El formato básico para hacer una consulta es el siguiente:

```
SELECT [DISTINCT] select_expr FROM tabla
```

donde `select_expr` es una expresión que incluye:

- el nombre de una columna de `tabla` o
- una lista de nombres de columnas de `tabla` separados con una coma o
- `*` (asterisco) es decir, todas las columnas de `tabla` o
- una expresión algebraica compuesta por operadores, operandos y funciones.

La expresión `select_expr` puede incluir la secuencia *AS* `alias` para cambiar el nombre a una columna en la tabla resultante a la consulta, donde *AS* es una palabra clave y `alias` es un nombre asignado a la columna por el usuario.

La palabra clave *DISTINCT* está entre corchetes para indicar que es opcional: se usa para mostrar sólo las tuplas con valores distintos.

Después de la cláusula *FROM* se indica el nombre de la tabla sobre la cual se va a ejecutar la consulta *SELECT*. A continuación se ilustran algunas consultas sobre la base de

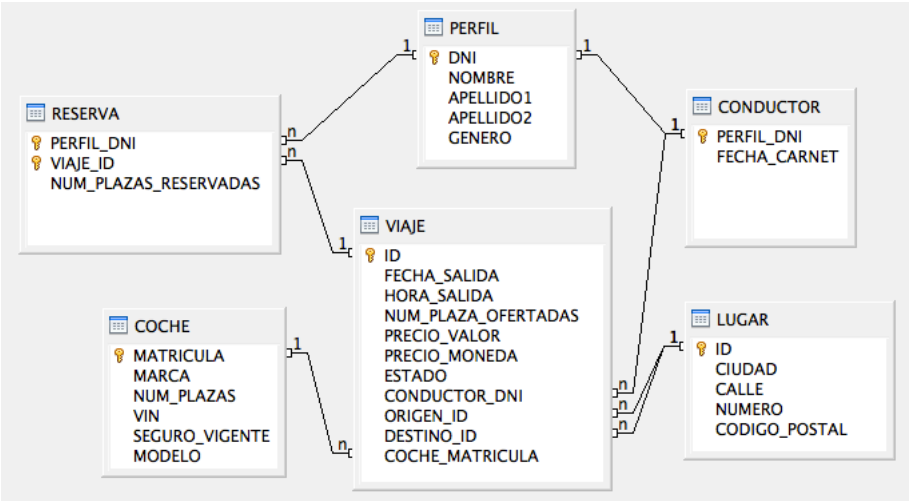


Figura 8.1 Base de datos (parcial) para el caso *CarShare*.

datos relacional de Figura 8.1 que almacena parte de los datos utilizados por la aplicación *CarShare* (véase Apéndice A). Un ejemplo de estado de la tabla PERFIL se muestra en el Cuadro 8.1.

DNI	NOMBRE	APELLIDO1	APELLIDO2	GENERO
11111111A	Paula	Silva	Mora	M
12345127G	Ana	Martínez	García	M
12345678D	Jorge	Pérez	González	V
21345656T	María	Rodríguez	Campos	
98789776O	Juan	Silva	Pérez	V

Cuadro 8.1 Estado de la tabla PERFIL (ejemplo).

La siguiente consulta produce la lista de los primeros apellidos de todos los perfiles:

```
-- Consulta 1
SELECT APELLIDO1 FROM PERFIL
```

El resultado que se obtendrá es el siguiente:

APELLIDO1
Silva
Martínez
Pérez
Rodríguez
Silva

Si se modifica la Consulta 1 añadiendo la cláusula *DISTINCT* -después del *SELECT*- la lista que se obtiene no contiene apellidos repetidos, es decir el apellido "Silva" aparece sólo una vez en la lista.

Se puede obtener una tabla con los dos apellidos y el nombre de los perfiles especificando las columnas correspondientes:

```
-- Consulta 2
SELECT APELLIDO1, APELLIDO2, NOMBRE FROM PERFIL
```

El resultado de la consulta se muestra a continuación:

APELLIDO1	APELLIDO2	NOMBRE
Silva	Mora	Paula
Martínez	García	Ana
Péréz	González	Jorge
Rodríguez	Campos	María
Silva	Pérez	Juan

Si se quiere obtener todos los datos de la tabla PERFIL (Cuadro 8.1) se utiliza el símbolo * (asterisco):

```
-- Consulta 3
SELECT * FROM PERFIL
```

En algunos casos puede ser útil concatenar el contenido de dos o más columnas, de la tabla sobre la que se realiza la consulta, de modo que el resultado de la consulta se muestre en una única columna. El operador de concatenación es la doble barra vertical "||". Por ejemplo la siguiente consulta SQL:

```
-- Consulta 4
SELECT APELLIDO1 || ' ' || APELLIDO2
FROM PERFIL
```

produce una tabla formada por una única columna con los dos apellidos de cada persona de la tabla PERFIL, separados por un espacio blanco:

APELLIDO1 || ' ' || APELLIDO2
Silva Mora
Martínez García
Péréz González
Rodríguez Campos
Silva Pérez

Se puede modificar la Consulta 4 para renombrar la columna resultante. La siguiente consulta usa palabra clave *AS* para asignar a la columna el nombre APELLIDOS:

```
-- Consulta 5
SELECT APELLIDO1 || ' ' || APELLIDO2 AS APELLIDOS
FROM PERFIL
```

El resultado que se obtiene es el siguiente:

APELLIDOS
Silva Mora
Martínez García
Péréz González
Rodríguez Campos
Silva Pérez

8.2.1. Filtros

Los filtros son condiciones que permiten seleccionar las tuplas (filas) de una tabla y mostrarlas como resultado de la consulta. En SQL hay que añadir una cláusula *WHERE*:

```
SELECT [DISTINCT] select_expr
FROM tabla
WHERE filtro
```

donde `filtro` es una expresión lógica que permite seleccionar las tuplas que la cumplen.

Por ejemplo la siguiente consulta SQL:

```
-- Consulta 6
SELECT APELLIDO1, APELLIDO2, NOMBRE
FROM PERFIL
WHERE APELLIDO1 = 'Silva'
```

selecciona los apellidos y nombre de las personas de la tabla PERFIL que tengan `Silva` como primer apellido:

APELLIDO1	APELLIDO2	NOMBRE
Silva	Mora	Paula
Silva	Pérez	Juan

En las consultas SQL, las constantes literales de tipo cadena de caracteres (por ejemplo, `Silva`) se ponen entre comillas simples. El filtro puede especificar condiciones lógicas sobre varias columnas.

Por ejemplo la siguiente consulta SQL:

```
-- Consulta 7
SELECT *
FROM PERFIL
WHERE (APELLIDO1 = 'Silva') AND (GENERO = 'V')
```

selecciona los datos de las personas de la tabla PERFIL que tengan `Silva` como primer apellido y de género masculino:

DNI	NOMBRE	APELLIDO1	APELLIDO2	GENERO
98789776O	Juan	Silva	Pérez	V

Filtros con test NULL

Los operadores *IS* y *IS NOT* se usan en un filtro para verificar si un campo es o no es nulo, respectivamente. De esta manera, es posible comprobar, por ejemplo, si hay personas en la tabla PERFIL con el género no asignado:

```
-- Consulta 8
SELECT *
FROM PERFIL
WHERE GENERO IS NULL
```

donde *NULL* es una palabra clave. La consulta produce la siguiente información:

DNI	NOMBRE	APELLIDO1	APELLIDO2	GENERO
21345656T	María	Rodríguez	Campos	

Filtros con operador de rango

La sintaxis del operador de rango *BETWEEN* es la siguiente:

```
nombre_columna BETWEEN valor1 AND valor2
```

El operador permite seleccionar las tuplas que tienen el `nombre_columna` incluido en el rango de valores [`valor1`,`valor2`].

Por ejemplo, considérese la tabla VIAJE (Cuadro 8.2) que almacena los viajes registrados por los conductores.

ID	FECHA_SALIDA	HORA_SALIDA	NUM_PLAZA_OFERTADAS	PRECIO_VALOR	PRECIO_MONEDA
1	14/01/15	09:00:00	4	15	euro
2	14/01/15	10:00:00	3	30	euro
3	27/01/15	15:00:00	4	20	euro
4	29/01/15	15:00:00	3	28	euro

ESTADO	CONDUCTOR_ID	ORIGEN_ID	DESTINO_ID	COCHE_MATRICULA
registrado	11111111A	1	3	4952 DDD
cancelado	12345678D	2	3	5000 DXX
registrado	11111111A	3	1	4952 DDD
registrado	12345678D	2	3	5000 DXX

Cuadro 8.2 Estado de la tabla VIAJE (ejemplo).

La siguiente consulta SQL muestra la lista de los viajes cuyo precio es entre 10 y 20 euros, ambos incluidos:

```
-- Consulta 9
SELECT *
FROM VIAJE
WHERE PRECIO_VALOR BETWEEN 10 AND 20
```

Filtros con operador de pertenencia a conjuntos

La sintaxis del operador de pertenencia a conjuntos *IN* es la siguiente:

```
nombre_columna IN (valor1, valor2, ...)
```

La siguiente consulta SQL muestra los viajes que salen el 27 o el 29 de enero de 2015:

```
-- Consulta 10
SELECT *
FROM VIAJE
WHERE FECHA_SALIDA IN ('2015-01-27', '2015-01-29')
```

Obsérvese la diferencia de formatos utilizados para las fecha en las tablas y en las consultas SQL. En particular, en las consultas siempre se utiliza el formato *AAAA-MM-DD* que corresponde al formato de almacenamiento en la base de datos.

Filtros con test de patrón

Cuando se necesita seleccionar las tuplas de una tabla en base a un patrón que tenga que cumplir una columna de tipo cadena de caracteres, se utiliza la cláusula *LIKE* y el carácter porcentaje (`%`). Este carácter es un comodín que se utiliza tanto como prefijo así como sufijo para referirse a cualquier cadena que tenga cero o más caracteres.

Por ejemplo, se quiere obtener la lista de personas de la tabla PERFIL cuyo primer apellido incluye las letras `il`. Se puede especificar la siguiente consulta SQL:

```
-- Consulta 11
SELECT *
FROM PERFIL
WHERE APELLIDO1 LIKE '%il%'
```

8.2.2. Ordenación

Para mostrar ordenado el resultado de una consulta se utiliza la cláusula *ORDER BY*:

```
SELECT [DISTINCT] select_expr
FROM tabla
WHERE filtro
ORDER BY nombre_columna [ASC| DESC]
```

Esta cláusula permite ordenar el conjunto de resultados de forma ascendente (*ASC*) o descendente (*DESC*), por una o varias columnas. Si no se indica la forma de ordenación, por defecto es ascendente.

La siguiente consulta SQL genera la lista de las personas de la tabla PERFIL ordenada alfabéticamente por los dos apellidos:

```
-- Consulta 12
SELECT *
FROM PERFIL
ORDER BY APELLIDO1 ASC, APELLIDO2 ASC
```

8.2.3. *Consultas de resumen*

En SQL se pueden generar consultas más complejas que resuman cierta información, extrayendo información calculada de varios conjuntos de tuplas. Para especificar consultas de resumen, hay que utilizar funciones predefinidas (se llaman *funciones de columnas*) que convierten un conjunto de tuplas en una información simple cuya representación es un cálculo. SQL ofrece las siguientes funciones:

- `COUNT(*)`: cuenta el número de tuplas de una tabla
- `COUNT(col)`: cuenta el número de valores no nulos de la columna `col`
- `SUM(expr)`: suma los valores indicados en `expr`
- `AVG(expr)`: calcula la media aritmética de los valores en `expr`
- `MIN(expr)`: calcula el mínimo de los valores en `expr`
- `MAX(expr)`: calcula el máximo de los valores en `expr`

A continuación, se consideran algunos ejemplos. La consulta SQL:

```
-- Consulta 13
SELECT COUNT(*) FROM PERFIL
```

cuenta el número de tuplas almacenadas en la tabla PERFIL:

COUNT(*)
5

Si se especifica una columna como parámetro de la función `count`, sólo se cuentan las tuplas que no tienen valor nulo en la columna seleccionada:

```
-- Consulta 14
SELECT COUNT(GENERO) FROM PERFIL
```

En este caso el resultado producido es 4 -véase el Cuadro 8.1.

La siguiente consulta SQL calcula la media de los precios de los viajes almacenados en la tabla VIAJE:

```
-- Consulta 15
SELECT AVG(PRECIO_VALOR)
FROM VIAJE
```

y produce el resultado:

AVG(VIAJE.PRECIO_VALOR)
23,25

Si se quiere obtener el precio de los viajes más baratos ofrecidos por un conductor en concreto, por ejemplo del conductor con DNI `12345678D`, hace falta añadir un filtro (cláusula *WHERE*):

```
-- Consulta 16
SELECT MIN(PRECIO_VALOR)
FROM VIAJE
WHERE CONDUCTOR_DNI = '12345678D'
```

En este caso la consulta produce 28 (euros) como resultado -véase el Cuadro 8.2.

8.2.4. Consultas con agrupaciones

Este tipo de consultas permite realizar agrupaciones de tuplas. Una agrupación de tuplas es un conjunto de tuplas que tienen una o varias columnas con el mismo valor. Hay que utilizar la cláusula *GROUP BY*:

```
SELECT [DISTINCT] select_expr
FROM tabla
GROUP BY expr
```

donde `expr` suele ser una columna -o conjunto de columnas separadas por comas- que representa el criterio de agrupación. Normalmente la agrupación se utiliza en consultas que utilizan además las funciones de columnas en la cláusula *SELECT* (consultas de resumen).

Por ejemplo, la siguiente consulta SQL produce una lista con fechas y, para cada fecha, el número total de viajes que tienen dicha fecha de salida:

```
-- Consulta 17
SELECT FECHA_SALIDA, COUNT(*) AS NUMERO_DE_VIAJES
FROM VIAJE
GROUP BY FECHA_SALIDA
```

El criterio de agrupación es la columna FECHA_SALIDA y la función predefinida es el contador de tuplas COUNT(*). El resultado que se obtiene considerando la tabla VIAJE del Cuadro 8.2 es la tabla siguiente:

FECHA_SALIDA	NUMERO_DE_VIAJES
14/01/15	2
27/01/15	1
29/01/15	1

Obsérvese que se ha seleccionado también el nombre de la columna por la cual se agrupa (FECHA_SALIDA) y que la segunda columna ha sido renombrada utilizando la palabra clave *AS*.

Definir en la cláusula *SELECT* tanto columnas como funciones de columnas produce un error si no se incluye también la cláusula *GROUP BY*, donde el criterio de agrupación sea el conjunto de las columnas que aparecen en el *SELECT*.

Por ejemplo la siguiente consulta SQL **no** es correcta:

```
SELECT FECHA_SALIDA, COUNT(*) AS NUMERO_DE_VIAJES
FROM VIAJE
```

El DBMS avisará con un mensaje de error del tipo:

> *Not in aggregate function or group by clause...*

8.2.5. Uso de filtros después de agrupar

Se pueden aplicar filtros al resultado de una consulta con agrupaciones. La cláusula utilizada para la especificación de este tipo de filtros es *HAVING*. No es posible utilizar la cláusula *WHERE*, ya que los filtros definidos de esa manera se ejecutan antes de realizar la agrupación. La sintaxis SQL es la siguiente:

```
SELECT [DISTINCT] select_expr
FROM tabla
GROUP BY expr
HAVING filtro_grupos
```

donde `filtro_grupos` es una expresión lógica que permite seleccionar las tuplas de la tabla producida por la agrupación que cumplen la condición.

Por ejemplo, se quiere reducir la lista producida por la consulta 17 a las fechas de salida cuyo número total de viajes es mayor que uno. La nueva consulta se especifica añadiendo un filtro *HAVING* a la consulta 17:

```
-- Consulta 18
SELECT FECHA_SALIDA, COUNT(*) AS NUMERO_DE_VIAJES
FROM VIAJE
GROUP BY FECHA_SALIDA
HAVING COUNT(*) > 1
```

Obsérvese que, en el filtro, se ha utilizado la función de columnas COUNT(*): la utilización del alias NUMERO_DE_VIAJES asignado a la segunda columna produciría un mensaje de error por el DBMS.

La tabla generada por la consulta 18, considerando la tabla VIAJE del Cuadro 8.2, es la siguiente:

FECHA_SALIDA	NUMERO_DE_VIAJES
14/01/15	2

8.2.6. Subconsultas

Las subconsultas se utilizan para realizar filtrados con los datos de otra consulta. Estos filtros pueden ser aplicados tanto en cláusulas *WHERE* para filtrar tuplas, como en cláusulas *HAVING* para filtrar grupos.

Por ejemplo, se quiere obtener todos los datos de los viajes más baratos:

```
-- Consulta 19
SELECT *
FROM VIAJE
WHERE PRECIO_VALOR = ( -- subconsulta
                       SELECT MIN(PRECIO_VALOR)
                       FROM VIAJE
                     )
```

La sentencia *SELECT ... FROM ...* entre paréntesis es la subconsulta que calcula el precio más barato y produce el siguiente resultado:

MIN(VIAJE.PRECIO_VALOR)
15

es decir, un número que permite filtrar en la consulta principal *SELECT ... FROM ... WHERE ...* las tuplas cuyo valor de la columna PRECIO_VALOR es igual a 15.

El resultado final de la consulta 19 sobre la tabla VIAJE del Cuadro 8.2, produce la primera tupla de la tabla.

8.2.7. *Consultas parametrizadas*

Los filtros (*WHERE*, *HAVING*) permiten seleccionar las tuplas cuyas columnas verifican condiciones lógicas. Las condiciones lógicas consideradas en los ejemplos anteriores (consultas 6, 7, 9, 10, 16 y 18) se concretan en expresiones de comparación entre columnas, o resultados de funciones predefinidas sobre columnas, y constantes.

Es posible crear consultas con parámetros (es decir, variables) en lugar de constantes: en este caso cuando se ejecute la consulta aparecerá una ventana *Entrada de parámetros* que solicita un valor para los parámetros definidos en la consulta.

La sintaxis para la definición de parámetros es la siguiente:

```
:nombreParametro
```

Por ejemplo, se quiere parametrizar la consulta 18 de manera que cada vez que el usuario ejecute la consulta, el DBMS le pida el umbral del número total de viajes para obtener la lista de las fechas de salida que tienen un número total de viaje superior al umbral. La consulta 18 se modifica como sigue:

```
-- Consulta 18bis
SELECT FECHA_SALIDA, COUNT(*) AS NUMERO_DE_VIAJES
FROM VIAJE
GROUP BY FECHA_SALIDA
HAVING COUNT( * ) > :NumeroMinimo
```

donde `NumeroMinimo` es el parámetro.

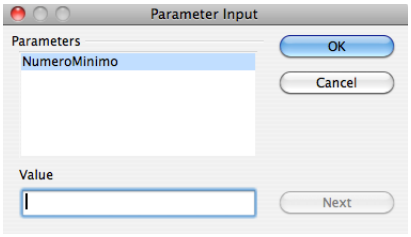


Figura 8.2 Ejecución de la consulta 18bis en Open Office Base (2015): ventana *Entrada de parámetros*.

Cuando se ejecute la consulta, el DBMS pide al usuario introducir el valor para el parámetro (véase la Figura 8.2), para poder obtener el resultado de la consulta ajustado al valor introducido.

8.3. Consultas multitabla

Una consulta multitabla es una consulta que involucra más de una tabla para obtener información de la base de datos. Se aprovechan las columnas relacionadas (clave primaria y clave externa) de las tablas para unirlas.

La sintaxis SQL no cambia con respecto a la sintaxis utilizada para las consultas sobre una única tabla:

```
SELECT [DISTINCT] select_expr
FROM tablas
WHERE filtro
GROUP BY expr
HAVING filtro_grupos
```

la diferencia es que en este caso la cláusula *FROM* incluye una lista de tablas, separadas por comas, y el filtro de la cláusula *WHERE* tendrá que especificar las relaciones entre las tablas.

La operación que une tablas considerando las relaciones que existen entre ellas se llama *join* y consiste en:

JOIN = PRODUCTO CARTESIANO + FILTRO.

8.3.1. Producto cartesiano

El producto cartesiano de dos tablas genera todas las combinaciones posibles entre las filas de las tablas. Por ejemplo, la siguiente consulta SQL:

```
-- Consulta 20
SELECT *
FROM VIAJE,COCHE
```

es el producto cartesiano de la tabla VIAJE y la tabla COCHE.

El Cuadro 8.3 muestra un ejemplo de estado de la tabla COCHE y el Cuadro 8.4 el resultado de la consulta 20 sobre la tabla VIAJE (véase Cuadro 8.2) y la tabla COCHE (Cuadro 8.3).

MATRICULA	MARCA	MODELO	NUM_PLAZAS	VIN	SEGURO_VIGENTE
4231 KVF	Porche	911 4S	1	TRQIYU4SX8A789123	☑
4952 DDD	Volkswagen		5	WVWZZZ3CZ9E334557	☑
5000 DXX	Skoda	Fabia	5		☑

Cuadro 8.3 Estado de la tabla COCHE (ejemplo).

Seleccionando todas las columnas (*) la tabla que se obtiene tiene las columnas de las dos tablas. El número de las tuplas obtenidas es igual al producto de las tuplas en la primera tabla (4) y en la segunda tabla (3): cada tupla de la tabla VIAJE se combina con cada tupla de la tabla COCHE.

8.3.2. JOIN

Aparentemente la consulta 20 no tiene mucha utilidad, sin embargo, si se aplica un filtro al producto cartesiano que escoja solo aquellas tuplas en las que la columna COCHE_MATRICULA de VIAJE coincida con la columna MATRICULA de COCHE:

ID	FECHA_SALIDA	HORA_SALIDA	NUM_PLAZA_OFERTADAS	PRECIO_VALOR	PRECIO_MONEDA	ESTADO	CONDUCTOR_ID	ORIGEN_ID	DESTINO_ID	COCHE_MATRICULA	MATRICULA	MARCA	MODELO	NUM_PLAZAS	VIN	SEGURO_VIGENTE
1	14/01/15	09:00:00	4	15	euro	registrado	11111111A	1	3	4952 DDD	4231 KVF	Porche	911 4S	1	TRQIYU4SX8A789123	☑
1	14/01/15	09:00:00	4	15	euro	registrado	11111111A	1	3	4952 DDD	4952 DDD	Volkswagen		5	WVWZZZ3CZ9E334557	☑
1	14/01/15	09:00:00	4	15	euro	registrado	11111111A	1	3	4952 DDD	5000 DXX	Skoda	Fabia	5		☑
2	14/01/15	10:00:00	3	30	euro	cancelado	12345678D	2	3	5000 DXX	4231 KVF	Porche	911 4S	1	TRQIYU4SX8A789123	☑
2	14/01/15	10:00:00	3	30	euro	cancelado	12345678D	2	3	5000 DXX	4952 DDD	Volkswagen		5	WVWZZZ3CZ9E334557	☑
2	14/01/15	10:00:00	3	30	euro	cancelado	12345678D	2	3	5000 DXX	5000 DXX	Skoda	Fabia	5		☑
3	27/01/15	15:00:00	4	20	euro	registrado	11111111A	3	1	4952 DDD	4231 KVF	Porche	911 4S	1	TRQIYU4SX8A789123	☑
3	27/01/15	15:00:00	4	20	euro	registrado	11111111A	3	1	4952 DDD	4952 DDD	Volkswagen		5	WVWZZZ3CZ9E334557	☑
3	27/01/15	15:00:00	4	20	euro	registrado	11111111A	3	1	4952 DDD	5000 DXX	Skoda	Fabia	5		☑
4	29/01/15	15:00:00	3	28	euro	registrado	12345678D	2	3	5000 DXX	4231 KVF	Porche	411 4S	1	TRQIYU4SX8A789123	☑
4	29/01/15	15:00:00	3	28	euro	registrado	12345678D	2	3	5000 DXX	4952 DDD	Volkswagen		5	WVWZZZ3CZ9E334557	☑
4	29/01/15	15:00:00	3	28	euro	registrado	12345678D	2	3	5000 DXX	5000 DXX	Skoda	Fabia	5		☑

Cuadro 8.4 Producto cartesiano de las tablas VIAJE y COCHE.

```
-- Consulta 21
SELECT *
FROM VIAJE, COCHE
WHERE VIAJE.COCHE_MATRICULA = COCHE.MATRICULA
```

se obtienen los datos de los viajes y de los coches utilizados en los viajes (véase el resultado de la consulta en el Cuadro 8.5).

ID	FECHA_SALIDA	HORA_SALIDA	NUM_PLAZA_OFERTADAS	PRECIO_VALOR	PRECIO_MONEDA	ESTADO	CONDUCTOR_ID	ORIGEN_ID	DESTINO_ID	COCHE_MATRICULA	MATRICULA	MARCA	MODELO	NUM_PLAZAS	VIN	SEGURO_VIGENTE
1	14/01/15	09:00:00	4	15	euro	registrado	11111111A	1	3	4952 DDD	4952 DDD	Volkswagen		5	WVWZZZ3CZ9E334557	☑
2	14/01/15	10:00:00	3	30	euro	cancelado	12345678D	2	3	5000 DXX	5000 DXX	Skoda	Fabia	5		☑
3	27/01/15	15:00:00	4	20	euro	registrado	11111111A	3	1	4952 DDD	4952 DDD	Volkswagen		5	WVWZZZ3CZ9E334557	☑
4	29/01/15	15:00:00	3	28	euro	registrado	12345678D	2	3	5000 DXX	5000 DXX	Skoda	Fabia	5		☑

Cuadro 8.5 Join de las tablas VIAJE y COCHE.

Mediante la consulta 21 se obtiene la información relacionada a las dos tablas: las dos columnas que se igualan en la cláusula *WHERE* son la clave externa de la tabla VIAJE (COCHE_MATRICULA) y la clave primaria de la tabla COCHE (MATRICULA).

Cuando se definen consultas que involucran dos o varias tablas es una buena costumbre especificar la pertenencia de una columna a una tabla utilizando la sintaxis:

```
tabla.columna
```

Esta notación es necesaria cuando hay columnas con el mismo nombre en tablas diferentes.

Las operaciones producto cartesiano y *join* que se han descrito en esta sección son operaciones definidas en la versión SQL1.

En la versión SQL2 existen diferentes tipos de consultas multitabla, entre ellos:

- la composición cruzada que corresponde al producto cartesiano y
- el join de equivalencia (*INNER JOIN*) que corresponde al join.

Sin embargo, aunque produzcan el mismo resultado de las correspondientes operaciones definidas en SQL1, se especifican con una sintaxis diferente.

En particular, la sintaxis del *INNER JOIN* es la siguiente:

```
SELECT [DISTINCT] select_expr
FROM tabla1 INNER JOIN tabla2 ON tabla1.clave_externa = tabla2.clave_primaria
```

Por ejemplo, la siguiente consulta utiliza el *INNER JOIN* y produce el mismo resultado (es decir, el Cuadro 8.5) de la consulta 21:

```
-- Consulta 22
SELECT *
FROM VIAJE INNER JOIN COCHE ON VIAJE.COCHE_MATRICULA = COCHE.MATRICULA
```

8.4. Consultas con tablas derivadas

Una consulta SQL produce siempre una tabla (virtual o derivada), por lo tanto es posible utilizar una consulta *SELECT...* en la cláusula *FROM* de otra consulta en lugar de nombres de tablas.

Por ejemplo la siguiente consulta SQL calcula el número de coches que han sido utilizados en más de un viaje:

```
-- Consulta 23
SELECT COUNT(*) AS NUMERO_COCHES
FROM
  ( -- subconsulta
    SELECT VIAJE.COCHE_MATRICULA, COUNT(*) AS NUMERO_VIAJES
    FROM VIAJE
    GROUP BY VIAJE.COCHE_MATRICULA
  ) AS TABLA_DERIVADA
WHERE TABLA_DERIVADA.NUMERO_VIAJES > 1
```

La subconsulta *SELECT...FROM...GROUP BY* en la cláusula *FROM* de la consulta 23 produce una tabla -renombrada TABLA_DERIVADA- que extrae, para cada coche, el número de matrícula y el número total de viajes en que ha sido utilizado -columna renombrada NUMERO_VIAJES.

Considerando las tabla VIAJE del Cuadro 8.2, la subconsulta produce la tabla derivada:

MATRICULA	NUMERO_VIAJES
4952 DDD	2
5000 DXX	2

La tabla es utilizada por la consulta 23 para contar las tuplas cuyo valor de la columna NUMERO_VIAJES es superior a 1. El resultado final calculado por la consulta 23 es dos (en el ejemplo, todas las tuplas que se obtienen con la subconsulta cumplen con la condición).

8.5. Casos prácticos

Ej. 1 — Considérese la base de datos *CarShare* de Figura 8.1 y el siguiente ejemplo de datos que podrían almacenar las tablas VIAJE, RESERVA y LUGAR:

VIAJE ../..					
ID	FECHA_SALIDA	HORA_SALIDA	NUM_PLAZA_OFERTADAS	PRECIO_VALOR	PRECIO_MONEDA
1	14/01/15	09:00:00	4	15	euro
2	14/01/15	10:00:00	3	30	euro
3	27/01/15	15:00:00	4	20	euro
4	29/01/15	15:00:00	3	28	euro

../.. VIAJE				
COCHE_MATRICULA	ESTADO	CONDUCTOR_ID	ORIGEN_ID	DESTINO_ID
4952 DDD	registrado	11111111A	1	3
5000 DXX	cancelado	12345678D	2	3
4952 DDD	registrado	11111111A	3	1
5000 DXX	registrado	12345678D	2	3

RESERVA		
PERFIL_DNI	VIAJE_ID	NUM_PLAZAS_RESERVADAS
12345127G	1	1
21345656T	1	1
12345678D	3	1

LUGAR				
ID	CIUDAD	CALLE	NUMERO	CODIGO_POSTAL
1	Zaragoza	Academia General Militar	12	50015
2	Zaragoza	María Agustín	8	50003
3	Madrid	Castellana	44	28046

Escribir el código SQL para obtener la siguiente información:

a) El número de viajes que tengan fecha de salida en 2015 y que hayan tenido como ciudad de origen ciudades cuyo nombre empiece por Z.
b) El identificador de cada viaje que tenga fecha de salida el mes de enero de 2015 y que haya tenido reservas, y el correspondiente número total de plazas reservadas.
c) La matrícula y el número de viajes realizados por los coches que hayan efectuado más de 10 viajes en 2015.
d) El DNI de los conductores que hayan actuado en 2015 también como pasajeros, es decir que hayan realizado alguna reserva en 2015. Por ejemplo, véase en las tablas anteriores el caso del cliente con DNI 12345678D.

Ej. 2 — La compañía *VolarFeliz* dispone de una base de datos para la gestión de la venta de asientos en sus vuelos diarios. La base de datos tiene la estructura de Figura 8.3.
Para facilitar la comprensión de la base de datos se presenta a continuación un ejemplo de datos que podría almacenar:

AVION		
ID	MODELO	CAPACIDAD
1	boeing	100
2	boeing	150

RUTA		
ORIGEN	DESTINO	DISTANCIA
MAD	BCN	500
MAD	ZGZ	250

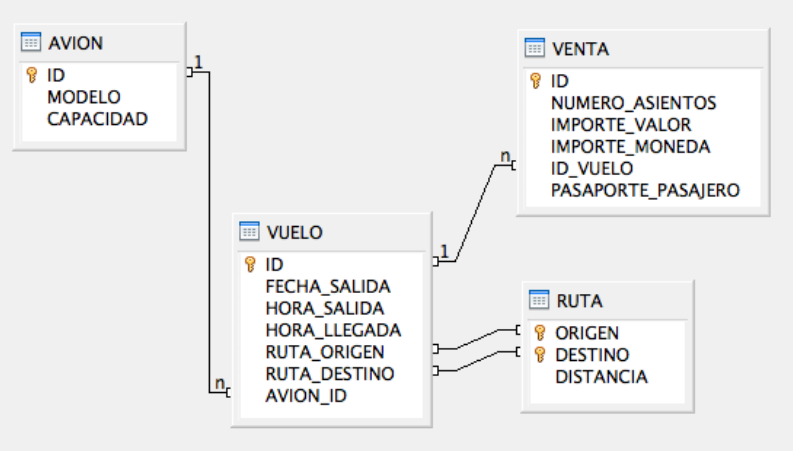


Figura 8.3 Estructura de la base de datos de *VolarFeliz*.

VUELO						
ID	FECHA_SALIDA	HORA_SALIDA	HORA_LLEGADA	RUTA_ORIGEN	RUTA_DESTINO	AVION_ID
11	15/10/14	14:00:00	17:00:00	MAD	BCN	1
22	15/10/14	15:00:00	17:00:00	MAD	ZGZ	2

VENTA					
ID	NUMERO_ASIENTOS	IMPORTE_VALOR	IMPORTE_MONEDA	ID_VUELO	PASAPORTE_PASAJERO
111	2	500	euro	11	AK1234567
221	1	200	euro	22	GH0983456

Obsérvese que en la tabla VENTA, la columna NUMERO_ASIENTOS se refiere al número de asientos vendidos en una transacción de venta.
Escribir el código SQL para obtener la siguiente información:

a) La hora de salida, hora de llegada, origen y distancia de todos los vuelos con fecha de salida '15/10/14' y con aeropuerto de destino Madrid (código MAD).
b) El identificador de los vuelos junto con el número de asientos vendidos.
c) La capacidad máxima de los aviones de la compañía.
d) Hora de salida, hora de llegada, origen y destino de los vuelos que utilizan el/los avión(es) de mayor capacidad de la compañía.

Ej. 3 — La biblioteca *Crecer Leyendo* dispone de una base de datos relacional para la gestión de sus libros. Una parte de la estructura de esta base de datos se muestra en Figura 8.4.

Para facilitar la comprensión de la base de datos se presenta a continuación un ejemplo de datos que podría almacenar:

AUTOR	
NOMBRE	ISBN_LIBRO
Antonio Fernández García	123-3-543-39511-6
Mar Fernández	434-1-543-45309-9
Luis Fernández	434-1-543-45309-9

EDICION		
ID	ISBN_LIBRO	EDICION
1	123-3-543-39511-6	1
2	123-3-543-39511-6	1
3	434-1-543-45309-9	2

LIBRO		
ISBN	TITULO	AÑO
123-3-543-39511-6	Historia universal contemporánea	2000
434-1-543-45309-9	Poesía	1990

Escribir el código SQL para obtener la siguiente información:

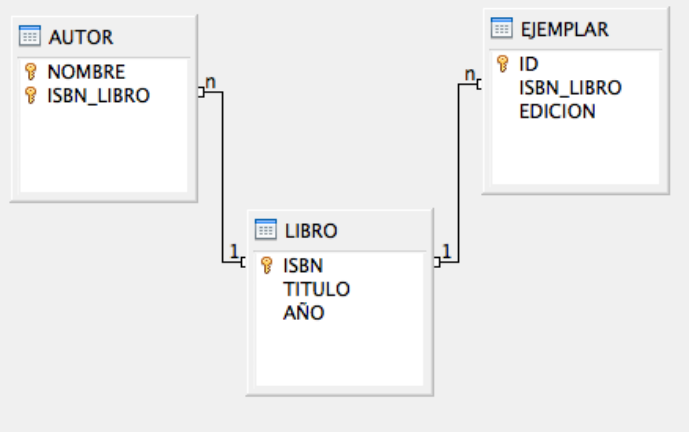


Figura 8.4 Estructura de la base de datos de *Crecer Leyendo*.

a) El año del libro más antiguo que se encuentra en la biblioteca.
b) El número de ejemplares del libro más antiguo de la biblioteca.
c) El título de los libros escritos a partir del 1 de enero de 2000 y el número de ejemplares en la biblioteca de cada uno.
d) El ISBN y el título de los libros que tienen al menos un autor que contenga en su nombre *Fernández*. Obsérvese que un libro puede tener varios autores con ese apellido.

Ej. 4 — La empresa *Memorabilia S.A.* dispone de una base de datos relacional para la gestión de las ventas de artículos deportivos a sus clientes. Una parte de la estructura de la base de datos se muestra en Figura 8.5.

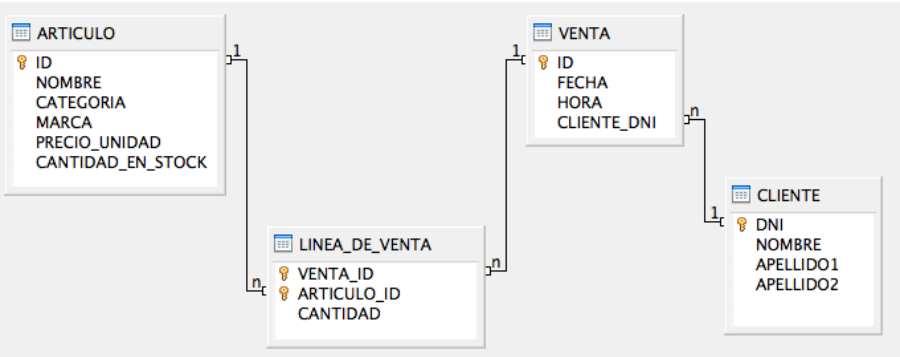


Figura 8.5 Estructura de la base de datos de *Memorabilia*.

Para facilitar la comprensión de la base de datos se presenta a continuación un ejemplo de datos que podría almacenar:

ARTICULO						
ID	NOMBRE	CATEGORIA	MARCA	TALLA	PRECIO_UNIDAD	CANTIDAD_EN_STOCK
1	gafas	natación	Speedo	U	15	5
2	casco seguridad	equitación	Fouganza	L	20	50

CLIENTE			
DNI	NOMBRE	APELLIDO1	APELLIDO2
2255667A	Juan	Sanchez	Pérez
1199666K	María	Silva	Jimenez

VENTA			
ID	FECHA	HORA	CLIENTE_DNI
1	19/01/15	08:45:00	2255667A
2	24/01/15	09:00:00	1199666K

LINEA_DE_VENTA		
VENTA_ID	ARTICULO_ID	CANTIDAD
1	1	1
2	2	2
2	1	1

Escribir el código SQL para obtener la siguiente información:

a) El número de ventas realizadas en 2015 a clientes que se llaman *María*.
b) El identificador de las ventas realizadas en el mes de enero de 2015 en las que se hayan vendidos dos o más artículos diferentes, y el correspondiente número total de artículos diferentes vendidos. En el ejemplo, en la venta con identificador 2 se han vendidos dos artículos diferentes. La venta con identificador 1 solo incluye un artículo, por lo que no debería aparecer en el resultado de la consulta.
c) El nombre de los artículos vendidos en el mes de enero de 2015 y la cantidad que queda en stock.
d) El nombre del artículo más caro disponible en stock.

Capítulo 9
Herramientas de inteligencia de negocios

Resumen Este capítulo repasa los tipos de sistemas de información en las organizaciones, haciendo hincapié en los sistemas informacionales. Se introducen las herramientas de *Business Intelligence* (inteligencia de negocios) con sus principales componentes: los almacenes de datos y las herramientas OLAP.

9.1. Tipos de sistemas de información

No todas las organizaciones son iguales por lo que tampoco todos los sistemas de información son iguales. Cada sistema de información refleja una determinada cultura organizacional. Sin embargo la mayoría comparten ciertas características que nos permiten diferentes clasificaciones.

9.1.1. Sistemas de información según niveles de decisión

Tradicionalmente, la clasificación de los sistemas de información se ha venido realizando según el tipo de decisiones a las que dan soporte. Hay que recordar que la clasificación de las decisiones en el modelo clásico piramidal tiene tres niveles - operativo, táctico y estratégico - (Chopo et al, 2011), tal como se puede apreciar en la Figura 9.1.

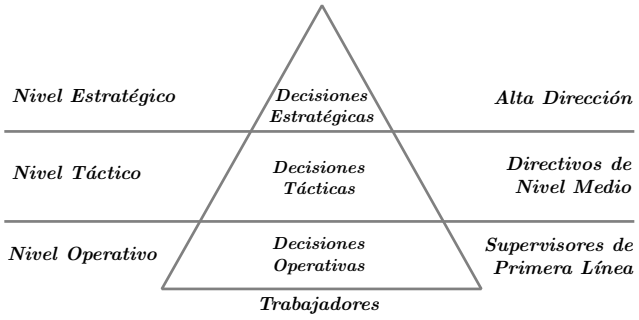


Figura 9.1 Tipos de decisiones: clasificación por niveles.

En base a estos niveles, se puede hablar de: sistemas de información (SI) circunscritos al núcleo de operaciones, a la línea media o al ápice estratégico.

SI circunscritos al núcleo de operaciones

Son sistemas de información que recolectan, almacenan, modifican y recuperan toda la información generada por las transacciones producidas en una organización. Este tipo de sistemas de información se denomina **TPS - Transaction Processing System** o sistemas de procesamiento de transacciones. Se trata de sistemas que dan soporte a decisiones operacionales rutinarias y la realización del seguimiento/control del flujo de transacciones de la organización.

Inicialmente los sistemas TPS se basaban en el procesamiento por lotes. Hoy en día es habitual el procesamiento en tiempo real, siendo estos sistemas de tipo **OLTP - On Line Transaction Processing** (procesamiento de transacciones en línea). La tecnología OLTP se utiliza en innumerables aplicaciones, como en gestión de pedidos, banca electrónica, comercio electrónico, supermercados o industria.

Este tipo de sistemas funcionan sobre grandes bases de datos (gestionadas a través de un DBMS) con muchos usuarios (del orden de cientos o miles) realizando transacciones de manera concurrente. En la actualidad el modelo lógico relacional sigue siendo el más utilizado en este tipo de sistemas, siendo la mayoría de los DBMS implantados relacionales. Se aprovechan así las ventajas inherentes del modelo relacional subyacente como la *integridad de los datos* o *independencia entre los datos y las aplicaciones*. Obviamente, en el mercado se encuentran también otros DBMS como los orientados a objetos, en red o jerárquicos. A todos estos tipos de bases de datos se les suele denominar también bases de datos tradicionales o transaccionales.

> Una *transacción*, en este contexto, es un evento que genera o modifica los datos que se encuentran almacenados en el sistema de información (acontecimiento de negocio).

En un entorno empresarial las transacciones podrían tener correspondencia con una actividad rutinaria de la empresa como una reserva de un viaje, una facturación, cobro, pago, etc.

A nivel de la base de datos, una transacción se entiende como un programa en acción o proceso que incluye una o más operaciones de acceso a la base de datos, sea(n) de recuperación (lectura: p. ej. la ejecución de una sentencia SQL SELECT), inserción, eliminación o modificación (escritura: p. ej. la ejecución de sentencias SQL INSERT, DELETE o UPDATE). En este contexto son muy importantes las capacidades que proporciona el DBMS utilizado en cuanto a *control de concurrencia* y *recuperación*. Para un correcto procesamiento de las transacciones en un sistema TPS, el DBMS subyacente debería cumplir las siguientes **propiedades ACID** (acrónimo de *Atomicity*, *Consistency*, *Isolation* and *Durability*):

- **atomicidad**: una transacción es tratada como una unidad atómica de procesamiento: o se realiza por completo o no se realiza en absoluto. Por ejemplo una transferencia bancaria entre dos cuentas que implique un cargo en una cuenta y un ingreso en otra, no puede quedarse a medias, solo con registrar el cargo.
- **consistencia**: la ejecución de una transacción lleva la base de datos de un estado consistente a otro estado consistente (válido).
- **aislamiento**: la ejecución de una transacción no interfiere con otras transacciones que se ejecuten concurrentemente.

- **permanencia**: una vez realizada la transacción, ésta persistirá y no se podrá deshacer aunque falle el sistema. Ante un fallo, el DBMS debe asegurar la recuperación de los datos.

SI circunscritos a la línea media

Son sistemas de información que deben proporcionar información para la toma de decisiones a nivel administrativo. El tipo de decisiones a este nivel son menos rutinarias, menos estructuradas que en el nivel operacional. En este nivel se encuentran:

- los sistemas **MIS - Management Information Systems** o sistemas de información gerencial
- los sistemas **DSS - Decision Support Systems** o sistemas de apoyo a la toma de decisiones

Los sistemas MIS dan soporte a la toma de decisiones normalmente mediante la realización, con cierta periodicidad, de informes. Se relacionan con las decisiones regulares y estructuradas de este nivel. El tipo de información que proporcionan suele tener relación con el desempeño actual de la organización y se nutren de los datos resumidos que les proporcionan los sistemas TPS de la organización.

Algunos ejemplos de decisiones a las que pueden dar apoyo los sistemas MIS son la administración de ventas, el control de inventarios, la elaboración del presupuesto anual.

Los sistemas DSS dan soporte a la toma de las decisiones menos rutinarias del nivel medio de la organización. Se centran en problemas únicos, que cambian rápidamente, para los cuales el procedimiento para llegar a una solución puede no ser totalmente predefinido de antemano. Se suelen apoyar en bases de datos optimizadas para el análisis de grandes volúmenes de datos para facilitar su análisis y hallar las causas, raíces de los problemas/pormenores de la organización.

SI circunscritos al ápice estratégico

Se trata de soluciones de alto nivel que permiten visualizar, de una forma rápida y fácil, el estado de una determinada situación empresarial, presente o pasada, con el fin de detectar por ejemplo anomalías o oportunidades y apoyar la toma de decisiones sobre la marcha.

Los sistemas de este tipo se denominan **ESS - Executive Support Systems** o sistemas de apoyo para la dirección. Son herramientas software diseñadas para incorporar información interna (a través de sistemas MIS y DSS) y externa a la organización y proveer a sus usuarios de un acceso *sencillo* a esa información. Realizan la filtración, compresión, seguimiento de datos críticos y la visualización de datos de mayor importancia a los ejecutivos. Cada vez más, estos sistemas incluyen herramientas de análisis para analizar tendencias, realizar pronósticos y permitir indagar en los datos a un mayor nivel de detalle.

Una herramienta cada vez más popular utilizada en la toma de decisiones a este nivel es el **Cuadro de Mando Integral - CMI**, también conocido como Balanced Scorecard (BSC) o dashboard. Los CMI están orientados al seguimiento de indicadores. A modo de ejemplo, cabe mencionar el CMI del SALE -Sistema de Apoyo Logístico del Ejército- (Díaz Osto, 2010), que engloba el seguimiento de varios indicadores (p. ej., *disponibilidad operativa del material*, *tiempo medio de mantenimiento*, etc.) con código de colores para sus valores: azul (muy bien), verde (bien/aceptable), amarillo (mejorable) y rojo (manifiestamente mejorable).

9.1.2. Sistemas de información para enlazar la organización

Conseguir que todos los diferentes tipos de sistemas de una empresa trabajen juntos ha demostrado ser un gran desafío. Una solución ha sido implementar aplicaciones empresariales: sistemas que abarcan las áreas funcionales de la empresa, se centran en la ejecución de procesos de negocio de un extremo a otro de la empresa, e incluyen todos los niveles de gestión.

Hoy en día las empresas utilizan sistemas empresariales conocidos como sistemas **ERP** (Enterprise Resource Planning). Los sistemas ERP integran los procesos del negocio en un único sistema software, dando soporte a las diferentes áreas funcionales de la empresa: fabricación, producción, finanzas y contabilidad, ventas y marketing, recursos humanos. Los sistemas ERP adoptan en general una **estructura modular** que a menudo guarda similitud con la división por áreas funcionales en una empresa. Sin embargo, cada proveedor de ERP define la modularización de su solución.

La información, que en soluciones pasadas podía estar fragmentada en diferentes sistemas, se almacena en un único repositorio, con la ventaja de poder ser utilizada luego desde las diferentes partes del negocio, en torno al concepto de un **dato único** y una **explotación múltiple** (p. ej. si una factura ha sido registrada en el módulo de clientes, ya no será necesario volver a introducirla en el módulo de contabilidad y finanzas que podrá utilizarla directamente).

Los sistemas ERP son soluciones genéricas con cierta capacidad de **parametrización** para adaptarlo a las necesidades concretas de la empresa que lo implante. Aspectos susceptibles de parametrizar en una solución pueden ser: la estructura organizativa y física de la empresa, flujo de procesos, automatización de tareas, gestión de alertas, reglas de negocio o estructura fiscal entre otros (Gómez and Suárez, 2011).

Ejemplo de uso de un sistema ERP (Laudon and Laudon, 2012): cuando un cliente hace un pedido, los datos del pedido fluyen automáticamente a las otras partes de la empresa que afecta. La transacción activa la orden al almacén para recoger los productos pedidos y programar el envío. El almacén informa a la fábrica para reponer lo que se ha agotado. El departamento de contabilidad es notificado para enviar al cliente una factura. Representantes del servicio al cliente puede realizar el seguimiento del progreso de la orden para informar a los clientes sobre el estado de sus pedidos. Los administradores pueden utilizar la información de toda la empresa para tomar decisiones más precisas y oportunas acerca de las operaciones diarias y la planificación a largo plazo. El ejemplo pone de manifiesto una característica de los sistemas ERP: la visión de gestión por procesos (flujos de actividades) que el punto de vista desde quién hace las cosas hacia en qué, cómo, para qué y para quién se hacen las cosas.

Otro tipo de grandes sistemas de información que se pueden encontrar en las empresas son los sistemas **SCM** (Supply Chain Management) para la gestión de la cadena de suministro. Se trata de sistemas inter-organizacionales que implican tanto a proveedores como a empresas compradoras, distribuidores, empresas de logística. A través de estos sistemas se comparte información acerca de los pedidos, la producción, los niveles de inventario y la entrega de productos y servicios para que puedan producir y entregar productos y servicios de manera eficiente.

La gestión de la relación con el cliente se lleva a cabo a través de sistemas **CRM** (Customer Relationship Management). Los sistemas CRM proporcionan información para coordinar todo el proceso de negocio que tiene que ver con los clientes: ventas, marketing, servicios para optimizar los ingresos, la satisfacción del cliente y retención de clientes. Este tipo de información ayuda a las empresas a:

- identificar, atraer y mantener los clientes más rentables.
- proveer un mejor servicio a sus clientes.
- mejorar las ventas.

Los ERP de nueva generación ofrecen en la actualidad de forma integrada algunas de las funcionalidades que tradicionalmente estaban siendo soportadas por aplicaciones CRM o SCM.

9.1.3. Sistemas informacionales

Como se ha visto en la Subsección 9.1.1, los sistemas transaccionales (TPS) dan soporte y automatizan los procesos de negocio de una empresa u organización, siendo enfocados a la captura (registro) de los datos relacionados con los distintos eventos de negocio. Cuando en un sistema transaccional se decide que ciertos datos no se utilizan con suficiente frecuencia, se catalogan como históricos y se suelen depositar en almacenamientos externos. Por lo tanto los datos que almacenan los sistemas transaccionales no suelen ser suficientes para dar respuestas a preguntas complejas para la toma de decisiones estratégicas, que a menudo necesitan datos históricos.

Para dar soporte a preguntas complejas, a menudo no son suficientes solo los datos históricos del propio negocio, sino además una gran cantidad de datos que a menudo provienen de distintas fuentes incluidas externas (INE, INEM, colegios profesionales, encuestas, ... hasta un 20 % de los datos que maneja una empresa). Además el analista de información o el responsable de la toma de decisiones necesita respuestas a este tipo de preguntas de una manera rápida y sencilla, y sobre todo, una interfaz fácil de utilizar por un no experto en TIC.

Debido a que muchos de los análisis que se realizan son cíclicos y predecibles, se han diseñado sistemas que soportan estas funciones. Se trata de sistemas informacionales, orientados al análisis de datos y simulación de alternativas para el soporte a la toma de decisiones. Los sistemas MIS, DSS y ESS (Subsección 9.1.1) son ejemplos de sistemas informacionales. Las soluciones actuales que se encuentran en el mercado se distribuyen con el nombre de **aplicaciones de *Business Intelligence*** (BI, inteligencia de negocios).

Las herramientas de *Business Intelligence* transforman los datos en información significativa y útil, utilizada para dar lugar a conocimientos estratégicos, tácticos y operativos en la toma de decisiones. Las aplicaciones de *Business Intelligence* no se limitan a dar soporte a la línea media de una organización, sino que se encuentran en todos los niveles de la organización, incluido el ápice estratégico.

Ejemplos de soluciones de *Business Intelligence* son: *MS SQL Server* (Microsoft), *Oracle Business Intelligence* (Oracle), *SAP BusinessObjects* (SAP) o *Pentaho BI Suite* (Pentaho), esta última de libre distribución.

Estos productos incluyen por lo general funcionalidades para:

- extracción de datos de diferentes fuentes,
- almacenamiento: los datos extraídos son transformados en un formato de datos estándar, y almacenados en el sistema BI,
- modelado y simulación,
- explotación de la información, generación de alertas.

Herramientas como las hojas de cálculo podrían cubrir funcionalidades ofrecidas por herramientas BI (sobre todo si se utilizan sus funciones más avanzadas de tratamiento de

datos, como las tablas dinámicas (Martínez et al, 2012), con limitaciones sobre todo en poder manejar grandes volúmenes de datos. Muchas de las herramientas BI ofrecen integración con herramientas ofimáticas (especialmente con Excel) para poder exportar datos a hojas de cálculo y habilitarlas para que funcionen como interfaz de usuario y capa de presentación de datos.

El almacenamiento de datos (*Data Warehouse* en inglés) junto con la tecnología **OLAP** (*Online Analytical Processing* - procesamiento analítico en línea) son normalmente el núcleo de la mayoría de las herramientas BI.

9.2. El almacenamiento de datos (data warehousing)

Un almacén de datos es una colección de datos **orientados por tema, integrados, variables en el tiempo** y **no volátiles** que se emplea como apoyo a la toma de decisiones de los directivos.

Los almacenes de datos, en comparación con las bases de datos tradicionales, tienen la característica distintiva de que están proyectados principalmente para las aplicaciones de toma de decisiones. Están optimizados para la recuperación de datos, no para el procesamiento de transacciones rutinarias. El usuario final puede realizar consultas pero no puede insertar, actualizar, borrar datos. La información no se modifica ni se elimina, una vez almacenado un dato, éste se convierte en información de sólo lectura, y se mantiene para futuras consultas. Comparados en este sentido con las bases de datos tradicionales, los almacenes de datos no son volátiles.

La información que contiene un almacén de datos cambia con menos frecuencia y debe ser considerada de tipo "no en tiempo real", con actualizaciones periódicas. En sistemas transaccionales, las transacciones son las que promueven el cambio en los datos. Por el contrario, la información de los almacenes de datos se refresca siguiendo una cuidadosa política, habitualmente incremental.

Las actualizaciones del almacén son manipuladas por el componente de adquisición del mismo que incluye la limpieza y el re-formateo de los datos antes de su carga en el almacén. Este proceso se realiza en la actualidad mediante herramientas **ETL** (*Extraction, Transformation and Loading* - extracción, transformación y carga). La parte de extracción se refiere al proceso de extraer los datos de fuentes múltiples y heterogéneas (por ejemplo bases de datos o cualquier otro lugar en el que exista información relevante). La transformación o limpieza hace referencia al proceso de resolver incoherencias entre datos procedentes de diferentes fuentes. Los datos deben examinarse y formatearse de forma coherente para poder luego comprobar la validez y calidad de los mismos. Los datos de distintas fuentes deben acomodarse al modelo de datos del almacén. Finalmente los datos deben ser cargados en el almacén. El elevado volumen total de los datos en los almacenes de datos hace que la carga de los mismos sea una tarea importante. Las cargas suelen ser de tipo incremental, es decir solo se cargan las actualizaciones.

9.2.1. El paradigma multidimensional

Los almacenes de datos proporcionan acceso a los datos para realizar análisis complejos, descubrimiento de conocimiento y toma de decisiones. Un almacén de datos es una agrupación de datos integrados procedentes de varias fuentes, procesados para su almacenamiento en un modelo *multidimensional*, modelo base del diseño de los almacenes de datos. El diseño del almacén está enfocado a responder eficientemente a consultas. Por ello, el almacén de datos está organizado de acuerdo con los temas más importantes para la organización. Se distinguen dos aspectos fundamentales:

- actividades de interés para el análisis: compra de productos, venta de vehículos, alquileres → también llamados **hechos**
- contexto del análisis para las actividades: clientes, proveedores, productos → también referidos como **dimensiones**

El modelo multidimensional estructura el diseño de los almacenes de datos en hechos y dimensiones.

Los modelos multidimensionales se benefician de las relaciones inherentes de los datos para rellenar unas matrices multidimensionales llamadas **cubos de datos**. Un hecho contiene medidas interesantes que son el objeto de análisis (por ejemplo venta de productos), mientras que las dimensiones representan diferentes perspectivas para analizar dichas medidas (productos, almacenes, tiempo).

La Figura 9.2 muestra un cubo de datos tridimensional con información relativa al censo, que organiza los datos de un hecho: núcleos familiares (concretamente su número) por los datos de diferentes dimensiones: *localización* (atributo comunidad autónoma de residencia), *vivienda* (atributo régimen de tenencia) y *hogar* (atributo tamaño del hogar).

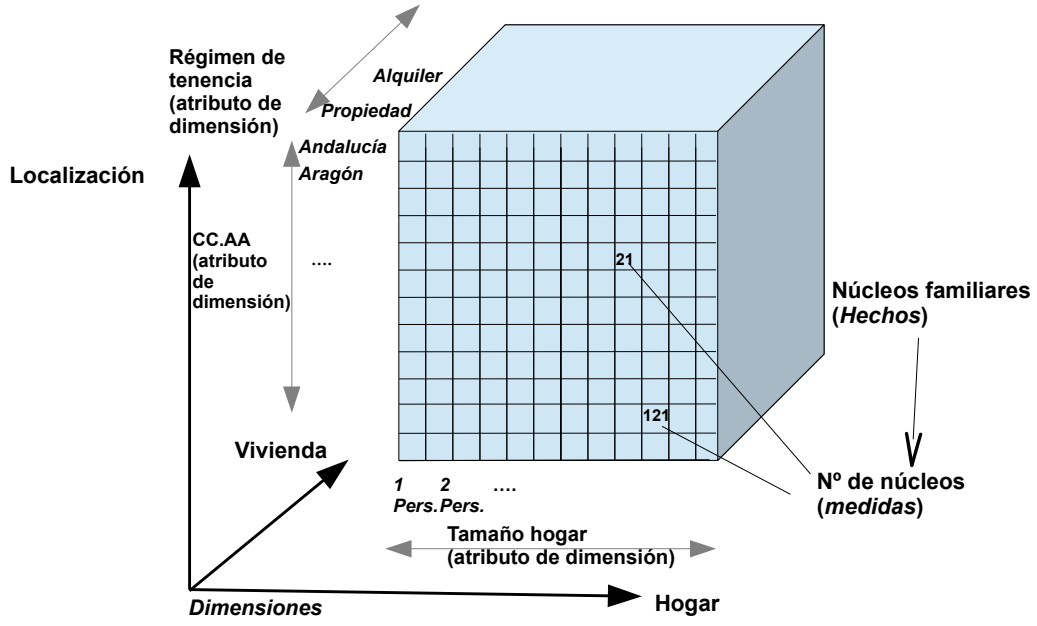


Figura 9.2 Ejemplo cubo de datos de tres dimensiones.

Los datos que se guardan en el almacén de datos son datos históricos, recogidos a lo largo de un gran intervalo de tiempo, con su referencia temporal bajo la que son válidos. El tiempo (medido en días, horas, semanas, etc.) suele ser una dimensión importante en un cubo.

Dimensiones

Una dimensión se compone de una serie de atributos organizados jerárquicamente. Los atributos determinan los diferentes niveles de análisis posibles dentro de una dimensión. Por ejemplo, para la dimensión *Localización* del cubo de datos en Figura 9.2 sus atributos pueden ser: CCAA, Provincia, Municipio, Comarca. Las jerarquías organizan los atributos dentro de la dimensión y orientan luego en el acceso a los datos mediante la navegación (OLAP). Las dimensiones son tablas que almacenan tuplas de atributos de dimensión.

Hechos

Los hechos contienen atributos de hecho o medidas a analizar (por ejemplo para el hecho *Núcleos familiares*, el atributo *Nº de núcleos familiares*). Representan algún aspecto cuantificable o medible de los objetos o eventos a analizar. Las medidas registradas de un hecho pueden ser atómicas (por ejemplo, *Nº de núcleos familiares*) o derivadas si se utiliza una fórmula para calcularlas (por ejemplo, *Porcentaje de núcleos respecto de la fila*). Los datos a analizar se guardan en la tabla de hechos, siendo las tablas de dimensión las que identifican cada tupla de estos datos. Uno de los diseños lógicos comunes para el almacén de datos es el *esquema estrella* que coloca en el medio la tabla de hechos y la relaciona con cada una de las tablas de dimensión, como en la Figura 9.3.

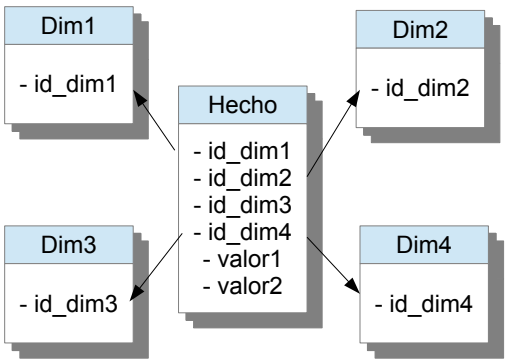


Figura 9.3 Ejemplo de modelado lógico: esquema estrella.

9.3. Análisis multidimensional

Un almacén de datos es una base de datos diseñada para favorecer el análisis y la divulgación eficiente de datos. Durante la fase del diseño del almacén, no existe forma de prever todas las posibles consultas y análisis que se realizarán. Sin embargo, el diseño sí de-

be soportar las **consultas ad-hoc**, es decir el acceso a los datos con cualquier combinación significativa de los valores de los atributos de las tablas de dimensión o de hechos.

La tecnología utilizada para realizar consultas y análisis ad-hoc sobre los datos suele ser **OLAP**, procesamiento analítico en línea. Las herramientas OLAP facilitan al usuario, no especialista TIC, el acceso intuitivo a los datos almacenados y le permite analizar la misma información según diferentes puntos de vista.

En la Figura 9.4 se muestra un ejemplo de informe realizado por una herramienta OLAP sobre el cubo de datos que aparece en la Figura 9.2. El informe considera datos relativos al censo en España del 2001 y muestra el número de núcleos familiares por comunidad autónoma y provincia de residencia contra el régimen de tenencia de la vivienda. Informes similares se pueden obtener de manera ad-hoc en la página del Instituto Nacional de Estadística[1].

CC.AA. de residencia	TOTAL	Aragón			
Provincia de residencia	TOTAL	TOTAL	22-Huesca	44-Teruel	50-Zaragoza
Régimen de tenencia (agregación)					
TOTAL	334.862	334.862	55.780	37.252	241.830
En propiedad	291.729	291.729	49.305	32.603	209.821
En alquiler	28.228	28.228	3.628	2.060	22.540
Cedida gratis o a bajo precio por otro hogar, la empresa..	7.447	7.447	1.431	1.286	4.730
Otra forma	7.458	7.458	1.416	1.303	4.739

Figura 9.4 Ejemplo de informe del censo realizado por una herramienta OLAP.

9.3.1. Operaciones OLAP

Los atributos de cada dimensión permiten analizar las medidas de los hechos a diferente nivel de detalle según se agreguen o desagreguen los datos. Estas operaciones se conocen como visualización roll-up y drill-down. Por ejemplo, el número de núcleos familiares se puede analizar por ciudad, comunidad, provincia o país. La **visualización roll-up** (compactar/subir) mueve hacia arriba la jerarquía, agrupando en unidades más grandes a lo largo de una dimensión. Por ejemplo, si se considera el número de núcleos familiares por ciudades se puede aplicar roll-up para obtener el número de núcleos familiares por provincias. La **visualización drill-down** (descomponer/expandir) ofrece la operación contraria, proporcionando una vista más fina (por ejemplo descomponiendo por provincias el número de núcleos familiares de una comunidad autónoma).

La operación que permite reorientar la vista multidimensional de los datos, es decir, cambiar la distribución de filas o columnas se conoce como **pivoting** (pivotaje o rotación).

Otras operaciones que una herramienta OLAP puede permitir son:

- **Drill-across**: navegación de un hecho a otro mediante el uso de dimensiones comunes.
- **Slice & dice**: definición de restricciones/filtros sobre niveles de jerarquías. *Slice* permite la selección de un solo valor para una de las dimensiones, lo que crea un nuevo cubo con una dimensión menos. Por ejemplo en el caso del censo (Figura 9.2), la dimensión tiempo

[1] `http://www.ine.es/censo/es/inicio.jsp`

no existe al reducir las posibilidades de las consultas únicamente al año 2001. La operación *dice* permite al analista seleccionar valores específicos de múltiples dimensiones.

9.4. Otras capacidades de BI

Entre las capacidades que se consideran esenciales para las plataformas de BI se encuentran: la presentación de informes, cuadros de mando, consultas ad-hoc, OLAP, visualización interactiva, el modelado predictivo y minería de datos.

En particular, las herramientas de **minería de datos** (*data mining*, en inglés) son otro tipo de técnicas para explotar el *almacén de datos*.

> La minería de datos hace referencia a la extracción o descubrimiento de nueva información en términos de patrones significativos o reglas procedentes de grandes cantidades de datos (Elmasri and Navathe, 2007).

Los patrones descubiertos no se pueden encontrar necesariamente mediante la consulta directa de los datos del almacén (p. ej. mediante la ejecución de una operación OLAP).

Entre los distintos tipos de herramientas que suelen encontrarse en los paquetes de minería de datos, incorporados en las soluciones de BI, destacan (Gómez and Suárez, 2011):

- **Herramientas estadísticas**: cálculo de medias, varianzas, etc.; técnicas bayesianas; test de hipótesis; regresión lineal; etc.
- **Técnicas de inteligencia artificial**: algoritmos genéticos; redes neuronales; etc.
- **Herramientas simbólicas**: árboles de decisión; reglas de asociación; etc.

Las oportunidades asociadas con los datos y su análisis en diferentes organizaciones han suscitado un alto interés en BI. Además del procesamiento de datos subyacente y las tecnologías analíticas, BI incluye prácticas centradas en el negocio y metodologías que se pueden aplicar a diversos ámbitos tales como: el comercio electrónico, inteligencia de mercado, e-gobierno, la salud, y la seguridad (Chen et al, 2012). A modo de ejemplo, se puede mencionar el uso de técnicas de minería de datos y aprendizaje automático en el ámbito de la seguridad, para el apoyo a la toma de decisiones en la lucha antiterrorista (Chen et al, 2008; Velasco et al, 2014), y en el ámbito de la ciberseguridad (Dua and Du, 2014) para de la detección de anomalías, intrusiones, usos impropios en sistemas de información, etc. Este último tipo de capacidades son incorporadas en productos específicos de apoyo a la toma de decisiones denominados SIEM (*Security Information and Event Management*).

9.5. Sistemas transaccionales versus informacionales

Los almacenes de datos y las herramientas OLAP, como núcleo de sistemas informacionales, están orientados al tema y el negocio, siendo su objetivo proporcionar respuesta a las consultas ah-hoc para apoyar la toma de decisiones. En cambio, los sistemas transaccionales están orientados a la aplicación y a procesar un gran número de transacciones (a menudo en tiempo real), dando soporte a las operaciones diarias de una organizaciones.

Las principales diferencias entre los dos tipos de sistemas se muestran en el Cuadro 9.5.

Sistemas transaccionales	Sistemas informacionales
Basados en:	
▪ tecnología OLTP	▪ tecnología OLAP
Orientados al:	
▪ procesamiento de datos ▪ registro detallado de los eventos de negocio ▪ soporte a operaciones diarias	▪ apoyo a la decisión ▪ análisis de los datos acumulados, simulación de alternativas
Base de datos:	
▪ diseño orientado a la aplicación (Modelo ER) ▪ datos actuales, aislados ▪ tamaño del orden de los MB, GB	▪ diseño orientado al tema o negocio (Estrella, ...) ▪ datos históricos, consolidados ▪ tamaño del orden de los GB, PB
Operaciones:	
▪ de lectura y escritura ▪ realizadas por muchos usuarios ▪ transacciones simples, estructuradas y repetitivas	▪ de lectura mayoritariamente ▪ realizadas por pocos usuarios ▪ consultas complejas, ad-hoc

Cuadro 9.1 Sistemas transaccionales versus informacionales.

9.6. Casos prácticos

1. ¿Qué tipo de sistema de información es el sistema *CarShare* analizado a lo largo del libro?
2. Identifique algunas dimensiones y hechos que podría tener un cubo OLAP que almacenara información histórica de *CarShare*. ¿Qué consultas podría realizar sobre ese cubo?
3. Conéctese a la página del INE:

   `http://www.ine.es/censo/es/inicio.jsp`

   que proporciona los censos de población y viviendas del 2001. A continuación:

   - Elija la opción Crear tablas.
   - Seleccione el ámbito geográfico para la consulta como CCAA.
   - Elija Aragón y añádalo en la columna derecha y pulse el botón Aceptar selección.
   - Seleccione el colectivo principal Parejas y otros núcleos familiares.
   - Seleccione *Datos de la vivienda* → *Régimen de tenencia* → *Régimen de tenencia (agregación)* y añádalo (pulsando flecha) en el recuadro *Columnas* como atributo de dimensión.
   - Pulse Ver tabla.

   Reflexione sobre el resultado obtenido y los pasos que ha realizado para alcanzarlo. ¿Qué tipo de operaciones OLAP ha aplicado en cada paso?
   Con la tabla obtenida:

   - Pulse sobre el nombre de alguna columna o fila y localice las funciones de navegación de las que dispone.
   - Modifique la tabla cambiando el atributo a analizar (opción Cambiar unidad de medida).
   - Intente aplicar las principales operaciones de visualización OLAP (pivotaje, roll-up, drill-down) y observe el resultado.

Apéndice A
Caso de estudio *CarShare*

CarShare (Bennet et al, 2004) es una franquicia cuyo objetivo de negocio es promover el uso compartido de coches (*car-sharing*). *CarShare* ofrece un servicio para los viajeros poniendo en contacto personas que viven y/o trabajan en lugares cercanos. La empresa está organizada en una dirección general y establecimientos locales franquiciados.

En un establecimiento de *CarShare* las operaciones más comunes son registrar nuevos clientes, poner en contacto viajeros y registrar del uso compartido de coches. Estas operaciones pueden ser realizadas por los empleados, operadores telefónicos y supervisores del establecimiento. El supervisor, además, es responsable de la redacción mensual de un informe de gestión.

Hoy en día, las actividades en un establecimiento se realizan por los trabajadores a través de un sistema software. Periódicamente se actualizan los resultados de las actividades en los establecimientos (registros de nuevos clientes, acuerdos de viajes, etc.) intercambiando los datos con el servidor de la dirección central.

El consejo de administración de *CarShare* ha planificado ampliar a medio plazo el negocio explotando el potencial del comercio electrónico. Con ese objetivo ha encargado un estudio de viabilidad a un equipo de analistas para mejorar su sistema de información de modo que pueda ser utilizado tanto por los establecimientos franquiciados como por sus clientes a través de Internet, siguiendo el ejemplo de muchas empresas de car-sharing (por ejemplo Bla Bla Car (2014)).

Como ejemplo de uso del sistema de información mejorado, se puede considerar el caso de Alex que quiere compartir su coche a través de *CarShare*. Cada vez que Alex visita a su familia los fines de semana utiliza *CarShare* para publicar sus viajes, escoger sus preferencias, *CarShare* incluso le ayuda a fijar el precio. María estudia, Juan es arquitecto y Javier profesor: los tres quieren hacer el mismo viaje, pero sus opciones son limitadas y cada vez son más caras, sobre todo en el último minuto. Con *CarShare* los tres pueden encontrar rápidamente conductores y, con las opiniones de los otros usuarios, elegir su coche y simplemente reservar una plaza. En el viaje, Alex comparte su pasión por el jazz, María cuenta su último viaje y Juan comparte sus galletas. Mientras tanto Javier decide echarse una siesta. Ya en casa valoran a sus compañeros rellenando una encuesta a través de la página web de *CarShare*. Gracias a *CarShare* los cuatro ahorran mucho dinero en sus viajes.

Los viajeros que tienen coche y buscan a compañeros de viaje para compartir los gastos, tienen que especificar sus datos de contacto, los datos del viaje y también la forma de cobro (por ejemplo, en efectivo al comienzo del viaje o a través de su cuenta *PayPal* o con tarjeta de crédito). Los viajeros que buscan una plaza también tienen que especificar sus datos de contacto y pagar al conductor.

A.1. Entrevista

El enunciado de *CarShare* no es suficientemente detallado para obtener un modelo de requisitos y de negocio del sistema. Por lo tanto, los analistas han organizado entrevistas entre su equipo de trabajo y el consejero delegado de *CarShare* para obtener más información sobre las necesidades de los futuros usuarios. A continuación se presenta el extracto de una entrevista.

Analista ¿Qué datos necesitamos para el registro de los clientes?

Consejero Los datos de contacto (nombre, apellidos, dirección, etc.). En particular, la dirección de correo electrónico, o por lo menos un número de teléfono, para avisarles en caso de haber encontrado compañeros de viaje.

Analista ¿Qué datos necesitamos para registrar un viaje?

Consejero Necesitamos definir el itinerario (lugar de origen y destino), la fecha y hora de salida, la marca y el modelo del coche. También hay que especificar el número de plazas del coche y el número de plazas ofertadas, el lugar donde se recogen y/o dejan los pasajeros (puede haber más de uno, por ejemplo, en las ciudades de paso). Se puede permitir al conductor insertar algún comentario (por ejemplo, ¿Cuánto espacio hay para el equipaje?). Además el conductor tiene que que declarar de estar en posesión de un permiso de conducir y de un seguro vigente ...

Analista ¿Quien establece el coste del viaje?

Consejero Lo establece el conductor, pero *CarShare* le ayuda a fijar el precio en base a varios parámetros (número km a recorrer, precio actual de la gasolina, número pasajeros, etc.).

Analista ¿Los pasajeros pagan al conductor o a *CarShare*?

Consejero Pagan al conductor. Hay diferentes formas de pago que establece el conductor cuando registre el viaje ...

Analista ¿Cuáles son las formas de pago?

Consejero Los pasajeros pueden pagar al conductor en efectivo, al comienzo del viaje, o *en línea* utilizando una tarjeta de crédito, PayPal, etc., al momento de la reserva de la plaza.

Analista ¿*CarShare* cobra una comisión de viaje?

Consejero Actualmente sí: cobra una comisión (11 % del coste del viaje) cuando el conductor registre un nuevo viaje llamando a nuestros empleados. Pero, si el conductor realizara el registro por Internet no cobraremos la comisión.

Analista ¿Quién combina los clientes para un viaje?

Consejero Actualmente, son nuestros empleados que trabajan en el establecimiento los que se preocupan de buscar combinaciones a través de la aplicación informática. Nos gustaría que los viajeros pudieran buscar directamente a los conductores que hacen el mismo trayecto y eligir la mejor oferta.

Analista ¿Cómo se avisa al conductor si hay pasajeros interesados?

Consejero La forma común es por correo electrónico. En caso de que no tenga, por teléfono.

Analista ¿Qué pasa cuando un viaje se anula?

Consejero Si un viaje se anula por parte del conductor o de un viajero antes del día concertado se devuelve el dinero pagado por adelantado.

Analista ¿Si un compañero de viaje no se presenta a la cita, el conductor no cobra la parte correspondiente?

Consejero El conductor se queda con el dinero pagado por el pasajero. En caso de forma de pago en efectivo, el conductor no obtiene un reembolso. Por eso, aconsejamos a los conductores eligir la forma de pago en línea en el momento del registro del viaje ...

Analista ¿El cuestionario de opinión del viaje pueden rellenarlo todos?

Consejero Sí claro. El cuestionario es opcional, pero aconsejamos rellenarlo, sobre todo si alguien no se presenta el día del viaje al lugar concertado ... Para nosotros es importante para tener traza de las incidencias ...

Analista ¿Quién actualiza los datos con el servidor central de *CarShare*?

Consejero Actualmente los empleados del establecimiento, al final del día. Se supone que con el nuevo sistema esta actividad ya no será necesaria.

Analista ¿Cambia la forma de redacción del informe mensual de gestión por parte del supervisor?

Consejero El supervisor deberá poder redactar el informe mensual en línea.

Apéndice B
Caso de estudio *UPS*

United Parcel Service, Inc. (UPS, 2015) es la empresa de entrega de paquetes -vía tierra y aire- más grande del mundo. Su sistema de información se apoya fuertemente en las TIC y, en particular, ofrece diferentes servicios dependiendo del tipo de usuario.

El sistema UPS permite a cualquier cliente que desee hacer una petición a través de su sitio web, visualizar en línea una guía de servicios con las tarifas aplicadas, las restricciones del servicio, los requisitos para el embalaje y los servicios opcionales. Para ver rápidamente las opciones de envíos y niveles de servicio, el cliente puede solicitar un presupuesto en línea; es suficiente introducir los datos de un envío para que el sistema UPS se encargue de calcular los costes de envío. Los datos necesarios para obtener un presupuesto se concretan en: 1) lugar de origen y destino (País, código postal, ciudad, dirección del domicilio) y la fecha de envío; 2) el tipo de servicio requerido (*express plus*: la entrega es garantizada a primera hora de la mañana, *express*: la entrega es garantizada a la mañana siguiente, *express saver*: la entrega es garantizada a lo largo del día laborable siguiente, y *standard*: para envíos no urgentes); 3) el tipo de envío (sobre, paquete, etc.) y sus dimensiones; 4) servicios adicionales como la confirmación de entrega con firma requerida, la entrega el sábado, el uso de carbon neutral para compensar el impacto climático del envío, la modalidad de entrega de la etiqueta de código de barra (electrónica, envío por correo, etc.). A partir de los datos introducidos, el sistema UPS genera un presupuesto con el coste, la fecha y hora estimadas de envío y de entrega.

Una vez obtenido el presupuesto, un remitente puede realizar la petición de envío con diferentes opciones de pago (tarjeta de crédito, PayPal, transferencia bancaria, etc.). La transacción se efectúa con un protocolo de comunicación seguro. Finalmente, el sistema UPS genera la etiqueta de código de barras, que contiene la información detallada sobre el remitente, destinatario, el día y hora de entrega, y transmite la petición de envío al centro de distribución UPS más cercano a su destino final. Los empleados del centro, mediante el sistema, pueden visualizar e imprimir los datos de la petición y realizar una asignación de ruta eficiente que considera las condiciones de tráfico, meteorológicas y los puntos intermedios de almacenaje. Una vez asignada la ruta, el sistema actualiza el estado de la petición a *listo para el envío*. Por otra parte, el cliente, una vez realizada la petición de envío (que puede ser a través del sitio web de UPS o mediante una aplicación en su *smartphone*), descarga, imprime y pega la etiqueta de código de barras al paquete. El conductor UPS accede al sistema, habitualmente mediante un terminal portátil tipo PDA (*Personal Digital Assistant*) para descargarse la ruta diaria y para registrar la firma de los clientes junto con la información de recogida y entrega. En los puntos intermedios de almacenaje a lo largo de la ruta, los empleados de los almacenes usan dispositivos portátiles de lectura de código

de barras para actualizar en el sistema el estado de envío del paquete. Cuando el paquete llega a la oficina UPS de destino el empleado actualiza el estado del paquete, utilizando un dispositivo portátil tipo PDA, y el sistema notifica al destinatario su llegada.

UPS ofrece también servicios dirigidos a empresas, previo registro del perfil. Las empresas pueden efectuar envíos en línea tanto de paquetes individuales como automatizar el proceso de envío por lotes utilizando las direcciones guardadas en una agenda y la información de las mercancías. En particular el envío por lotes permite crear hasta 250 envíos simultáneos gracias a la importación de un archivo de texto (formateado .csv - *comma separated values*). Además las empresas pueden acceder a los historiales de envíos y obtener notificaciones por correo electrónico. Finalmente, UPS ofrece un servicio de seguimiento avanzado (*UPS tracking*) a las empresas para: 1) acceder a la información de sus envío 24 horas al día, 7 días a la semana, y 2) habilitar a sus propios clientes para que puedan comprobar el estado de la entrega de sus pedidos.

Hay diversas formas de seguimiento. La más sencilla es a través de la página web de UPS: el cliente puede recuperar la información relacionada con el paquete introduciendo el número de seguimiento. Éste es creado en el momento del envío y la empresa puede elegir entre el número de orden o el número de cliente.

Apéndice C
Lenguaje Unificado de Modelado

A continuación se presentan ejemplos de aplicación de los tres tipos de diagramas UML, utilizados a lo largo de este libro, al caso de estudio *CarShare* (véase Apéndice A): los diagramas de casos de uso, los diagramas de clase y los diagramas de actividades. En la descripción de cada diagrama, se hace hincapié en los elementos de modelado más importantes.

C.1. Diagrama de casos de uso

Los diagramas de casos de uso se utilizan principalmente para crear modelos de contexto que representan los requisitos funcionales de un sistema, es decir las funcionalidades del mismo (*"Lo que un sistema tiene que hacer"*) - véase Capítulo 3. Existen también extensiones del diagrama de casos de uso, definidas con la técnica de *perfiles*, para la especificación de requisitos no funcionales como, por ejemplo, de seguridad y de supervivencia (Sindre and Opdahl, 2005; Bernardi et al, 2013). Finalmente, los diagramas de casos de uso se utilizan, en la etapa de pruebas (*testing*, en inglés) del proceso de desarrollo del software, para identificar los casos de prueba a realizar.

Hay que observar que **los diagramas de casos de uso no son los casos de uso**: estos últimos - como explicado en el Capítulo 3 - son historias escritas de uso de un sistema por parte de un actor para conseguir sus objetivos.

La Figura C.1 muestra un ejemplo de diagrama de casos de uso que especifica los requisitos funcionales del sistema software *CarShare* (véase Apéndice A).

Los elementos básicos de un diagrama de casos de uso son: los actores y los casos de uso. Los primeros son muñecos esbozados e identifican las partes interesadas en un sistema (por ejemplo, el *Conductor* y el *Pasajero*). Una representación alternativa de los actores es un rectángulos estereotipado «`actor`», que se utiliza para indicar un actor de soporte, típicamente otro sistema que ofrece servicios al sistema que se está analizando (por ejemplo, el *Sistema de autorización al crédito*).

Un *caso de uso*, dibujado como forma elíptica, representa una funcionalidad del sistema que aporta un resultado perceptible a los actores relacionados con este: en el diagrama, el caso de uso se identifica con una descripción concisa, que empieza con un verbo, por ejemplo *Registrar viaje*.

La *asociación* entre un actor y un caso de uso, indica que la parte interesada usa la funcionalidad del sistema a través de interacciones. Las interacciones entre un actor y el sistema, y viceversa, se describen textualmente en el caso de uso. Las herramientas CASE,

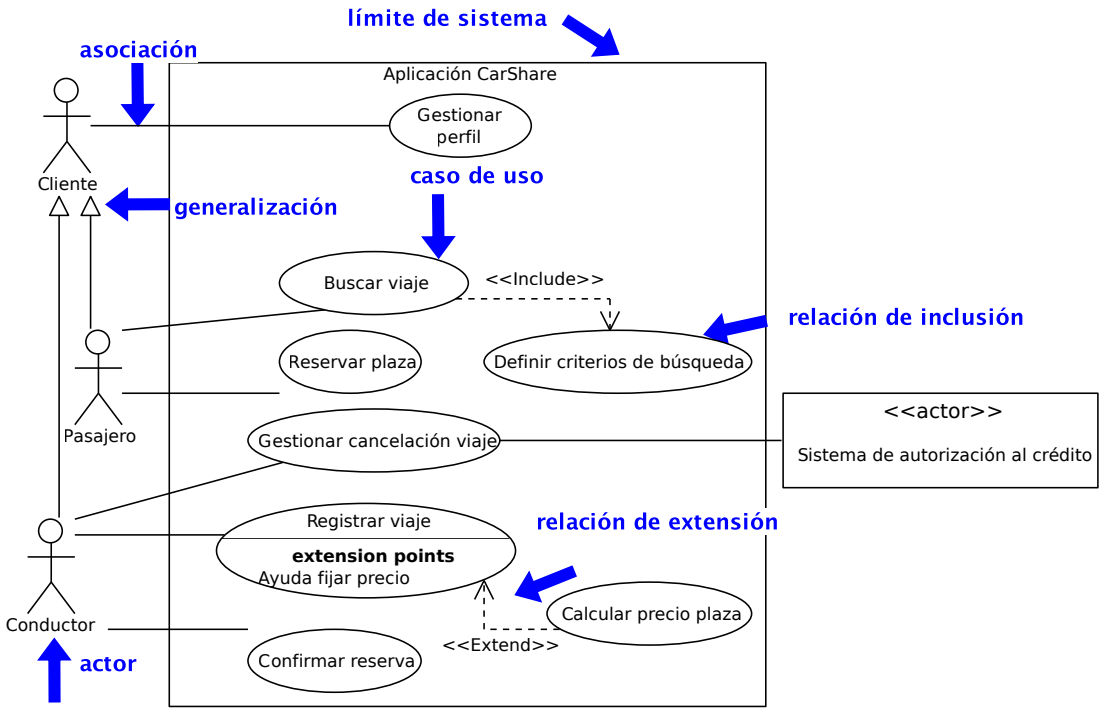


Figura C.1 Ejemplo de diagrama de casos de uso.

como por ejemplo Visual Paradigm (2015), enlazan editores de texto a los elementos de modelado para poder detallar los casos de uso.

Los casos de uso se pueden relacionar entre ellos; hay dos tipos principales de relación: de *inclusión* y de *extensión*. Un caso de uso incluye («`include`») a otro caso de uso cuando depende del resultado del caso de uso incluido: por ejemplo, la búsqueda de un viaje (caso de uso *Buscar viaje*) incluye siempre la definición de los criterios de búsqueda (*Definir criterios de búsqueda*). Esta relación es útil para extraer comportamientos/funcionalidades comunes desde múltiples casos de uso a una descripción individual.

Un caso de uso puede extender («`extend`») a otro caso de uso cuando el comportamiento/ funcionalidad del caso de la extensión se utiliza en el segundo. A diferencia de la inclusión, el caso de uso extensión puede ser insertado en el caso de uso extendido bajo ciertas condiciones (los *puntos de extensión*). Esta relación se usa para considerar casos especiales o para acomodar nuevos requisitos durante el proceso de desarrollo o mantenimiento del sistema. Por ejemplo, en la Figura C.1 el caso de uso principal *Registrar viaje* es extendido considerando el cálculo del precio de la plaza (el caso de uso extensión es *Calcular precio plaza*): esta sub-funcionalidad es opcional, es decir durante el registro de un viaje si el *Conductor* pide ayuda al sistema para fijar un precio (*punto de extensión*), el sistema tiene que calcular el precio de la plaza.

La *generalización* es la tercera forma de relación que se utiliza también en los diagrama de clases (véase Sección C.2). En un diagrama de casos de uso se suele utilizar principalmente para relacionar actores y, en la práctica, este tipo de relación es útil para factorizar comportamientos comunes a uno o más actores.

La notación UML para la generalización es una flecha que conecta el actor especializado con el actor genérico, apuntando al actor genérico: el actor especializado hereda el comportamiento del actor genérico y puede tener otros específicos. Por ejemplo, en la Figura C.1 hay dos generalizaciones: una entre el actor genérico *Cliente* y el actor específico *Conductor* y la otra entre el *Cliente* y el actor específico *Pasajero*. Se han introducido para factorizar el comportamiento común de los actores específicos: es decir, tanto el *Conductor* como el *Pasajero* pueden utilizar el sistema con el rol de *Cliente* para gestionar su perfil (caso de uso *Gestionar perfil*).

En la fase de identificación de requisitos es muy importante establecer el contexto o *límite del sistema* que se está analizando, sobre todo cuando hay varios sistemas o subsistemas relacionados entre sí: el límite del sistema es un rectángulo que permite agrupar casos de uso que pertenecen al sistema. Es buena costumbre asignar siempre un nombre al límite del sistema que identifica el sistema que se está analizando (*aplicación CarShare*).

C.2. Diagrama de clases

Los diagramas de clases describen la estructura estática de un sistema, en términos de clases y asociaciones. Se pueden utilizar para diferentes propósitos: en particular, para representar los conceptos del dominio en la etapa de análisis del problema (véase Capítulo 4) o para definir las clases software durante la etapa de diseño de una aplicación software. La Figura C.2 muestra un ejemplo de diagrama de clases que representa los principales conceptos del dominio *CarShare*.

Una *clase* describe un conjunto de objetos con estructura, comportamiento, relaciones y semántica similar. Por ejemplo, cada viaje - es decir, un objeto, instancia o ejemplo de clase *Viaje* - tiene una fecha y una hora de salida, un número de plazas ofertadas; estos son valores específicos asignados a los atributos *fechaSalida*, *horaSalida* y *númeroPlazasOfertadas*, respectivamente.

La estructura de los objetos de una clase se define con *atributos*, el comportamiento con *operaciones*. Una clase se representa con un rectángulo, el nombre de la clase aparece en la parte superior del rectángulo, la parte central contiene los atributos y la parte inferior contiene las operaciones. Los atributos y operaciones de una clase son opcionales. En el ejemplo de Figura C.2 no hay operaciones, puesto que el diagrama representa conceptos del dominio y no clases software.

Las relaciones entre clases se definen con *asociaciones* y *generalizaciones*. Una asociación bidireccional se representa con una línea que enlaza dos clases: los viajes *se efectúan en* coches. Las asociaciones pueden tener un nombre y los extremos de una asociación también. En este último caso son los nombres de los *roles* de las clases involucradas en la asociación. Por ejemplo, un trayecto está caracterizado por dos lugares: uno de salida (rol *sale-de*) y uno de llegada (rol *llega-a*).

Los números de los extremos de una asociación indican las *multiplicidades*: por defecto, la multiplicidades son iguales a uno. Así que se puede especificar que un viaje *incluye* por lo menos un trayecto (multiplicidad 1..$*$), un lugar puede ser de salida (o de llegada) en cero, uno o muchos trayectos (multiplicidades $*$ -o de forma equivalente 0..$*$- de un extremo de las asociaciones bidireccionales entre *Trayecto* y *Lugar*) y un trayecto tiene un sólo lugar de salida y un sólo lugar de llegada (multiplicidad 1 del otro extremo de las asociaciones entre *Trayecto* y *Lugar*).

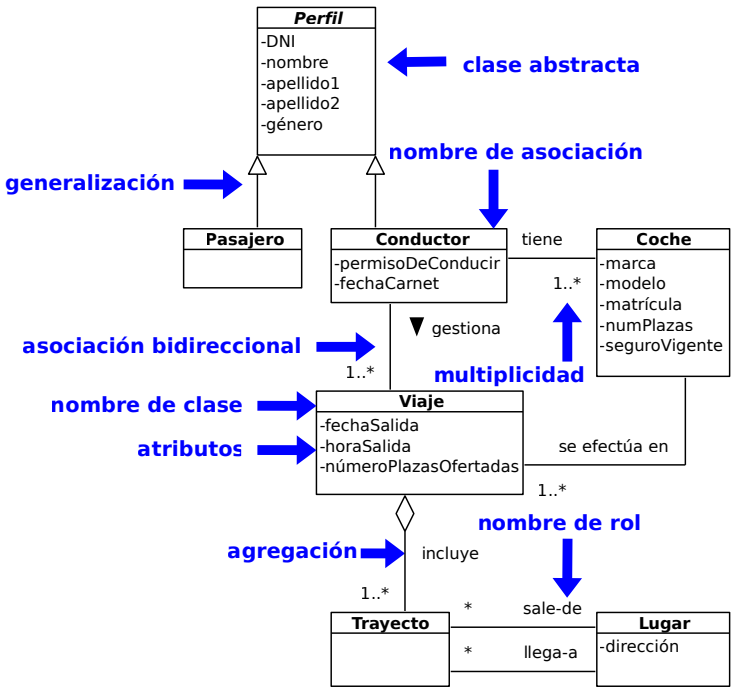


Figura C.2 Ejemplo de diagrama de clases.

La *agregación* es un tipo particular de asociación: en un diagrama de clase, se identifica con una forma de diamante cerca de la clase que agrega, mientras la clase al otro lado representa la parte de la agregación. La Figura C.2 muestra un ejemplo de agregación: la asociación *incluye* entre la clases *Viaje* y *Trayecto*.

Otro tipo de relación entre clases es la *generalización*. Las clases *Pasajero* y *Conductor* son clases especializadas (sub-clases) de la clase genérica (super-clase) *Perfil*: tienen las mismas características de un perfil - es decir, en el ejemplo heredan los atributos *DNI, nombre, apellido1, apellido2, género* - y además pueden tener otros atributos y operaciones específicos (*permisoDeConducir* y *fechaCarnet* en la clase *Conductor*). La notación UML para la generalización -ya comentada en la descripción del diagrama de casos de uso (Sección C.1)- es una flecha que conecta la super-clase con sus sub-clases, apuntando a la super-clase.

La clase *Perfil* es un ejemplo de clase *abstracta* y su nombre se indica en cursiva. Una clase es abstracta cuando no contiene objetos concretos, en este caso un perfil sólo puede corresponder al perfil de un pasajero o al perfil de un conductor.

C.3. Diagrama de actividades

Los diagramas de actividades son grafos directos y se usan para mostrar la secuencia y concurrencia de acciones. En particular, muestran el flujo de trabajo desde el punto de inicio hasta el punto final detallando muchas de las rutas de decisiones que existen en el progreso de ejecución de las acciones. Estos también pueden usarse para detallar situaciones donde el proceso paralelo puede ocurrir en la ejecución de algunas actividades.

La semántica de los diagramas de actividades se basa sobre las redes de Petri (Silva Suárez, 1985), un lenguaje formal de modelado importante en el marco de la automática y de la computación informática.

Un ejemplo de diagrama de actividades se muestra en Figura C.3: representa el modelo de proceso de negocio para el acuerdo de un viaje entre un pasajero y un conductor. Los principales elementos de modelado de un diagrama de actividades se describen a continuación.

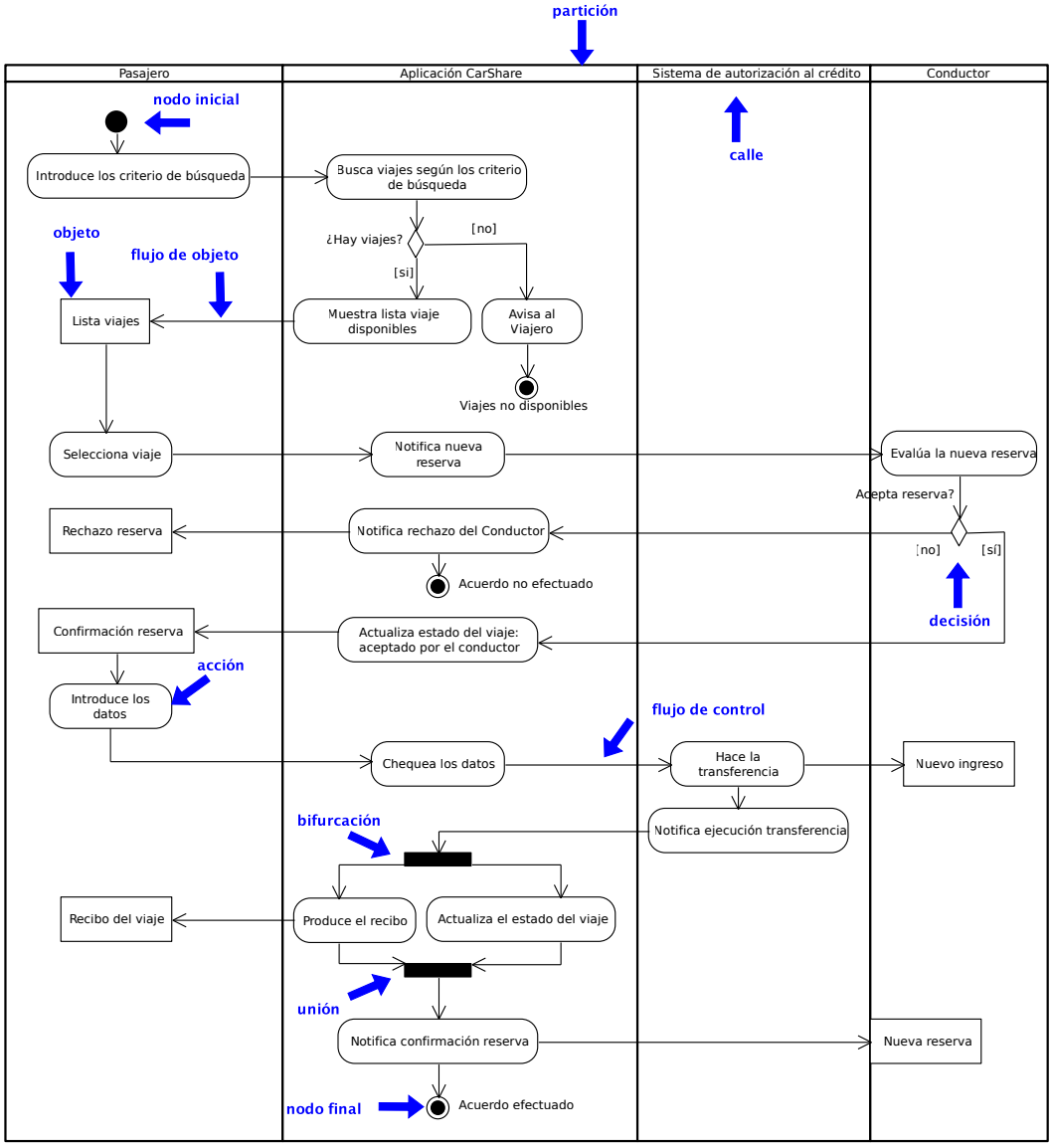


Figura C.3 Ejemplo de diagrama de actividades.

Una *acción* es un solo paso dentro de un diagrama de actividades. Las acciones se denotan por rectángulos con las puntas redondeadas (por ejemplo, *Introduce los datos*). El *nodo*

inicial -o de comienzo- se representa con un gran punto negro, mientras el *nodo final* de actividad se describe como un círculo con un punto dentro del mismo. En general, puede haber varios nodos iniciales y finales. Sin embargo, por razones de inteligibilidad del diagrama, suele haber un sólo nodo inicial -que indica la primera acción que se lleva a cabo- y uno o más nodos finales -que especifican el estado alcanzado por efecto de un escenario de interacción entre los participantes. Por ejemplo, en Figura C.3, hay tres nodos finales: *Viajes no disponibles*, *Acuerdo no efectuado* y *Acuerdo efectuado*.

En los diagramas de actividades se puede especificar tanto el *flujo de control* como el *flujo de objetos* (o datos). Un flujo de control muestra el orden de ejecución de las acciones dentro de una actividad: se representa con flechas, cada flecha enlaza dos nodos y su dirección indica el orden de ejecución, es decir el nodo de salida de la flecha se ejecuta antes que el nodo de llegada. En la Figura C.3, la aplicación *CarShare* verifica los datos introducidos por el pasajero para el pago al conductor (acción *Chequea los datos*) y, después de haber recibido los datos, el sistema de autorización al crédito hace la transferencia (acción *Hace la transferencia*).

Los *objetos* se muestran cómo rectángulos (*Lista viajes*) y un flujo de objetos es la ruta a lo largo de la cual pueden pasar objetos o datos. El flujo de objetos, igual que el flujo de control, se representa con flechas, pero, en este caso, cada flecha tiene un objeto en uno de sus extremos y la dirección de la flecha especifica la dirección a la cual se está pasando el objeto. Por ejemplo, la *Lista viajes* es el resultado de la acción *Muestra lista viaje disponibles* y es un dato de entrada para la acción *Selecciona viaje*.

Un diagrama de actividades puede incluir otros nodos, a parte de los nodos acción, inicial y final. Los *nodos de decisión, combinación, bifurcación* y *unión* sirven básicamente para describir el flujo de control de las acciones.

Las decisiones se representan con forma de diamante. Los flujos de control que provienen de un nodo de decisión tendrán *condiciones de guarda* que permitirán el control para fluir si la condición de guarda se realiza (en el ejemplo, una vez recibida la notificación de nueva reserva, el conductor puede aceptar - *[sí]* - o rechazar - *[no]* - la reserva). La misma notación se usa para combinar flujos de control procedentes de una decisión (es decir, flujos de control alternativos).

Las bifurcaciones y uniones se usan para especificar acciones en paralelo (concurrentes) y tienen la misma notación: una barra negra horizontal o vertical. La orientación depende de si el flujo de control va de derecha a izquierda o de arriba abajo. Estos nodos indican el comienzo y final de hilos actuales de control.

Una unión es diferente de una combinación ya que la unión sincroniza dos flujos de entrada y produce un solo flujo de salida. El flujo de salida desde una unión no se puede ejecutar hasta que todos los flujos de entrada se hayan recibido. Una combinación pasa cualquier flujo de control directamente a través de esta. Si dos o más flujos de entrada se reciben por un símbolo de combinación (el *diamante*), la acción a la que el flujo de salida apunta se ejecuta en cuanto al menos uno de los flujos de entrada haya completado la ejecución.

Una *partición* de una actividad se muestra como *calle* horizontal o vertical: hay tantas calles como partes interesadas en una actividad o proceso (en el ejemplo, *Pasajero, Aplicación CarShare, Sistema de autorización al crédito* y *Conductor*). Las particiones se usan para separar acciones realizadas por las partes interesadas.

C.3.1. Modelo de flujos de datos

Los diagramas de flujos de datos (*Data Flow Diagrams* - DFD), definidos por Gane and Sarson (1978), han sido tradicionalmente utilizados para especificar el flujo de datos en un sistema de información. En los DFD, los principales elementos de modelado son: las *entidades externas*, que producen o reciben la información; los *flujos de datos*, que indican el flujo de información a través del sistema; los *procesos*, que transforman la información que les llega a través de los flujos de datos de entrada en la información que sale a través de los flujos de datos de salida; y los *almacenes de datos*, que guardan los datos para su procesamiento posterior.

UML no incluye los DFD entre sus diagramas, sin embargo -como se ha mencionado en esta sección- los diagramas de actividades permiten especificar, además del flujo de control, también el flujo de objetos de manera similar a los DFD. Por lo tanto, es posible utilizar los diagramas de actividades en lugar de los DFD (Larman, 2004), donde las calles reemplazan las entidades externas, los flujos de objetos reemplazan los flujos de datos y las acciones reemplazan los procesos. Finalmente, los almacenes de datos se pueden modelar en UML como objetos estereotipados «*datastore*».

Referencias

van der Aalst W (2011) Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer Verlag

Abran A, Moore W, Bourque P, Dupuis R (eds) (2004) Software Engineering Body of Knowledge. IEEE Computer Society

Acero R, Pastor J, Sancho J, Torralba M (2012) Ingeniería de la Calidad. In: Colección Textos Docentes, Centro Universitario de la Defensa (Zaragoza)

Amazon Inc. (2014) Amazon.com: Online shopping for electronics, apparel, computers, books, dvds & more. Sitio web: `www.amazon.com`. Accedido el 26/12/2014

Bennet S, Skelton J, Lunn K (2004) Schaum's Outline of UML. McGraw-Hill Professional, 2nd edition

Bernardi S, Dranca L, Merseguer J (2013) Un enfoque guiado por el modelado para la obtención y análisis de requisitos de supervivencia de C2IS. In: I Congreso Nacional de I+D en Defensa y Seguridad, Madrid, España

Bla Bla Car (2014) Conectamos Conductores con Pasajeros para compartir coche. Sitio web: `www.blablacar.es`. Accedido el 24/11/2014

Brown A, Wallnau K (1996) Engineering of Component-Based Systems. In: Proceedings of the Second IEEE International Conference of Complex Computer Systems, IEEE, pp 414 –422

Chen H, Reid E, Sinai J, Silke A, Ganor B (2008) Terrorism informatics: Knowledge management and data mining for homeland security, vol 18. Springer

Chen H, Chiang R, Storey V (2012) Business Intelligence and Analytics: From Big Data to Big Impact. MIS quarterly 36(4):1165–1188

Chopo S, Delgado M, Medrano L, Muñoz F, Sáez C (2011) Fundamentos de administración de empresas. In: Colección Textos Docentes, Centro Universitaro de la Defensa (Zaragoza)

Davenport T, De Long D, Beers M (1997) Building Successful Knowledge Management Projects. `http://www.providersedge.com/docs/km_articles/Building_Successful_KM_Projects.pdf`

Debrauwer L, VanDerHeyde F (2011) UML2: Iniciación, ejemplos y ejercicios corregidos. ENI, 2^a Edición

DoDAF-MODAF (2012) Unified Profile for the US Department of Defense Architecture Framework (DoDAF) and the UK Ministry of Defence Architecture Framework (MODAF). Documento técnico OMG: ptc/2012-01-03

Dranca L, Bernardi S, Lozano M (2013) Desarrollo de un sistema de información para el CUD (SICUD): un caso de estudio real para reforzar el aprendizaje de técnicas de modelado en grupos auto-organizados. PIIDUZ_12_1_342

Dranca L, Bernardi S, Lozano M (2014) Herramientas colaborativas en la nube como soporte a la planificación docente y a la realización de proyectos en grupos auto-organizados. PIIDUZ_13_375

Dua S, Du X (2014) Data mining and machine learning in cybersecurity. CRC press

Díaz Osto P (2010) El Cuadro de Mando Integral, Poderosa Herramienta de Dirección del SALE. REVISTA EJÉRCITO (826):94–102

Elmasri R, Navathe S (2007) Fundamentos de Sistemas de Bases de Datos. Addison Wesley

Fernández-Alarcón V (2006) Desarrollo de sistemas de información: una metodología basada en el modelado. Ediciones UPC

Fowler, M (2004) UML Distilled: A Brief Guide to the Standard Object Modeling Language. The Addison-Wesley object technology series, Addison-Wesley

Frankel D (2003) Model Driven Architecture. Wiley

Gane C, Sarson T (1978) Structured Systems Analysis: Tools and Techniques. Prentice-Hall, Englewood Cliffs, NJ

Gómez A, Suárez C (2011) Sistemas de información. Herramientas prácticas para la gestión empresarial. Ra-Ma, 4^a Edición ampliada y actualizada

HSQLDB (2014) HyperSQL, HSQLDB 100 % Java Database. `http://hsqldb.org/`

IEEE-1220 (1998) Application and Management of the Systems Engineering Process. IEEE Computer Society

IEEE-830 (1998) IEEE Recommended Practice for Software Requirements Specifications. Std 830-1998, IEEE Computer Society

INCOSE (2006) System engineering handbook. Tech. rep., International Council on System Engineering, version 3

Inteco (2009) Ingeniería del software: metodologías y ciclos de vida. Instituto Nacional de Tecnología de la Comunicación, Laboratorio Nacional de Calidad del Software

ISO-IEC15288 (2008) System and software engineering - System life-cycle processes. International Organization for Standardization/ International Electrotechnical Commission

Jacobson I, Booch G, Rumbaugh J (1999) The Unified Software Development Process. Addison Wesley

Kelly S, Tolvanen J (2008) Model-Driven Software Development. Wiley

Larman C (2004) Applying UML and patterns. Prentice Hall, third edition

Laudon K, Laudon J (2012) Management information systems. Pearson, 12^a Edición

López Montalbán I, Castellano Pérez M, Ospino Rivas J (2011) Bases de Datos. Garceta grupo editorial

Martínez J, Olmo J, Rodríguez M, Vilariño S (2012) Introducción a la estadística. In: Colección Textos Docentes, Centro Universitaro de la Defensa (Zaragoza)

METRICA (2012) Metodología de Planificación, Desarrollo y Mantenimiento de sistemas de información. Versión 3 - Ministerio de Hacienda y Administraciones Públicas

MIL-STD-499 (1969) System Engineering Management. Department of Defence (US)

MIL-STD-499B (1993) System Engineering Management. Department of Defence (US)

Monforte Moreno M (2012) Operaciones en red: una revolución en la doctrina militar. Atenea - Seguridad y Defensa 35:46–50

Monforte Moreno M, Hinarejos Rojo A, Herrero Santos C (2010) Introducción a los sistemas de información para el mando y control militar. Ministerio de Defensa

OMG (2011) Common Object Request Broker Architecture (CORBA). Versión 3.2

Open Office Base (2015) Open Office: The Free and Open Productivity Suite. Sitio web: `http://www.openoffice.org/product/base.html`. Accedido el 15/01/2015

Silva Suárez M (1985) Las redes de Petri en la automática y la informática. Editorial AC, D.L.

Sindre G, Opdahl A (2005) Eliciting security requirements with misuse cases. Requirements Engineering 10(1):34–44

Stahl T, Vólter M (2006) Model-Driven Software Development. Wiley

Stair R, Reynolds G (2012) Principles of Information Systems. Cengage Learning, 10^a Edición

UML (2010) Unified Modeling Language: Superstructure. Documento técnico OMG: ptc/2010-11-14

UPS (2015) Shipping, freight, logistics and supply chain management from ups. Sitio web: `www.ups.com`. Accedido el 13/01/2015

Velasco M, Bernardi S, Dranca L, López P, Oller A, Sánchez A, Umpiérrez F, Vígara R (2014) Estudio de la aplicabilidad de técnicas de minería de datos para el apoyo a la toma de decisiones en la lucha antiterrorista. In: II Congreso Nacional de I+D en Defensa y Seguridad, Zaragoza, España

Visual Paradigm (2015) Software design tools for agile teams, with UML, BPMN and more. Sitio web: `http://www.visual-paradigm.com/`. Accedido el 05/01/2015

Yurcik W, Doss D (2001) Achieving Fault-Tolerant Software with Rejuvenation and Reconfiguration. IEEE Softw 18(4):48–52

Índice alfabético