Nótese que la capacidad de justificar una asociación en función de necesito-conocer depende de los requisitos; obviamente, un cambio en éstos —como que se necesite mostrar en el recibo el ID del cajero— cambia la necesidad de recordar una relación.

Basado en el análisis anterior, *podría* justificarse la eliminación de las asociaciones en cuestión.

Asociaciones necesito-conocer vs. comprensión

Un criterio necesito-conocer estricto para el mantenimiento de las asociaciones generará un "modelo de información" mínimo de lo que se necesita para modelar el dominio del problema —limitado por los requisitos actuales que se están considerando—. Sin embargo, este enfoque podría crear un modelo que no transmite (a nosotros o a los demás) una comprensión completa del dominio.

Además de ser un modelo necesito-conocer de información sobre las cosas, el modelo del dominio es una herramienta de comunicación con la que estamos intentando entender y comunicar a otros los conceptos importantes y sus relaciones. Desde este punto de vista, eliminando algunas asociaciones que no se exigen estrictamente en una base necesito-conocer, puede crear un modelo sin interés —no comunica las ideas claves y relaciones—.

Por ejemplo, en la aplicación del PDV: aunque, tomando como base las relaciones necesito-conocer de manera estricta, podría no ser necesario registrar *Venta Iniciada-por Cliente*, su ausencia deja fuera un aspecto importante para entender el dominio —que un cliente genera las ventas—.

En cuanto a las asociaciones, un buen modelo se sitúa en alguna parte entre un modelo necesito-conocer mínimo y uno que ilustra cada relación concebible. ¿El criterio básico para juzgar su valor? —¿Satisface todos los requisitos necesito-conocer y además comunica claramente un conocimiento esencial de los conceptos importantes en el dominio del problema?—.

> Céntrese en las asociaciones necesito-conocer, pero contemple las asociaciones de sólo-comprensión para enriquecer el conocimiento básico del dominio.

Capítulo 12

MODELO DEL DOMINIO: AÑADIR ATRIBUTOS

Cualquier error tardío es indistinguible de una característica.

Rich Kulawiec

Objetivos

- Identificar los atributos del modelo del dominio.
- Distinguir entre atributos correctos e incorrectos.

Introducción

Resulta útil identificar aquellos atributos de las clases conceptuales que se necesitan para satisfacer los requisitos de información de los actuales escenarios en estudio. Este capítulo explora la identificación de los atributos adecuados, y añade los atributos al modelo del dominio de NuevaEra.

12.1. Atributos

Un **atributo** es un valor de datos lógico de un objeto.

> Incluya los siguientes atributos en un modelo del dominio: aquellos para los que los requisitos (por ejemplo, los casos de uso) sugieren o implican una necesidad de registrar la información.

Por ejemplo, un recibo (que recoge la información de una venta) normalmente incluye una fecha y una hora, y la dirección quiere conocer las fechas y horas de las ventas por múltiples motivos. En consecuencia, la clase conceptual *Venta* necesita los atributos *fecha* y *hora*.

12.2. Notación de los atributos en UML

Los atributos se muestran en el segundo compartimento del rectángulo de la clase (ver Figura 12.1). Sus tipos podrían mostrarse opcionalmente.

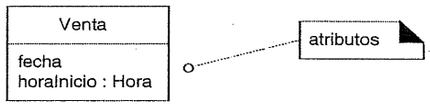

Figura 12.1. Clases y atributos.

12.3. Tipos de atributos válidos

Hay algunas cosas que no deberían representarse como atributos, sino como asociaciones. Esta sección presenta los tipos válidos.

Mantenga atributos simples

Intuitivamente, la mayoría de los atributos simples son los que, a menudo, se conocen como tipos de datos primitivos, como los números. El tipo de un atributo, normalmente, no debería ser un concepto de dominio complejo, como *Venta* o *Aeropuerto*. Por ejemplo, el siguiente atributo *registroActual* en la clase *Cajero* de la Figura 12.2, no es deseable porque su tipo tiene la intención de ser un *Registro*, que no es un tipo de atributo simple (como *Numero* o *String*). La manera más útil para expresar que un *Cajero* utiliza un *Registro* es con una asociación, no con un atributo.

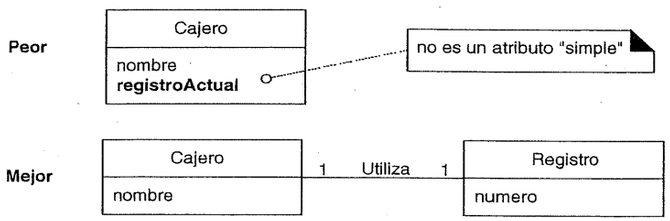

Figura 12.2. Relaciones con asociaciones, no atributos.

Los atributos en un modelo del dominio deberían ser, preferiblemente, **atributos simples** o **tipos de datos.**
Los tipos de datos de los atributos muy comunes incluyen: *Boolean, Fecha, Numero, String (Texto), Hora.*

Otros tipos comunes comprenden: *Direccion, Color, Geometrico (Punto, Rectangulo), Numero de Telefono, Numero de la Seguridad Social, Codigo de Producto Universal (UPC; Universal Product Code), SKU, ZIP o códigos postales, tipos enumerados.*

Repitiendo un ejemplo previo, un error típico es modelar un concepto del dominio complejo como un atributo. Para ilustrarlo, un aeropuerto de destino no es realmente una cadena de texto; se trata de una cosa compleja que ocupa muchos kilómetros cuadrados de espacio. Por tanto, *Vuelo* debería relacionarse con *Aeropuerto* mediante una asociación, no con un atributo, como se muestra en la Figura 12.3.

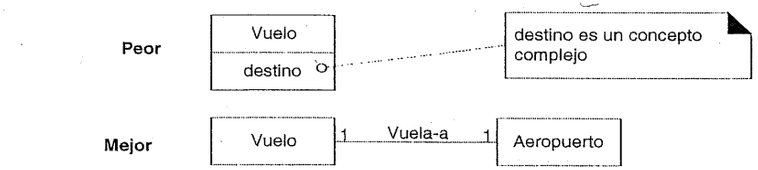

Figura 12.3. Evite la representación de conceptos del dominio complejos como atributos; utilice asociaciones.

Relacione las clases conceptuales con una asociación, no con un atributo.

Perspectiva conceptual vs. implementación: ¿qué sucede con los atributos en el código?

La restricción de que el tipo de los atributos en el modelo del dominio sea sólo un tipo de datos simple *no* implica que los atributos C++ o Java (miembros de datos, campos de una instancia) sólo deban ser de tipos de datos primitivos, o simples. El modelo del dominio se centra en declaraciones conceptuales puras sobre un dominio del problema, no en componentes software.

Posteriormente, durante el trabajo de diseño e implementación, se verá que las asociaciones entre objetos representadas en el modelo del dominio, a menudo, se implementarán como atributos que referencian a otros objetos software complejos. Sin embargo, ésta es sólo una de las posibles soluciones de diseño para implementar una asociación, y, de ahí que la decisión se deba posponer durante el modelado del dominio.

Tipos de datos

Los atributos deben ser, generalmente, **tipos de datos**. Esto es un término UML que implica un conjunto de valores para los cuales no es significativa una identidad única (en el contexto de nuestro modelo o sistema) [RJB99]. Por ejemplo, no es (normalmente) significativo distinguir entre:

- Diferentes instancias del *Numero* 5.
- Diferentes instancias del *String* 'gato'.

- Diferentes instancias del *NumeroDeTelefono* que contiene el mismo número.
- Diferentes instancias de la *Direccion* que contiene la misma dirección.

Por el contrario, es significativo distinguir (por identidad) entre dos instancias de *Persona* cuyos nombres son, en los dos casos, "Luis García", puesto que las dos instancias pueden representar individuos diferentes con el mismo nombre.

En cuanto al software, existen pocas situaciones donde uno compararía las direcciones de memoria de las instancias de *Numero, String, NumeroDeTelefono o Direccion*; sólo son relevantes las comparaciones basadas en los valores. Por el contrario, es comprensible comparar las direcciones de memoria de las instancias de *Persona*, y distinguirlas, incluso si tienen los mismos valores de los atributos, porque es importante su identidad única.

Por tanto, todos los tipos primitivos (número, string) son tipos de datos UML, pero no todos los tipos de datos son primitivos. Por ejemplo, *NumeroDeTelefono* es un tipo de dato no primitivo.

Estos valores de tipos de datos también se conocen como **objetos valor**.

La noción de tipos de datos puede ser sutil. Como regla empírica, sea fiel a la prueba básica de tipos de atributos "simples": hágalo un atributo si se considera de manera natural como un número, string, booleano, fecha u hora (etcétera); en otro caso, represéntelo como una clase conceptual aparte.

> En caso de duda, defina algo como una clase conceptual aparte en lugar de como un atributo.

12.4. Clases de tipos de datos no primitivos

El tipo de un atributo podría representarse como una clase no primitiva por derecho propio en un modelo del dominio. Por ejemplo, en el sistema de PDV hay un identificador de artículo. Generalmente se ve simplemente como un número. Así que, ¿se debería representar como una clase no primitiva? Aplique esta guía:

> Represente lo que podría considerarse, inicialmente, como un tipo de dato primitivo (como un número o string) como una clase no primitiva si:
> - Está compuesto de secciones separadas.
> - Número de teléfono, nombre de persona.
> - Habitualmente, hay operaciones asociadas con él, como análisis sintáctico o validación
> - Número de la seguridad social.
> - Tiene otros atributos.
> - el precio de promoción podría tener una fecha (efectiva) de comienzo y fin.
> - Es una cantidad con una unidad.
> - La cantidad del pago tiene una unidad monetaria.
> - Es una abstracción de uno o más tipos con alguna de estas cualidades.
> - El identificador del artículo en el dominio de ventas es una generalización de tipos como el Código de Producto Universal (UPC) o el Número de Artículo Europeo (EAN)

Aplicando estas guías a los atributos del modelo del dominio del PDV llegamos al siguiente análisis:

- El identificador del artículo es una abstracción de varios esquemas de codificación comunes, incluyendo UPC-A, UPC-E y la familia de esquemas EAN. Estos esquemas de codificación numéricos tienen subpartes que identifican al fabricante, producto, país (para EAN), y un dígito de control para validarlos. Por tanto, debería existir una clase no primitiva *ArticuloID*, puesto que satisface muchas de las guías anteriores.
- Los atributos del *precio* y *cantidad* deberían ser clases no primitivas *Cantidad* o *Moneda* porque se trata de cantidades en una unidad monetaria.
- El atributo de la *direccion* debe ser una clase no primitiva *Direccion* porque tiene secciones diferentes.

Las clases *ArticuloID*, *Direccion* y *Cantidad* son tipos de datos (no es significativa la identidad única de las instancias) pero merece la pena considerarlas como clases independientes debido a sus cualidades.

¿Dónde representamos las clases de tipos de datos?

¿Debería mostrarse la clase *ArticuloID* como una clase conceptual independiente en el modelo del dominio? Depende de lo que quiera resaltar en el diagrama. Puesto que *ArticuloID* es un *tipo de datos* (no es importante la identidad única de las instancias), se podría mostrar en el compartimento de los atributos del rectángulo de la clase, como se muestra en la Figura 12.4. Pero, puesto que es una clase no primitiva, con sus propios atributos y asociaciones, podría ser interesante mostrarla como una clase conceptual en su propio rectángulo. No existe una respuesta correcta; depende de cómo se esté utilizando el modelo del dominio como herramienta de comunicación, y la importancia de los conceptos en el dominio.

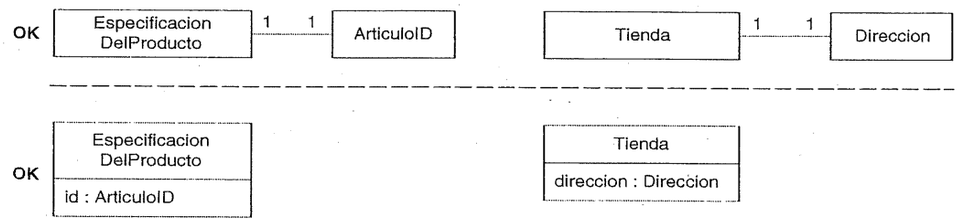

Figura 12.4. Si la clase del atributo es un tipo de datos, podría mostrarse en el rectángulo del atributo.

Un modelo del dominio es una herramienta de comunicación; las elecciones sobre lo que se muestra deben hacerse con esa consideración en mente.

12.5. Deslizarse al diseño: ningún atributo como clave ajena

No se deberían utilizar los atributos para relacionar las clases conceptuales en el modelo del dominio. La violación más típica de este principio es añadir un tipo de **atributo de clave ajena**, como se hace normalmente en el diseño de bases de datos relacionales, para asociar dos tipos. Por ejemplo, en la Figura 12.5, no es deseable el atributo *numeroRegistroActual* en la clases *Cajero* porque el propósito es relacionar el *Cajero* con un objeto *Registro*. La mejor manera de expresar que un *Cajero* utiliza un *Registro* es con una asociación, no con un atributo de clave ajena. Una vez más, relacione los tipos con una asociación, no con un atributo.

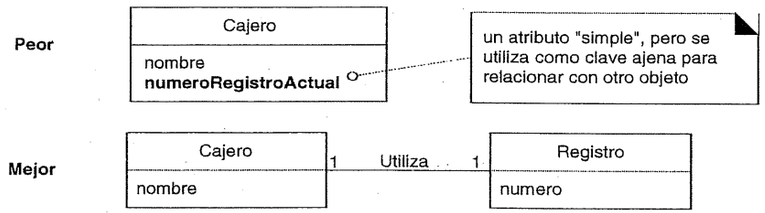

Figura 12.5. No utilice atributos como claves ajenas.

Hay muchas formas de relacionar objetos —siendo las claves ajenas una de ellas— y pospondremos el modo de implementar la relación hasta el diseño, para evitar el **deslizamiento al diseño**.

12.6. Modelado de cantidades y unidades de los atributos

La mayoría de las cantidades numéricas no deberían representarse simplemente como números. Considere el precio o la velocidad. Son cantidades con unidades asociadas, y es habitual que se necesite conocer la unidad y dar soporte a las conversiones. El software del PDV NuevaEra es para un mercado internacional y necesita soportar los precios en diferentes monedas. En el caso general, la solución consiste en representar la *Cantidad* como una clase conceptual aparte, con una *Unidad* asociada [Fowler96]. Puesto que las cantidades se consideran tipos de datos (no es importante la identidad única de las instancias), es aceptable recoger su representación en la sección de atributos del rectángulo de clase (ver Figura 12.6). También es común mostrar especializaciones de *Cantidad*. El *Dinero* es una clase de *Cantidad* cuyas unidades son monedas. El *Peso* es una cantidad cuyas unidades son kilogramos o libras.

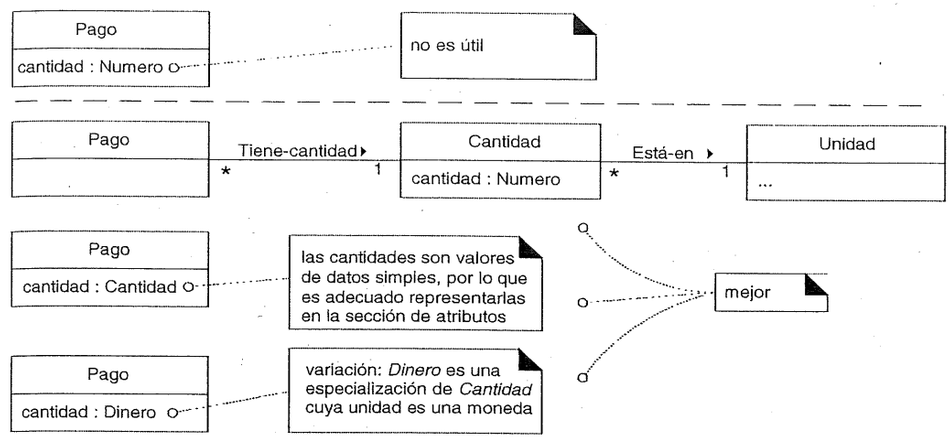

Figura 12.6. Modelado de cantidades.

12.7. Atributos en el Modelo del Dominio de NuevaEra

Los atributos elegidos reflejan los requisitos de esta iteración —los escenarios de *Procesar Venta* de esta iteración—.

Pago	*cantidad:* Se debe capturar una cantidad (también conocida como "cantidad entregada") para determinar si se proporciona el pago suficiente y calcular el cambio.
EspecificacionDelProducto	*descripcion:* Para mostrar la descripción en una pantalla o recibo.
	id: Para buscar una *EspecificacionDelProducto*, dado un articuloID introducido, es necesario relacionarlas con un *id*.
	precio: Para calcular el total de la venta y mostrar el precio de la línea de venta.
Venta	*fecha, hora:* Un recibo es un informe en papel de una venta. Normalmente muestra la fecha y la hora de la venta.
LineaDeVenta	*cantidad:* Para registrar la cantidad introducida, cuando hay más de un artículo en una línea de venta (por ejemplo, *cinco* paquetes de tofu).
Tienda	*direccion, nombre:* El recibo requiere el nombre y la dirección de la tienda.

12.8. Multiplicidad de la LineaDeVenta al Articulo

Es posible que el cajero reciba un grupo de artículos iguales (por ejemplo, seis paquetes de tofu), introduzca el *articuloID* una vez, y después introduzca una cantidad (por