microcódigos

diciembre 7, 2009

Asociaciones en UML

Filed under: UML y patrones — David Liñán @ 5:54 pm
Uno de los elementos del lenguaje *UML* que más llevan a confusión son las asociaciones: cuándo usarlas durante el modelado y qué representan exactamente. Con frecuencia, cuando se intentan "traducir" asociaciones a un lenguaje de programación como *Java* o *C#* se piensa en un puntero o una referencia a un objeto. Pero con un puntero no hay forma de saber desde el objeto accedido nada del objeto que lo referencia. Esta es la diferencia con una asociación: que ambos objetos tienen una referencia del otro.

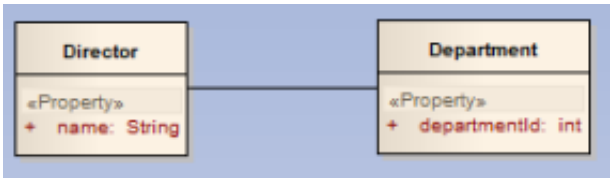
Asociaciones y otras relaciones

Hay varios tipos de relaciones en UML y como su propio nombre indica, sirven para relacionar elementos. En función de su uso hay cuatro tipos de relación, que se enumeran brevemente a continuación:

- *Dependencias*: indican que un elemento utiliza a otro, ya sea invocando sus métodos o accediendo a sus propiedades. Las dependencias definen por tanto una relación de uso. Se representan mediante una flecha discontínua desde el elemento principal al elemento usado.
- *Generalización*: es una relación entre un elemento general y otro que define un caso más específico. O si se prefiere, una relación de herencia. Se representa con una flecha con la cabeza hueca apuntando al elemento más general, siendo la línea sólida.
- *Realización*: es una relación entre un elemento general abstracto y otro más concreto. Por ejemplo, una implementación de un interfaz o una clase abstracta. Se representa como en el anterior caso, pero siendo la línea discontínua.
- *Asociaciones*: representan relaciones estructurales entre elementos de un mismo nivel. Ambos elementos están relacionados de una forma más íntima y es posible navegar de uno a otro. Se

representan con una línea sólida que une ambos elementos. Este es el tipo de relación objeto de este post.

Las asociaciones en *UML* se usan por tanto cuando se quiere definir una relación estructural en el modelo: pedidos que contienen productos, empleados que trabajan en departamentos, estudiantes que asisten a cursos… En todos esos casos es necesario poder navegar entre los elementos relacionados (qué empleados trabajan en qué departamento y viceversa, qué productos tiene un pedido, etc…).



Tipos de asociación

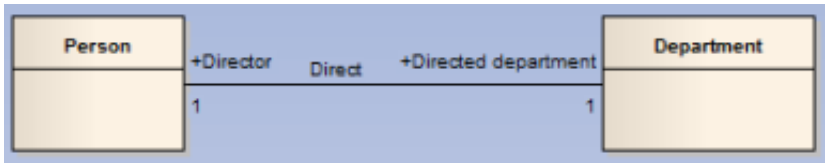
Las asociaciones tienen algunos adornos o atributos, que siempre están presentes en cualquier herramienta *UML*. Estos atributos son los siguientes:

- Nombre: describe de una forma más concreta la naturaleza de la relación entre los objetos. No suele ser necesario, salvo que haya varias asociaciones: por ejemplo, 'trabaja en' o 'dirige' para una asociación entre empleados y departamentos.
- Roles: definen el nombre de un objeto desde el otro. Por ejemplo, 'detalles' y 'pedido' para una relación entre un pedido y varios productos (estos nombres coinciden con las variables definidas en el código).
- Mutiplicidad: define el número de objetos de ambos tipos. Puede ser 1-1 (un director dirige un departamento), 1-n (un departamento tiene varios empleados) o n-n (varios empleados trabajan en varios proyectos).

Las asociaciones se clasifican en función de su multiplicidad y las tres posibilidades antes mencionadas se exponen a continuación con más detalle.

Asociaciones 1-1

Son aquellas que relacionan exactamente dos elementos. Por ejemplo, un empleado dirige un departamento. Este tipo de relaciones se implementan a nivel de código incluyendo en cada clase un atributo del tipo de la otra.

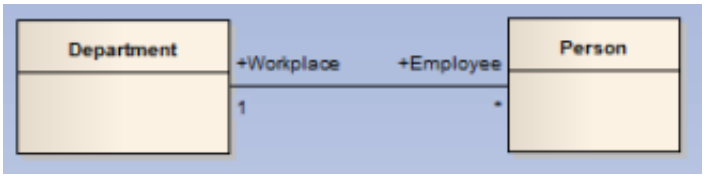


¿Cómo se traduciría esto en código *Java*? Cada clase tendría una propiedad del tipo de la otra clase, a través de la cual accederíamos al objeto asociado.

```
1  public class Person {
2    private Department directedDepartment;
3
4    public Department getDepartment() {
5      return directedDepartment;
6    }
7
8    public void setDepartment(Department department) {
9      this.directedDepartment= department;
10   }
11   ...
12 }
13
14 public class Department {
15   private Person director;
16
17   public Person getDirector() {
18     return director;
19   }
20
21   public void setDirector(Person director) {
22     this.director = director;
23   }
24   ...
25 }
```

Asociaciones 1-n

En este tipo de relaciones hay un elemento que se asocia a varios de otro tipo: un departamento tiene varios empleados. La única diferencia a nivel de código es que ahora el elemento a la izquierda (*Department*) tendrá una colección de elementos del otro tipo (*Person*).



El código fuente para este tipo de asociación se muestra a continuación. Para la clase *Person* no varía de forma significativa, pero en *Department* tenemos ahora una colección:

```
1  public class Person {
2    private Department workplace;
3
4    public Department getWorkplace() {
5      return workplace;
6    }
```

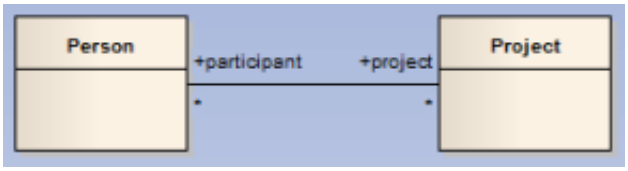
```
  7
  8     public void setWorkplace(Department workplace) {
  9       this.workplace = workplace;
 10     }
 11     ...
 12   }
 13
 14   public class Department {
 15     private ArrayList employees = new ArrayList();
 16
 17     public void AddEmployee(Person employee) {
 18       employees.add(employee);
 19     }
 20
 21     public void RemoveEmployee(Person employee) {
 22       employees.remove(employee);
 23     }
 24
 25     public Person GetEmployee(int index) {
 26       return (Person) employees.get(index);
 27     }
 28
 29     public int GetEmployeeCount() {
 30       return employees.size();
 31     }
 32     ...
 33   }
```

Cuando se tiene una multiplicidad de *1-n* hay dos opciones para representar los elementos del extremo múltiple:

- Set: cada elemento sólo puede aparecer una vez. Es el caso del ejemplo, donde cada empleado está en un departamento sin repetirse.
- Set ordenado: igual que antes, pero los elementos aparecen ordenados.

Asociaciones n-n

Se trata de relaciones de múltiples elementos de un tipo con múltiples elementos de otro. Por ejemplo, varios empleados pueden estar asignados a más de un proyecto en un momento dado.



En este caso se tienen ahora colecciones en ambas clases para asociar los elementos de la otra clase:

```
 1   public class Person {
 2     private ArrayList projects = new ArrayList();
```

```
3
4     public void AssignProject(Project project) {
5       projects.add(project);
6     }
7
8     public void DesassignProject(Project project) {
9       projects.remove(project);
10    }
11
12    public Project GetProject(int index) {
13      return (Project) projects.get(index);
14    }
15
16    public int GetProjectCount() {
17      return projects.size();
18    }
19    ...
20  }
21
22  public class Project {
23    private ArrayList team = new ArrayList();
24
25    public void AddMember(Person member) {
26      team.add(member);
27    }
28
29    public void RemoveMember(Person member) {
30      team.remove(member);
31    }
32
33    public Person GetMember(int index) {
34      return (Person) team.get(index);
35    }
36
37    public int GetMemberCount() {
38      return team.size();
39    }
40    ...
41  }
```

Cuando se tiene una multiplicidad de *n-n* hay dos opciones para representar los elementos:

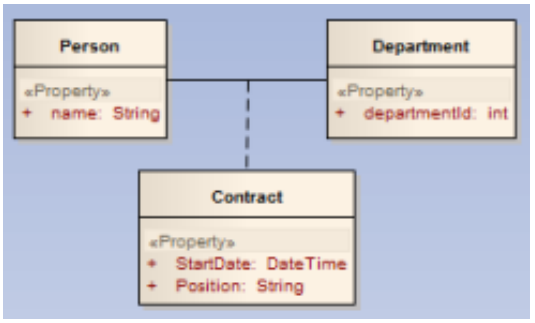
- Bag: los elementos pueden repetirse. Por ejemplo, un empleado podría ser reasignado al mismo proyecto más de una vez en distintas fechas. El otro extremo de la relación también sería de tipo *Bag* o bien *Sequence*.

- Sequence: al igual que antes los elementos pueden repetirse, pero aparecen ordenados. El otro extremo de la relación también sería de tipo *Sequence* o *Bag*.

Otros tipos de asociación

Clases asociación

En una asociación entre dos clases la propia asociación puede tener propiedades. Por ejemplo, entre un empleado y una compañía pueden considerarse propiedades como salario, cargo, fecha de alta en la compañía, etc…que realmente no pertenecen al empleado y tampoco a la compañía. En este caso se usa una clase asociación, representada con un símbolo de clase unido por línea discontínua a una asociación. En el siguiente ejemplo la clase asociación *Contract* relaciona a una persona con un departamento:



El código fuente para implementar la clase asociación incluye propiedades del tipo de ambas clases relacionadas (*Person* y *Department*), como se muestra en el listado:

```
1  public class Contract {
2    private Person person;
3    private Department department;
4
5    public Contract(Person person, Department department) {
6      this.person = person;
7      this.department = department;
8    }
9    ...
10 }
```

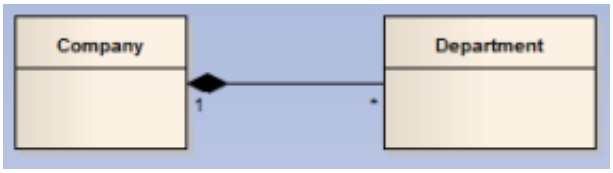
```
1  public class Person {
2    private String name;
3    private String surname;
4    ...
5  }
```

```
1  public class Department {
2    private int Id;
3    private String Name;
4    ...
```

```
5 }
```

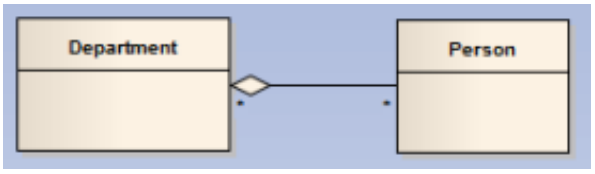
Composiciones y agregaciones

Son tipos especiales de asociaciones; en ambos casos uno de los elementos se considera como un 'todo' que contiene a otros elementos, los cuales estructuralmente son 'partes'. Una composición se representa con un rombo relleno pegado al elemento contenedor:



La composición supone una relación fuerte de pertenencia y ciclo de vida entre el elemento contenedor y las partes. El contenedor se ocupa del ciclo de vida (creación y destrucción) de las partes, que solamente pueden pertenecerle a él (de lo que se deduce que una composición no podría tener nunca multiplicidad *n*-*n*). Un ejemplo es la asociación entre una empresa y sus departamentos: éstos no podrían existir de forma independiente, no tendría sentido.

La agregación en cambio no vincula tan fuertemente el elemento contenedor de las partes y el ciclo de vida de ambos es independiente. Se representa con un rombo hueco. Un ejemplo es la asociación entre departamento y empleados: éstos existen de forma independiente y pueden pasar de un departamento a otro a lo largo del tiempo.



1 Comentario »

1. exelente

   Comentario por Emanuel — abril 5, 2010 @ 7:30 pm

   Responder

RSS (Really Simple Syndication) feed para los comentarios de esta entrada. TrackBack URI (Uniform Resource Identifier)

El tema Silver is the New Black. Blog de WordPress.com.

Seguir

Seguir “microcódigos”

Ofrecido por WordPress.com