Programación 4

Un resumen de conceptos

Compilado por Liber Dovat el 5 de marzo de 2011
Con el aporte de Gonzalo Cedrés y Pablo Yaniero

Versión 1.7

Índice general

Análisis

1.1. Elementos

Orientación a objetos Puede ser entendida como una forma de pensar basada en abstracciones de conceptos existentes en el mundo real.

Análisis orientado a objetos Consiste en considerar el dominio de la aplicación y su solución lógica en términos de objetos (cosas, conceptos, entidades).

Objetivo-1 Busca de modelar el dominio del problema para comprender mejor el contexto del problema y para obtener una primera *aproximación* a la estructura de la solución.

Objetivo-2

- Modelar el dominio el problema
  Para comprender mejor el contexto del problema.
  Para obtener una primera *aproximación* a la estructura de la solución.
- Especificar el comportamiento del sistema
  Para contar con una descripción mas precisa de que es lo que se espera del sistema.

Actividades en el análisis

- Modelado de dominio
  Consiste en encontrar y describir los objetos (o conceptos) en el dominio de la aplicación.
- Especificación del comportamiento del sistema
  Consiste en entender a cada caso de uso en términos de intercambios de mensajes entre los actores y el sistema, y en especificar el comportamiento de cada uno de esos mensajes (Pero sin decir como funcionan).

1.2. Conceptos básicos

Concepto Es una idea, cosa u objeto.

Clase Es un descriptor de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y comportamiento.

Objeto Es una entidad discreta con limites e identidad bien definidos.
Encapsula estado y comportamiento.
Es una instancia de una clase.

Identidad Es una propiedad inherente de los objetos de ser distinguibles de todos los demás.
Dos objetos son distintos aunque tengan exactamente los mismos valores en sus propiedades.

Atributo Es una descripción de un comportamiento de un tipo especificado dentro de una clase.
Puede ser:

De instancia:
: cada objeto de esa clase mantiene un valor de ese tipo en forma independiente.

De clase:
: Todos los objetos de esa clase comparten un mismo valor de ese tipo.

Operación Es una *especificación* de una transformación o consulta que un objeto puede ser llamado a ejecutar.
Tiene asociada un nombre, una lista de parámetros y un tipo de retorno.

Método Es la *implementación* de una operación para una determinada clase.
Especifica el algoritmo o procedimiento que genera el resultado o efecto de la operación.

Polimorfismo Es la capacidad de asociar diferentes métodos a la misma operación.

Interfaz Es un conjunto de operaciones al que se le aplica un nombre.

Datatype Es un descriptor de un conjunto de valores que *carecen de identidad*.

Datavalue Es un valor único que carece de identidad.
Es la instancia de un Datatype.
Típicamente son usados como valores de atributos.

Referencia Es un valor en tiempo de ejecución que es *void* o *attached*.

- Si es attached la referencia identifica a un único objeto.
- Si es void la referencia no identifica a ningún objeto.

Tipo estático y dinámico El tipo estático de un objeto es el tipo del cual fue declarada la referencia adjunta a él; se conoce en tiempo de compilación.
El tipo dinámico es el tipo del cual es instancia directa.
En ciertas situaciones ambos tipos coinciden por lo que pierde sentido realizar tal distinción.

Asociación Describe una relación semántica entre clasificadores.
es una relación entre conceptos que indica alguna conexión interesante o significativa entre ellos.

Generalización Es una relación taxonómica entre un elemento más general y un elemento más específico.
El elemento más específico es consistente con el más general y puede tener información adicional.

Realización Es una relación entre una especificación y su implementación. Por ejemplo, entre una interface y una clase.

Tipo asociativo Es un elemento que es tanto clase como asociación.
Agrega propiedades a las asociaciones.

Roles Especifican el papel que juegan las clases en una asociación.

Cuando utilizar el rol Se utiliza para eliminar la ambigüedad cuando existen mas de dos asociaciones entre dos clases.

Agregación Significa que un elemento es parte de otro.
Existen dos variantes:

- Compartida:
  Es una agregación en la que las partes no son exclusivas del compuesto. Las partes pueden estar incluidas en otros compuestos.
- Compuesta:
  Es una agregación en la que las partes son exclusivas del compuesto. Generalmente una acción sobre el compuesto se propaga a las partes (típicamente en la destrucción).

Subsumption Es una propiedad que deben cumplir todos los objetos, también conocida como *intercambiabilidad*.
Un objeto de clase base puede ser sustituido por un objeto de clase derivada (directa o indirecta).
Por lo tanto: $b : B \wedge B <: A \Longrightarrow b : A$

Acceso a propiedades La propiedades de una clase tienen aplicadas calificadores de acceso:

- Public:
  Puede se accedida desde cualquier punto desde el cual se tenga visibilidad sobre el objeto.
- Private:
  Puede ser accedida solamente desde los métodos de la propia clase.

Por defecto, los atributos deben ser privados y las operaciones públicas.

Descriptores

Full descriptors:
: Es la descripción completa que es necesaria para describir a un objeto. Contiene la descripción de todos los atributos, operaciones y asociaciones que el objeto contiene.

Segment descriptor:
: Son los elementos que efectivamente se declaran en un modelo o en el código (por ejemplo, clases) y contienen las propiedades heredables que son: Los atributos, las operaciones, los métodos y la participación en asociaciones.

Herencia Es el mecanismo por el cual se permite compartir propiedades entre una clase y sus descendientes.

Clase abstracta Ningún objeto puede ser creado directamente a partir de ellas.
No son instanciables.
Existen para que otras hereden las propiedades declaradas por ellas

Operación abstracta Es una operación en una clase sin método.

Modelo Es una abstracción de un sistema desde un punto de vista determinado: Funcionalidad, estructura, lógica, estructura física, etc.

Relación entre caso de uso, escenario y DSS Para un caso de uso pueden existir varios escenarios (típicos y alternativos).
Cada escenario de caso de uso se puede representar con un DSS.

Relación entre caso de uso y colaboración Una colaboración realiza uno o más casos de uso.

1.3. Modelo de dominio

Modelos de las actividades Durante el modelado de dominio, se construye el modelo del dominio, mientras que en la especificación del comportamiento del sistema se completa el modelo de casos de uso.

Invariante Es un predicado que expresa una condición sobre los elementos del modelo de dominio y que siempre debe ser verdadero. Invariantes habituales:

- Unicidad de atributos (identificación de instancias):
  Un atributo tiene un valor único dentro del universo de instancias de un mismo tipo (una instancia es identificada por ese valor).
- Dominio de atributos:
  El valor de un atributo pertenece a cierto dominio.
- Integridad circular:
  no puede existir circularidad en la navegación.
- Atributos calculados:
  El valor de un atributo es calculado a partir de la información contenida en el dominio.
- Reglas de negocio:
  Invariante que restringe el dominio del problema.

1.4. OCL

Colección de objetos Serán tratados como meros contenedores de objetos. Proveerán solamente operaciones que permitan administrar los objetos contenidos.
En general, las interfaces de diccionario (add, remove, find, member, etc.) e iterador (next, etc.) son suficientes.

Subtipos Se distinguen tres subtipos de collection: Set, Bag y Sequence.
Se corresponden con los tipos abstractos conjunto, bolsa y secuencia respectivamente.
Un valor de set es una colección de elementos donde ellos no se repiten y no existe un orden entre ellos.
Un valor de bag es una colección de elementos donde ellos se pueden repetir pero no existe un orden entre ellos.
Un valor de sequence es una colección de elementos donde ellos se pueden repetir y existe un orden entre ellos.

Ejemplo 1

```
context Etapa inv:
self.Pareja->forAll(p|p.concurso = self.concurso)
```

Ejemplo 2

```
context Famoso inv:
self.Pareja->forAll(s1,s2| s1 <>s2 impies s1.concurso <>s2.concurso)
```

Ejemplo 3

```
context Aplicacion inv:
Aplicacion.allInstances()->IsUnique(id)
```

Ejemplo 4

```
context Empleado inv:
self.trabaja.horas->sum() <= 10
```

Ejemplo 5

```
context Vendedor inv:
self.empresa.producto->includes(self.producto)
```

1.5. Casos de uso

1.5.1. Descripción

Narra la historia completa (junto a todas sus variantes) de un conjunto de actores mientras usan el sistema.

1.5.2. Diagrama de secuencia del sistema

Es un artefacto incluido en el modelo de casos de uso, que define e ilustra la interacción entre los actores y el sistema en uno o varios escenarios de un caso de uso.
Definen la conversación entre los actores y el sistema, enfocándose en los mensajes que el sistema recibe.

Elementos del DSS

Incluye una instancia representando al sistema, una instancia por cada actor que participe y los mensajes enviados entre ellos para el escenario del caso de uso que corresponda.

- Incluir una instancia que represente al sistema como una unidad.
- Identificar cada actor que participe en el escenario considerado e incluir una instancia para cada uno.

- De la descripción del caso de uso, identificar aquellos eventos que los actores generen y sean de interés para el sistema e incluir cada uno de ellos como un mensaje.
- Opcionalmente, incluir junto a cada mensaje una descripción.

1.5.3. Contratos de software

Especifican declarativamente mediante pre y post condiciones el comportamiento o efecto de una operación.
Existe un contrato de software por cada operación del DSS.
El Consumidor se compromete a satisfacer la precondición al invocar la operación.
El Proveedor se compromete a satisfacer la postcondición al finalizar la operación solamente cuando la precondición fue satisfecha al momento de la invocación.

Precondición Es a lo que debe acceder el consumidor para obtener el resultado deseado.
Es lo que debe exigir el proveedor para llegar al resultado.
Especifican los valores de los parámetros de la operación y el estado del sistema antes de ejecutar la operación, en términos de:

- Que un objeto existe.
- Que un objeto no existe.
- Que un link existe.
- Que un link no existe.
- Propiedades sobre valores de atributos de objetos.

Postcondición Es a lo que accederá el consumidor.
Es a lo que se compromete el proveedor.
Especifican el valor de retorno de la operación y el estado del sistema luego de ejecutar la operación, en términos de:

- Que un objeto existe.
- Que un objeto no existe.
- Que un link existe.
- Que un link no existe.
- Especificar el valor de retorno.

Memoria del sistema Son los datos que el sistema debe guardar temporalmente mientras se esté ejecutando un caso de uso.
Representa el estado de la conversación entre usuario y sistema.

Diseño

2.1. Elementos

Diseño orientado a objetos Consiste en definir objetos lógicos (de software) y la forma de comunicación entre ellos para una posterior programación.

Arquitectura lógica Conjunto de componentes lógicos relacionados entre si, con responsabilidades específicas.

2.2. Modelo de diseño

2.2.1. Descripción

Es una abstracción de la solución lógica al problema.

Controladores Es una clase que implementa las operaciones del sistema.
Se destacan tres tipos de controladores:

- Fachada
  Contiene todas las operaciones del sistema.
- Caso de uso
  Contiene todas las operaciones de un caso de uso.
- Mini fachada
  Contiene operaciones de varios casos de uso relacionados.

Criterios GRASP Son criterios que ayudan a resolver el problema de asignar responsabilidades.
Sugieren a quien asignar responsabilidades:

Expert:
Responsabilizar a quien tenga la información necesaria.

Creator:
A quien responsabilizar de la creación de un objeto.

Bajo acoplamiento:
Evitar que un objeto interactúe con demasiados objetos.

Alta cohesión:
Evitar que un objeto haga demasiado trabajo.

No hables con extraños:
Asegurarse que un objeto realmente delega trabajo.

Controller:
A quien responsabilizar de ser el controlador.

Clases fuertes

Clases débiles Son clases de menor importancia a cuyas instancias se accede a través de alguna instancia de clase fuerte.

Manejador Es una clase singleton que contiene el universo de instancias de cierta clase y operaciones para la manipulación de ese universo.

Las operaciones pueden ser CRUD (create, retrieve, update y delete) y operaciones mas complejas que involucren *solamente* las instancias fuertes del manejador.

Permiten que los controladores no dependan entre si para acceder a instancias fuertes ni que deban compartir colecciones de esas instancias.

Interfaces del sistema Buscan quebrar las dependencias entre los elementos de la capa de presentación que invocan operaciones del sistema y los controladores de la capa lógica que las implementan.
Contienen las operaciones del sistema que son implementadas por los controladores.

Fábrica Es un objeto que tiene la responsabilidad de crear instancias que realicen una interfaz determinada.

Cómo diseñar una colaboración

- Definir la estructura y luego generar las diferentes interacciones respetando la estructura.
- Definir las interacciones (según GRAPS) y luego definir la estructura necesaria para que ocurran las interacciones.

Visibilidad Es la capacidad de un objeto de tener referencia a otro.
Existen cuatro formas básicas de que un objeto A tenga visibilidad sobre otro B:

- Por atributo:
  B es un pseudo atributo de A.
- Por parámetro:
  B es un parámetro de un método de A.
- Local:
  B es declarado localmente en un método de A.
- Global:
  B es visible en forma global.

2.3. Diagrama de comunicación

Descripción Son artefactos mediante los cuales se expresarán las interacciones.

2.4. Patrones de diseño

2.4.1. Descripción

Explica un diseño general que se aplica a un problema de diseño a objetos.

Strategy Busca definir una familia de algoritmos, encapsularlos y hacerlos intercambiables.
Esto permite que el algoritmo varíe dependiendo del cliente que lo utiliza.

Composite Componer objetos en estructuras arborescentes para representar jerarquía de objetos compuestos y tratar uniformemente los mismos.

Singleton Asegurar que una clase tenga una sola instancia y proveer un acceso global a ella.

State Permite que un objeto varíe su comportamiento cuando su estado interno cambie.
El objeto parecerá haber cambiado de clase.

Proxy

Template method

Observer

Adapter

Implementación

3.1. Elementos

Implementación orientada a objetos Consiste en codificar en un lenguaje de programación orientada a objetos los mecanismos definidos en el diseño.

3.2. Ejemplos de código

Atributo

```
En el .h:
   static int max;

En el .cc:
   int <clase>::max = 0;

------------------------

Dato::Dato(int A, int B){

   key_a       = A;
   this->key_b = B;

} // Constructor
```

Singleton

```
En el .h:

   public:
      static <clase>* getInstance();

   private:
      static <clase>* instance;
      <clase>(); // constructor

En el .cc:
```

```
<clase>* <clase>::instance = 0; // igual a null

<clase>* <clase>::getInstance(){
   if(instance == 0)
      instance = new <clase>();

   return instance;
} // getInstance
```

Dynamic cast

```
bool IntKey::Equals(IKey *sk){

   IntKey *k = dynamic_cast<IntKey>(sk);
   if (k != null)
      return k->key == key;

   return false;

} // equals
```

Interface

```
class Interface {

   public:
      virtual void operacion1()       = 0;
      virtual bool operacion2(bool A) = 0;
      ...
      virtual ~Interface(); // no es virtual puro

}; // interface
```

Subclase

```
class Dato: public ICollectible { // es igual para ↘
    → implementar una interface

   public:
      int key_a;
      int key_b;

}; // Dato
```

Iterador

```
void Print(ICollection *col){

   Iterator *li = col->getIterator();
```

```
    while(li->hasNext()){
       Data_t *elem = (Data_t *)(li->getCurrent());
       cout << elem << endl;
       li->next();
    } // while

    delete li;

} // Print
```