LAS ESPECIFICACIONES ALGEBRÁICAS: UNA APUESTA POR LA ORIENTACIÓN A OBJETOS

Nicolás Padilla[1]

[1]*Dpto. Lenguajes y Computación. U. de Almería*
e-mail: npadilla@ualm.es

Resumen: La construcción de software orientado a objetos está siendo analizada para su incorporación en las fases iniciales de la enseñanza de la programación. Dada su facilidad de uso, junto con otras características como su correctitud, robustez, extendibilidad y reusabilidad, hacen de esta técnica un candidato único para este propósito. Sin embargo, su utilización exige revisar los métodos de especificación que se están empleando en la actualidad. En este documento, proponemos un método de especificación para Tipos de datos abstractos, las Especificaciones algebráicas, que podría utilizarse como base para la enseñanza de la construcción de software orientada a objetos.

1.- INTRODUCCIÓN

El propósito de la Ingeniería del software es encontrar la forma de construir software de calidad. La construcción de software orientada a objetos persigue este objetivo, con factores como correctitud, robustez, extendibilidad y reusabilidad.

La enseñanza de la construcción de software orientada a objetos es un tema que está siendo abordada en la curricula de muchos planes de estudios. Dicha materia aparece principalmente como asignaturas optativas en cursos avanzados de enseñanza de la programación. Sin embargo, dada su facilidad de uso junto con otros factores ya comentados, se está estudiando el hecho de incluir esta materia en fases tempranas de la enseñanza de la programación. Esto ha quedado ya reflejado en las jornadas de Jenui'98 donde hubo un debate sobre este tema. También se está viendo avalado por una propuesta de reforma en la troncalidad de los estudios de Ingeniero técnico en informática de Sistemas y Gestión donde aparece esta

materia en el bloque temático de Metodología y tecnología de la programación. Sin embargo, la incorporación de esta nueva perspectiva del software nos plantea diversas cuestiones que debemos analizar cuidadosamente. Así, ¿debemos comenzar la enseñanza de la programación directamente con la Orientación a objetos?. ¿Son útiles los métodos de especificación utilizados actualmente en la enseñanza de la Orientación a objetos?. ¿Existen técnicas de especificación para la Orientación a objetos?.

Para abordar estas cuestiones, en los siguientes apartados examinaremos las ventajas de la especificación como paso previo a la implementación. Posteriormente, examinaremos algunas de las técnicas de especificación que están siendo utilizadas actualmente en la enseñanza y en la investigación, y por último, analizaremos un método de especificación formal, las Especificaciones algebráicas, que podría utilizarse como base para la enseñanza de la construcción de software orientada a objetos.

2.- LA ESPECIFICACIÓN EN LA ENSEÑANZA DE LA PROGRAMACIÓN

Especificar consiste en esencia en contestar a la pregunta ¿qué hace el programa?, sin dar detalle alguno sobre cómo se consigue dicho efecto, que corresponde a la implementación. Separar estos dos aspectos es una de las tareas más importantes del buen programador. La especificación permite descomponer la tarea de razonar en unidades manejables que corresponden a la estructura modular del programa. Para ello, la especificación ha de ser lo suficientemente *precisa* para responder a cualquier pregunta sobre el uso del algoritmo sin tener que acudir a la implementación del mismo en busca de la respuesta. La mayoría de las especificaciones que se escriben en la práctica industrial de la programación están hechas en lenguaje natural. El lenguaje natural no permite la precisión requerida si no es sacrificando otra propiedad que ha de tener toda buena especificación: la *brevedad*. La consecuencia de ambos objetivos, precisión y brevedad, requiere una *especificación formal* ([Wing, J. M., 1990]). Debido a su base matemática, la especificación formal es más concisa que una informal ya que proporciona la única forma de tratar con pruebas rigurosas de propiedades y con la correctitud de sistemas software.

Una forma de clasificar los lenguajes de especificación formal divide a éstos en *orientados al modelo, orientados al tipo y orientados a propiedad* ([Wirsing M., 1994]). Usando un método orientado al modelo, un especificador define el comportamiento de un sistema directamente construyendo un modelo del sistema en términos de estructuras matemáticas

como túplas, relaciones, funciones, conjuntos y secuencias. Algunos ejemplos son VDM, Z, redes de petri o TSL. Los métodos orientados al tipo se basan en teorías de tipos; o sea una lógica con una estructura rica de tipos. Usando un método orientado a propiedad, una especificación define el comportamiento del sistema indirectamente presentando un conjunto de propiedades (restricciones), habitualmente en la forma de axiomas, que el sistema debe satisfacer. Los axiomas se expresan en alguna lógica, generalmente una extensión de la lógica ecuacional. Este método se divide a su vez en axiomáticos y algebráicos. El método axiomático, ampliamente utilizado en los primeros cursos de la enseñanza de la programación, usa lógica de predicados de primer orden con pre-postcondiciones para especificar operaciones. O sea, se buscan aserciones que tengan que cumplirse antes y después de la ejecución de ese programa ({P} S {Q} siendo S un programa). Se utiliza por tanto una abstracción procedural. Ejemplos de métodos axiomáticos son OBJ, Anna y Larch. En el método algebráico, los tipos de datos son vistos como algebras y los axiomas se usan para presentar propiedades de las operaciones de los tipos de datos; por tanto, se usa abstracción de datos. Ejemplos de métodos algebráicos son Clear, Act One, Logica temporal, LOTOS, etc.

Las técnicas orientadas al modelo y las algebráicas sirven para especificar formalmente el comportamiento de componentes software, en particular de *Tipos de datos abstractos* (TDAs). La diferencia principal entre ambas técnicas concierne a la forma en que se da la semántica de las operaciones. En las especificaciones orientadas al modelo, se elige un objeto abstracto para representar el tipo de dato a ser especificado (p.e., una secuencia para representar una pila). La semántica de las diversas operaciones se dan especificando el efecto de estas operaciones sobre los objetos abstractos. Esto se puede indicar mediante pre-postcondiciones de la operación. Si el tipo a ser especificado puede ser fácilmente modelizado con algún constructor de tipos proporcionado, la especificación será muy natural. Sin embargo, si el nuevo tipo tiene alguna propiedad que no puede ser fácilmente expresado en el modelo abstracto, entonces la especificación llegará a ser compleja. La semántica en las especificaciones algebráicas se dan a través de relaciones entre las operaciones. Estas relaciones se llaman axiomas. Las especificaciones algebráicas no tienen el problema de las técnicas orientadas al modelo al no utilizar estructuras matemáticas para representar el nuevo tipo. Sin embargo, la especificación algebráica puede tener problemas de consistencia y completitud en el conjunto de axiomas del tipo especificado.

3.- LAS ESPECIFICACIONES ALGEBRÁICAS COMO BASE PARA LA CONSTRUCCIÓN DE SOFTWARE ORIENTADO A OBJETOS

a) La especificación algebráica de tipos de datos abstractos.

La *especificación algebráica* es una técnica formal para especificar *Tipos de datos abstractos (TDAs)* ([Peña, R., 1998]). El objetivo de la misma es definir el conjunto de valores del tipo y el efecto de cada una de las operaciones permitidas para el tipo. La principal virtud de esta técnica es que define un nuevo tipo de manera totalmente independiente de cualquier posible representación. Además nos permite unanimidad de interpretación de los distintos usuarios del tipo de dato, la posibilidad de realizar determinadas verificaciones formales, la deducción, a partir de la especificación, de propiedades satisfechas por cualquier implementación válida del tipo de dato, y la posibilidad, en ciertos casos, de obtener una implementación automática a partir de la especificación algebráica.

Una especificación algebráica consta de dos partes: (1) La parte sintáctica, en la que se declara los nombres de los nuevos tipos de datos junto con los nombres y los dominios de las operaciones que se realizarán con el nuevo tipo, y (2) la parte semántica, en la que se listan las propiedades que la implementación de las funciones debe cumplir. En la figura 1 podemos ver la especificación algebráica de una pila genérica. La sintaxis empleada para definirla es irrelevante. En la bibliografía podemos encontrar otras nomenclaturas utilizadas ([Horebeek, I. V., et al., 1989]).

```
Module Pila [Elem];
  Import Boolean;
  Operations
    new: -> Pila;
    push: Pila, Elem -> Pila;
    pop: Pila -> Pila;
    top: Pila -> Elem;
    empty: Pila -> Boolean;
  Declare x : Elem, s: Pila;
  Axioms
   pop (push (s, x)) = s;
   top (push (s, x)) = x;
   empty (new) = true;
   empty (push (s, x)) = false;
   empty (s) => pop (s) = error;
   empty (s) => top (s) = error;
End module
```

Fig. 1 Especificación del tipo abstracto Pila

Como se puede apreciar, la especificación de un tipo de datos consta de varias cláusulas. El nombre se indica en la cláusula Module, donde Elem representa el elemento genérico que se va a almacenar en la pila. La cláusula Import especifica otros tipos abstractos que han sido previamente definidos y que van a ser utilizados en esta especificación. A continuación aparece la cláusula Operations que se utiliza para definir las operaciones permitidas con el nuevo tipo. El símbolo -> se utiliza para indicar el resultado (único) de la función. Así, la función "push: Pila, Elem -> Pila" indica que la función push necesita como entrada dos datos, una Pila y un elemento y devolverá una nueva Pila. Con estas cláusulas se define la sintaxis del nuevo tipo abstracto. Para especificar la semántica se utiliza la cláusula Axioms. En dicha cláusula se utiliza tantos axiomas como sean necesarios para definir el significado de cada operación. Note que hay operaciones (en concreto new y push) que no tienen axiomas. Esto ocurre porque dichas operaciones definen los elementos de la pila; o sea son operaciones constructoras (o generadoras) del tipo que se está definiendo. El resto de las operaciones, las cuales definen su significado mediante axiomas, son operaciones modificadoras u operaciones observadoras si el tipo que devuelve se corresponde con el tipo que se está definiendo o no. Cuando tenemos una operación parcial, es necesario expresar bajo que condiciones es congruente. Esto es lo que ocurre con el axioma "empty (s) => pop (s) = error". Si la pila está vacía no tiene sentido eliminar un elemento de la pila. El símbolo => se utiliza para separar la condición en el axioma.

Lo que podemos comprobar es la sencillez "relativa" con la que se puede especificar tipos abstractos de datos. Ahora nos interesa saber como podemos utilizar esta especificación para realizar la implementación.

b) De los Tipos de datos abstractos a las Clases.

En el apartado anterior se ha utilizado una teoría matemática para modelizar estructuras de datos, pero ¿cómo podemos emplear esta especificación para desarrollar software?. Los TDAs nos proporcionan la base de una estructura modular basada en tipos de objetos. Por ello, un sistema orientado a objetos podría ser construido como un conjunto de TDAs, parcial o totalmente implementados. El concepto clave en esta relación es la *clase*. Una clase es un tipo de dato abstracto equipado con una implementación. Para implementar totalmente una clase tenemos que proporcionar todos los detalles de la implementación. En caso contrario, la clase será implementada parcialmente. Ésto lo decide el especificador.

Para producir una clase implementada totalmente, necesitamos 3 elementos ([Meyer, B., 1997]): (1) Una especificación de un TDA, o sea, un conjunto de funciones con sus axiomas asociados, (2) una elección de representación, y (3) una asignación desde las funciones a la representación en la forma de un conjunto de mecanismos, cada uno representando a las funciones en términos de la representación, así como satisfaciendo los axiomas y precondiciones. Muchos de estos mecanismos serán rutinas (subprogramas) en el sentido habitual, aunque algunos pueden aparecer como campos de datos o "atributos". Por ejemplo, en el TDA Pila podemos elegir como representación el siguiente fragmento de código:

```
contador: INTEGER
representacion: array [1 .. capacidad] of Elem
```

donde *capacidad* es el número máximo de elementos de la pila. Para la implementación de las funciones new, push, pop, etc., utilizaremos rutinas. Por ejemplo, la función push podría ser implementada de la siguiente forma:

```
push (x: Elem) {    // Inserta x en la Pila (No se comprueba un posible overflow de la pila)
  contador := contador + 1
  representacion [contador] := x
}
```

La combinación de los elementos obtenidos en (1), (2) y (3) producirá una clase, la estructura modular de la tecnología de objetos. Cuando nos falte los elementos (2) ó (3), la clase será implementada parcialmente. Estas clases serán útiles durante las fases de análisis, donde no es necesario o deseado ningún detalle de la implementación, y en la fase de diseño, donde nos debemos concentrar en las funcionalidades que proporciona cada módulo y no cómo lo proporciona. También se utiliza la implementación parcial cuando diseñamos de un forma gradual, de menos detalle hasta la implementación completa.

4.- REFERENCIAS

Horebeek, I. V., & Lewi, J. (1989). Algebraic Specifications in Software Engineering. An Introduction. Springer-Verlag.

Meyer, B., (1997). Object-Oriented Software Construction. 2°. Edición. Prentice Hall PTR.

Peña, R. (1998). Diseño de programas. Formalismo y abstracción. 2ª edición. Prentice Hall.

Wing, J. M., (1990). A Specifier's Introduction to formal Methods. Computer. Sept. 8-22.

Wirsing M., (1994). Algebraic Specification Languajes: An Overview. 81-115