Actas de las XX JENUI. Oviedo, 9-11 de julio 2014
ISBN: 978-84-697-0774-6
Páginas: 201-205

Pooi, un intérprete de un lenguaje orientado a objetos basado en prototipos para la educación

Baltasar García Perez-Schofield
Departamento de Informática
Universidad de Vigo
Ourense
jbgarcia@uvigo.es

Francisco Ortín Soler
Departamento de Informática
Universidad de Oviedo
Oviedo
ortin@uniovi.es

Resumen

En cursos avanzados del grado en Ingeniería Informática, es interesante explorar otras opciones al margen del típico modelo de programación orientado a objetos basado en clases. El modelo basado en prototipos es de hecho muy interesante, pues llega a ser capaz de representar el modelo basado en clases. Sin embargo, no es sencillo encontrar lenguajes basados en prototipos que permitan que el estudiante interaccione con ellos de una manera sencilla y didáctica. Fruto de este problema nace Pooi (Prototype-based Object-Oriented Interpreter), un pequeño intérprete de objetos cuyo fin último es el de servir de ejemplo sencillo del funcionamiento interno de estos lenguajes, así como de ser una herramienta con la que los alumnos puedan experimentar en su tiempo no presencial.

Abstract

In advanced courses of the Computer Science Engineering is interesting to explore other options apart from the typical object-oriented programming model, based on classes. The prototype-based model is indeed interesting, since is able to represent the class-based one. However, it is not easy to find prototype-based languages that are simple for the student to interact with, in a didactic way. Pooi (Prototype-based Object-Oriented Interpreter), was developed as a way to cover this need, to be a tool suitable for demonstrations of the internal workings for this kind of programming languages, as well as for the students to experiment on their own.

Palabras clave

Lenguaje, programación, objetos, prototipos.

1. Motivación

La mayor parte de lenguajes de programación orientados a objetos están basados en clases, como por ejemplo C++, Java o C#. Sin embargo, este no es el único paradigma disponible en el campo de la orientación a objetos, pues también existe el basado en prototipos. Este último no es un paradigma demasiado extendido en lenguajes de programación empleados en la industria, si bien ha sido investigado ampliamente [4], y ha influenciado fuertemente el diseño de lenguajes hasta el punto de que algunos usados habitualmente, como JavaScript [1] o Python, sí lo emplean.

La asignatura *Tecnología de Objetos* (1999/00-2012/13) en la Escuela Superior de Ingeniería Informática (Universidad de Vigo), se impartía en el último curso, y su propósito era formar en profundidad a los alumnos en todo el campo de la orientación a objetos. Para esta asignatura también se desarrolló Zero [3], un entorno de programación mucho más ambicioso. Sin embargo, era necesario un entorno de aprendizaje extremadamente simple que permitiera la ejecución rápida de ejemplos significativos, así como la experimentación por parte del alumno.

Si bien existen lenguajes de programación usados en la industria, o de investigación (como los mencionados), que siguen este modelo, desafortunadamente todos presentan algún tipo de característica que no lo hace apropiado para docencia.

Bajo estas premisas nació Pooi[1], un intérprete de un lenguaje de programación orientada a objetos y basado en prototipos efectivamente simple, con un entorno integrado que favorece su uso didáctico y la interacción sencilla.

El resto del presente documento se estructura como sigue: en primer lugar se discute el modelo basado en prototipos, para después analizar cómo Pooi representa ese modelo. Se muestra también el diseño

[1]Pooi puede descargarse libremente de: http://webs.uvigo.es/jbgarcia/prys/

de la herramienta, para finalmente analizar el trabajo relacionado , y presentar las conclusiones.

2. El modelo basado en prototipos

En esta sección se discuten en profundidad las características del modelo basado en prototipos.

2.1. El punto del modelo

En la mayor parte de lenguajes de programación orientados a objetos, se emplea el modelo basado en clases. Según este modelo, las clases son como moldes de los que se extraerán los futuros objetos, de tal forma que a) todos los objetos de una clase serán estructuralmente iguales, y b) es necesaria la información contenida en la clase para interpretar un objeto determinado.

En los lenguajes de programación basados en prototipos, no existen las clases. Pueden crearse objetos en cualquier momento, y estos son absolutamente independientes unos de otros. Self fue uno de los primeros en ofrecer estas características [4, 5].

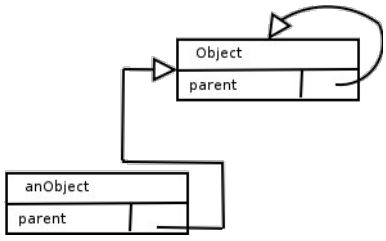

Figura 1: Un objeto vacío recién creado.

En la Figura 1, se muestra un objeto vacío recién creado. Todos los objetos tienen al menos un atributo fijo, que es *parent*. Este atributo señala al objeto padre de este objeto, y por defecto siempre es **Object** o similar, la raíz de la jerarquía de herencia. Aunque Self [4] e IO [2] sí proporcionan herencia múltiple, esta no es obligatoria, y de hecho Pooi no la ofrece.

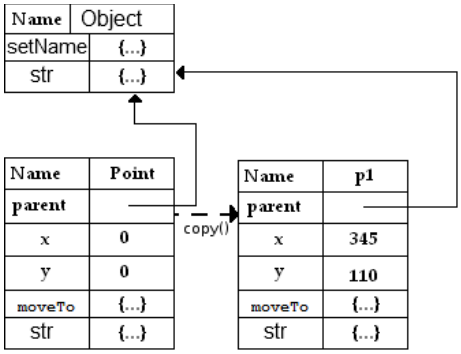

Figura 2: El prototipo **Point**.

A cualquier objeto se le pueden insertar métodos y atributos nuevos (así como borrarlos o modificarlos). En la Figura 2, se ha creado el objeto **Point**, con su atributo *parent* apuntando a **Object**, y dos atributos, *x* e *y*. **Point** es un prototipo, es el objeto a partir del cual, mediante copia, se crearán el resto de objetos **Point**, como *p1* en la figura. El comportamiento de **Point** es lo más parecido a una clase en los lenguajes basados en clases; sin embargo, una vez creado el objeto *p1* por copia (que tomaría el papel de instancia), el objeto es totalmente independiente del prototipo, y de hecho puede incorporar nuevos métodos y atributos.

Al ser una copia, *p1* tiene los mismos valores para *x* e *y* que **Point**, y su atributo *parent* señala al mismo objeto que señalaba **Point**. Los métodos *moveTo*, que permite cambiar los valores de *x* e *y*, y *str*, que devuelve una representación textual, también se copian, de manera que se duplica el código en el nuevo objeto.

Aunque es técnicamente posible utilizar el mismo prototipo para realizar cualquier tarea (un prototipo no se distingue de cualquier otro objeto), esto consiste en una mala práctica, pues quizás las modificaciones realizadas lo volverían inútil para crear nuevos objetos mediante copia.

Una critica muy extendida hacia este tipo de lenguajes, es que las implementaciones de los mismos adolecen de un buen rendimiento, al tener que evaluar todas las llamadas a métodos y atributos en tiempo de ejecución. Así mismo, la copia de métodos no es eficiente, aspecto este que sí se podrá paliar, como se discutirá en las siguientes secciones.

2.2. Herencia y polimorfismo

La implementación típica de los lenguajes basados en prototipos es mediante la comprobación dinámica de tipos. Esto quiere decir que apenas se realizan comprobaciones en tiempo de compilación, y los posibles errores de métodos, atributos, u objetos que no existen, se resuelven mediante el uso de excepciones.

La herencia se obtiene, tal y como se ha discutido previamente, mediante el uso de un atributo llamado *parent*. Este atributo señala al objeto padre del objeto que lo contiene, y siguiendo la cadena de enlaces se debe poder llegar,de forma transitiva, a la raiz de la jerarquía de herencia.

Al ser un lenguaje dinámico, todos los métodos son en realidad polimórficos. Tomando como ejemplo de nuevo la Figura 2, la llamada al método *setName* (que permite cambiar de nombre al objeto), en **Point** o en *p1* no será exitosa, pues estos objetos no tienen dicho método. Al plantearse esta situación, el objeto delega en su padre el cumplimiento del mensaje, de manera que se envía el mensaje *setName* a **Object**. En caso de alcanzar la raíz de la jerarquía y no poder satisfacer el mensaje, se lanza una excepción de método no encontrado. Esto se conoce como herencia

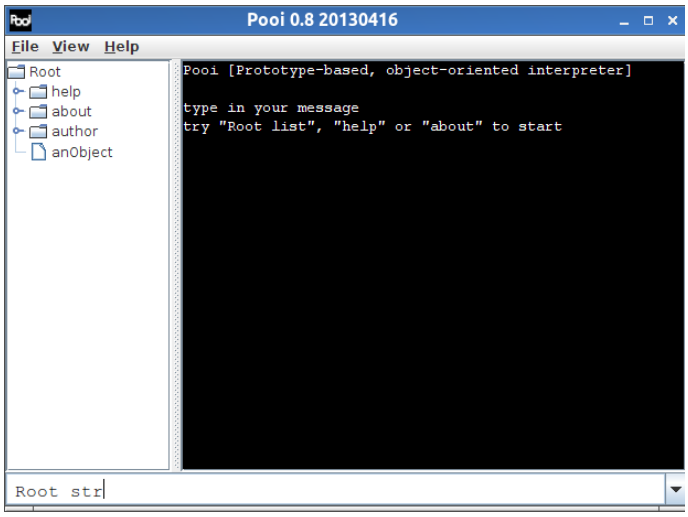

Figura 3: El entorno de trabajo que ofrece Pooi.

por delegación, y es típica de los lenguajes de programación basados en prototipos [4].

2.3. Representación del modelo de clases

Como se ha indicado anteriormente, el modelo basado en prototipos es capaz de representar el modelo basado en clases [5].

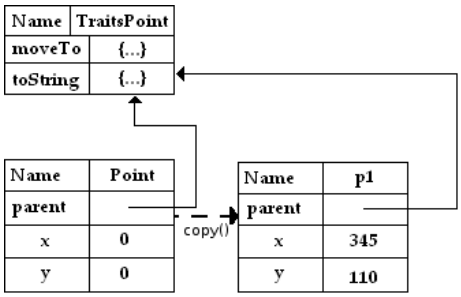

Figura 4: El *traits object* **TraitsPoint**, y el prototipo **Point**.

Para ello, se usa el concepto de *traits object* (objeto de rasgos). Este objeto **TraitsPoint** guarda los métodos (es decir, el comportamiento) del prototipo. Como se puede apreciar en la Figura 4, en el prototipo se mantiene tan solo el estado, manteniendo un enlace con el *traits object* mediante su atributo *parent*.

Al realizar la copia para crear *p1*, se mantienen todos los valores de los atributos (incluyendo *parent*, que seguirá apuntando a **TraitsPoint**). Cuando se llame a un método en *p1*, en realidad se delegará la satisfacción del mensaje en **TraitsPoint**, que tiene el rol (del modelo basado en clases), de clase al aglutinar todo el comportamiento. El objeto *p1*, en realidad, sólo mantiene el estado, teniendo el rol de instancia. Por ello, se puede afirmar que el modelo basado en prototipos es capaz de representar el modelo basado en clases [5].

3. Pooi

En esta sección se presenta la herramienta docente Pooi, con sus principales características y funcionalidades.

3.1. El entorno de trabajo en Pooi

En la Figura 3, se puede apreciar la interfaz de usuario que presenta Pooi. En la parte izquierda, se mantiene un árbol de objetos creados, mientras a la derecha una ventana de texto muestra los resultados de los mensajes enviados. En la parte inferior, una entrada de texto (con un histórico asociado), permite ejecutar mensajes. Así, Pooi funciona de manera intuitiva como un terminal, con la ventaja de que pueden escogerse los objetos a los que enviar mensajes del árbol de objetos.

Una forma de comenzar es pulsar sobre el objeto help, en el árbol de objetos, con lo que su nombre se copia a la entrada. Si se pulsa Enter, ante la ausencia de ningún mensaje, se asume *str*, que describe el contenido del objeto. Es decir, "*help*" y "*help str*" producen el mismo resultado en esta entrada para el usuario.

3.2. Copia, herencia y polimorfismo

En el modelo basado en prototipos, los nuevos objetos se crean mediante copia. Por eso precisamente se ofrece un prototipo llamado **anObject**, que se puede emplear para que el usuario cree sus propios prototipos.

```
(anObject copy) setName "Point"
Point.x = 0
Point.y = 0
```

Con los mensajes anteriores, se crea un nuevo objeto vacío, copiado de **anObject**, y cuyo *parent* señala a **Object**. Los mensajes pueden agruparse encerrándolos entre paréntesis, de forma que sobre el objeto creado con el mensaje *copy*, se lanza el mensaje *setName*, para cambiar su nombre por *Point*.

Nótese cómo el entorno trata de informar en lenguaje natural de lo que va sucediendo. La respuesta a este mensaje es: "***anObject*** *was copied into* ***Root*** *as 'Root.anObject1', Root.anObject1 renamed to* ***Root.Point***".

Siguiendo las directrices de la sección anterior, hemos creado el prototipo **Point**. A continuación, será necesario crear el objeto de rasgos **TraitsPoint**.

```
(anObject copy) setName "TraitsPoint"
Point.parent = TraitsPoint
```

Se ha creado con los dos mensajes anteriores el objeto de rasgos **TraitsPoint**, y el atributo *parent* de **Point** señala a **TraitsPoint**.

```
TraitsPoint.str =
     {: ((x str) + ", ") + (y str) }
```

Con la instrucción anterior, se crea el método *str*. Debido a la naturaleza dinámica del lenguaje, y a la herencia por delegación, este método *str* se encuentra antes que el método *str* de Object, por lo que ante ***Point*** *str*, se ejecuta este código, devolviendo "0, 0".

Los métodos se crean enmarcáncolos dentro de llaves ('{' y '}'), y se indican los parámetros como los primeros identificadores al comienzo del método, separados del código en si mediante dos puntos (':'). La última instrucción no tiene por qué llevar punto y coma (';'), esta se utiliza exclusivamente para separar unos mensajes de otros.

```
TraitsPoint.moveTo =
     { a b: self.x = a; self.y = b }
```

El objeto que está ejecutando el método se denota mediante la referencia *self*. Nótese que es importante distinguir el objeto que ejecuta el método del objeto que está siendo apuntado por *self*. Por ejemplo, si se ejecuta "*Point moveTo 10 10*", el objeto apuntado por *self* es **Point**, mientras el método *moveTo* reside en **TraitsPoint**. Así, cuando dentro del método se ejecutan "*self.x = 10; self.y = 10*", es el objeto **Point** quien va a recibir los valores 10 y 10 para sus atributos *x* e *y*, no **TraitsPoint**.

Sólo resta crear un objeto *p1* a partir del prototipo **Point** para haber terminado de replicar la situación de la sección anterior mostrada en la Figura 4, demostrando que el modelo basado en prototipos es capaz de representar el modelo basado en clases.

```
(Point copy) setName "p1"
p1 moveTo 20 25
p1 str
```

Con los mensajes ejecutados anteriormente, se crea el objeto *p1* y se guardan en sus atributos *x* e *y* los valores 20 y 25, respectivamente. Finalmente, con el mensaje final se devuelve "20, 25" como resultado.

4. Diseño de Pooi

Pooi sigue el esquema de un intérprete, creando un árbol de sintaxis y ejecutándolo inmediatamente (en contraste con generar código ejecutable).

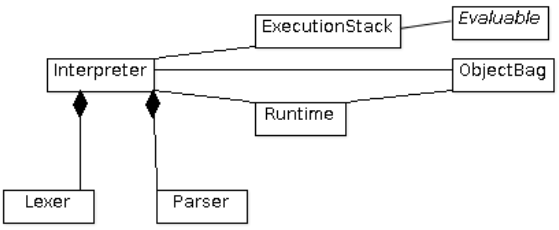

Figura 5: Estructura básica de clases de Pooi.

La Figura 5, muestra el esquema de clases básico necesario para interpretar mensajes. Los mensajes se traducen mediante un **Lexer** y **Parser**, y se guardan (ya como objetos listos para la ejecución, derivados de **Evaluable**), en un objeto **ExecutionStack** (pila de ejecución). Finalmente, son ejecutados contra el **Runtime**.

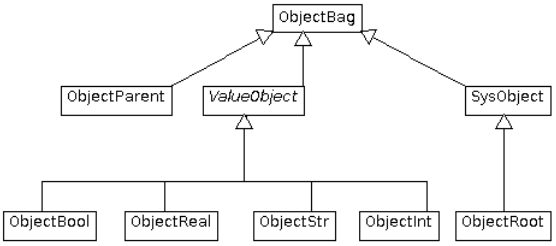

Figura 6: Los diferentes objetos en Pooi.

La Figura 6, muestra los diferentes tipos de objetos que soporta Pooi. Los literales encontrados en el código son traducidos a objetos **ValueObject** (ya que en Pooi todo es un objeto). Y también es posible apreciar **ObjectParent** (el objeto que será conocido como Object por el usuario), así como el objeto **ObjectRoot**, que será conocido como **Root**, el objeto que sirve de almacenamiento al resto de objetos.

La Figura 7, muestra los diferentes tipos de objetos que pueden ser evaluados como parte de un mensaje complejo, como puede ser desde el cuerpo de un método, hasta un simple literal.

Pooi ha sido implementado en Java 7, y probado con éxito, verificando funcionalidad completa, en Windows 7 y Ubuntu Linux 13.10.

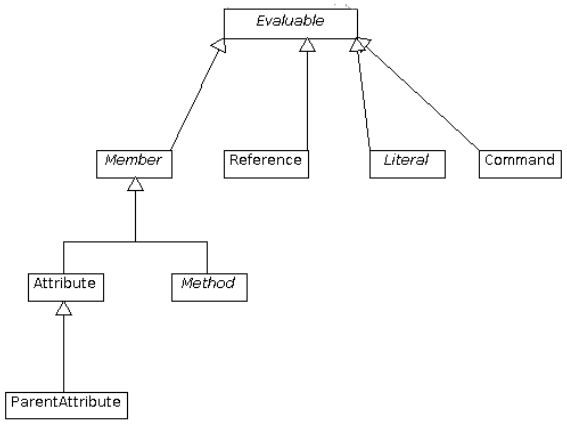

Figura 7: Evaluables en Pooi.

5. Trabajo relacionado

La investigación en lenguajes basados en prototipos [4, 5] comenzó en la decada de 1980, por lo que existen bastantes trabajos basados en ellos (discutidos a continuación), como el propio Zero [3], un entorno de ejecución completo soportando incluso persistencia.

La principal ventaja de Pooi es que es un entorno sencillo, completo, y que sin conocer en profundidad el modelo de objetos empleado puede comenzar a utilizarse sin más (el tutorial en la página de Pooi[2] cubre toda la herramienta, con la librería estándar disponible, en apenas 19 páginas).

Es importante reseñar que la mayoría de las alternativas disponibles, como el propio Self [4], o IO [2] (un lenguaje basado en Self), soportan sintaxis demasiado chocantes para el principiante (del tipo postfijo), lo que unido a un cambio de modelo de objetos, provoca rechazo en el estudiante.

Finalmente, sin duda el lenguaje más popular que implementa el modelo basado en prototipos es JavaScript [1]. La sintaxis que emplea es bastante parecida a C, y además está disponible en cualquier navegador. Sin embargo, JavaScript es también conocido por ser un lenguaje de programación muy laxo, con diferentes formas de ofrecer características como herencia, por ejemplo, y confiando todas sus posibilidades sobre los *closures* (cierres). Es decir, según se emplee un *closure* se obtiene una característica u otra, lo cual puede resultar confuso.

6. Conclusiones

En este documento se ha presentado Pooi como una herramienta de ayuda a la docencia, que ha sido empleada con éxito tanto en demostraciones en clases, como sirviendo de vehículo de experimentación para los propios alumnos en su tiempo no presencial.

El modelo de orientación a objetos basado en prototipos es muy interesante como una forma alternativa de orientación a objetos al margen de la predominante basada en clases. Su estudio en cursos avanzados de ingeniería, permite abordar la orientación a objetos desde una perspectiva totalmente distinta, lo cual enriquece los conocimientos del estudiante.

Pooi es multiplataforma, por lo que independientemente del sistema operativo elegido, es posible empezar a utilizarlo con un solo click de ratón. Una vez que se empieza a trabajar con él, gracias a su modo interactivo mensaje-respuesta, es muy sencillo para el estudiante interaccionar poniendo en práctica sus conocimientos sobre el modelo basado en prototipos.

Finalmente, el docente puede emplearlo de forma muy sencilla y directa en clases para realizar demostraciones.

La herramienta ha sido utilizada con éxito especialmente en demostraciones durante las clases magistrales de *Tecnología de Objetos*, 5º curso de Ingeniería Informática, y continúa su uso en la asignatura del mismo nombre del *Máster Universitario en Sistemas Software Inteligentes y Adaptables*, en la Escuela Superior de Ingeniería Informática de la Universidad de Vigo.

Referencias

[1] Douglas Crockford. *JavaScript: The Good Parts: The Good Parts*. O'Reilly Vlg. Gmbh & Co., Mayo de 2008. ISBN 978-0596517748.

[2] Steve Dekorte. Io: a small programming language. *Actas de OOPSLA '05. Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. Pgs. 166-167. ACM New York, NY, USA. 2005. ISBN 1-59593-193-7.

[3] J. Baltasar García Perez-Schofield, Emilio García Roselló, Francisco Ortín Soler, Manuel Pérez Cota. Visual Zero: A persistent and interactive object-oriented programming environment. *Journal of Visual Languages and Computing*. Volumen 19, número 3. Pgs. 380-398. Junio, 2008.

[4] David Ungar, Randal B. Smith. Self: The power of simplicity. *ACM SIGPLAN Notices*. Volumen 22, número 12. Pgs. 227-242. Diciembre 1987.

[5] David Ungar, Craig Chambers, Bay-wei Chang. Organizing programs without classes. *Lisp and Symbolic Computation*. Pgs. 223-242 . Kluwer Academic Publishers. 1991.

[2]Disponible desde la misma web de Pooi.