

Implementando “True Delegation” en VisualWorks*

Nicolás Cañibano¹

¹ Telefónica Móviles TM+
Buenos Aires, Argentina
Nicolas.Canibano@telefonicomoviles.com.ar

Abstract

Hoy en día cuando hablamos de delegación en el contexto del paradigma de objetos, todos creemos saber a que nos estamos refiriendo, pero basta revisar los ensayos con respecto a este tema, para sembrar la incertidumbre y darnos cuenta que en realidad podemos llegar a estar equivocados por completo. El planteo de lo que debiera entenderse por delegación se remite a fines de la década de los 80', tiempos en los que en el campo teórico los trabajos de H. Lieberman en [3] y de L. A. Stein en [4] sentaban un precedente fundamental. Mientras que para la misma época, un grupo de entusiastas que se proponía replantear las bases que habían sido establecidas desde el surgimiento de Smalltalk, llevaba a la práctica alguna de estas ideas. Se trataba del equipo de investigación de los laboratorios Sun al frente del proyecto Self [6]. Todos ellos acordaron en definir en [5] los mecanismos fundamentales que pueden encontrarse en todo lenguaje orientado a objetos: delegación y herencia (como mecanismos para compartir estado o comportamiento); prototipos y clases (como mecanismos para compartir la forma o molde, a partir del cual se pueden crear otros objetos). De forma tal que todo lenguaje orientado a objetos puede ser descrito mayormente teniendo en cuenta la manera en que combina estos mecanismos. A priori podemos catalogar y encasillar a Smalltalk dentro de lo que se considera el paradigma “clásico” de objetos, donde las clases y la herencia son los mecanismos sobre los cuales se basa el lenguaje. Pero tratándose de Smalltalk, todo intento de encuadrarlo dentro de alguna categoría es en vano. Aquí mostraremos como puede integrarse de manera transparente el mecanismo de delegación en el ambiente Smalltalk, sin incurrir en el agregado de nuevos artilugios sintácticos en el lenguaje, simplemente haciendo uso de lo que se tiene a mano en un ambiente Smalltalk, los objetos.

Lo que comúnmente se entiende por delegación, debiera mejor denominarse “forwarding” o simplemente envío de mensajes. Básicamente cuando un mensaje es enviado desde un objeto *A* hacia otro objeto *B*, dentro de los métodos de *B*, *self* se refiere al objeto *B*. En cambio, cuando un mensaje es delegado desde un objeto *A* (delegador) hacia un objeto *B* (delegado), dentro de los métodos de *B*, *self* se refiere a *A*, no a *B*. De esta manera, cuando hablamos de “true delegation” nos referimos al mecanismo de delegación cuya semántica se corresponde con lo expuesto en este último caso. Por lo tanto, al delegar, el delegador puede sobrescribir comportamiento del delegado.

* Trabajo desarrollado durante el año 2006 como parte del proyecto de investigación del Context-Aware Group, LIFIA (Laboratorio de Investigación y Formación en Informática Avanzada), La Plata, Argentina.

No es el objetivo de este trabajo reflotar la polémica sobre cuál de los mecanismos es más poderoso, si la delegación o la herencia, si no el de reconocer la importancia de la delegación como mecanismo alternativo y complementario a la herencia; y el de mostrar cómo haciendo uso de las capacidades de reflexión y de meta-programación inherentes al ambiente (algunas de las cuales han sido enumeradas en [1]), se puede integrar este mecanismo en Smalltalk, en particular en VisualWorks.

El mecanismo de delegación puede llegar a ser útil para la resolución de diversos problemas específicos, por ejemplo para simular herencia múltiple o herencia dinámica, pero tal vez el mayor incentivo para la utilización de la delegación sea su gran utilidad a la hora de lidiar con wrappers o proxies [2], principalmente en sistemas que hacen uso de “transparent forwarders”, como pueden ser los sistemas de mapeo objeto-relacional, Opentalk y GemStone. Estos sistemas pueden ser implementados de manera más robusta. Esto es así porque la delegación puede ser usada para prevenir que las referencias a objetos decorados (detrás de un wrapper o un proxy) escapen por fuera del objeto decorador, lo que se conoce comúnmente como “The Self Problem” [3]. Esto mismo no puede llevarse a cabo con el mecanismo de “forwarding”. Con la delegación, las referencias a un wrapper se mantienen como tal, preservando la habilidad de los wrappers de mantenerse delante y ocultar la existencia del objeto decorado.

Nuestra implementación del mecanismo de delegación, permite que cualquier par de objetos preexistentes en el ambiente, y pertenecientes a cualquier clase, puedan delegarse explícitamente mensajes entre sí. Facilitando de esta manera la construcción de wrappers o proxies transparentes y robustos, que cumplan con las características mencionadas anteriormente.

El presente trabajo se pretende sea acompañado de una demo en vivo, para mostrar los detalles implementativos y ejemplos de uso del mecanismo de delegación. El mismo puede encuadrarse dentro de las categorías de meta-programación, reflexión o extensiones al ambiente y puede llegar a apuntarse dentro de la sección de Reportes de Experiencias de Desarrollo o Investigación.

Referencias

1. Brian Foote and Ralph E. Johnson, “Reflective Facilities in Smalltalk-80”, OOPSLA '89, Octubre 1989.
2. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison Wesley, 1995.
3. Lieberman, H., “Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems”, OOPSLA'86, Octubre 1986.
4. Stein, L. A., “Delegation Is Inheritance”, OOPSLA'87, Octubre 1987.
5. Lynn Andrea Stein, Henry Lieberman, David Ungar, “A Shared View of Sharing: The Treaty of Orlando”, Object-Oriented Concepts, Databases and Applications, 1989.
6. Ungar, D. and Smith B., “Self: The Power of Simplicity”, OOPSLA'87, Octubre 1987.