Programación Orientada a Agentes en el Marco de Lenguajes Multiparadigmas

Maximiliano Klemen Claudio Vaucheret

Departamento de Ciencias de la Computación
Facultad de Economía y Administración
Universidad Nacional del Comahue
{maximilianoklemen,vaucheret}@gmail.com

Resumen

La presente línea de investigación surge de dos observaciones. Por un lado, observamos que el paradigma orientado a agentes aún hoy no se encuentra consensuadamente definido dentro de las diferentes áreas de las ciencias de la computación. Por el otro lado, la combinación de conceptos y herramientas de los diferentes paradigmas de programación en lenguajes actuales demostraron ser la vía más adecuada para la elaboración de herramientas de desarrollo de software que permitan representar adecuadamente los problemas y soluciones de dominios de la vida real. A partir de estas observaciones comenzamos una investigación cuyo objetivo es establecer las bases para dar origen a una definición definitiva del paradigma orientado a agentes, mostrando que constituye la convergencia entre los demás paradigmas. Con este fin nos proponemos desarrollar un lenguaje de programación que incorpore sus ideas elementales de manera armónica. Para su desarrollo escogimos un lenguaje multiparadigma basado en la programación lógica, por sus numerosas extensiones y la facilidad con la que permite definir nuevas estructuras sintácticas, semánticas y de control. Como conclusión de este trabajo utilizaremos el nuevo lenguaje para desarrollar aplicaciones prácticas en diversos campos como la robótica, la domótica, el desarrollo de aplicaciones móviles, etc.

1. Contexto

Presentamos aquí una línea de investigación contenida dentro del proyecto de investigación *Técnicas Avanzadas y Análisis para el Desarrollo Multiparadigma* de la Secretaría de Investigación de la Universidad Nacional del Comahue, código 04/E073. Entre los principales objetivos de esta investigación se encuentran establecer las bases para dar origen a una definición definitiva del paradigma orientado a agentes y su relación con los demás paradigmas de programación.

2. Introducción

Según el paradigma orientado a objetos el mundo puede ser representado como un conjunto de objetos que interaccionan entre ellos a través del envío de mensajes. Estos mensajes provocan

la ejecución de porciones de código que pueden modificar el estado interno del objeto que lo recibe o enviar a su vez nuevos mensajes a otros objetos. La adhesión estricta a este paradigma demostró ser viable sobre todo con la aparición de Smalltalk [14], un lenguaje orientado a objetos puro que aún hoy es utilizado en parte de la industria del desarrollo de software.

Pero fue debido a la amplia utilización de lenguajes basados en el paradigma estructurado que la programación orientada puramente a objetos no gobernó la industria, y en su lugar lo hizo una fusión entre ambos paradigmas. Ejemplos claros de lenguajes ampliamente difundidos en la actualidad que adoptaron este modelo son Java [15] y C++ [6]. Fue esta combinación de conceptos lo que hizo de ambos paradigmas una base sólida sobre la cual pudieron sostenerse y crecer las herramientas de diseño y desarrollo de software actuales.

Las capacidades cada vez mayores de hardware y el veloz avance de las redes hicieron absolutamente necesario que los programas y sistemas sean capaces de ejecutarse distribuídamente en distintas plataformas, interoperar con otros componentes y aprovechar múltiples unidades de procesamiento. Por este motivo los lenguajes de programación debieron incorporar elementos que les permitiesen modelar esta nueva situación. Fue así como se hicieron comunes conceptos como *procesos livianos* o *lightweight processes*, *hilos de ejecución* o *threads*, *sincronización de procesos*, etc. Se denominó a la programación que se enfocaba en este modelo *programación concurrente* [1] o paralela, dependiendo si se compite por los recursos subyacentes o si pueden utilizarse simultáneamente. Si bien esta manera de programar no fue sustentada por un paradigma bien definido, su importancia actual es definitiva. Poco a poco los sistemas comenzaron a estructurarse a través de entidades autónomas que realizaban sus propios cómputos y se comunicaban entre sí a través de algún tipo de protocolo de intercambio de mensajes. Estas entidades no necesariamente estaban escritas en un mismo lenguaje de programación, o incluso se ejecutaban en plataformas muy distintas. Más tarde, la ingeniería del software denominó a estas entidades autónomas como *agentes*.

Paralelamente, el área de la Inteligencia Artificial adoptó el concepto de agente [12, 17] como una entidad activa que actúa percibiendo distintos estímulos de su entorno y actuando en consecuencia de estas percepciones, con el propósito de alcanzar una o varias metas.

Para el desarrollo de agentes racionales [12] resultó muy adecuado el paradigma lógico, por su claridad para representar estados mentales gracias a su naturaleza declarativa. Así, el comportamiento de un agente podía representarse estableciendo los hechos de su entorno, y codificando las reglas mediante las cuáles podría inferir nuevo conocimiento. Variantes y extensiones interesantes de este paradigma fueron surgiendo [13, 11, 8] para acercarse cada vez más a las estructuras y mecanismos de pensamiento de los seres humanos.

Si bien han habido intentos de acercamiento de estas dos concepciones de *agencia* [7, 5, 10], no creemos que una combinación definitiva haya surgido aún. Esto se demuestra claramente con la falta de acuerdo respecto a cuáles son las características básicas que un agente debe poseer. Como consecuencia de esta falta de claridad, todavía no es común oír hablar del diseño de sistemas en función de agentes, y sí lo es en función de threads, componentes, objetos distribuidos, etc. En esta línea de investigación tenemos como objetivo hallar una combinación adecuada de estos conceptos que de origen al paradigma orientado a agentes definitivo. Consideramos que la mejor manera de encontrar esta fusión es la implementación de un lenguaje de programación que incorpore estos conceptos de manera nativa y clara, permitiendo su utilización en el desarrollo de sistemas inteligentes distribuidos.

3. Líneas de Investigación y Desarrollo

Para alcanzar los objetivos propuestos por esta línea de investigación, se abordarán los ejes principales detallados a continuación.

Características esenciales y deseables de un lenguaje orientado a agentes Determinaremos cuál es el conjunto minimal de características que debe poseer un lenguaje para cumplir con los lineamientos de la orientación a agentes, clasificándolas entre características *inter-agentes* o *exógenas* e *intra-agentes* o *endógenas*. Para esto, analizaremos varias plataformas y lenguajes que actualmente incorporan en alguna medida conceptos de agencia.

Representación del conocimiento y razonamiento mediante *FLUX* Desarrollaremos las estructuras mentales que podrán incorporar los agentes de nuestro lenguaje adaptando el método de programación lógica *FLUX* basado en el *cálculo fluent* [16]. Éste método utiliza *Constraint Handling Rules* [9] para implementar el razonamiento en dominios parcialmente observables, por lo que será necesario su estudio y la eventual colaboración con el grupo de investigación.

Implementación del lenguaje en *Ciao Prolog* Implementaremos el lenguaje sobre *Ciao Prolog* [3], una plataforma de desarrollo y ejecución multiparadigma desarrollada y mantenida por el grupo *CLIP*. Esta herramienta brinda facilidades para incorporar nuevas características mediante las técnicas de *metaprogramación* [4] y *compilación de control* [2]. Además, en versiones recientes de Ciao Prolog se incorporaron construcciones para la comunicación entre agentes que nos resultarán útiles para construir sobre ellas nuestro lenguaje permitiendo abstraernos de esas cuestiones.

4. Resultados esperados

A través de la presente línea de investigación pretendemos alcanzar dos resultados fundamentales. Por un lado, establecer las bases sobre las cuáles poder llegar a una definición definitiva y consensuada entre las diferentes áreas de la comunidad de ciencias de la computación de la orientación a agentes. Por otro lado, implementar un lenguaje que respete estrictamente la definición antes mencionada, demostrando su utilidad práctica a través del desarrollo de diversas aplicaciones en diferentes dominios de la vida real.

5. Formación de recursos humanos

La presente línea de investigación constituirá el tema para la tesis de fin de carrera de Maximiliano Klemen, para alcanzar el título de grado de *Licenciatura en Ciencias de la Computación*, proyectándose a realizar estudios de postgrado en colaboración con algunos de los grupos de investigación mencionados anteriormente.

Referencias

[1] Gregory R. Andrews and Fred B. Schneider. Concepts and notations for concurrent programming. *ACM Computing Surveys*, 15:3–43, 1983.

[2] Maurice Bruynooghe, Danny de Schreye, and Bruno Krekels. Compiling control. *J. Log. Program.*, 6(1-2):135–162, 1989.

[3] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López, and G. Puebla. *The Ciao Prolog System - A Next Generation Multi-Paradigm Programming Environment.* Grupo Clip, http://www.ciaohome.org/, the ciao system documentation series edition, August 2004.

[4] D. Cabeza and M. Hermenegildo. A New Module System for Prolog. In *International Conference on Computational Logic, CL2000*, LNCS. Springer-Verlag, July 2000.

[5] S Costantini and A Tocchio. The dali logic programming agent-oriented language. In *Logics in Artificial Intelligence, Proc. of the 9th European Conference, Jelia 2004*, pages 685–688. Springer-Verlag, 2004.

[6] C++'s web site. http://www.research.att.com/∼bs/C++.html.

[7] Winton H E Davies and Peter Edwards. Agent-k: an integration of aop and kqml. In *in Proceedings of the CIKM'94 Intelligent Information Agents Workshop, Y. Labrou & T. Finin (Eds*, 1994.

[8] Marc Denecker. Ailp: abductive inductive logic programming. In *In Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1201–1207. Morgan Kaufmann, 1995.

[9] Thom Fruhwirth. J. logic programming 1994:19, 20:1–679 1 theory and practice of constraint handling rules, 1994.

[10] Martin L Griss, Steven Fonseca, Dick Cowan, and Robert Kessler. Smartagent: Extending the jade agent behavior model. In *In Proceedings of the Agent Oriented Software Engineering Workshop, Conference in Systemics, Cybernetics and Informatics.* ACM Press, 2002.

[11] V. Wiktor Marek. Reflexive autoepistemic logic and logic. In *Programming, in L M Pereira & A Nerode (eds), Logic Programming and Non-Monotonic Reasoning.* MIT Press, 1993.

[12] Stuart Russell and Peter Norvig. *Artifficial Intelligence: A modern Approach.* PEARSON Prentice Hall, 2004.

[13] Chiaki Sakama. Nonmonotonic inductive logic programming. In *In Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, page 80.

[14] John F. Shoch. An overview of the programming language smalltalk-72. *ACM SIGPLAN Notices*, 14:64–73, 1979.

[15] Sun. Java's official web site. http://java.sun.com.

[16] Michael Thielscher. Programming of reasoning and planning agents with flux. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR*, pages 435–446. Morgan Kaufmann, 2002.

[17] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995.