Análisis de lenguajes para especificación e implementación de agentes

Sebastián Gottifredi **Alejandro J. García**

Laboratorio de Investigación y Desarrollo de Inteligencia Artificial,
Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur,
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET),
Email: {sg,ajg}@cs.uns.edu.ar

ABSTRACT

Esta línea de investigación analiza las principales características que debe poseer un lenguaje de especificación de agentes, con el objetivo de cubrir las necesidades relacionadas con la implementación de sistemas multi-agente. En este trabajo se plantean las características mas importantes que debe poseer un lenguaje de implementación de agentes y se analizan dos alternativas de lenguajes de implementación de agente que utilizan un lenguaje formal para la especificación de sus agentes.

Palabras claves: Sistemas Multi-Agentes, Agentes, Lenguaje de Especificación de Agentes, Lenguaje de Implementación de Agentes, Comunicación.

1 INTRODUCCIÓN

En este trabajo se hará un análisis de las características que debería poseer un lenguaje de implementación de agentes que utiliza como base un lenguaje de especificación formal de agentes. Se enunciarán las diferentes características deseables de un lenguaje de implementación de agentes, mostrando los beneficios de las mismos a través de ejemplos. Por último se realizará una evaluación de dos lenguajes de implementación de agentes que utilizan un lenguaje formal para especificar sus agentes.

Esta línea de investigación está orientada a sistemas multi-agente colaborativos, con agentes cognitivos que pueden llegar a poseer capacidades deliberativas. Por mas complejo que sea el agente la percepción sobre el entorno será parcial y al actuar la influencia sobre el entorno también será parcial. Por ende, una acción ejecutada por el agente dos veces puede tener efectos diferentes, el agente debe estar preparado para que la acción falle, o que los efectos no sean los previstos. [6]

Los Sistemas Multi-Agente (SMA) son considerados como una muy buena solución para aplicaciones comerciales e industriales de gran escala [5]. Sin embargo, en gran parte de las aplicaciones de SMA, los agentes son especificados e implementados en forma Ad-hoc utilizando lenguajes de propósito general mas algún *plug in* para la comunicación como es el caso de JACK [2] o JADE [1]. Esto hace muy heterogénea la gama de estructuras internas de los agentes, lo que produce que los agentes, y por ende los SMA, sean mucho mas difíciles de diseñar, implementar, verificar y mantener. Una forma de aliviar este problema es utilizar reglas formales y estándares para la construcción de los agentes. Esto se puede conseguir si la implementación de los agentes del SMA se basa en reglas de sintaxis, semántica y pragmática de un lenguaje de especificación formal para el desarrollo de agentes. Las reglas de este lenguaje formal determinan la forma de las estructuras deliberativas generales del agente. La interacción y comunicación entre agentes del SMA es provista por reglas del lenguaje formal. Esto permite al desarrollador diseñar al agente a nivel de sus estados mentales y capacidades efecto-sensoriales. La importancia de diseñar agentes a este nivel es dada en [6]. Existen varios lenguajes de implementación de agentes que utilizan como base lenguajes de especificación formal de agentes. Ejemplos de ellos son 3APL [3] y Agent-O [7], los cuales serán analizados en este trabajo.

2 CARACTERÍSTICAS DESEABLES

En esta sección se hará una presentación de las principales características con las que debería contar un lenguaje de implementación y de especificación de agentes. Para esto se utilizará como dominio de aplicación un sistema

Financiado parcialmente por Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) PIP 5050, y por la Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 Nro 13096)

de intercambio de archivos por internet, donde los agentes interactúan entre ellos para compartir/solicitar/descargar archivos para sus usuarios. Cada agente:

- Posee información incompleta y capacidades limitadas.
- Ofrece a otros agentes los archivos que comparte el usuario.
- Se comunica con otros agentes para buscar los archivos que cumplan los criterios establecidos por el usuario.
- Toma decisiones acerca de que archivos bajar, en que orden y de que agente.

Como ejemplo particular sobre este dominio de aplicación se trabajará con 3 agentes conectados al sistema de intercambio, desde computadoras diferentes conectadas a internet. El agente 1 tendrá como meta obtener archivos de audio y ofrece archivos de video. El agente 2 tendrá como meta obtener archivos de texto y ofrece archivos de audio. El agente 3 tendrá como meta obtener archivos de video y ofrece archivos de audio y texto.

Figure 1: Ejemplo de los sistemas de intercambio

Las principales características con las que debe contar un lenguaje de especificación de agentes son: arquitecturas mentales flexibles, reglas para modelar la relación comunicación-estados mentales, soporte para planning, manejar la base de conocimiento del agente y el razonamiento.
Las principales características con que debe contar un lenguaje de implementación de agentes son: primitivas de comunicación entre agentes que sigan estándares, versatilidad en la recepción de mensajes, interacción con sistemas de administración de agentes y manejadores de directorio, soporte para la implementación de agentes desarrollados en un Lenguaje de especificación de agentes, código producido por los agentes debe ser portátil y posibilidad de desarrollar agentes móviles.
A continuación se desarrollarán algunas de las características enunciadas para ambos lenguajes.

2.1 Lenguajes de implementación de agentes

Un lenguaje para implementar agentes que participan de un SMA debe contar, sin lugar a dudas, con algún tipo de **primitivas de comunicación e interacción**. Supóngase que en el ejemplo el agente 1 quiere descargar los archivos de audio que ofrece el agente 3. Para esto el agente 1 primero debe solicitar el archivo, el agente 3 aceptar el intercambio y luego comenzar la transferencia. De no existir primitivas de comunicación, el agente 1 debe establecer algún tipo de conexión con el agente 3 para realizar el pedido y posteriormente la transferencia de los archivos solicitados. Desde nuestro punto de vista los desarrolladores deben abstraerse de la forma de transferencia de los mensajes y sólo preocuparse por el contenido de los mismos y como reaccionan sus agentes a los mensajes recibidos, ya que en la transferencia de mensajes hay muchos detalles de bajo nivel que deberían ser transparentes a la implementación de los agentes. Por ejemplo La conexión puede ser TCP, UDP, etc. y la comunicación se realiza a nivel de paquetes de red. Para ello los los agentes deben manejar numeros de IP, puertos, cuestiones relacionadas con los protocolos, etc. Si se cuenta con primitivas de comunicación los agentes sólo deben especificar los mensajes para realizar un pedido, aceptarlo u otras cuestiones pero no con lidiar con los aspectos técnicos de la transferencia del mensaje.

El lenguaje de implementación debe brindar diferentes **alternativas para la recepción** de los mensajes. Supóngase que en el ejemplo el agente 1 consultó al agente 3 sobre los archivos de audio y ejecutó una instrucción de para recibir que lo bloquea hasta que el agente 3 no le responda. Si el lenguaje de implementación de agentes no posee un sistema de eventos para la recepción de mensajes el agente 1 podría quedar esperando eternamente si el agente 3 no le contesta.

Además de proveer primitivas de comunicación, estas deberían ser los suficientemente versátiles como para poder modelar cualquier tipo de interacción y seguir un **estándar** común en los SMA, como los de FIPA [4]. Si se utilizan los estándares de FIPA se pueden aprovechar los protocolos de interacción para estructurar las comunicaciones.

En el ejemplo, los agentes podrían utilizar el utilizar el protocolo FIPA-Request para realizar los pedidos por archivos.

Los agentes deben conectarse con entidades a las cuales se les pueda consultar que agentes hay conectados al SMA y que servicios ofrece cada uno de ellos. Para esto FIPA [4] promueve el uso de un **Sistema de Administración de Agentes** (SAA) y un **Facilítador de Directorio** (FD), en combinación con los estándares para las primitivas de comunicación. Supóngase que en el ejemplo el agente 1 quiere comunicar a todos los agentes que ofrece un nuevo archivo de video y el SMA no cuenta con un sistema de administración de agentes, este agente debe tener en su código los identificadores de cada uno de los agentes con los quiere interactuar. Si en alguno momento se conectan nuevos agentes al SMA se debe re-codificar al agente 1 para que pueda comunicarse con ellos. Si el SMA contase con un sistema administrador de agentes, el agente 1 sólo debería consultar al SAA para saber que agentes hay conectados al sistema y con esta información enviar el mensaje a cada uno ellos. De esta forma, el desarrollador de agentes se abstrae de que agentes están conectados a SMA y sólo se preocupa por la lógica de comunicación de su agente.
Ahora Supóngase que el sistema no cuenta con un facilítador de directorio, todos aquellos agentes que se conecten luego de que el agente 1 ofreció su nuevo archivo, no sabrán de la existencia del nuevo servicio ofrecido, a no ser que el agente 1 vuelva a enviar el mensaje a todos los agente ahora conectados. Si el sistema cuenta con un facilítador de directorio, el agente 1 puede publicar su nuevo servicio en el FD, de forma tal que cuando haya algún agente interesado en ese tipo de servicios consulte al FD y este lo introduzca con el Agente 1. De esta forma, los desarrolladores de agentes pueden abstraerse de quienes serán los que conozcan sus servicios y sólo preocuparse por la lógica del servicio en si.

Como fue mencionado en la introducción, es importante que el lenguaje de implementación de soporte completo para implementar agentes que fueron especificados formalmente a través de un **lenguaje de especificación** para agentes.

2.2 Lenguajes de especificación de agentes

El lenguaje de especificación, como fue mencionado en la introducción, permite diseñar agentes a través de sus estados mentales y capacidades efecto-sensoriales. El lenguaje de especificación impone cierta arquitectura interna del agente, esta debe ser lo **suficientemente flexible como para poder diseñar agentes con diferentes arquitecturas mentales** (reactivos, de estado Interno, BDI). Siguiendo el ejemplo propuesto, el agente 1 tendría como estados mentales los objetivos de obtener nuevos archivos de audio y ofrecer los archivos de video disponibles, ciertas reglas para intentar resolver estos objetivos, como efectores tendría la capacidad de enviar mensajes/archivos, y como sensores la capacidad de recibir mensajes/archivos.

El Lenguaje de especificación debe proveer reglas para modelar la comunicación, y que sea posible diseñar fácilmente la **influencia de la comunicación con los estados mentales del agente**. Si el agente 1 se comunica con el agente 3 para solicitar un archivo de audio, la respuesta positiva del agente 3, generará un evento en el agente 1 que hará que este último agente agregue una nueva meta que representa su el objetivo de obtener un archivo de audio particular de el agente 3. Cuando el agente 1 decida cumplir esta nueva meta, seleccionará la regla que la resuélva y, probablemente, esta última envié un mensaje al agente 3 solicitando el comienzo de la descarga. Entonces, y aprovechando los estándares de FIPA es posible especificar al agente 1 de la siguiente manera: Para el objetivo "obtener archivos de audio", el agente puede tener una regla que transforma la meta en submetas que se resolverán en el siguiente orden. Primero inicia una conversación con el facilítador de directorio, con el objetivo de obtener una lista con los agentes que ofrecen este tipo de archivos. Esto podrá resolverlo utilizando el protocolo de interacción FIPA Query. Una vez conseguida la lista podrá iniciar una conversación con cada uno de los agentes conectados y de ellos seleccionar de quien descargar los archivos. Para esto podrá utilizar el protocolo de interacción FIPA Contract-Net. Cada una de las submetas podrá dividirse a su vez en submetas que determinan el comportamiento del agente dentro los protocolos de comunicación de similar manera a como fue ejemplificado en el párrafo anterior. La especificación del objetivo "ofrecer archivos de video", es análoga.

El lenguaje de especificación debería dar soporte para que el agente pueda hacer ***planning***, o por lo menos, permitirlo. La secuencia de submetas especificadas para el agente 1 en el párrafo anterior asumen un plan para la descarga de un archivo de audio. Sin embargo, este plan puede que falle debido a que, por ejemplo, el agente con que negociaba la transferencia se desconecta del SMA. Entonces, el agente 1 debería poder contar con las facilidades para poder cambiar su plan de acción y buscar a otro agente que le ofrezca archivos de audio.

El lenguaje de especificación debe dar soporte para el **manejo de la base de conocimientos del agente**. Además debería automatizar el proceso de actualización con el conocimiento, para que el desarrollador solamente se preocupe por agregar o sacar cosas de ella. El agente 1, por ejemplo, utiliza la Base de conocimiento para mantener aquellos agentes con los que se encuentra transfiriendo archivos y con los que se esta comunicando actualmente.

3 ANALISIS DE DOS LENGUAJES DE IMPLEMENTACIÓN DE AGENTES

En esta sección se hará un análisis de los lenguajes de implementación de agentes, 3APL y Agent-O. Para el análisis se tendrán en cuenta las características enunciadas en la sección anterior.

3.1 3APL

El lenguaje 3APL [3] permite implementar agentes cognitivos. Provee primitivas para manejar metas, creencias y capacidades básicas (como la actualización de conocimiento, acciones externas o acciones de comunicación) y provee un conjunto de reglas de razonamiento a través de las cuales las metas del agente se actualizan. 3APL a sido desarrollado por el *3APL group* del Departamento de las Ciencias de la Computación de la universidad de Utrech en el año 2003.
Características positivas

- Es un lenguaje para especificar/implementar agentes deliberativos.
- Sus agentes siguen una estructura fija, pero permite realizar llamadas a métodos JAVA.
- Provee primitivas para manejar las metas, creencias y capacidades básicas.
- Brinda un conjunto de reglas de razonamiento a través de las cuales las metas del agente se actualizan.
- El interprete es Open-Source y en su informe, explica como funcionan y se pueden cambiar las primitivas deliberativas que manejan toda la simulación.
- Es un Lenguaje declarativo.
- Provee mecanismos de comunicación entre agentes.
- Posee una sintaxis similar a la de Prolog.

Características negativas

- Brinda facilidades Multi-Agente pero sólo para agentes corriendo en una misma máquina.
- No brinda ningún tipo de manejador de directorio o facilítador.
- Los agentes participantes del SMA deben ser creados por algún agente activo del sistema.
- Funciona por ”turnos” (ciclos deliberativos).
- Las posibilidades de sincronización entre agentes son pobres.
- No sigue estándares para la comunicación.
- La capacidad para planificar es limitada.
- Las actualizaciones de la base de conocimiento dependen del usuario.

En la figura 2 se muestra el código para las funcionalidades básicas del agente 1 utilizado en el ejemplo de la sección anterior. Este agente tiene como objetivos ofrecer archivos de video y descargar archivos de audio

```
Agente1
CAPABILITIES:
{} RecibirArchivo(Agente,Item) { ... }
{} EnviarArchivo(Agente,Path) { ... }

BELIEFBASE:
agente(agente2),agente(agente3),
producto(video,hola,"C:\hola.avi")

GOALBASE:
Buscar(Audio),Ofrecer(Video)

RULEBASE:
Buscar(Prod) <- agente(X) and X <> 0 |
    BEGIN Para todos los X send(X,busco,Producto);
    Obtener(Producto) END,

Obtener(Prod) <- received(A,hay,Prod,Item) |
    BEGIN send(A,Transferir,Item);
    RecibirArchivo(A,Item);
    Buscar(Prod) END,

Obtener(Prod) <- | Buscar(Prod)
Ofrecer(Prod) <- received(A,transferir,Prod)
    AND producto(Prod,Name,Path) |
    BEGIN EnviarArchivo(A,Path);
    Ofrecer(Prod) END

Ofrecer(Prod) <- received(A,busco,Prod)
    AND producto(Prod,Name,Path) |
    BEGIN send(A,hay,Prod,Name);
    Ofrecer(Prod) END
```

Figure 2: Agente1.3apl

Para que funcione este sistema los agentes 1, 2 y 3 deben ser creados por un mismo agente. Esto es muy restrictivo y es debido al pobre soporte de comunicación que provee 3APL. Como se puede ver, el hecho de que 3APL no cuente con un sistema de administración de agentes, hace que el agente 1 deba tener almacenado en su base de conocimiento los agentes que con los que se comunicará. La forma del agente comunicación es ad-hoc por que no se impone el uso de estándares.

3.2 Agent-O

La plataforma Agent-O [7] esta basada en el paradigma Programación Orientada a Agentes (AOP), presentado en 1993 por Shoham. En esta plataforma los agentes pueden ser vistos como entidades autónomas que poseen componentes mentales como creencias, capacidades, decisiones y obligaciones. Actualmente existe un sólo intérprete para el paradigma planteado por Shoham, que está implementado en Lisp. Sin embargo, varios de los frameworks Multi-Agente utilizan en mayor o menor medida conceptos del paradigma AOP.
Características positivas

- Plantea a los agentes como entes deliberativos.
- Para la comunicación se utilizan sólo 3 formas comunicativas.
- Los componentes mentales son planteados por separado.
- Es un Lenguaje declarativo.

Características negativas

- No brinda una forma explícita de especificar las metas de los agentes.
- En este lenguaje se asume explícitamente que el interprete da soporte para el manejo de los mensajes.
- El funcionamiento de la herramienta es en ciclos y no en tiempo real.
- No brinda ningún tipo de manejador de directorio o facilítador.
- La implementaciones encontradas en internet no poseen entornos amigables para el desarrollador.
- En Agent-O se plantea a las obligaciones como acciones directas para los agentes, es decir se utiliza planning limitado.
- En los informes no se aclara como se actualiza la base de conocimiento.

4 CONCLUSIÓN

En este trabajo se enunciaron las diferentes características deseables de un lenguaje de implementación de agentes, mostrando los beneficios de las mismas a través de ejemplos, como el soporte para primitivas de comunicación estándar, el uso del sistema de administración de agentes y facilítador de directorio, y el uso de un lenguaje formal de especificación de agentes en la herramienta de desarrollo. Por último se realizo una evaluación de dos lenguajes de implementación de agentes que utilizan un lenguaje formal para especificar sus agentes.

Del análisis de Agent-O y 3APL, y ejemplo presentados para este último lenguaje, se puede concluir que estos lenguajes brindan buenas abstracciones para los estados metales de los agentes, pero dejan de lado importantes aspectos de la comunicación como son la utilización de estándares y la utilización de sistemas de administración de agentes. Lo ideal para el desarrollo de agentes que participan de un SMA, sería que estas herramientas, además de contar con un importante soporte para el lenguaje de especificación de agentes, brinden todas las características importantes para la comunicación entre los agentes.

References

[1] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE — A FIPA-compliant agent framework. In *Proceedings of the 4th International Conference on the Practical Applications of Agents and Multi-AgentSystems (PAAM-99)*, 1999.

[2] Paolo Busetta, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. Jack - summary of an agent infrastructure. Technical report, 1999.

[3] M. Dastani, B. van Riemsdijk, F. Dignum, and J. Meyer. A programming language for cognitive agents: Goal-directed 3apl, 2003.

[4] FIPA. Foundation for intelligent physical agents. http://www.fipa.org.

[5] N. Jennings and M. Wooldridge, editors. *Agent Technology : Foundations, Applications, and Markets*. Springer-Verlag, Berlin, 1998.

[6] Haycinth S. Nwana. Software agents: An overview. In *Knowledge Engineering Review*, 1996.

[7] Mark C. Torrance and Paul A. Viola. The agent-o manual. Technical report, 1991.