Desarrollando Sistemas Multiagentes sobre AgentNet

Gerardo Rossel[1], Andrea Manna[2]
CAETI- Facultad de Tecnología Informática - UAI
Departamento de Computación - FCEyN - UBA
(1)grossel@computer.org (2)amanna@dc.uba.ar

Resumen. En este artículo se presenta el desarrollo de sistemas multiagentes utilizando la plataforma AgentNet. Se describe la implementación y facilidades que provee AgentNet para la realización de sistemas multiagentes y la posibilidad de embeber el micro-núcleo en diferentes aplicaciones. Se verá, además, una descripción de los detalles arquitectónicos de la plataforma y su integración con E-MOBI . Finalmente un caso de estudio es presentado.

Palabras Clave. Inteligencia Artificial, Sistemas Multiagentes, Agentes Inteligentes, Plataformas de Agentes.

1. Introducción

En la última década, ha crecido enormemente la popularidad de los sistemas basados en agentes multiagentes debido a que éstos brindan a las aplicaciones de software, mayor facilidad para lograr autonomía, razonamiento y movilidad. A su vez, el paradigma de agencia permite modelar aplicaciones en una forma más cercana a la que los humanos perciben el dominio del problema, yendo un paso más allá en la reducción del "gap" semántico que los sistemas orientados a objetos. De todos modos, una visión más justa indicaría que el paradigma de agentes se complementa con el paradigma de orientación a objetos para el desarrollo de software[8].
El presente artículo se estructura como sigue: en esta sección se presenta una introducción a los conceptos básicos de agencia. Luego, en la segunda sección, se realiza una descripción de la arquitectura de AgentNet y su facilidad para el desarrollo tanto de agentes de grano fino (como los utilizados en vida artificial) como de grano grueso. En la sección 3 se describe brevemente la integración con E-MOBI. En la sección 4 se muestra, a partir de un ejemplo, como se implementan sistemas multiagentes en AgentNet [1]. Para finalizar, las conclusiones y planes de trabajos futuros.
Si bien no hay una definición universalmente aceptada del término *agente,* sí existen criterios más o menos generales acerca de algunas de las características que deben poseer los sistemas basados en agentes. En ese, sentido surgen las nociones débiles y fuertes de agencia [4]. Se considera entonces a un agente como a "*un sistema de computación que está ubicado en algún ambiente y que es capaz de accionar autónomamente en dicho ambiente en orden a cumplir sus objetivos*" [6]. La definición dada se basa en la noción débil de agencia que implica que los agentes tienen autonomía, habilidad social, son reactivos y pro-activos. La noción fuerte de agencia, en donde se ubican también los agentes inteligentes e incluyen una serie de actitudes mentales, implica además: movilidad, veracidad, benevolencia y racionalidad[4].
La concepción del software como un conjunto de entidades interactuando, acorde a las tendencias actuales en el desarrollo de aplicaciones distribuidas, dan lugar a los sistemas multiagentes. Los sistemas multiagentes, son aquellos consistentes de un conjunto de agentes interactuando entres sí. Los agentes pueden ubicarse en el mismo equipo o en equipos remotos sobre una red. Esto exige nuevas habilidades, no presentes en agentes aislados, relacionadas con la interacción, tales como coordinación, cooperación y negociación.

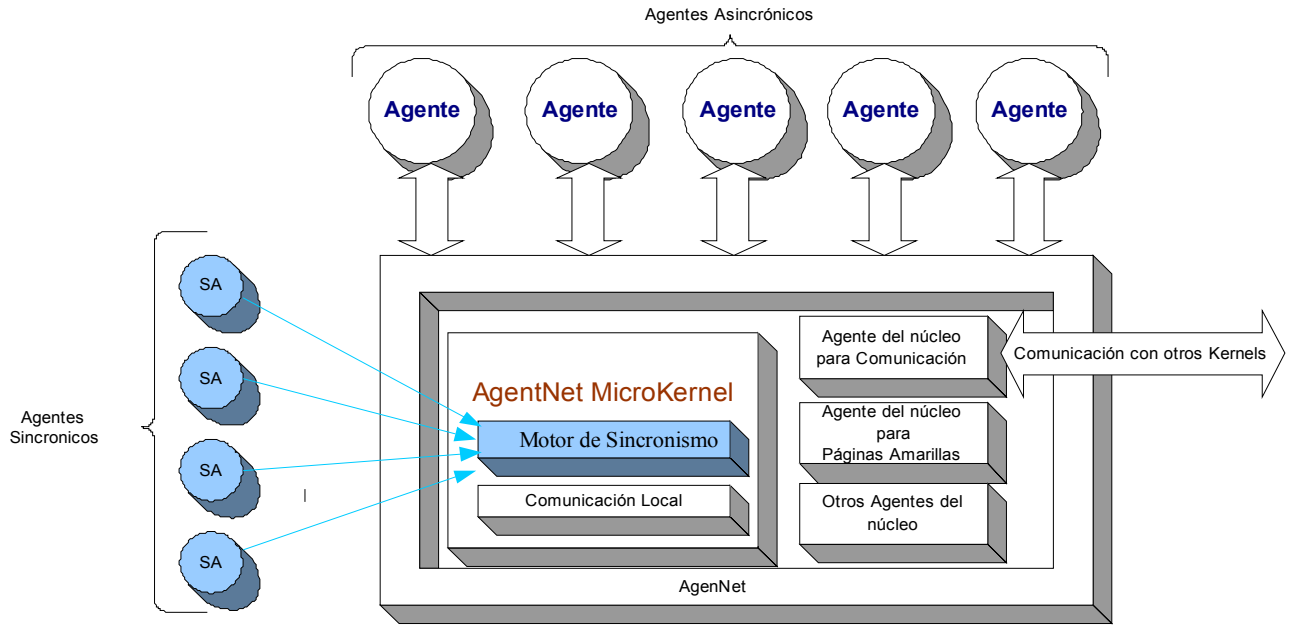
Por lo tanto, el desarrollo de sistemas basado en agentes trata con dos aspectos: por un lado la arquitectura interna de los agentes individuales y por otro la organización, distribución e interacción de los mismos, es decir el aspecto social.

2. AgentNet

En está sección, se describen los aspectos arquitecturales de AgentNet y la utilización del *Microsoft .Net framework* para su construcción. AgentNet es una plataforma y un *framework* para el desarrollo de sistemas multiagentes basada en micro-núcleo (*microkernel*) desarrollado en el *Grupo de Investigación y Desarrollo en Agentes y Sistemas Inteligentes* del CAETI. Es una plataforma porque provee la infraestructura necesaria para la comunicación y la ejecución de agentes y es un *framework* [25] ya que provee una serie de estructuras que facilitan la construcción de agentes mediante la instanciación de las mismas. Es posible extender la plataforma AgentNet con nuevas funcionalidades que permitan luego instanciar diversas arquitecturas de agentes. La integración con E-MOBI con el soporte de múltiples estrategias de razonamiento, es un ejemplo de ello [3].
Actualmente, existen diversas herramientas, frameworks y plataformas para la construcción de sistemas multiagentes ([10,11,12,13]). La mayoría ellos son desarrolladas con la tecnología de la plataforma Java 2 (ya sea en las versiones empresarial o estándar J2EE o J2SE) usando el lenguaje Java [9]. Una lista de estos y otros trabajos puede obtenerse desde [14].
Una de las limitaciones de estos trabajos está dada por la imposición de un lenguaje preferencial para el desarrollo de los agentes. Si bien plataformas como Madkit o Jade permiten el desarrollo utilizando otros lenguajes, la integración de los mismos no es directa y presenta dificultades a la hora de encarar la construcción de sistemas. Java no cumple con los requisitos de ser un lenguaje adecuado para la programación orientada a agentes [6], lo que llevó a que se construyeran extensiones al mismo para facilitar el soporte de actitudes mentales [15].
El lanzamiento del *.Net framework* da una nueva perspectiva al desarrollo de aplicaciones de agentes y sistemas multiagentes. Las características remarcables de .Net como ser: neutralidad del lenguaje, facilidades de acceso y construcción de servicios de red, servicios Web, *remoting,* entorno de ejecución virtual, introspección y carga dinámica, lo tornan en un entorno adecuado para una nueva generación de herramientas orientadas a la construcción de agentes.
Una de las características que se consideró importante para la construcción de AgentNet, es la posibilidad de desarrollo multilenguaje. Si bien el *microkernel* y algunos de los principales agentes del núcleo fueron construidos utilizando el lenguaje C# [16], los agentes para la plataforma AgentNet pueden desarrollarse en otros lenguajes (VB, Delphi, Eiffel, ComponentPascal, etc.), de hecho cualquier lenguaje que compile al CLI. La integración con E-MOBI, por ejemplo, es una muestra del desarrollo de agentes en el lenguaje Eiffel [17].
El diseño de AgentNet tiene como base la arquitectura de microkernel y la agentificación de servicios. El microkernel tiene como tareas principales la comunicación local que se realiza mediante el envío de mensajes asincrónicos, el registro de agentes y el soporte de sincronismo para agentes de grano fino. La comunicación remota se logra mediante la utilización de .Net *remoting y* es realizada por agentes especiales llamados *agentes del núcleo* de la misma forma que el resto de los servicios (páginas amarillas, *blackboard* de comunicación, etc.).

2.1 Arquitectura de AgentNet

El corazón de AgentNet es el microkernel, quien se encarga fundamentalmente de la comunicación local, del soporte de sincronismo y la administración del ciclo de vida de los agentes. La siguiente figura muestra un esquema de la arquitectura de AgentNet.



Los agentes asincrónicos son los que se utilizan en aplicaciones distribuidas y cada uno se ejecuta en su propio hilo. Algunos agentes asincrónicos pueden tener una interfaz gráfica de usuario y otros trabajar en segundo plano. La comunicación entre ellos es lograda mediante el intercambio asincrónico de mensajes. AgentNet provee una librería para definir distintos protocolos y lenguajes de comunicación de agentes. Por ejemplo, es posible enviar y recibir mensajes utilizando FIPA ACL [18] o simplemente como una cadena de caracteres.
Los agentes sincrónicos son aquellos agentes de grano fino que no se ejecutan en su propio hilo sino que el motor de sincronismo se encarga de ir turnando su ejecución. Este tipo de agentes son ideales para modelar aplicaciones de simulación de vida artificial y colonia de hormigas.
El microkernel es el responsable de la asignación de un identificador único para cada agente. El identificador se compone de un identificador único local y la dirección del *núcleo*. Esta última se compone de la dirección IP y el puerto principal para las comunicaciones remotas. De esta forma, cada agente tiene un único identificador a lo largo de una red de núcleos interconectados.

2.2 Agentes en AgentNet

Se describirán los dos tipos generales agentes: sincrónicos y asincrónicos. Todo agente tiene un ciclo de vida administrado por el micro-núcleo. El mismo cuenta con las siguientes etapas:

- Nacimiento
- Vida
- Muerte

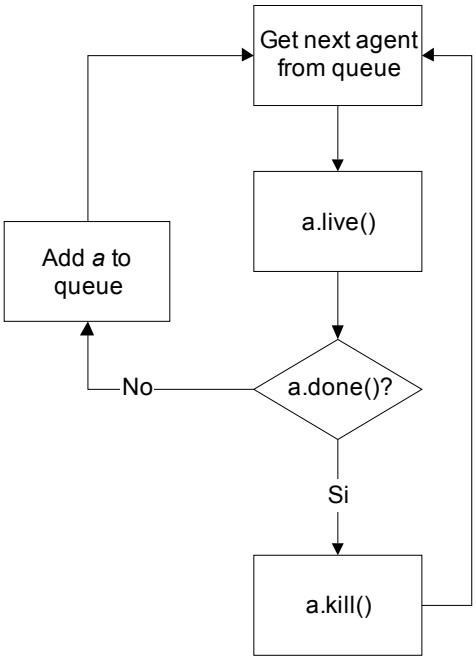
De estas etapas, sólo la segunda es obligatoria. Es responsabilidad del desarrollador implementar el comportamiento en cada estado del ciclo de vida. El agente no es eliminado una vez finalizada la ejecución de su etapa de vida, sino mediante la destrucción explícita del mismo, ya sea por el núcleo o por otro agente autorizado. El siguiente fragmento de código muestra un simple agente asincrónico “Hola Mundo”. Para definir un agente, hay que definir la clase correspondiente al mismo como subclase de *Agent* o *AgentGUI* (en caso de tener interfaz gráfica).

```
/// <summary>
/// Simple HelloWorldAgent.
/// </summary>
public class HelloWorldAgent: Agent
{

    public override void born()
    {
        Console.WriteLine("I'm alive!!!");
    }
    public override void live()
    {
        Console.WriteLine("Hellow World");
    }
    public override void death()
    {
       Console.WriteLine("I was killed!!!");
    }

}
}
```

En sistemas donde es necesario lanzar cientos o miles de agentes (como aquellos de colonias de hormigas o vida artificial), es muy costoso en términos de recursos que cada uno se ejecute en su propio hilo de ejecución. Para estos casos, el motor de sincronismo provee el soporte para la planificación o *scheduling* de muchos agentes utilizando un solo hilo de ejecución para los mismos. Cada agente sincrónico debe implementar un método denominado *done()* que indica que el agente ha terminado su tarea y entonces el motor de sincronismo puede eliminarlo de la lista de agentes sincrónicos. El siguiente diagrama muestra un esquema del flujo de ejecución desde la perspectiva del motor de sincronismo.

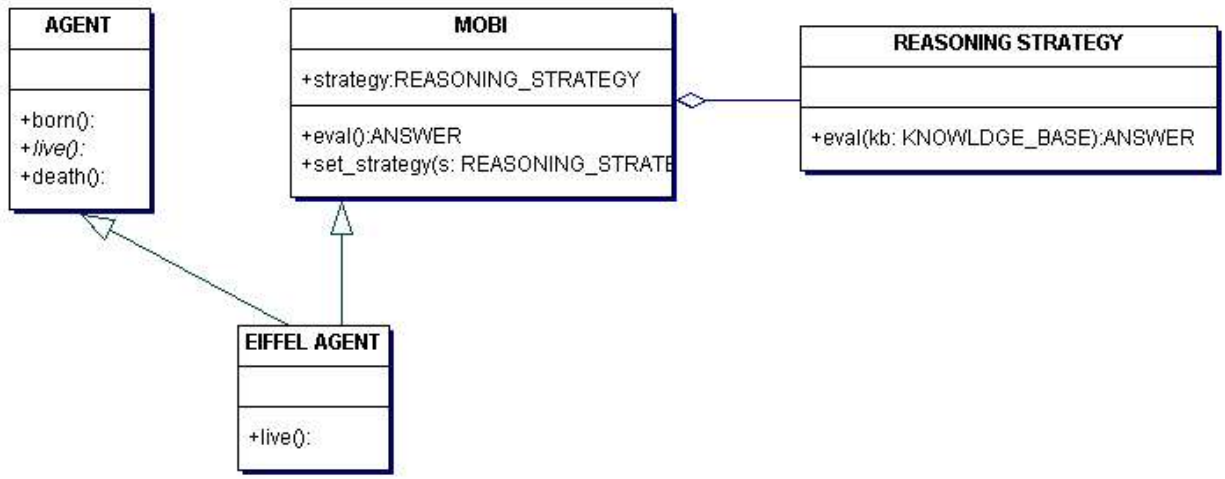


Como puede verse en el diagrama, la ejecución de agentes sincrónicos no es *preemptive* como en el caso de los agentes sincrónicos de grano grueso, sino cooperativa. El desarrollador es responsable

de permitir la ejecución de otros agentes. Un mecanismo similar es implementado en JADE para administrar la planificación de la ejecución de los comportamientos asociados a los agentes [12], pero la diferencia radica en que los agentes en sí tienen un mecanismo de planificación de comportamientos , mientras que, en este modelo, lo que se planifica es la ejecución de los diferentes agentes.

3. Integración con E-MOBI

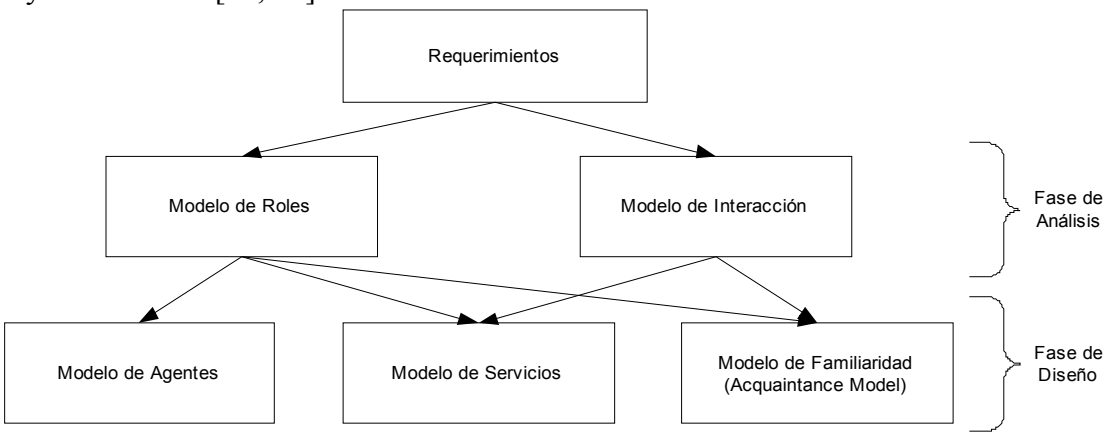
E-MOBI [2] es una implementación en el lenguaje Eiffel[17] de un modelo de objetos inteligentes. Esta implementación, que originalmente permitía la incorporación de reglas en el formato de cláusulas de Horn y utilizaba *SLD-Resolution* (Linear Resolution for Definite Clauses with Selection function) ([19,20]) como mecanismo de inferencia, actualmente soporta otras estrategias de razonamiento y es posible incorporarle estrategias ad-hoc[3]. La utilización de múltiples estrategias de razonamiento permite la construcción flexible de agentes que adapten su mecanismo de inferencia al problema que intentan resolver.
La integración de E-MOBI se logró mediante la traslación del código original de E-MOBI para adaptarlo al framework de .NET. Se aprovecharon las facilidades de introspección del framework y las nuevas características del lenguaje Eiffel como el soporte de rutinas como objetos[21]. La construcción de agentes en Eiffel para la plataforma de AgentNet se logra mediante la utilización de múltiple herencia como se ve en el siguiente diagrama UML simplificado:



4. Desarrollo de un Sistema Multiagente

La plataforma AgentNet permite reducir la complejidad y el tiempo del desarrollo de sistemas multiagentes. La misma puede utilizarse con diversas metodologías como Gaia [22,23] o MaSE [24].
Gaia es una metodología de desarrollo de sistemas multiagentes de propósito general orientada fundamentalmente a las fases de análisis y diseño. Gaia permite tratar tanto con el aspecto de macro-nivel, o sea el aspecto social de interacción entre agentes, como con los aspectos de diseño individual de agentes o micro-nivel. Gaia tiene un fuerte punto de vista organizacional. Tomando como punto de partida los requerimientos, la fase de análisis permite obtener modelos de la estructura del sistema en términos de dos modelos: el modelo de roles y el de interacción, sin referenciar a detalles de implementación. La fase de diseño toma como punto de partida los modelos de análisis y define la estructura del sistema de agentes en forma de tres modelos: modelo de

agentes, modelo de servicios y modelo de familiaridad (Acquaintance). La siguiente figura muestra las fases y sus modelos [22, 23]:



El modelo de agentes se corresponde directamente con la plataforma AgentNet. Cada tipo de agente del modelo de agentes se implementa como una subclase de *Agent* en AgentNet. El modelo de servicios tiene correspondencia con los métodos en la clase correspondiente al agente que brinda el servicio. A su vez, a partir del modelo de servicios y el modelo de *acquaintance*, puede establecerse la estructura de mensajes que se utilizará en la interacción. La misma puede hacer uso de las clases de mensajes ya predefinidas como las que implementan FIPA-ACL o KQML, o utilizarlas como clases base para crear nuevos tipos de mensajes.
Si la metodología es MaSE, la utilización de AgentNet es también bastante directa. MaSE tiene una fuerte influencia de los métodos orientados a objetos y también cuenta con una fase de análisis y una fase de diseño. Las tareas de la fase de análisis son:

- Capturar *Goals*
- Aplicar casos de uso
- Refinar Roles

mientras que las tareas la fase de diseño serían:

- Crear Clases de agentes
- Construir conversaciones
- Definir la arquitectura interna de cada agente
- Definir la estructura del sistema mediante el uso de diagramas de *deployment*.

Claramente, MaSE asigna una clase para implementar los tipos de agentes, lo cual se traduce en una subclase de *Agent* o de *AgentGUI* en AgentNet. La estructura de las conversaciones indica la secuencia de mensajes a enviarse. Luego, al definir la estructura del sistema, se definen los núcleos de AgentNet a utilizar y su distribución, por ejemplo:

- Todos los agentes en un solo núcleo embebido en una aplicación
- Núcleos en diversas ubicaciones interactuando entre sí
- Núcleos embebidos en diversas aplicaciones

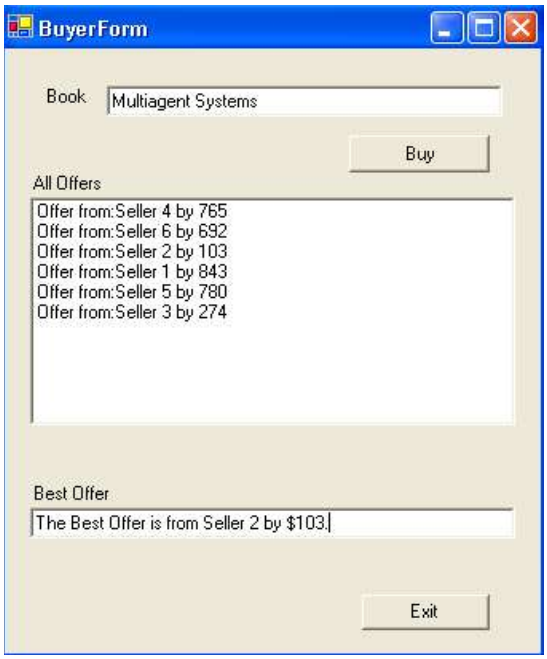
Junto a ello, se define en que núcleos deben ubicarse los agentes, es decir, la política de distribución de agentes. La comunicación entre agentes remotos es resuelta por la plataforma. Cada metodología tiene sus fortalezas y debilidades, por lo cual AgentNet no impone restricciones en ese sentido.

4.1 Un ejemplo

Supongamos que en un modelo de agentes para un sistema multiagente de compra venta de libros existen dos agentes:

- Un agente comprador que actúa en representación de un usuario y cuenta con una interfaz gráfica para que se puedan ingresar los requerimientos de libros a comprar.
- Un agente vendedor que actúa en representación de una librería y no provee interfaz gráfica. Simplemente espera una solicitud y luego envía una oferta.

El agente comprador realiza una solicitud de ofertas, recibe las mismas y luego selecciona en función de las preferencias del usuario. La implementación del mecanismo de selección puede ser realizada en un ambiente de reglas como E-MOBI, JESS o Prolog. La siguiente figura muestra la interfaz GUI del agente comprador luego de recibir las ofertas de los vendedores:



Para implementar al comprador, es necesario crear una subclase de *AgentGUI*. En la implementación el agente se mantiene a la espera de solicitudes por parte del usuario, para lo cual la plataforma provee las facilidades de bloqueo/activación necesarias. La clase *AgentGUI* cuenta con un método abstracto *intiForm()* que es el lugar donde las subclases deben inicializar las interfaces gráficas. Dado que la GUI se ejecuta en un hilo diferente del hilo principal del agente, la comunicación entre la interfaz GUI y el agente correspondiente se debe realizar en forma *thread-safe*, lo cual se logra mediante el uso de delegados y el método *Invoke*.

En el siguiente fragmento de código, se ve un esquema de los métodos que administran el ciclo de vida del agente comprador. Se puede ver que la solicitud de oferta es enviada en forma de *broadcast* pero podría usarse el servicio de páginas amarillas para obtener una lista de agentes vendedores. De esta forma, los agentes no interesados o con otras tareas/servicios, no reciben la solicitud:

```
public class AgBookBuyer : AgentGUI
{
    private BuyerForm bf;
    private bool moreBuys = true;
    public string bookName;

    public override void initForm()
    {
        bf = new BuyerForm();
        bf.myAgent = this;
    }
    public override void born()
    {
        showForm(bf);
    }
    public override void live()
    {
        while (moreBuys)
        {
            Suspend();//wait for user input
            if (moreBuys)
            {
                MessageACL msg = new MessageACL(PerformativeACL.CFP, bookName);
                broadcastMessage(msg);
                Sleep(1000);
                handleMessages();
            }
        }
    }
    public override void death()
    {
        if (bf != null )bf.Close();
    }
```

Los mensajes entre el comprador y los vendedores son enviados como mensajes FIPA-ACL con *performative* RFP (*Request For Proposal*) [18] y el nombre del libro como contenido del mensaje. Los agentes vendedores se implementan como subclase de *Agent.* El siguiente fragmento de código muestra una implementación simplificada del método *live()* del agente vendedor:

```
public override void live()
{
  while (true)
  {
    Message msg = waitMessage();
    if (msg is MessageACL)
    {
      MessageACL mac1 = (MessageACL) msg;
      if (mac1.Performative == PerformativeACL.CFP )
      {
        offer = findPrice(msg.Content);
        if ( offer > 0 ){
          MessageACL reply = new MessageACL(PerformativeACL.PROPOSE,offer.ToString());
          sendMessage(mac1.SenderID, reply );
        }
      }
    }
  }
}
```

El agente vendedor se bloquea a la espera de que arribe un mensaje. El kernel se encargará de despertar al agente al llegar el mensaje, tal es la semántica del método *waitMessage()*. Luego controla que el mensaje sea un mensaje de FIPA-ACL y que la preformativa sea un CFP. Obtiene el contenido del mensaje (el título del libro), busca el precio adecuado y envía la oferta como una

réplica. Toda la manipulación de mensajes, bloqueo a la espera de algún suceso, etc., forman parte de la plataforma.
Por su parte, el agente comprador cuenta con método específico, llamado *handleMessage()*, para la recepción y administración de las ofertas. El método *addResponse()* es el responsable de dar la valorización a la respuesta según los parámetros del usuario (menor tiempo de entrega, precio más bajo, etc.). A continuación se ve el código simplificado de la administración de mensajes:

```
private void handleMessages()
{
    Message msg = waitMessage();
    addResponse(msg);
    while (!isMessageBoxEmpty())
    {
        msg = getNexMessage();
        addResponse(msg);
    }
}
```

Como se ve, la plataforma facilita la construcción de sistemas de agentes permitiendo que el desarrollador se concentre en la arquitectura de los agentes, los caminos de interacción, la inteligencia apropiada para los agentes, etc. Las tareas rutinarias de mensajería, implementación de hilos de ejecución, tiempos de espera, etc., quedan en manos de AgentNet.
El desarrollo de sistemas multiagentes obliga a flexibilizar las posibilidades de implementación de la plataforma. Por dicho motivo, se provee con la posibilidad de embeber la plataforma en otras aplicaciones. De esta manera, es posible diseñar aplicaciones que hagan uso de los agentes embebiendo el micro-núcleo en las mismas. Esto facilita el desarrollo de ambientes de prueba con el micro-núcleo embebido y que puedan disparar agentes que sean cargados dinámicamente. Dicha facilidad es importante para modelos de simulación, desarrollo de prototipos, investigación y la enseñanza de la tecnología en los cursos universitarios.

5. Conclusiones y trabajo futuro

Se ha presentado la utilización de la plataforma AgentNet para la construcción de sistemas multiagentes y ejemplificado en el caso del comprador/vendedor de libros. Se mostró que puede utilizarse AgentNet conjuntamente con metodologías como Gaia o MaSE y la posibilidad de incorporar agentes de E-MOBI.
Se planifica extender este trabajo en diversas direcciones. Por un lado, agregar más agentes especializados del micro-núcleo para resolver temas claves como es el de seguridad. Por otro lado, se pretende crear una estructura de agentes que facilite el desarrollo de agentes con arquitectura BDI *(belief-desire-intention*) [25]. Si bien nada impide desarrollar agentes BDI sobre AgentNet, aún no hay facilidades específicas para ello. Se tiene la intención de desarrollar dichas facilidades en forma de *framework*[26]. Además, se proyecta la realización de una herramienta de diseño asistido que permita el desarrollo de sistemas desde los modelos de análisis hasta su implementación en AgentNet.

Bibliografía

[1] Gerardo Rossel, Andrea Manna. *La plataforma AgentNet*. Reporte Técnico CAETI (2004)

[2] Gerardo Rossel, Andrea Manna. *E-MOBI Smart Object Model and Implementation* en *Journal of Object Technology* vol. 2, no. 6, November-December (2003).

[3] Gerardo Rossel, Andrea Manna. *Implementando Múltiples Estrategias de Razonamiento en E-MOBI*. . IV Workshop de Agentes y Sistemas Inteligentes. IX Congreso Argentino de Ciencias de la Computación: CACIC (2003)

[4] Michel Wooldridge, Nicholas Jennings, *Intelligent Agents: Theory and Practice* en *Knowledge Engineering Review*. (1995).

[5] Yoav Shohan. Agent Oriented Programming. *Artificial Intelligence*, Vol. 60, No 1. (1993).

[6] Michel Wooldridge. *Intelligent Agents* en *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Chapter 1. The MIT Press. Gerhard Weiss editor.(1999)

[7] Michel Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons (2002)

[8] Michel Wooldridge, Nicholas Jennings, *Pitfalls of Agent-Oriented Development* en *Proc. Second Int'l Conf. Autonomous Agents*, ACM Press, New York, pp. 385–391.(1998)

[9] Gosling J, Joy B., Steele G, Bracha G *The Java Language Specification, Second Edition* Addison-Wesley. (2000)

[10] BBN Technologies, *Cougaar Architecture Document V10.0.* http://www.cougaar.org/ (2003) [última visita Julio 2004]

[11] Bellifemine, F., Caire, G., Trucco, T., Rimassa, G *Jade Programmer's Guide* JADE 2.5 http://sharon.cselt.it/projects/jade/. (2002)

[12] Bellifemine, F., Rimassa, G., and Poggi A.. *JADE – A FIPA- compliant agent framework* en *4th International Conference on the Practical Applications of Agents and Multi-Agent Systems* (PAAM-99) . (1999)

[13] Gutknecht, O., Ferber, J., Michel, F. *The Madkit Agent Platform Architecture* Rapport de Recherche, Universit´e Montpellier II. (2000)

[14] AgentLink Web Page http://www.agentlink.org/resources/agent-software.php [última visita Julio 2004].

[15] A. Zunini, L. Berndon, A. Amandi. *JavaLog un lenguaje para la Programación de Agentes.* Revista Iberoamericana de Inteligencia Artificial. No. 13 (2001).

[16] Hejlsberg, A. and Wiltamuth, S. *Microsoft C# Language Specification*". Microsoft Press, Redmond, WA. (2001)

[17]Bertrand Meyer. *Eiffel: The language* Prentice Hall. (1992)

[18] FIPA Standards. *FIPA ACL Message Structure Specification.* Document SC00061G, (2002).

[19] Robinson J.A. *A Machine Oriented Logic Based on the Resolution Principle* Journal of the ACM, 12, 1965.

[20] Loveland, D.W. *Mechanical theorem-proving by model elimination* Journal of the ACM,15. 1968.

[21] Bertrand Meyer. *Eiffel the Language Third Edition*. Trabajo en Progreso (versión borrador accedida en Enero del 2004).

[22] Michael Wooldridge, Nicholas R. Jennings, David Kinny. *The Gaia Methodology for Agent-Oriented Analysis and Design*. Kluwer Academic Publishers, (2000).

[23] Franco Zambonelli, Nicholas R Jennings, Andrea Omicini, Michael Wooldridge. *Agent Oriented Software Engineering for Internet Applications* en *Coordination of Internet Agents Models Technologies and Applications* Capítulo 13 (2000).

[24] Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman. *Multiagent systems engineering* en. *International Journal of Software Engineering and Knowledge Engineering*. (2001)

[25] Fayad, M.; Schmidt, D. *Building Application Frameworks: Object-Oriented Foundations of Design*. First Edition, John Wiley & Sons(1999).