Revista Venezolana de Computación
ISSN: 2244-7040
http://www.svc.net.ve/revecom
Vol. 4, No. 1, pp. 47-58, Junio 2017
Fecha recepción: 28/04/2017, Fecha aceptación: 10/10/2017

VMAS-Modeller: Una Aplicación Visual para el Modelado de Sistemas Multi-Agentes Guiado por la Metodología MASINA

Sredny Buitrago[1], Manuel Sánchez[1]
sredny.nevermind92@gmail.com, mbsanchez@unet.edu.ve

[1] Departamento de Ingeniería en Informática, Universidad Nacional Experimental del Táchira, San Cristóbal, Venezuela

Resumen: El presente artículo describe un software para el modelado y generación de código de Sistemas Multi-Agentes siguiendo las fases y modelos establecidos por la metodología MASINA El código generado corresponde a la estructura principal de clases que conforman el proyecto que está siendo desarrollado por el usuario, dicho código se ejecutará en la plataforma de agentes JADE; además, incluye una serie de funcionalidades que hacen al software bastante atractivo para desarrolladores y diseñadores de SMA. El principal propósito de la aplicación es generar una estructura básica de clases que integren el proyecto partiendo de los modelos de MASINA: modelo de agentes, modelo de tareas, modelo de comunicaciones, modelo de coordinación y modelo de inteligencia, además permite la exportación del proyecto en formato XML versión 1.0, copiar la descripción de los modelos en el portapapeles para facilitar la creación de documentación, ensayos, artículos académicos, etc. Por otro lado, se utilizó FDD (Desarrollo Guiado por Funcionalidad, en español) como metodología de desarrollo y MDA (Arquitectura Guiada por Modelos) como metodología para la interpretación y generación de código a través de los modelos.

Palabras Clave: Sistemas Multi-Agentes; Herramientas CASE; JADE; MASINA; Generación de Código.

Abstract: This paper describes a software solution for modeling and code generation of Multi-Agents Systems following the steps and models established in MASINA methodology. The generated code is the main structure of the classes that compounds the whole project in JADE platform; also, it involves a series of functionalities that makes the developed tool very attractive for SMA designers. The main purpose of the application is to generate a basic structure of the classes that integrate the project in JADE starting from the models of MASINA, these models are: agents model, task model, communication model, coordination model and intelligence model, also provides some others helpful features as exporting the current project in a XML v-1.0 file and copy the models in the clipboard to generate documentation, papers, academics articles, etc. In other hand, the methodology used in the development of this software was FDD (Feature Driven Development) and MDA (Model Driven Architecture) for the code generation.

Keywords: Multi-Agents Systems; CASE Tools; JADE; MASINA; Code Generation.

I. Introducción

La automatización industrial ha traído consigo la aparición de nuevos retos con respecto a cómo dar solución a cada uno de los problemas de manera eficiente y eficaz, muchos de estos problemas tienen que ver con tiempos de respuestas muy lentos, inadaptabilidad de los sistemas a nuevos estímulos del entorno, autonomía, descentralización, entre otros. A lo largo del tiempo se han propuesto distintos tipos de soluciones que permiten resolver los problemas mencionados de forma computacional; una de ellas son los Sistemas Multi-Agentes (SMA), éstos sistemas han comenzado a acrecentar con fuerza su presencia en el mundo del software debido a las posibilidades que ofrecen, la diversa aplicabilidad y la cantidad de plataformas que los soportan. En Efecto, un SMA puede representar una solución adecuada y óptima de acuerdo a la naturaleza del problema que se afronta, puesto, que éstos sistemas tienen como premisa la división del trabajo en tareas que serán posteriormente ejecutadas por los agentes de forma paralela, de tal forma que se aumenta la eficiencia del sistema proveyendo un tiempo de respuesta aceptable. Por otra parte, el área de diseño y creación de SMA aplicado a la automatización industrial se va expandiendo día a día, debido a que se presenta como solución a problemas de diversa índole y complejidad; Sin embargo, la creación de un SMA es un proceso extenso en el que se deben tomar en cuenta muchos factores [1], por lo que se han diseñado y postulado diversas metodologías para la creación de estos sistemas, entre las que se encuentra la metodología MASINA [2] (MultiAgent Systems for INtegrated Automation), la cual es una propuesta metodológica para la especificación e implementación de sistemas basados en agentes en ambientes de automatización industrial. En consecuencia, MASINA abarca la creación de SMA pasando a

través de 5 fases: conceptualización, análisis, diseño, codificación y pruebas. En cada una de las fases de ésta metodología, se crean diagramas, mediante el llenado de las plantillas que aporta la misma, lo que permite al diseñador del SMA, contar con información importante que permite definir las propiedades, objetivos, tareas, modelos de interacción, comunicación, inteligencia, entre otros, de los agentes del sistema. En algunas ocasiones, resulta confuso para los diseñadores del SMA llenar apropiadamente (con la información adecuada para cada campo) las plantillas que define la metodología, situación que es frecuente entre los usuarios que tienen poco conocimiento de la metodología, lo que puede conllevar al fracaso o al mal uso de la misma, obteniendo como resultado un sistema que no se ajusta a los requerimientos de diseño definidos inicialmente. Como se mencionó anteriormente, el diseño y desarrollo de los SMA es una tarea que requiere dedicación y una gran cantidad de tiempo, por lo que es necesario que sea apoyada a través de herramientas de software especializadas que usen una metodología de diseño y un ciclo de desarrollo adecuados. Esto ha motivado a desarrollar una aplicación CASE [3] (Computer-Aided Systems Engineering) que permita automatizar el proceso de diseño y desarrollo de un SMA. En este sentido, se dice que una herramienta CASE es una aplicación que ayuda en el proceso de desarrollo de software en forma automatizada. En consecuencia, ésta investigación se enfoca en el desarrollo de una aplicación CASE para el diseño de Sistemas Multi Agentes guiados por medio de la metodología MASINA.

La herramienta desarrollada debe contar con la capacidad de generar el código inicial del proyecto para la plataforma de despliegue de agentes JADE [4] (Java Agente Development Framework) a partir de la información captada por medio de los modelos de la metodología MASINA. Por su parte JADE es un middleware que facilita el desarrollo de sistemas multi-agente bajo el estándar FIPA, que cuenta con un entorno de ejecución en el que los agentes co-existen. De esta manera, la creación de la aplicación propuesta en esta investigación va a favorecer la construcción de sistemas Multi-agentes, facilitando la creación de los diagramas de la metodología, al guiar al diseñador paso a paso a través que aporta la metodología, lo que asegura que el SMA sea creado de forma metodológica, y a su vez, facilitar a los usuarios inexpertos el uso correcto de la metodología MASINA. De igual forma, la aplicación permite ingresar la información, de una manera sencilla para el usuario, parametrizando aquellos campos con valores conocidos y validando la información ingresada por el mismo, para asegurar que el código generado sea correcto. Finalmente, la aplicación permitirá generar una gran cantidad de código que beneficiará el tiempo de desarrollo del SMA.

En general, este artículo se estructura de la siguiente manera: En la Sección II se presenta el marco teórico, en la Sección III se muestran los trabajos relacionados, la Sección IV presenta el desarrollo de la aplicación y en la Sección V se define un caso de estudio; en la Sección VI se hace una pequeña experimentación; para finalizar con algunas conclusiones acerca del producto desarrollado.

II. MARCO TEÓRICO

Los sistemas Multi-agentes son parte del paradigma de inteligencia artificial distribuida [5], se trata de soluciones de software que afrontan problemas dividiéndolos como una serie de tareas que serán ejecutadas por los entes o agentes que conforman el sistema. Un agente es esencialmente un componente de software especial que posee autonomía y provee una interfaz interoperable a un sistema principal o que se puede comportar como un agente humano, trabajando para ciertos clientes en función de cumplir su propia agenda [6]. Por tanto, es vital la comunicación entre los diferentes agentes de tal forma que se pueda alcanzar el objetivo de diseño eficazmente, por lo que es necesario contar con un protocolo que coordine la comunicación entre los diferentes entes.

Al ser el desarrollo de estos sistemas un proceso de alta complejidad se han propuesto diversas metodologías para el modelado de SMA, entre ellas tenemos las orientadas a objetos [7-14] y las metodologías basadas en el paradigma de agentes [15-22]. Sin embargo, éstas presentan deficiencias, debido a que dificultan el modelado de partes complejas del sistema mientras que otras permiten inconsistencia con respecto a la información proporcionada. MASCommonKADS, es una de las metodologías de diseño de agentes ampliamente conocidas, la cual surge como una extensión o mejora de la metodología de ingeniería del conocimiento CommonKADS al incluir componentes orientados a objetos (OO) y de la ingeniería del protocolo para la definición de los protocolos del agente [6]. MASCommonKADS se considera como la más completa y adecuada para el modelado de SMAs de automatización industrial, no obstante, ella no toma en consideración los fundamentos inteligentes del agente. Debido a lo anterior, Aguilar et. al proponen MASINA [2] como una metodología de modelado de Sistema Multiagentes supliendo así las carencias que presentan las demás metodologías. En consecuencia, MASINA es una extensión de MASCommonKADS, cuya diferencia primordial es que esta última agrega los modelos para la especificación de la inteligencia del agente, además, MASINA se orienta al modelado de procesos relacionados a la automatización industrial. Aunado a esto, MASINA permite especificar el modelo de inteligencia individual e inteligencia colectiva, así como de establecer las tareas que un agente debe realizar para lograr sus objetivos. Es importante aclarar, que la inteligencia en sistemas multi-agentes se entiende como la capacidad que tiene los agentes para ir tras sus objetivos y ejecutar las tareas de una manera optimizada [5], a la vez que permite detallar lo concerniente a la coordinación y comunicación entre los agentes involucrados en el sistema, es decir, intercambio de mensajes, y métodos para la resolución de conflictos. En consecuencia, la metodología MASINA define 5 modelos para el diseño de todo el sistema, ellos son: modelo de agente, modelo de tareas, modelo de comunicación, modelo de coordinación y modelo de inteligencia (colectiva e individual). Finalmente, cabe mencionar que MASINA agrega nuevos atributos a los modelos existentes en MASCommonKADS, de tal manera que se puede ser más específico en la descripción de los agentes. Entre esos tributos se tiene: componentes del SMA y marco de referencia al modelo de agentes. En tal sentido, en el modelo de tareas se especifican las sub-tareas y cuáles de

ellas hacen uso de técnicas de inteligencia, mientras que, en el modelo de coordinación se definen las conversaciones que se efectúan entre los agentes registrados.

Por tanto, al ser MASINA una metodología que cubre las deficiencias de las metodologías mencionadas anteriormente, y debido a las ventajas que ésta presenta respecto a las demás, se consideró para ser implementada en esta investigación, con lo que se pretende crear un software denominado Visual MAS Modeller (V-MAS) que tiene la capacidad de generar código para el despliegue de Sistemas Multi-Agentes en JADE, a partir de los modelos de la metodología MASINA.

Por otra parte, JADE se presenta como una plataforma, que provee los mecanismos necesarios para que los entes de un SMA puedan coexistir (descubrimiento, comunicación, coordinación, etc.), así como las herramientas para la detección y búsqueda de servicios que ofrecen los agentes [23]. Esta plataforma se caracteriza por su alto uso como plataforma de despliegue de agentes y su orientación al desarrollo de sistemas multi-agentes bajo el estándar FIPA [24]; Esta son las razones por las cuales se considera JADE como plataforma objetivo de Visual MAS Modeller lo que permitirá cubrir una amplia cantidad de usuarios y sistemas. Sin embargo, el sistema ha sido desarrollado con la capacidad de que en el futuro se puedan agregar generadores de código para otras plataformas de despliegue de agentes.

En el mismo orden de ideas, las herramientas CASE son programas de software cuyo objetivo es aumentar la productividad en el desarrollo de aplicaciones computacionales [25], es decir, éstas se encargan de automatizar ciertas áreas del ciclo de vida del software, ahorrando tiempo de desarrollo y, en consecuencia, los costos asociados al mismo. Por definición la ingeniería de sistemas asistida por computador es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias del desarrollo de sistemas. El propósito de esta tecnología es acelerar el proceso de desarrollo de sistemas y mejorar la calidad de los sistemas resultantes [25]. En este mismo sentido, se puede decir que hoy en día existe una amplia gama de este tipo de herramientas, que abarcan desde el modelado de sistemas, gestión de tiempos y costos de proyectos; hasta la generación de código para ciertas plataformas específicas, siendo este último el que mayor se adapta a los objetivos de V-MAS. Los generadores automáticos de código sirven como herramientas de apoyo a los desarrolladores, debido a que se puede procesar una gran cantidad de información en cuestión de segundos y ahorrar un trabajo que muy bien puede ser automatizado, produciendo, además, un código libre de errores. De esta forma V-MAS, es capaz de disminuir el tiempo de creación de un SMA, al crear de forma automatizada la estructura principal del proyecto del SMA en cuestión, de tal forma que el desarrollador del SMA pueda centrarse en actividades más específicas y complejas como la implementación de algoritmos de aprendizaje, algoritmos de razonamiento o algoritmos para la resolución de conflictos, entre otros, siendo así como se establece la relación entre VMAS y las herramientas CASE.

III. Trabajos Relacionados

Es indiscutible que no existe una herramienta que reúna las características principales que presenta V-MAS (generación de código para JADE, uso de MASINA como metodología de modelado, exportación de modelos en forma tabular, usabilidad, etc.), se han desarrollado otras herramientas de software similares, que partiendo de algunos datos ingresados por el usuario son capaces de generar código inicial hacia otras plataformas de destino. Éstas herramientas sirvieron como base o referencia para la presente investigación, de tal forma que para lograr que V-MAS sea una aplicación estable, consistente con la metodología y usable se tomaron en cuenta los aspectos mejorables, fallas y debilidades que estos trabajos presentan, entre ellas se tiene: no son multiplataforma, escasa validación de la información ocasionando inconsistencias y errores en el código generado, usabilidad de usuario limitada, lo que indica que son difíciles de usar y de instalar, no generan código para plataformas de gestión de agentes, lo que le agrega mayor carga de trabajo al usuario que debe ocuparse de todos los mecanismos necesarios para que los agentes puedan co-existir, como registro, búsqueda, comunicación, coordinación, entre otro de los agentes; no toman en cuenta a los usuarios inexpertos, lo que hace que la curva de aprendizaje en la creación de SMAs sea más pronunciada, entre otros. Dentro de los trabajos previos se tiene:

EDISMA [25] (Entorno de Desarrollo Integrado para la creación de Sistemas MultiAgentes) desarrollada por Rivero en la Universidad de Los Andes (ULA), ésta herramienta presenta una interfaz gráfica de usuario (IGU) que permite captar la información de los modelos de MASINA, para posteriormente generar el código en lenguaje C++, es decir, que crea archivos que contienen los métodos e instanciación de los agentes en lenguaje C++, así como el archivo makefile para compilar el SMA en cuestión. Por otra parte, Mészáros, de la Universidad Comenius de Bratislava, Eslovaquia en su tesis de maestría titulada code generation from AML (Archetype Modeling Language) to jadex [27], desarrolló un generador de código que parte de los modelos AML para generar el código de los agentes en JADE, dicha traducción de los modelos a código se realizó haciendo uso de las fases que propone la metodología MDE (Model Driven Engineering). Seguidamente, en el año 2009 en La Universidad de Los Andes (ULA) Aguilar y Bravo desarrollaron un proyecto que denominaron SISGECOMA [23] (Sistema Generador de Código para MASINA) cuyo objetivo principal fue diseñar e implementar un sistema traductor de los modelos de MASINA a C++, para lo cual se desarrollaron 3 agentes, uno que se encarga de desplegar la IGU y recolectar la información, otro se encarga de la validación y verificación de la información obtenida y finalmente un agente que realiza la generación de código [28].

V-Mas por su parte guía al usuario paso a paso en el uso de la metodología, permitiendo ingresar y validar los datos paso a paso y de manera sencilla, logrando que el usuario pueda crear un SMA sin necesidad de ser un experto en la metodología, pero asegurando que el desarrollo del mismo sea adecuado al ciclo metodológico de la metodología MASINA, lo que a su vez certifica que se cumplan los objetivos de diseño. De igual manera, V-MAS asegura que el código generado esté libre de

errores de sintaxis, lo que deriva en un ahorro de tiempo en la fase de codificación, a la vez que el usuario no debe preocuparse por implementar algoritmos para la co-existencia del SMA, ya que la plataforma JADE será la responsable de ello y de asegurar que el SMA cumpla los estándares de la FIPA.

IV. Visual MAS Modeller (V-MAS)

V-MAS Modeller es una aplicación que busca facilitar el trabajo de los desarrolladores de SMA basándose en los conceptos, modelos y pasos propuestos en MASINA. Así V-MAS apoya el modelado y desarrollo del SMA, además de facilitar la utilización de la metodología MASINA. De igual manera, V-MAS, es una aplicación de alta usabilidad, al proveer los formularios para la introducción de la información de los modelos de manera secuencial y simple. Por otra parte, V-MAS permite guardar el proyecto en formato XML, lo que aumenta la integración con otras aplicaciones que pueden tomar el modelo como entrada para realizar operaciones sobre el mismo, como, por ejemplo, generar el código para otras plataformas de agentes, generar gráficos que muestren la interacción entre los agentes, etc. V-MAS se presenta como un medio de apoyo para aquellos desarrolladores de SMA que conozcan o no la metodología MASINA, ya que V-MAS hace más fácil y eficiente la implementación e ingreso de la información requerida por los modelos de la metodología, lo que significa un ahorro de tiempo, esfuerzo y costos en la programación. Por otra parte, la integración con diversas aplicaciones de diagramado UML hace que los diseñadores no tengan que abandonar sus IDE's de modelado UML tradicionales, sino que, por el contrario, vienen a ser un soporte sobre las tareas y funciones que se desarrollarán en el sistema. En consecuencia, el propósito de V-MAS Modeller, es apoyar al usuario en el desarrollo de los SMA, basándose en el ciclo de vida la metodología MASINA y de los sistemas multiagentes, a la vez de contar con funcionalidades básicas de un modelador como son: guardar, exportar y generar código, compartir el proyecto con otros usuarios, e importar los diagramas UML desarrollados en modeladores como StarUML y ArgoUML, según se describe en la Figura 1, además V-MAS facilita la exportación de la descripción de los modelos a distintos procesadores de texto como Microsoft Word a través del portapapeles del Sistema operativo (copiar y pegar) esto es una ventaja para los desarrolladores a la hora de crear documentos como informes, artículos para publicaciones, etc. V-MAS Modeller surge como una solución a la inexistencia de modeladores guiados por MASINA y la generación de código para JADE.

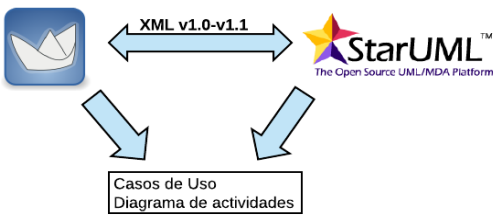

Figura 1: Importación de Archivos XML desde los Modeladores UML

El desarrollo de la aplicación se guió a través de 2 metodologías: MDA (Model Driven Architecture) y FDD (Feature Driven Development). Por un lado, MDA establece la forma en que se traducirá la información introducida por el usuario en los modelos de MASINA a código Java, es decir, instaura la relación entre la información introducida en los modelos y el código que debe ser generado por VMAS. Seguidamente, FDD es una metodología ágil que guía el desarrollo de herramientas de software y gestiona cada una de las funcionalidades a implementarse, ella cubre las etapas clásicas de desarrollo del software, como son: definición de requisitos, diseño y documentación, desarrollo, pruebas y despliegue haciendo uso de iteraciones cortas [29]. Específicamente, las etapas de desarrollo propuestas por FDD [30] son:

- Desarrollo de un modelo general.
- Construcción de una lista de características.
- Plan por característica.
- Diseño por característica.
- Construcción por característica.

Por tanto, con el fin de lograr un software robusto, al momento de listar las características o funcionalidades de V-MAS, se clasificaron en 6 diferentes áreas o módulos, ellos fueron: creación de agentes, creación de modelos, configuración de parámetros de la aplicación, generación de código, generación de documentación (copiar modelos, así como la relación tarea-servicios mediante el portapapeles) y administración del proyecto. Así pues, FDD se trata de una metodología que permitirá el acoplamiento con MDA perfectamente sin descuidar los aspectos relacionados a la calidad del software. Inicialmente el desarrollo de V-MAS es guiado por FDD, y en el momento de desarrollar la característica de generación de código se conmuta a MDA, de esta manera se integran ambas metodologías.

Una de las funcionalidades de V-MAS, es la de proveer las interfaces, formularios y mecanismos adecuados para la correcta y eficiente captura de la información. De esta manera, el ciclo de vida de un proyecto de SMA en V-MAS comienza con el registro de los agentes necesarios, posteriormente para cada uno de ellos se debe ingresar la información correspondiente a los modelos de MASINA tales como: modelo de agentes, modelo de tareas, modelo de comunicación, modelo de coordinación, modelo de inteligencia individual, modelo de inteligencia colectivo; para finalmente proceder con la generación del código. Es importante resaltar que mediante V-MAS no es posible saltarse pasos en la metodología, lo que asegura la consistencia y completitud de la información, por ejemplo, no es posible cargar el modelo de tareas si aún no se ha cargado el modelo de agentes. Esto no se ve como una falla sino como una fortaleza ya que asegura un buen uso de la metodología y la calidad del código generado.

De esta manera, una vez se cuenta con la información provista por el usuario V-MAS procederá a generar la mayor cantidad de elementos de código que sea posible, es decir, clases, métodos, atributos de clases, clases auxiliares y librerías. Para cumplir este objetivo es necesario que V-MAS interprete

correctamente los modelos de MASINA. En consecuencia, La fase de generación de código fue realizada utilizando la metodología MDA, ella propone primeramente realizar un modelo independiente de la plataforma (PIM) seguidamente transformarlo a un modelo específico de la plataforma (PSM) y por último proceder a la generación del código [23]. En la presente aplicación, el modelo PIM está representado por los modelos de MASINA ingresados por el usuario en V-MAS (V-MAS permite la especificación de cada uno de los modelos de MASINA, incluso los de inteligencia colectiva e individual). De, todos estos diagramas, se tomarán en cuenta para la generación del código el modelo de agentes y modelo de tareas ya que estos proveen la información necesaria para generar la estructura inicial de las clases que determinarán los agentes (la estructura de los agentes se puede generar solo usando estos dos diagramas, el funcionamiento en sí del agente debe ser codificado por el usuario y queda para futuras investigaciones), por su parte lo referido a los procesos de inteligencia y aprendizaje deben ser desarrollados manualmente por el usuario debido a que son procesos muy complejos y queda a disposición personal de dicho usuario definir cómo implementarlos. Es así, como el modelo de comunicaciones, coordinación, inteligencia individual y colectiva se utilizarán para generar documentación y comentarios en los archivos generados para el proyecto. En efecto, V-MAS genera una clase de Java para cada agente en la que se integran los objetos de tipo Servicio que hayan sido definidos por el modelador, así como las tareas correspondientes a cada servicio. A su vez, debido a que la plataforma destino se basa en el lenguaje Java, se toman en cuenta factores como sintaxis del lenguaje, específicamente la estructura de clases aceptadas en la plataforma JADE, tipo de datos soportados, herencia, modificadores de acceso, así como métodos heredados de clases abstractas. A su vez, el PSM está representado por cada una de las clases generadas por VMAS, para cada agente registrado se generará un archivo con extensión *.java* que contendrá la clase principal del agente, las clases relacionadas a cada uno de sus servicios, así como los métodos y/o clases que representan las tareas que lleva a cabo cada servicio del agente. De igual forma, el archivo de agente generado contendrá clases auxiliares que servirán como artificio para almacenar los datos de retorno de cada uno de los servicios y tareas registrados.

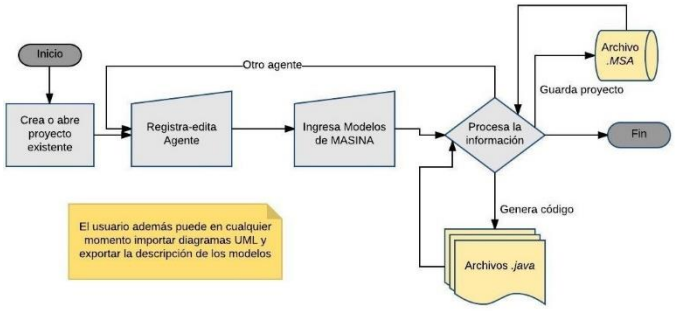

Figura 2: Diagrama de Flujo de las Actividades Desarrolladas en la Aplicación

En la Figura 2 se describen las actividades que los usuarios pueden desarrollar en V-MAS a través de un diagrama de flujo (en ella se especifica el ciclo de vida de un proyecto). El usuario comienza creando un proyecto, crea los agentes del SMA especifica la información de los modelos para cada uno de ellos, posteriormente puede generar el código Java y/o guardar el proyecto en un archivo con extensión MSA.

La clase principal del agente generado por V-MAS se define según se muestra en la Figura 3. En ella se declaran y se registran cada uno de los servicios del agente en el Directorio Facilitador (DF) de JADE, de acuerdo a lo que el usuario especifique en el modelo de agentes ingresado en V-MAS; además, como parte de las especificaciones de JADE, la clase Agente debe heredar de la clase *Agent* y contener el método *setup* que se ejecutará al agregar el agente al contenedor (este método registra los servicios del agente en el DF de JADE) y el método *takeDown* que se ejecuta cuando el agente se libera, a fin de realizar tareas de limpieza, como es el caso de eliminar los servicios del agente que fueron registrados en el DF.

Vale destacar, que cada servicio del agente representa un Comportamiento (Behaviour) de JADE, para el cual se genera una clase interna. Asimismo, las tareas registradas en el modelo de tareas, representaran una nueva clase interna que hereda de un comportamiento o un método de la clase, de acuerdo a como lo haya definido el usuario a la hora de ingresar la información.

```
public class Mi_Agente extends Agent{
    protected void setup(){
        DFAgentDescription dfd =  new DFAgentDescription();
        dfd.setName(getAID());

        ServiceDescription miServicio = new ServiceDescription();
        miServicio.setType("ServiceMiServicio");
        miServicio.setName(getLocalName());
        addBehaviour( new MiServicio(myAgent));
        try {
            DFService.register(this, dfd);
        }catch (Exception e){}
    }

    protected void takeDown(){
        try { DFService.deregister(this); }
        catch (Exception e){}
    }
}
```

Figura 3: Código Generado para la Clase Agente

```
public class Mi_Servicio extends Behaviour{
    private int calidad = 0;
    private MiServicioReturnValue valorRetorno;

    public void action(){...3 lines }

    public int getCalidad(){...3 lines }

    public void setCalidad(int calidad){...3 lines }

    public void miTarea1 (String ingrediente1, String ingrediente2){.

    public class miTarea2 extends Behaviour {...9 lines }

    public boolean done() {...3 lines }

    public class MiServicioReturnValue {...14 lines }
}
```

Figura 4: Código Generado para los Servicios

Por otro lado, la estructura básica de las clases que representan los servicios se define en la Figura 4, es una clase con muchos elementos que contiene los métodos relacionados a las tareas asociadas, un método *action*, atributos del servicio. Esta clase

debe heredar del comportamiento JADE adecuado, según lo que haya indicado el usuario al momento de especificar el servicio.

Seguidamente, el usuario puede registrar por cada agente uno o más servicios. De cada servicio se crea una clase cuyo nombre se corresponde con el nombre del servicio, y cuyos métodos y propiedades dependerán de las propiedades registradas y del tipo de comportamiento definido por el usuario. Además, los parámetros de salida se plantean como una serie de variables que se agrupan en un objeto de una clase auxiliar denominado *ValorRetorno* (ver Figura 5), el nombre de éstas clases se genera a través de la siguiente fórmula: *<nombre del servicio> + < "ReturnValue">*.

Finalmente, se deben agregar los métodos o clases correspondientes a cada una de las tareas asociadas al servicio, ésta información se obtiene del modelo de tareas del agente en el cual se especifica nombre de la tarea, ingredientes (parámetros), el servicio al que pertenece, y opcionalmente, el tipo de comportamiento que ésta posee. En caso que la tarea se defina como un comportamiento, se creará una clase de Java que hereda del comportamiento indicado, de otra manera la tarea pasará a ser un método de la clase del servicio al cual está ligada. En la Figura 5 se define la estructura básica de las clases auxiliares, ésta información se extrae de los parámetros de salida especificados para cada servicio, es decir, para la clase a generar se definirá una serie de atributos consistentes con los parámetros de salida del servicio y según el tipo de dato ingresado, además de establecerle los modificadores de acceso, y los respectivos métodos *getters* y *setters*.

```
public class MiServicioReturnValue {
    char miParametro1;

    public MiServicioReturnValue() {
    }

    public char getMiParametro1() {
        return miParametro1;
    }

    public void setMiParametro1(char miParametro1) {
        this.miParametro1 = miParametro1;
    }
}
```

Figura 5: Código Generado para las Clases Auxiliares

Esencialmente, la evolución del archivo de salida para cada uno de los agentes va cambiando según la información que se introdujo en los modelos, así como de la configuración de generación de código establecida por el usuario. En la Figura 6 se describe el aporte que realiza cada modelo sobre el archivo generado.

Finalmente, pero no menos importante, la aplicación ofrece una serie de funcionalidades que ayudarán al usuario en el alcance de ciertas actividades que son de uso común en las herramientas CASE, entre ellas la exportación de los modelos de MASINA de forma tabular para cualquier procesador de texto, de tal forma que se apoye al desarrollador en la creación de documentos académicos, documentación del proyecto, artículos científicos, etc., esto se logra haciendo uso del portapapeles del sistema operativo. A su vez, cuenta con un módulo de administración de proyecto en el cual se incluyen varias funcionalidades, ellas son: guardar proyecto, guardar como y abrir proyecto, un módulo de configuración de parámetros y la posibilidad de importar la información UML correspondiente a cada uno de los agentes modelados.

Las tecnologías involucradas en el desarrollo de Visual MAS Modeller son diversas, principalmente el desarrollo se rigió bajo la plataforma Java versión 8 e interactúa con archivos de entrada/salida XML y UML. Por otro lado, SQLite se utiliza como base de datos de la aplicación y patrones de diseño para las tareas y procesos que la herramienta deberá ejecutar. Se seleccionó JADE como plataforma objetivo, es decir, la plataforma para la cual sería generado el código debido a que es ampliamente utilizada y existe abundante documentación en la web. Por su parte, JADE [4] define la plataforma como: es un software totalmente implementado en lenguaje Java. Simplifica la implementación de sistemas multi-agentes a través de un middleware que cumpla con las especificaciones FIPA y a través de un conjunto de herramientas que soportan las fases de depuración y despliegue.

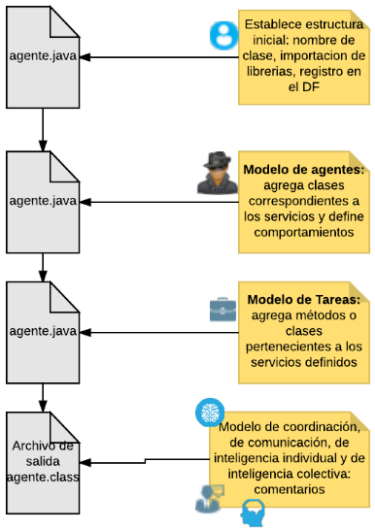

Figura 6: Aporte de cada Modelo sobre el Código Generado

V. Caso de Estudio

Con el fin de exponer el uso de la aplicación en un ambiente real, se propone un caso de estudio basado en el juego de TicTacToe, éste consiste en 2 tipos de agentes, un agente que representa al jugador y otro para representar el tablero del juego. Éstos agentes interactúan entre sí para informar la jugada seleccionada por parte de los jugadores; mientras que el agente tablero informará acerca del estado actual de la partida. A fin de abreviar la información, se especificará solo los modelos de MASINA que son cruciales para la generación del código, siendo ellos las secciones objetivos y servicios, modelo de agentes y el modelo de tareas.

A. *Agente Jugador*

Agente que representa una entidad de tipo jugador, se encarga de evaluar las jugadas e informar acerca del próximo movimiento. La descripción del objetivo del modelo de

agentes para esta entidad se cargó en la aplicación, según se observa en la Figura 7.

Por otro lado, se definió un servicio encargado principalmente de solicitar jugadas al tablero, según se describe en la Figura 8.

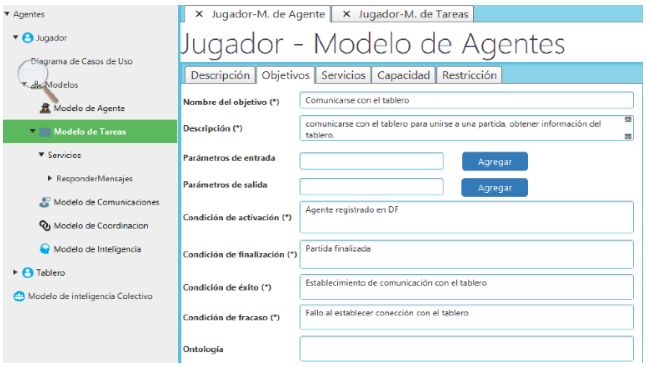

Figura 7: Definición del Objetivo Comunicarse con el Tablero

Además, a este servicio se le definió una tarea denominada "*Preparar Respuesta"*, como un método encargado de gestionar los mensajes de entrada y salida cuya descripción se detalla en la Figura 9. De igual forma el servicio cuenta con otra tarea que se representa como un comportamiento que extiende de *Behaviour*. Esta tarea se denomina "*Enviar Movimiento*" y representa el proceso responsable de llevar la secuencia de la partida y enviar el siguiente movimiento del jugador definido en la Figura 10.

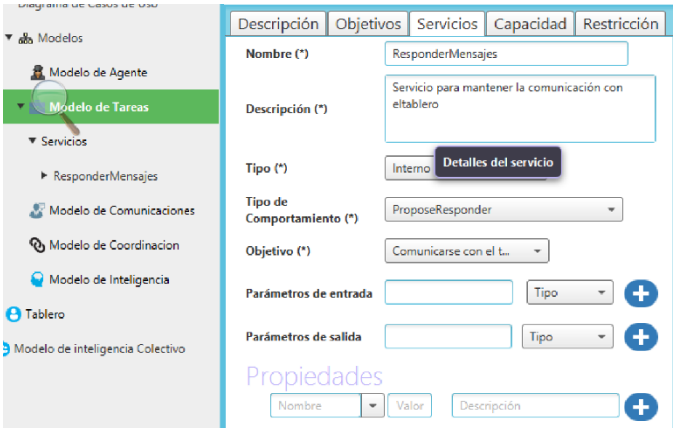

Figura 8: Definición del Servicio Solicitar Jugada

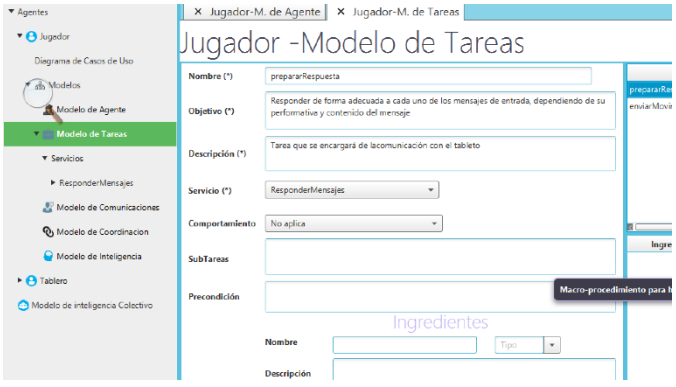

Figura 9: Definición de la Tarea Preparar Respuesta

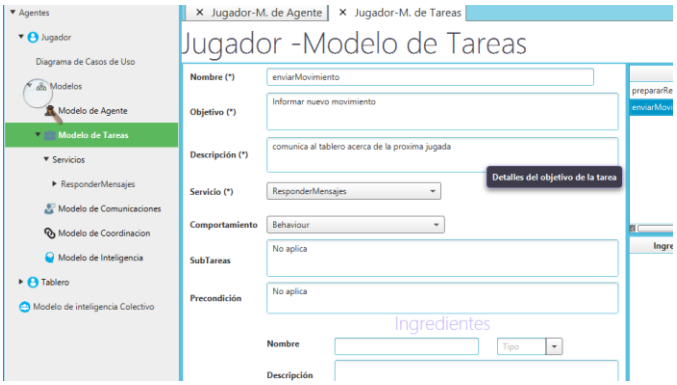

Figura 10: Definición de la Tarea Enviar Movimiento

B. Agente Tablero

Este agente crea y gestiona cada una de las partidas que se dan en la aplicación y coordina los mensajes que envían cada uno de los jugadores para informar acerca de su próximo movimiento en el tablero. Su objetivo es encargarse de gestionar las partidas, es decir, iniciarlas, coordinar movimiento de los jugadores, decidir ganador y mostrar resultados. La Figura 11 muestra la información cargada para este agente en V-MAS.

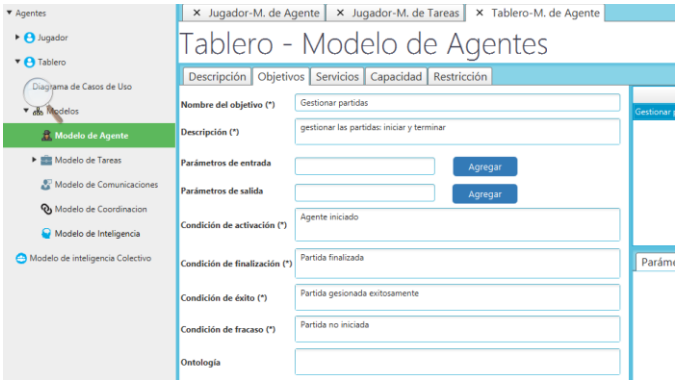

Figura 11: Definición del Objetivo Gestionar Partidas

Para este agente, se registraron 3 servicios en V-MAS Modeller, ellos son: *ComenzarPartida* cuya descripción se muestra en la Figura 12, *CrearPartida* definido según la Figura 13 y finalmente *Mover* que se describe en Figura 14.

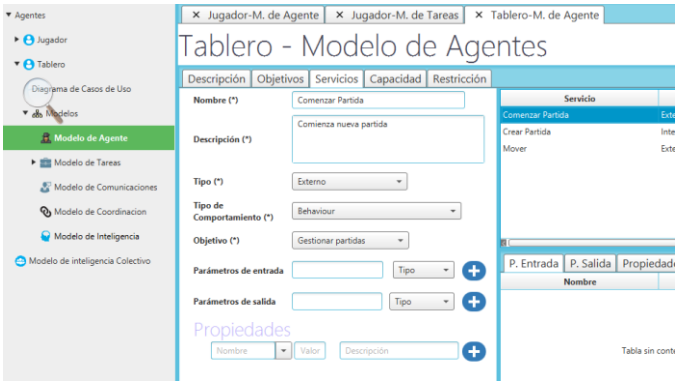

Figura 12: Definición del Servicio Comenzar Partida

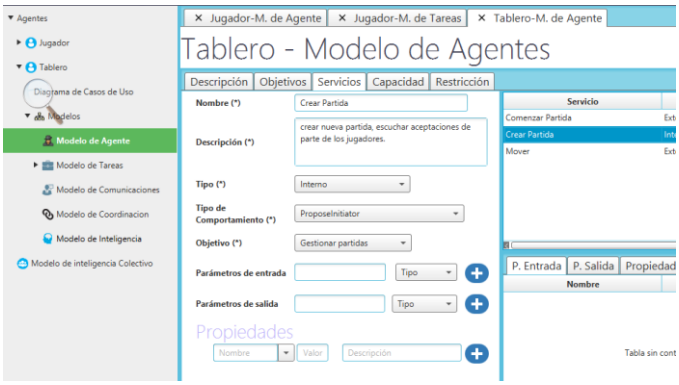

Figura 13: Definición del Servicio Crear Partida

Seguidamente, se definió una serie de tareas asociadas a los servicios, a continuación, se detalla con más profundidad cada una de ellas:

Aceptar Propuesta: esta tarea hace parte del servicio *Crear Partida* y se encargará de realizar las acciones correspondientes para iniciar una partida cuando un jugador acepte unirse a ella. La descripción de ésta tarea se define a continuación en la Figura 15.

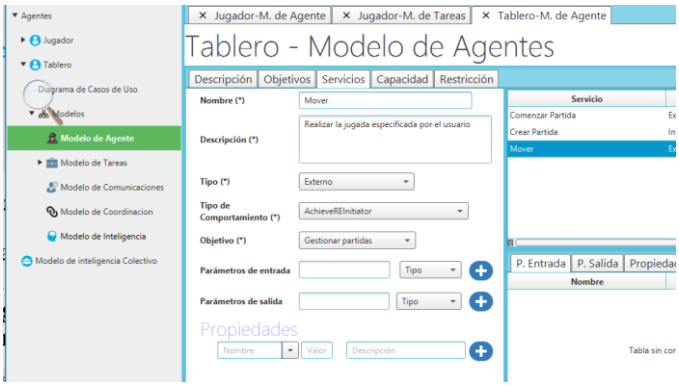

Figura 14: Definición del Servicio Mover

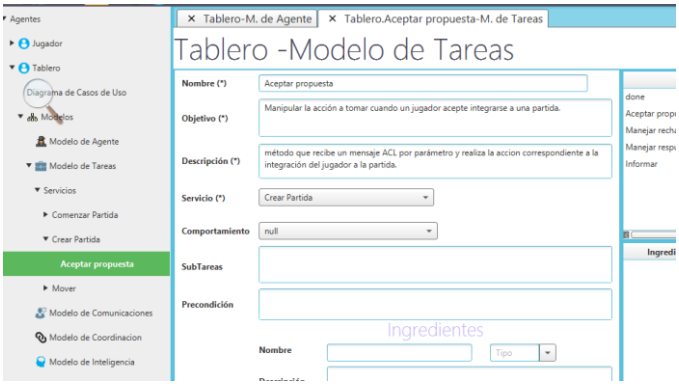

Figura 15: Definición de la Tarea Aceptar Propuesta

Manejar rechazo: Esta tarea hace parte del servicio *Mover* y se encarga principalmente de realizar las acciones necesarias, cuando un jugador abandona la partida, la descripción de ésta tarea se muestra en la Figura 16.

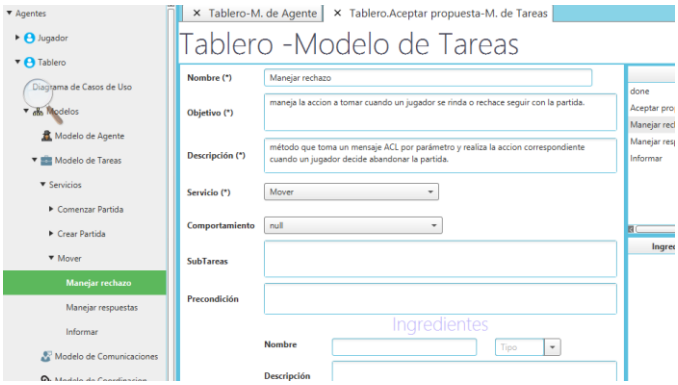

Figura 16: Definición de la Tarea Manejar Rechazo

Manejar respuestas: Esta tarea hace parte del servicio *Mover*, su función primordial es escuchar mensajes de parte de los jugadores y tomar las acciones requeridas según el contenido de los mismos para efectuar los movimientos en el tablero. La Figura 17 describe con detalle la definición de esta tarea, siguiendo el modelo MASINA.

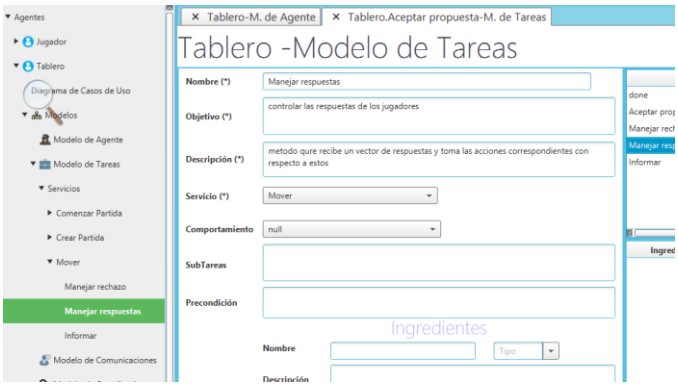

Figura 17: Definición de la Tarea Manejar Respuestas

Informar: Esta tarea hace parte del servicio *Mover*, principalmente se encarga de difundir el estado actual de la partida: posiciones, condición del tablero y ganador. En la Figura 18 se especifica la descripción de ésta tarea siguiendo el modelo MASINA.

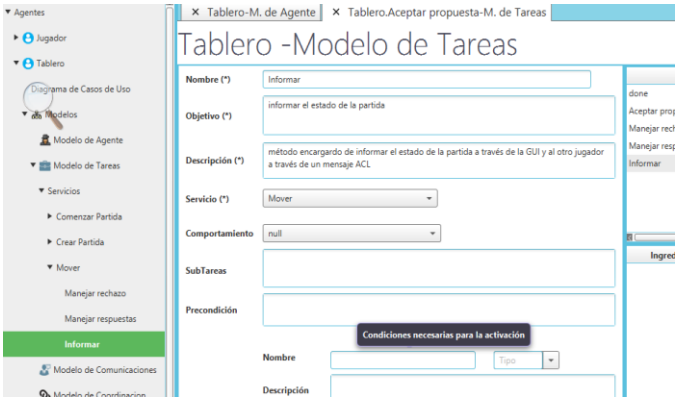

Figura 18: Definición de la Tarea Informar

Preparar Respuesta: tarea que hace parte del servicio *Responder Mensajes* y su función principal es responder a cada uno de los mensajes de entrada de manera correcta. En la Tabla

I se detalla las propiedades de esta tarea, siguiendo la plantilla propuesta por MASINA para el modelo de tareas, ésta tabla se exportó directamente desde la aplicación haciendo uso de la funcionalidad *copiar modelo*.

Tabla I: Descripción de la Tarea Preparar Respuesta

Modelo de Tareas	
Nombre	Preparar Respuesta
Objetivo	Responder de forma adecuada a cada uno de los mensajes de entrada, dependiendo de su performativa y contenido del mensaje
Descripción	Tarea que se encargará de la comunicación con el tablero
Servicios asociados	ResponderMensajes
Subtareas	No aplica
Ingredientes	No aplica

Enviar Movimiento: ésta tarea hace parte del servicio *Responder Mensajes* y su función principal es reportar al tablero acerca de la jugada que el participante desea realizar. En la Tabla II se detalla esta tarea siguiendo la plantilla propuesta por MASINA para el modelo de tareas, ésta tabla se exportó directamente desde la aplicación haciendo uso de la funcionalidad *copiar modelo*.

Tabla II: Descripción de la Tarea Enviar Movimiento

Modelo de Tareas	
Nombre	Enviar Movimiento
Objetivo	Informar nuevo movimiento
Descripción	comunica al tablero acerca de la próxima jugada
Servicios asociados	Responder Mensajes
Subtareas	No aplica
Ingredientes	No aplica

Seguidamente, se procedió a generar el código base para cada uno de los agentes registrados (la Figura 19 muestra como el usuario puede generar el código utilizando V-MAS), para los cuales V-MAS Modeller creó un archivo con extensión *.java* para cada agente, especificando elementos que constituyen la estructura de la clase del agente, constructores y clases asociadas a los servicios, entre otros. La configuración seleccionada (ver Figura 19) para la generación de código constó en no agregar comentarios y suprimir los espacios en blanco para aquellos elementos que definen un nombre de clase, atributo o método.

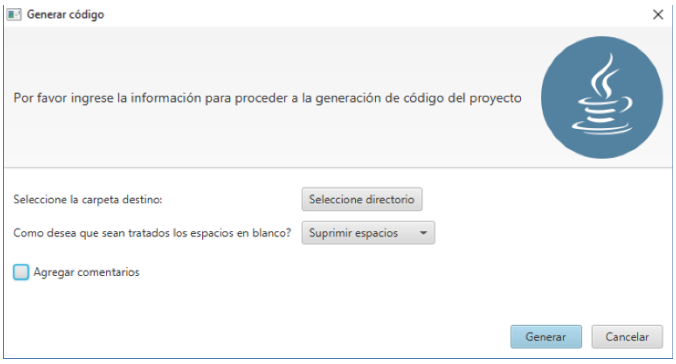

Figura 19: Interfaz para Configurar la Generación del Código

La Figura 20 muestra parte del código generado por V-MAS para el agente Jugador, mientras que en la Figura 21 se muestra un segmento de código correspondiente al agente Tablero, donde se detallan los métodos setup y takeDown los cuales han sido generados completamente por V-MAS, y no requieren de mucha configuración adicional por parte del usuario.

Una vez obtenidas estas clases desde V-MAS se procedió a desarrollar el código para la implementación de la lógica de cada uno de los agentes, seguidamente, se compiló el código en Java y se ejecutó la aplicación en la plataforma JADE. Posteriormente se desplegaron 2 agentes de tipo jugador denominados jugador1 y jugador2, y uno de tipo Tablero denominado tablero.

En el mismo sentido, en la Figura 22 se muestra una imagen de los agentes que se encuentran desplegados en la plataforma JADE.

```
public class Jugador extends Agent {

    protected void setup() {

        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());

        ServiceDescription respondermensajes = new ServiceDescription();
        respondermensajes.setType("respondermensajes");
        respondermensajes.setName(getLocalName());
        addBehaviour(new Respondermensajes(this, mt));
        try {
            DFService.register(this, dfd);
        } catch (Exception e) {
        }

    }

    //*Fin de metodo*//

    protected void takeDown() {
        try {
            DFService.deregister(this);
        } catch (Exception e) {
        }
    }
```

Figura 20: Sección del Código Generado Automáticamente para el Agente Jugador

```
public class Tablero extends Agent {

    protected void setup() {

        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());

        ServiceDescription comenzarPartida = new ServiceDescription();
        comenzarPartida.setType("comenzarPartida");
        comenzarPartida.setName(getLocalName());
        addBehaviour(new ComenzarPartida(this));

        ServiceDescription crearPartida = new ServiceDescription();
        crearPartida.setType("crearPartida");
        crearPartida.setName(getLocalName());
        addBehaviour(new CrearPartida(this, msg));

        ServiceDescription mover = new ServiceDescription();
        mover.setType("mover");
        mover.setName(getLocalName());
        addBehaviour(new Mover(this, msg));
        try {
            DFService.register(this, dfd);
        } catch (Exception e) {
        }

    }
```

Figura 21: Sección del Código Generado Automáticamente para el Agente Tablero

Por su parte, en la Figura 23 se muestra la interacción que se da entre los agentes Jugador y Tablero haciendo uso de la herramienta *sniffer* que provee la plataforma JADE. Allí se observa como fluyen los mensajes entre los agentes del sistema mientras se desarrolla el juego.

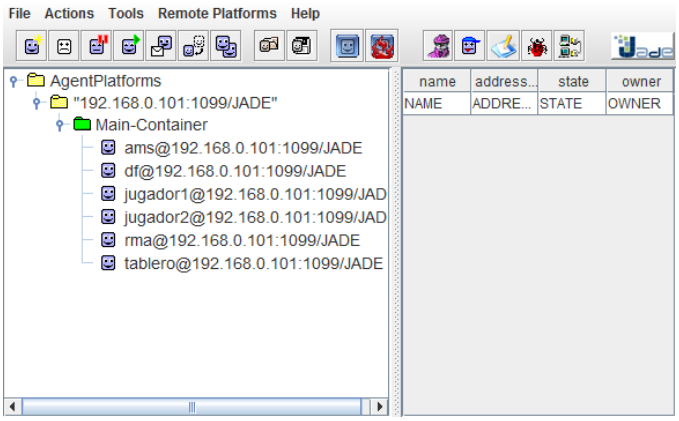

Figura 22: Contenedor de JADE con los Agentes en Ejecución

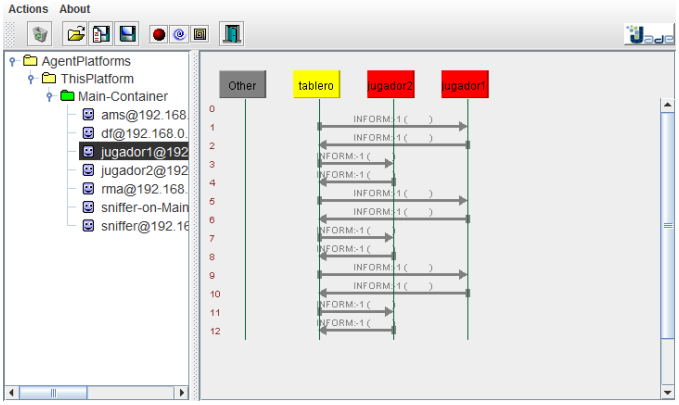

Figura 23: Visualización de la Interacción entre los Agentes a través del Sniffer de JADE

VI. Experimento

Para realizar una validación más científica de la herramienta, se tomó una sección de Inteligencia Artificial de la Universidad Nacional Experimental del Táchira de 57 estudiantes y se dividió en clases de la siguiente forma:

- No conocen MASINA.
- Nivel medio en el uso de MASINA.

Cada clase se dividió en grupos de tres estudiantes y se les instruyó para que crearan un SMA simple de solo dos agentes (el clásico ejemplo productor-consumidor o vendedor-comprador). Luego se crearon sub clases de la siguiente manera:

- No conocen MASINA.
  - 5 grupos deben hacer todo manualmente.
  - 5 grupos deben usar V-MAS
- Nivel medio en el uso de MASINA.
  - 4 grupos deben hacer todo manualmente.
  - 4 grupos deben usar V-MAS.

A cada grupo se les tomo el tiempo en horas que duraron para desarrollar el SMA (sólo debían crear el modelo de agentes, el modelo de tareas y codificar), a fin de poder comparar si disminuía usando MASINA o no. Todos los grupos tenían conocimiento medios de JADE y se les indicó en forma general las tareas y servicios con la que debía contar cada agente, de tal forma que esto no influyera en el tiempo de desarrollo.

Las métricas a tomar en cuenta son el tiempo de desarrollo, él número de errores y la necesidad de un experto para poder completar los modelos.

De los grupos que no conocían MASINA, y debían hacer todo manualmente, ninguno pudo crear los modelos sin ayuda del profesor, mientras que de los que usaron V-MAS sólo uno necesitó ayuda del profesor para ingresar los modelos. Aun así, los que recibieron ayuda del profesor y trabajaron manualmente requirieron mayor tiempo para finalizar la fase de diseño y sus modelos contaban con errores, en algunos parámetros.

Por su parte de los grupos que conocían MASINA a nivel medio, ninguno requirió ayuda del profesor, pero los que lo hicieron manualmente terminaron la fase de diseño con unos pocos minutos de retraso. De igual forma sus modelos contenían algunos errores, en los parámetros.

El profesor instruyó a los estudiantes para que corrigieran los errores, con lo cual perdieron aún más tiempo.

Seguidamente se pasó a la fase de desarrollo; los estudiantes que trabajaron con V-MAS tuvieron el código inicial del proyecto libre de errores en apenas unos segundos, mientras que los que hicieron el sistema manualmente, tardaron entre 20 y 30 minutos en completar el código inicial del proyecto; en ese tiempo, los que usaron V-MAS ya habían terminado de codificar y probar todas las funcionalidades del sistema.

VII. Conclusiones y Trabajo Futuro

La presente investigación concluyó en el desarrollo de una aplicación CASE que permite a los diseñadores de sistemas multi-agentes modelar aplicaciones siguiendo la arquitectura o

enfoque de SMA, a su vez ofrece a los usuarios la posibilidad de exportar los modelos a través de las plantillas propuestas por MASINA y generar el código base en JADE para cada uno de los agentes registrados en el proyecto. Esta aplicación permite a los desarrolladores y/o diseñadores crear, guardar, editar y visualizar un proyecto, además permite crear Sistemas multi-agentes siguiendo una metodología apropiada para tal fin, así como facilitar al usuario la creación de los modelos de la metodología en sí. Por otro lado, la aplicación desarrollada ayuda a ahorrar tiempo de desarrollo ya que se automatizan ciertas actividades que normalmente se desarrollan en forma manual.

Visual MAS Modeller se diferencia principalmente de las herramientas previamente desarrolladas (SISGECOMA y EDISMA) en que ellas generan código para plataformas poco conocidas y usadas, su código no se rige bajo el estándar FIPA, mientras que V-MAS Modeller genera el código base para una plataforma de agentes ampliamente usada y conocida como lo es JADE.

Por otro lado, V-MAS realiza a lo largo del proceso de introducción de datos un conjunto de validaciones que permiten al usuario la obtención de un código válido y libre de errores de sintaxis, además, que éste cumple con los estándares de definición de clases, atributos y métodos establecidos en Java. A su vez, la aplicación fue desarrollada pensando en el usuario, para facilitar el proceso de introducción de información, proveyendo una interfaz intuitiva, clara y efectiva para hacer este proceso bastante sencillo. Aunado a esto, V-MAS Modeller apoya al diseñador del SMA en la generación de documentación o informes ofreciéndole la posibilidad de copiar los modelos de forma tabular que coinciden con las plantillas establecidas en MASINA. A continuación, en la Tabla III se establece una comparación entre las herramientas que han sido desarrolladas previamente y Visual MAS Modeller.

De la Tabla III, se observa que herramientas similaresa V-MAS, no generan código para plataformas de gestión de agentes, como es el caso de C/C++, lo que le impone al desarrollador del sistema el trabajo de tener que codificar todos los elementos que le permitan al sistema multi agentes co-existir (registrar, descubrir, comunicar, interactuar, etre otros), lo cual genera aún más trabajo para el usuario. De la misma forma, la información tomada por otras herramientas no cuenta con validación de datos, lo que permite que el usuario pueda saltarse pasos importantes en la metodología, o ingresar paramétros que no se corresponde con los valores requeridos por la metodología, lo cual inducirá a errores de sintaxis en el código generado. De igual forma, la usabilidad de V-MAS es superior a la de las demás herramientas, ya que cuenta con datos parametrizados que son mostrados en listas y otros componentes gráficos y a su vez muestra ayudas que le indican al usuario que iformación ingresar en cada campo, algo de lo que carecen las demás herramientas.

Por otra parte, aunque V-MAS solo genera código para la plataforma JADE, su arquitectura se ha desarrollado de manera flexible usando tecnicas de desarrollo de software orientado a objetos y patrones de diseño que permitiran incorporar en el futuro nuevas plataformas de agentes como código objetivo de V-MAS.

Tabla III: Comparación de las Herramientas

Aplicación	**Características**
V-MAS Modeller	**Salida:** archivos de clase .java para cada agente **Metodología:** MASINA **Plataforma Destino:** JADE **Validación de datos**: Sí **Exportación de modelos:** Sí **Importación de información UML:** StarUML y ArgoUML **Usabilidad:** Alta
EDISMA	**Salida:** Archivos de clases en C++. **Metodología:** MASINA **Plataforma Destino:** C++ **Validación de información:** Deficiente **Exportación de modelos:** No soportado **Importación de información UML:** Umbrello **Usabilidad:** Media
SISGECOMA	**Salida:** Archivos compilados en C **Metodología:** MASINA **Plataforma Destino:** C **Validación de información:** Deficiente **Exportación de modelos:** No soportado **Importación de información UML:** No soportado **Usabilidad:** Media

El experimento realizado con los estudiantes logró demostrar que V-MAS no solo ayuda a reducir el tiempo de desarrollo sino que facilita el entendimiento a la hora de diseñar el sistema multi-agentes. Lo que permitió validar que V-MAS cumple con los objetivos de diseño.

Finalmente, con el fin de lograr que V-MAS sea un software que responda correctamente ante las diversas situaciones que se pueden presentar en el modelado de SMA se presentó un caso de estudio donde se diseñaron y se ejecutaron una serie de pruebas que permiten verificar el funcionamiento del software,tomando en consideración conceptos de calidad de software, como lo son valores al límite, criterio de clases válidas y escenarios.

El trabajo futuro, se enfoca principalmente en generar código para otras plataformas de agentes conocidas, así como la utilización de otros modelos de MASINA para ampliar aún más el código generado.

REFERENCIAS

[1] M. Maalal and A. Addou. *A New Approach of Designing Multi-Agent Systems*. International Journal of Advanced Computer Science and Applications, Vol. 2, No. 11. Casablanca, Marruecos, 2011.

[2] J. Aguilar, I. Bessembel, M. Cerrada, F. Hidrobo y F. Narciso. *Una Metodología para el Modelado de Sistemas de Ingeniería Orientado a Agentes*. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, Vol. 12, No. 38, pp. 39-60 Asociación Española para la Inteligencia Artificial. España, 2008.

[3] R. Mall. *Fundamentals of Software Engineering*. Prentice Hall of India, New Delhi, 2003.

[4] JADE. http://jade.tilab.com.

[5] G. Weiss. *Multiagents Systems 1st ed*. Cambridge MA. MIT Press, 2013.

[6] C. Iglesias, M. Garijo, J. González, and J. Velasco. *Analysis and Design of Multiagent Systems using MAS-CommonKADS*. Intelligent Agents IV, 1365, pp. 313-327, 1998.

[7] B. Burmeister. *Models and Methodology for Agent-Oriented Analysis and Design*. In Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems, Eindhoven, The Netherlands, 1996.

[8] S. A. DeLoach. *Modeling Organizational Rules in the Multi-agent Systems Engineering Methodology*. In Canadian Conference on AI, pp. 1–15, 2002.

[9] J. DiLeo, T. Jacobs, and S. A. DeLoach. *Integrating Ontologies into Multiagent Systems Engineering*. In AOIS@CAiSE, 2002.

[10] C. A. Iglesias, M. Garijo, and J. Centeno- González. *A Survey of Agent-Oriented Methodologies.* In ATAL 98: Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages, pp. 317–330, London, UK, 1999.

[11] D. Kinny, M. Georgeff, and A. Rao. *A Methodology and Modelling Technique for Systems of BDI Agents*. In Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, 1996.

[12] A. L. Self and S. A. DeLoach. *Designing and Specifying Mobility within the Multiagent Systems Engineering Methodology*. In SAC'03: Proceedings of the 2003 ACM Symposium on Applied Computing, pp. 50–55, New York, NY, USA. ACM Press, 2003.

[13] S. Vafadar, A. Barfouroush, and M. Ayatollahzadeh. *Towards a More Expressive and Refinable Multiagent System Engineering Methodology*. In AOIS, pp. 126–141, 2003.

[14] M. F. Wood and S. A. DeLoach. *An Overview of the Multiagent Systems Engineering Methodology.* In First International Workshop on Agent-Oriented Software Engineering (AOSE 2000), pp. 207–221, Secaucus, NJ, USA. Springer-Verlag New York, Inc, 2001.

[15] F. Alonso, S. Frutos, L. Martinez, and C. Montes. *SONIA: A Methodology for Natural Agent Development.* In Fifth International Workshop Engineering Societies in the Agents World, Toulouse, France, October 2004.

[16] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. *Tropos: An Agent-Oriented Software Development Methodology*. Autonomous Agents and Multi-Agent Systems, 2004.

[17] A. Collinot, P. Carle, and K. Zeghal. *Cassiopeia: A Method for Designing Computational Organizations.* In The First International Workshop: Decentralized Multi-Agent Systems DIMAS'95, pp. 124–131, Crakow, Poland, November 1995.

[18] M. Elammari and W. Lalonde. *An Agent-Oriented Methodology: High-level and Intermediate Models.* In the First International Bi-Conference Workshop on Agent-Oriented Information Systems, Seattle, USA and Heidelberg, Germany, May-June 1999.

[19] A. Omicini. *SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems.* In Agent-Oriented Software Engineering, pp. 185–194. Springer, LNCS 1957, 2001.

[20] L. Padgham and M. Wimikoff. *Prometheus: A Methodology for Developing Intelligent Agents*. In Agent-Oriented Software Engineering III, pp. 174–185. Springer, LNCS 2585, 2003.

[21] M. Wooldridge, N. R. Jennings, and D. Kinny. *The Gaia Methodology for Agent-Oriented Analysis and Design.* Autonomous Agents and Multi-Agent Systems, 2000.

[22] F. Zambonelli, N. R. Jennings, and M. Wooldridge. *Developing Multiagent Systems: The Gaia Methodology.* ACM Transaction on Software Engineering and Methodology, 2003.

[23] R. Bravo. *SISGECOMA. Sistema Generador de Código para la Metodología MASINA.* Universidad de Los Andes, Venezuela. 2009.

[24] FIPA.http://www.fipa.org.

[25] J. Whitten and L. Bentley. *Análisis y Diseño de Sistemas de Información*. México: McGraw-Hill. 2003.

[26] P. Rivero. *EDISMA: Entorno de Desarrollo Integrado para la Creación de Sistemas MutiAgentes*. Universidad de Los Andes, Venezuela. 2013.

[27] A. Mészáros. *Code Generation from AML to Jadex*. Comenius University in Bratislava. Eslovaquia. 2010.

[28] J. Aguilar y R. Castellanos. *Modelo Ontológico de Verificación de Sistemas Multiagentes Diseñados bajo MASINA*. Avances en Sistemas e Informática, September 1997.

[29] S. Goyal. Major *Seminar on Feature Driven Development.* Munich. Technical University Munich. 2007.

[30] L. Calabria. *Metodología FDD*. Universidad ORT Uruguay. Facultad de Ingeniería. 2003.