UNIVERSIDAD DE LOS ANDES

FACULTAD DE INGENIERÍA

DIVISIÓN DE ESTUDIOS DE POSTGRADO

POSTGRADO EN COMPUTACIÓN

UNIVERSIDAD
DE LOS ANDES

EDISMA: Entorno de Desarrollo Integrado para la creación de Sistemas MultiAgentes

Autora: **Paola S. Rivero P.**
Tutor: **Francisco Hidrobo**
Cotutor: **Addison Ríos Bolívar**

Trabajo de grado presentado ante la ilustre Universidad de Los Andes
como requisito parcial para optar al grado de
Magister Scientiae en Computación

Mérida, Abril, 2013

UNIVERSIDAD
DE LOS ANDES

EDISMA: Entorno de Desarrollo Integrado para la creación de Sistemas MultiAgentes

Tesis entregada a la Universidad de los Andes
en cumplimiento parcial de los requisitos
para optar al grado de
Magíster en Computación
Facultad de Ingeniería

por

Paola S. Rivero P.

Mayo, 2013

Tutor: **Francisco Hidrobo**

VEREDICTO DEL TRABAJO DE GRADO

Los suscritos, Miembros del Jurado designado por el Consejo Técnico del Postgrado en Computación para conocer y evaluar el Trabajo de Grado *"EDISMA: Entorno de Desarrollo Integrado para la Creación de Sistemas Multiagentes"*, realizado por la Ingeniera ***Paola Stephanie Rivero Pérez***, cédula de identidad N°V-17.662.012, acuerdan, según lo establecido en el Artículo 11 de las Normas de Funcionamiento del referido, el siguiente veredicto:

Trabajo de Grado APROBADO

Observaciones:

Prof. Francisco Hidrobo
Tutor

Prof. Addison Ríos
Cotutor

Profa. Mariela Cerrada
Jurado

Profa. Haydemar Núñez
Jurado

Mérida, 31 de Mayo de 2013

AGRADECIMIENTOS

Deseo agradecer a mi familia, y los profesores Francisco Hidrobo y Addison Ríos.

...

Este trabajo ha sido parcialmente financiado por CDCHTA-ULA, proyecto No. I-1237-10-02-AA, titulado : *Diseño y Construcción de una Plataforma para el Desarrollo e Implantación de Sistemas Multiagentes.*

Índice

Índice de figuras

Índice de tablas

RESUMEN

En este trabajo se presenta un entorno de desarrollo integrado para la creación de sistemas multiagentes denominado EDISMA, modelados en MASINA y que pueden ser instanciados en el MGS desarrollado en la Universidad de Los Andes. MASINA es una metodologa desarrollada para especificar sistemas multiagentes en ambientes de automatización industrial y que las siglas significan "MultiAgent Systems for INtegrated Automation"

EDISMA está compuesto por un entorno gráfico, un editor de textos y un analizador de diagramas de UML creados con Umbrello. Mediante la interfaz gráfica de usuario es posible especificar las planillas que documentan el diseño de los modelos de agente, tareas, coordinación y comunicación. EDISMA esta desarrollado en QT, haciendo uso de librerias creadas en C++ que permiten ejecutar operaciones sobre las estructuras de datos implantadas.

Para implementar un SMA deben crearse una serie de elementos de software, los cuales pueden ser generados con EDISMA, conformados por la documentación de los modelos de MASINA, archivos del tipo cabecera que realiza las definiciones de los agentes y de los métodos, archivos que realicen especificación de los métodos definidos, archivos principales que permite la instanciación de los agentes, un disparador que cree las instancias de los agentes y finalmente un constructor (makefile) que compile el SMA. Estos archivos generados deben completarse, ya que aspectos como la ejecución de las tareas, integración de librerias externas y estructura de los mensajes, no están contemplados en esta primera versión de EDISMA.

Para verificar los códigos generados por EDISMA, se realizó una prueba para un caso de estudio de un SMA compuesto por cuatro agentes, los cuales poseen distintas tareas y servicios. Estos agentes llevan a cabo una conversación conformada por actos de habla, comunicandose entre sí para enviar y recibir mensajes, que poseen un determinado contenido. Finalmente se creó con EDISMA dicho SMA, y se comprobó que lo archivos generados, implentan gran parte del mismo, dejando al programador la actividad de completar el código que implementa una parte del modelo de tarea y la estructura de los mensajes.

Capítulo 1

Introducción

Los agentes inteligentes poseen características que permiten representar, de una manera adecuada, ciertos comportamientos humanos tales como: reactividad, autonomía, colaboración, comunicación, inferencia, movilidad [21,24,48]. Aún cuando existen diferentes teorías de agentes, lenguajes, arquitecturas y aplicaciones exitosas basadas en agentes inteligentes, hay pocos trabajos que especifican metodologías, métodos, técnicas o herramientas para desarrollar aplicaciones usando esta tecnología [4,9,15,34,52].

El objetivo de este proyecto es construir un entorno de desarrollo integrado (IDE del inglés *Integrated Development Environment*), que permita crear agentes mediante una interfaz gráfica. Para ello es necesario definir, previamente, un conjunto estandarizado de conceptos, prácticas y criterios, para enfocarse a un tipo de problemática en particular, que servirá como referencia para enfrentar y resolver problemas de naturaleza similar. Tomaremos como principio la teoría de Sistemas Multiagentes (SMA). Es importante destacar que en este caso en particular, es necesario que los agentes estén diseñados bajo un mismo método o metodología, es por ello que se asume que los agentes que forman parte del SMA, se han diseñado utilizando MASINA [19].

El Entorno de Desarrollo Integrado para la creación de SMA (EDISMA) se diseñará con la intención de facilitar la creación de agentes modelados con MASINA la cual es una metodologa desarrollada para especificar sistemas multiagentes en ambientes de automatizacin industrial y que las siglas significan "MultiAgent Systems for INtegrated Automation".

EDISMA permite a los investigadores, programadores y usuarios disminuir el tiempo de implantación con los detalles de bajo nivel necesarios para proveer un agente funcional. Es decir, este proceso será transparente hasta cierto punto, obtendiéndose un código fuente génerico sobre el cual, varias instancias (en este caso los agentes) son integrados para una solución dada. Esta herramienta incluye la generación de código fuente en C++ y haciendo uso de las librerias del MGS, que posterior a su compilación, es incorporado a la plataforma de Medio de Gestión de Servicios (MGS) el cual es "*el conjunto básico de módulos de software que implantan las abstracciones mínimas para la ejecución y manipulación de agentes en un ambiente computacional*" [7] .

1.1. Antecedentes

Un agente puede definirse como "*Un sistema informático situado en un entorno (ambiente), capaz de realizar acciones autónomas dentro de ese entorno para alcanzar sus objetivos*" [11]. Debido a la amplitud de este concepto, es necesario enmarcar esta definición en un subconjunto de especificaciones más detalladas.

Para ello, inicialmente es necesario tomar en cuenta una metodología para especificar agentes, en este caso SMA, que permita modelar cada uno de los aspectos básicos que conforman este tipo de entidades. Posteriormente, es necesario tener una plataforma que permita instanciar estas entidades, tal y como lo es el MGS; el cual es una plataforma que provee los servicios fundamentales para el funcionamiento del SMA, y finalmente se debe contar con un IDE, que sirva como receptor de los parámetros externos que permiten modelar los agentes. A continuación se especifican los antecedentes de cada uno de los aspectos considerados anteriormente.

En [3] se propone una metodología fundamentada en MASINA [10], con las funcionalidades ofrecidas por el Lenguaje Unificado de Modelado (UML), el cual permite modelar y especificar los métodos o procesos de los agentes, y por la Técnica de Desarrollo de Sistemas de Objetos (TDSO) [41], para la definición del universo de clases, así como la especificación

de las mismas con sus atributos y métodos .

MASINA a través de su fase de conceptualización permite definir los servicios y la arquitectura preliminar del SMA, luego en la fase de análisis se hace uso de los modelos de agentes, tareas, comunicación y coordinación para describir las características básicas del SMA. En [5, 8, 12, 20] proponen modelos de referencia descritos con MASINA [10] y basados en el modelo genérico basado en agentes para Sistemas de Control Distribuido Inteligentes basado en Agentes (SCDIA). Dichos trabajos hacen hincapié en el concepto de autonomía de los agentes y el comportamiento basado en servicios, haciendo de los SMA una alternativa de modelado adecuada para desarrollar sistemas en ambientes distribuidos, abiertos y heterogéneos, con herramientas precisas para su implantación. A continuación se muestran una serie de trabajos realizados con MASINA y que están relacionados directamente con el IDE a desarrollar. En las Tablas 1.1, 1.2, 1.3 y 1.4 se encuentran especificados los aspectos más resaltantes de cada uno de estos trabajos.

Titulo	Sistema Generador de Código para la metodología MASINA [18].
Descripción	Propone un sistema de generación automática de código SISGECOMA para MASINA, creado en C++ y tomando los modelos de: agente, tareas, coordinación y comunicación como entradas. En SISGECOMA, el usuario realiza la especificación de los agentes llenando las plantillas correspondientes a los modelos, mediante una interfaz gráfica basada en formularios. Luego una vez recopilada la información, se construye una estructura de datos llamada concepto de agente, que finalmente produce la generación de código.
Debilidad	Se debe validar los modelos de MASINA para que el código a generar no sea inconsistente.

Tabla 1.1: SISGECOMA

Titulo	Sistema multi-agente para crear agentes de control para el SCDIA [15].
Descripción	Propone un sistema llamado SIGECO que permite la generación de agentes de control del SCDIA que incluye la generación de código fuente del agente, su compilación e incorporación al SCDIA. Para el desarrollo de SIGECO, se utilizó la plataforma de agentes JADE y se definieron tres agentes: Agente Central, Agente Generador de Código y Agente de Comportamiento. Estos agentes se comunican entre sí para generar agentes del SCDIA, mediante el uso de una ontología de generación de código
Debilidad	El SIGECO fue desarrollado en JAVA lo cual rompe esquemas en cuanto a la dependencia de una máquina virtual de java que no es código abierto

Tabla 1.2: SISGECO

Titulo	Modelo ontológico de verificación de sistemas multiagentes diseñados bajo MASINA [6].
Descripción	Esta investigación presenta un modelo ontológico para la verificación de los diseños de SMA hechos con MASINA, donde las propiedades a verificar son formalizadas semánticamente y son expresadas como sentencias en Lógica de Primer Orden. Para lograr ésto se utilizan dos técnicas: la verificación de manera composicional y el cruce de modelos. Dicho modelo fue implementado en Protégé-OML y probado en un caso de Estudio (Diseño de un sistema Operativo Web usando SMA)
Debilidad	La ontología fue verificada instanciando entidades en Protégé y para la validación sintáctica se usó un proveedor de servicios web WonderWeb, el cual es un servicio código cerrado

Tabla 1.3: Modelo ontológico para la verificación de los diseños de SMA

Titulo	Generación Automática de Código a Partir de Máquinas de Estado Finito [51].
Descripción	En este trabajo presenta una herramienta de generación automática de código fuente en lenguajes orientados a objetos para modelos abstractos expresados en UML, generando código fuente en el lenguaje C++, a partir de éstos. Dicha herramienta podrá ser integrada a herramientas CASE de modelado, con capacidades apropiadas de exportación de modelos del UML en formato XMI.
Debilidad	Carencia de validaciones sintácticas y semánticas. No considera la generación de código fuente para elementos que expresan procesamiento concurrente en los diagramas de estados

Tabla 1.4: Generación automática de código fuente

1.2. Definición del problema

Existen plataformas que permiten la creación de aplicaciones de agentes. En la Tabla 1.2 se especifican diversos software, arquitecturas y herramientas que permiten modelar agentes, en su mayoría desarrolladas en JAVA tales como MASON, ZEUS, MaDKit y AgentBuilder algunos de licencia libre como los tres primeros y privativos como el AgentBuilder. Actualmente, sólo MASON, MaDKit y JADE son actualizados periódicamente, incorporando funcionalidades para mantenerse a la vanguardia de la tecnología, y actualizando de tal modo sus versiones, incluyendo además aspectos que permiten compatibilidad con sistemas operativos actuales.

En la tabla 1.2 se muestran aplicaciones que permiten gestionar SMA, las cuales carecen de un estándar para el formato de los archivos que estás generan, es decir la documentación obtenida solo puede ser utilizada con la herramienta que fue creada.

Tabla 1.5: Sistemas que implementan SMA

Nombre	Especificaciones
Mason [35]	Permite simular sistemas multiagentes. Permite la Visualización en 2D y 3D. Es multiplataforma (UNIX, Windows, MacOS X) Cuenta con versions actuals Esta desarrollado en JAVA
Zeus [44]	ZEUS permite modelar un agente que puede ser personalizado para aplicaciones específicas modelando recursos, competencias, información, relaciones de la organización y los protocolos de coordinación. Provee un conjuntos de elementos: Buzón de correo, un motor de Coordinación, una base de datos que describe las relaciones del agente con otros agentes en la sociedad, un Planificador y Programador de tareas , una ontología de base de datos , una base de datos de tareas/Planes, la Ejecución de un monitor que mantiene el reloj interno del agente. Esta desarrollado en JAVA
ADEPT [45]	ADEPT produjo una arquitectura de agente (o marco) para los procesos de negocio. Este trabajo describe un agente que ofrece varias capas de lenguajes reutilizable y servicios para la construcción de sistemas de agentes: coordinación y lenguajes de comunicación, la gestión de descripción de la lógica basada en el conocimiento, cooperación entre la distribución de la información modelos de organización y gestión de conflictos
MaDKit [28]	Es una plataforma multiagente, modular y escalable desarrollado en Java y construido bajo el enfoque AGR (agente,grupo,rol). Los Agentes creados MaDKit desempeñar funciones en grupos y crean sociedades artificiales. MaDKit es software libre lincencia GNU que permite: creación de agentes asi como la gestión del ciclo de vida de los mismo. Una infraestructura organizativa para la comunicación entre los agentes alta heterogeneidad en las arquitecturas de agentes: no hay un modelo predefinido agente. Simulación de sistemas MultiAgentes locales y distribuidos.
Open Agent Architecture (OAA) [36]	Emplea una arquitectura abierta para el desarrollo sistemas multiagentes. Su característica clave es que posee un agente facilitador poderoso que coordina todos los agentes en el entorno , y que también es un planificador. El facilitador puede recibir tareas de los agentes, las descomponen y delega actividades.
AgentBuilder [1]	Es una suite de herramientas integradas para la construcción de agentes inteligentes de software. Su ultima versión posee compatibilidad con Windows 98/ME/NT/2000/XP, Solaris y Linux Existen dos versiones disponibles: LITE, ideal para construir un agente, aplicaciones individuales y una versión PRO, que contiene una serie avanzada de herramientas para probar y construir SMA. AgentBuilder es construido bajo el enfoque DBI. Es un software privativo desarrollado en JAVA.

El software más destacado es JADE [17]; JADE provee servicios para implantar SMA. Sin embargo, para propósitos prácticos de implantación en ambientes reales, específicamente industriales, puede resultar poco atractiva debido a:

1. Restricciones para la implantación de modelos de coordinación emergentes.
2. Restricciones para la implantación de modelos de inteligencia (mecanismos de razonamiento, marcos ontológicos, mecanismos de aprendizaje, entre otros).
3. Generación de código en un único lenguaje (JAVA).
4. Dependiente de la Máquina Virtual de Java.

Así pues, esta investigación está orientada hacia la obtención de un IDE que permita el desarrollo e implantación de Sistemas Multiagentes con las siguientes características generales:

1. Creación de SMA mediante un Interfaz Gráfica de Usuario (IGU).
2. Independencia del lenguaje de programación. Aunque, en una primera versión, se escogerá un lenguaje de propósito general como C++; en versiones posteriores se podrá incorporar soporte para otros lenguajes.

1.3. Objetivos

En base al problema planteado, se definen los siguientes objetivos:

1.3.1. Objetivo General

Construir un IDE que permita la creación de sistemas multiagentes de manera genérica y a través de una interfaz gráfica de usuario.

1.3.2. Objetivos Específicos

1. Diseñar la arquitectura de EDISMA.
2. Diseñar el esquema de interacción de la herramienta con el usuario.
3. Definir la integración de los productos que genere la herramienta con el MGS.
4. Construir el generador de Agentes.
5. Validar la herramienta con un caso de estudio, instanciando un SMA compuesto por cuatro agentes: Agente Aplicación, Agente Repositorio, Agente Especializado y Agente Negocio.

1.4. Metodología

En el desarrollo del IDE se utilizará el método "White Watch" [16], el cual es "*un marco metodológico que describe, el conjunto estructurado de actividades necesarias para producir un producto de software sencillo y pequeño con documentación precisa*" [39] . Este método se divide en tres modelos: modelo de productos, modelo de procesos y modelo de actores, en el Capítulo 2 se describen las actividades consideradas más relevantes para la herramienta a desarrollar.

1.5. Alcance y limitaciones

Un fin preciso para este proyecto es implementar y dejar de manera operativa una herramienta gráfica para la creación de SMA, modelados con MASINA, y con el MGS creado en [11], permitiendo a los investigadores, programadores y usuarios disminuir el tiempo de desarrollo, ocultando algunos detalles de bajo nivel y facilitar el proceso de implantación de SMA.

Capítulo 2

Marco de referencia teórico

Este capítulo abarca las definiciones que serán mencionadas a lo largo del documento y que, principalmente, hacen referencia a la teoría de agentes y a la estructura de los elementos de software.

2.1. Agentes Inteligentes

En el estado del arte de la teoría de agentes existen diversas definiciones de lo que es un agente [22,25,55,58,59,66], por lo cual aún no existe una definición universalmente aceptada. Entre estas definiciones podemos resaltar [13]:

Shoham: Usualmente, cuando la gente usa el término agente se refiere a una entidad que funciona continua y autónomamente, en un entorno en el cual otros procesos ocurren y existen otros agentes [59].

Russel: Un agente es una entidad que percibe su entorno y actúa bajo estas percepciones [55].

Franklin y Gasser: Un agente autónomo es un sistema situado en, y como parte de, un entorno, que detecta dicho entorno y actúa en él, en búsqueda de sus propios objetivos. [25].

Wooldridge y Jennings: Es un sistema de hardware o, principalmente, software, que es autónomo, reactivo y social [66].

Al definir un agente, se toma en cuenta ciertas características ó propiedades que permiten describirlos tales como : Reactividad, Proactividad, Comunicación, Sociabilidad, Movilidad,

Autonomia, Veracidad, Benevolencia, Racionalidad, Inteligencia y Adaptatividad [22,25,66]. Además poseen una serie de características adicionales que permiten diferenciarlos de los objetos, entre las cuales se encuentran: Autonomía, Flexibilidad y Estado de actividad.

2.1.1. Sistemas MultiAgentes

Un Sistema MultiAgente (SMA) comprende un conjunto de agentes que poseen determinados comportamientos, los cuales se comunican para conocer un problema y buscar una(s) solución(es), utilizando para ello protocolos y lenguajes de programación de alto nivel. En la Figura 2.1 [13] se observa un ejemplo de un SMA, compuesto por una serie de agentes los cuales gráficamente se encuentran agrupados mediante ovalos, conforman un Agente de Supervisión y un Agente Mantenimiento, en donde ambos cooperan entre sí para ejecutar diversas funcionalidades, entre ellas enviar y solicitar la información al Agente de Visualización, el cual permite el despliegue de datos en diferentes dispositivos de salida, a solicitud de cualquier otro agente del SMA. Algunas de las características de los SMA son:

- Cada agente tiene capacidad para solucionar parcialmente el problema.
- No hay un sistema global que los controla.
- Los datos no están centralizados.
- La computación es asíncrona, ya que no es neceario que exista una relacin temporal entre las instancias que transmiten información y la las que reciben.

Un SMA, por su naturaleza, es un sistema distribuido el cual produce un resultado inteligente [22,37,46,63,65] y que puede poseer 3 niveles de organización:

- **Organización microsocial**, que se interesa esencialmente por las interacciones y conexiones que existen entre los agentes.
- **Organización en grupo**, que se interesa en las estructuras que se producen en la composición de una organización compleja.

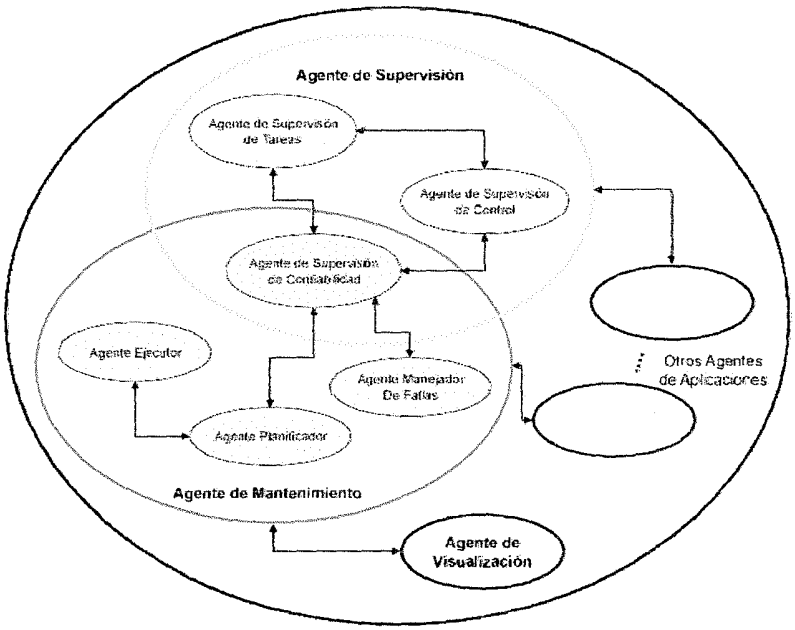


Figura 2.1: Ejemplo de un SMA para la automatizacin industrial

- **Organización de sociedad global**, que se interesa en la dinámica de un gran número de agentes, así como en la estructura general del sistema y su evolución.

Los SMA son construidos tomando en cuenta dos enfoques [22, 37, 46, 63, 65] : el primero de ellos denominado **enfoque formal**, en el cual se desea incorporar la mayor cantidad de inteligencia posible al agente, utilizando descripciones formales del problema a resolver; y un segundo enfoque denominado **enfoque constructivista**, en el cual se incorpora al agente algún mecanismo ingenioso de interacción para que éste genere un comportamiento inteligente, el cual no haya sido prediseñado ó predefinido en él.

Los SMA poseen una estructura organizativa que puede clasificarse de dos maneras [22,65]:

- **Definida a priori por el diseñador**, donde las relaciones y transformaciones son previamente definidas es decir son predecibles.

- **Definidas a posteriori**, se habla entonces de organizaciones emergentes, en donde los roles y las relaciones de los agentes no son predefinidos sino aparecen en el transcurso de la dinámica del SMA.

Los SMA por ser sistemas organizados poseen relaciones de subordinación y éstas son de dos tipos [22,37,65]; la primera de ellas son las **estructuras jerárquicas**, que suponen que estas relaciones forman una estructura piramidal, y la segunda son las **estructuras igualitarias**, en las cuales un agente puede pedirle a cualquier otro agente realizar una tarea, y este último eventualmente negarse. Estas estructuras son más características de organizaciones en las cuales todos los agentes intervienen uniformemente en la decisión final. En los SMA aparecen un conjunto de conceptos de gran interés, los cuales son: interacción, comunicación, actos de habla, conversaciones, lenguajes de comunicación, ontologías para comunicación, entre otros, los cuales son definidos a continuación [22,37,46,63,65].

2.1.2. Interacción en los SMA

En la teoría de agentes la interacción estudia, analiza y concibe los diferentes mecanismos que les permiten obrar recíprocamente, para realizar sus tareas y satisfacer sus objetivos [22, 37,46,63,65]. La interacción permite, por medio de la comunicación, transmitir información para modificar el estado de otro agente receptor o su ambiente. A continuación se explican cada uno de los elementos que forman parte de la interacción entre los SMA:

- **La comunicación**: en los SMA consiste en intercambiar información entre los agentes, a través de la producción y percepción de signos en un sistema compartido de símbolos convencionales [2,22,37,46,54,65].

- **Los actos de habla**: definen las acciones intencionales efectuadas en el transcurso de una comunicación (su conjunto constituyen una conversación) [22,37,62,65].

- **Las conversaciones**: son una secuencia de actos de habla que persiguen alcanzar un objetivo, o realizar una actividad del SMA [2,22,37,61,62,65].

- **Los lenguajes de comunicación**: para agentes el ACL (*Agent Communication Language*) [22,37,42,61,62,65] es el lenguaje más conocido y que ha venido estandarizando FIPA (*Foundation for Intelligent Physical Agents*) [23].

- **Las ontologías para comunicación**: las cuales son usadas como una especificación de un sistema de términos compartidos por los agentes (esto amplía la definición clásica de ontología para representar conocimiento).

2.1.3. Coordinación

La coordinación de acciones puede describirse como el conjunto de actividades suplementarias necesarias a realizar en una comunidad de agentes para poder actuar colectivamente [2, 22, 37, 57, 61, 62, 65]. La actividad de coordinación se vuelve fundamental en el marco de la cooperación, y puede ser definida como la articulación de las acciones individuales realizadas por cada uno de los agentes, de manera que el conjunto tenga éxito [2, 22, 61]. La coordinación engloba una serie de conceptos tales como: cooperación, negociación, asignación de tareas y recursos, y la planificación [2, 22, 61, 62, 65]

2.2. MASINA

Es una extensión de MAS-CommonKADS [33], la cual consta de las fases de a) conceptualización, b) análisis, c) diseño, d) codificación y pruebas, e) integración, y f) operación y mantenimiento [10].

a) Fase de conceptualización: define los servicios requeridos del sistema, quienes lo requieren y se realiza una identificación de aquellos componentes del sistema que pueden ser considerados agentes y se propone una arquitectura preliminar del SMA.

b) Fase de análisis: está compuesta por cinco modelos los cuales se consideran suficientes para describir las características básicas de los SMA. Particularmente, MASINA hace uso de los modelos de agente, tareas, comunicación, coordinación y el modelo de inteligencia, los cuales son explicados a continuación.

- **Modelo de agente**: especifica las características de un agente tales como : aspectos básicos, habilidades, y servicios. Además, a este modelo se le agregan dos atributos: componentes del agente, que permite indicar si el mismo es un SMA (permite repre-

sentar niveles de abstracción o jerarquías en el diseño del SMA), y marco de referencia para indicar si la arquitectura del agente bajo diseño está basada en un modelo de referencia dado. A modo de ejemplo, en la Tabla 2.1 [10] se puede observar la plantilla que se genera del modelo de agentes, también existen plantillas para cada uno de los modelos explicados a continuación [13]

- **Modelo de tareas**: se especifican las tareas que se requieren y se describe el macro-procedimiento (sub-tareas) que se debe seguir para la ejecución de dicha tarea. Los agentes pueden usar técnicas inteligentes (Redes Neuronales Artificiales, Algoritmos Genéticos, etc.) en función del tipo de tareas que realizan (sean o no ellos inteligentes).

- **Modelo de inteligencia**: describe los aspectos necesarios para incorporar la noción de inteligencia a un agente, y además se propone un esquema que integra conceptos como experiencia, representación de conocimiento (conocimiento de dominio, conocimiento estratégico y conocimiento de tareas), mecanismo de aprendizaje, y mecanismo de razonamiento. Este modelo se activa a través del modelo de tareas, mediante tareas específicas que conduzcan a procesos de razonamiento, aprendizaje, etc.

- **Modelo de coordinación**: permite especificar las conversaciones que se dan entre los agentes. Este modelo se concentra en la definición de las conversaciones que permiten una comunicación coordinada entre los agentes, y no en los actos de habla específicamente involucrados, los cuales pasan a ser detallados en el modelo de comunicación. Las interacciones (actos de habla) presentes en una conversación dada entre agentes, se representan, a nivel gráfico, a través de un diagrama de secuencia de UML. Este modelo permite una descripción detallada de cada conversación, en donde se especifican los agentes que integran la conversación y los actos de habla entre ellos. Además, se especifica el esquema de coordinación, de planificación, el mecanismo de comunicación directa o indirecta, el metalenguaje y la ontología.

- **Modelo de comunicación**: considera las interacciones de una manera amplia, y pro-

Tabla 2.1: Plantilla del Modelo de Agente

Agente		
Nombre		Nombre único para el agente
Posición		Ubicación del agente dentro del sistema multiagente
Componentes		Agentes que lo componen (si es un SMA)
Marco de referencia		Marco de referencia para el modelado de agentes
Descripción del agente		Lo que hace el agente
Objetivos del Agente		
Nombre		Nombre del objetivo
Descripción		Detalles del objetivo
Parámetro de entrada		Parámetros necesarios para el cumplimiento del objetivo
Parámetro de salida		Parámetros que se esperan obtener una vez cumplido el objetivo
Condición de activación		Condiciones que activan las tareas asociadas al cumplimiento del objetivo
Condición de finalización		Condiciones que indican la terminación de las tareas asociadas al cumplimiento del objetivo
Condición de éxito		Condiciones que indican el cumplimiento del objetivo
Condición de fracaso		Condiciones que indican el no cumplimiento del objetivo
Ontología		Descripción de la ontología
Servicios del Agente		
Nombre		Nombre del servicio ofrecido por el agente
Descripción del Servicio		Detalles del servicio
Tipo de Servicio		Clasificación (Interno, Externo o ambos: servicio dual)
Parámetros de entrada		Parámetros necesarios para el cumplimiento del servicio
Parámetros de salida		Parámetros obtenidos al finalizar el servicio
Propiedades del Servicio		
Nombre	Valor	Descripción
Calidad		Valor de la calidad del servicio
Auditable		Valor del nivel de auditabilidad del servicio
Garantía		Valor del nivel de garantía de recibir la solicitud del servicio
Capacidad		Valor de la capacidad del agente de cumplir con el servicio (capacidad de respuesta)
Confiabilidad		Valor de confiabilidad del servicio
Capacidad del Agente		
Habilidades del agente		Descripción de las habilidades generales del agente
Representación del Conocimiento		Lenguaje de representación del conocimiento
Lenguaje de Comunicación		Lenguaje de comunicación que usa el agente
Restricción del Agente		
Normas		Descripción de las normas del agente para el cumplimiento del servicio
Preferencias		Preferencias del agente en el momento de atender las solicitudes de servicio
Permisos		Accesos a información permitidos para el agente para el cumplimiento de su objetivo

pone un modelo que describe los actos de habla involucrados en las conversaciones entre los agentes del SMA, especificados en el modelo de coordinación.

c) Fase de diseño: se obtienen los siguientes modelos, como paso previo a la implementación del SMA:

- Diseño del SMA: se toman en consideración los agentes resultantes de los modelos generados en la fase de análisis, para generar la arquitectura final del SMA. Se plasmarán los niveles de abstracción, es decir, la visión holística del SMA.
- Diseño de red: se describen los aspectos relevantes a la plataforma del SMA, como las bases de conocimiento, la arquitectura de red, etc.
- Diseño de la plataforma: se determina la plataforma de desarrollo del SMA, es decir, se escogen las tecnologías (hardware y software) para implantar la plataforma.

d) **Fase de codificación y pruebas**: se codifica y prueba cada agente, utilizando las herramientas escogidas para tal fin.

e) **Fase de integración**: se realiza el acoplamiento entre los agentes del SMA, y de éstos con la plataforma real donde funcionarán.

f) **Fase de operación y mantenimiento**: consiste en el funcionamiento del sistema propiamente dicho. Durante la operación del sistema se realizan tareas de actualización (mantenimiento) de los componentes del sistema, para adaptarlos a la evolución del entorno o a nuevos requisitos.

2.3. Medio de Gestión de Servicios

Para la operación de un SMA es necesario contar con una plataforma básica que provea los servicios fundamentales para el funcionamiento de los agentes. Estos servicios, clásicos en los sistemas distribuidos, deben incluir: creación, nombramiento, localización, búsqueda, comunicación, entre otros. En este sentido, se requiere de un Medio de Gestión de Servicios

(MGS), que permite la operación de los agentes de una manera segura y eficiente, sin afectar los objetivos propios del SMA.

Actualmente existe una organización que ha venido creando los estándares en teoría de agentes, para brindarles servicios a las comunidades de SMA, denominada FIPA, la cual fue originalmente creada en Suiza en 1996, para producir estándares de software para agentes y sistemas basados en agentes heterogéneos, y fue oficialmente aceptada por la IEEE en Junio del 2005 [23].

El MGS es el conjunto básico de módulos de software que implantan las abstracciones mínimas para la ejecución y manipulación de agentes en un ambiente computacional. La especificación FIPA define la plataforma de agentes como un sistema constituido por los recursos de hardware y software (sistema operativo, software de comunicaciones, software de gestión de agentes) necesarios para que los agentes puedan ser ejecutados. El MGS propuesto inicialmente en [19], extendido en trabajos posteriores [14], pretende conformidad arquitectural con el estándar FIPA. El MGS esta compuesto por 3 niveles (Ver Figura 2.2) : a) nivel interfaz, b) nivel medio y c) nivel de acceso a recursos.

a) Nivel interfaz. Define la interfaz entre el SMA y los componentes del sistema distribuido y se encarga de establecer las pautas de conversación entre los componentes del sistema distribuido y el SMA. Está constituido por cinco agentes:

- Agente Administrador de Agentes (AAA): gestiona el sistema de agentes (controla, registra y administra), de modo específico, este brinda una serie de servicios los cuales son: creación, destrucción, movimiento, localización y cambio de estado de los agentes. Opera en conjunción con el nivel base; de esta manera se garantiza la operatividad del sistema de agentes.

- Agente Gestor de Recursos (AGR): permite la administración de los recursos que deben ser compartidos entre los agentes del sistema.

- Agente Gestor de Aplicaciones (AGA): permite la gestión general de los agentes de

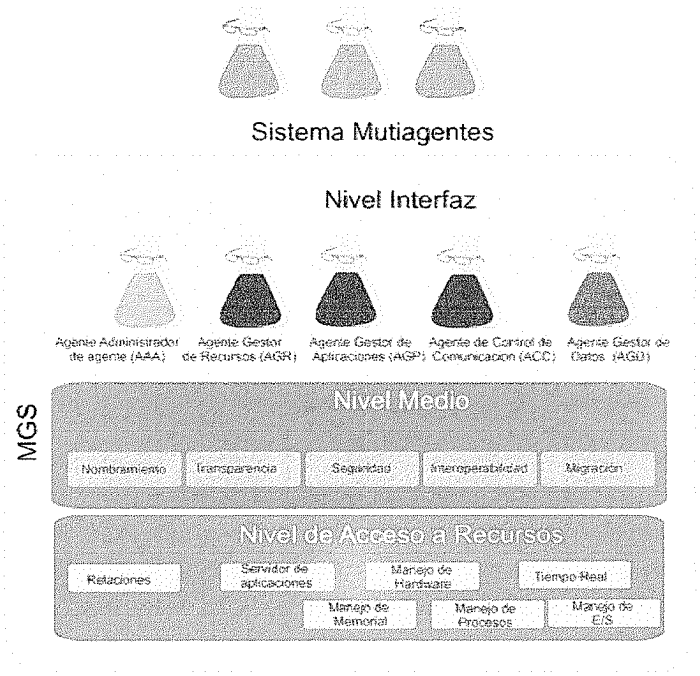


Figura 2.2: Arquitectura básica del MGS

aplicaciones especializados; entre los servicios del AGA se encuentran la localización de agentes de aplicaciones (se comporta como un servidor de páginas amarillas).

- Agente Gestor de Datos (ACD): se encarga de establecer el enlace con los lugares donde existan datos de interés para el proceso (agente) que se esté ejecutando, ya sea que estos datos provengan de bases de datos de otra fuente. Además, permite el traslado de los datos entre los diferentes dispositivos y/o aplicaciones de una manera transparente, para lo cual realiza las transformaciones requeridas, y el mantenimiento de los medios de almacenamiento de los datos.

- Agente de Control de Comunicación (ACC): se encarga de la comunicación entre los agentes a través del envio y recepción de mensajes, es decir éste recibe los mensajes con la información del emisor y del destinatario, determina la localización del agente (local o remota); y envía el mensaje a dicha localización.

b)Nivel medio: constituye el núcleo del sistema distribuido, provee servicios de software

que requieren los agentes para poder interactuar entre sí y con el nodo de ejecución. Proporciona transparencia y seguridad en las transacciones, interoperabilidad de las aplicaciones y componentes de software, migración de agentes, objetos y/o recursos, comunicación interprocesos, localización de recursos (agentes y objetos), y un sistema de nombramiento para la localización de agentes y/o objetos.

c) Nivel de acceso a recursos: está integrado por el núcleo básico del sistema operativo, el cual maneja las funcionalidades de tiempo real, cuando sean necesarias, y además manejadores de acceso a hardware específico que requiera el sistema. Se asume como sistema operativo Linux, en sus versiones tradicionales y su versión tiempo real; y por ello los agentes residen en procesos Linux. Cada proceso debe enlazar e invocar una biblioteca que implanta las llamadas al MGS. A este nivel, el MGS está compuesto por dos módulos que deben ser instanciados obligatoriamente, por los agentes del Nivel Interfaz (FIPA) para la realización de sus actividades: el manejador de agentes que se encarga de corresponder agentes hacia procesos Linux (realiza funciones como creación, destrucción y manejo de recursos del sistema operativo para la manipulación de agentes) y el manejador de comunicación que se encarga de proveer comunicación confiable de red orientada a invocación.

Para la investigación a desarrollar se utilizará un MGS creado en C++ y que implanta las tres capas descritas anteriormente.

2.4. Componente de *Software*

Se conoce como software al equipamiento lógico o el soporte de un sistema informático, comprende el conjunto de los componentes que hacen posible la realización de tareas específicas. Debido a la amplitud de este concepto, es necesario definir algunos elementos de *software* de acuerdo a sus características estructurales y funcionales, para de este modo identificar el producto a obtener en la implantación de la investigación.

Componente de *software*: es una parte importante, casi independiente y remplazable de un sistema que satisface una función clara en el contexto de una arquitectura bien definida.

Además, puede considerarse como una unidad de composición que sólo tiene dependencias de contexto explícitas y especificadas en forma contractual [50].

Entornos de Desarrollo Integrado: es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un sólo lenguaje de programación, o bien poder utilizarse para varios. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Interfaz de Programación de Aplicaciones (cuyas siglas en ingles es API acrónimo de *Aplication Programing Interface*): interfaz, generalmente especificada como un conjunto de operaciones, definidas por un programa de aplicación que permite acceder a la funcionalidad del programa. Esto significa que no sólo puede acceder a esta funcionalidad a través de la interfaz de usuario, sino que otros programas pueden utilizarla directamente.

Interfaz: especificación de atributos y operaciones asociados con un componente de software. La interfaz es utilizada como un medio para tener acceso a la funcionalidad de un componente.

CASE (acrónimo del ingles *Computer Asisted Software Engineer* que en español es Ingeniería del Software Asistidas por Computador): es un software que permite automatizar el ciclo de vida de desarrollo de los sistemas, y tambin se utiliza para la planificación estratégica de los datos. La CASE supone la aplicación de principios científicos a través de una metodología que ayude a producir software de alta calidad en un tiempo mucho más reducido [27].

En [49] definen la CASE como "un conjunto de herramientas de software que soporta las metodologías de ingeniería de software, y ayuda en la automatización del desarrollo de software . Comprende un amplio abanico de diferentes tipos de programas que se utilizan para ayudar a las actividades del proceso de software, como el análisis de requerimientos, el modelado de sistemas, la depuración y las pruebas.

Herramienta CASE: herramienta de software, como un editor de diseño o un depurador de programas, utilizada para apoyar una actividad en el proceso de desarrollo de software.

De acuerdo a estos elementos de software definidos, la implantación de la investigación será un IDE, ya que es la unidad de software que posee características estructurales que permiten implementar la construcción de SMA.

Capítulo 3

Marco metodológico

En este capítulo se definirán los métodos y herramientas utilizados para la implantación de la propuesta.

3.1. Método *White Watch*

El método *White Watch* [16] es la versión liviana del método *Watch* [39]. *White Watch* es un marco metodológico que describe el conjunto estructurado de actividades necesarias para construir un producto de software con un equipo de desarrollo pequeño, y además se fundamenta en la máxima reutilización de componentes de software como medio para acortar el tiempo de entrega de versiones parciales de producto y por consiguiente, el tiempo total de desarrollo del software. El método *WATCH* está compuesto por tres modelos que describen los tres elementos claves de todo método: el producto que se quiere elaborar, los actores que lo elaboran y el proceso que los actores deben seguir para elaborar el producto (ver Figura 3.1).

En *White Watch* se trata de disminuir la elaboración detallada de documentos y/o especificaciones de apoyo parcial al proceso de desarrollo, permitiendo al equipo de desarrollo pequeño (1 o 2 personas) dedicar más tiempo a las actividades de implementación de versiones operativas y evolutivas del producto. El rol de líder de proyecto es ejecutado en paralelo, y sin sobrecarga, por un miembro del equipo; mientras ejecuta otros roles técnicos. Así, las actividades gerenciales de control de calidad y de configuración, las cuales son indispensables

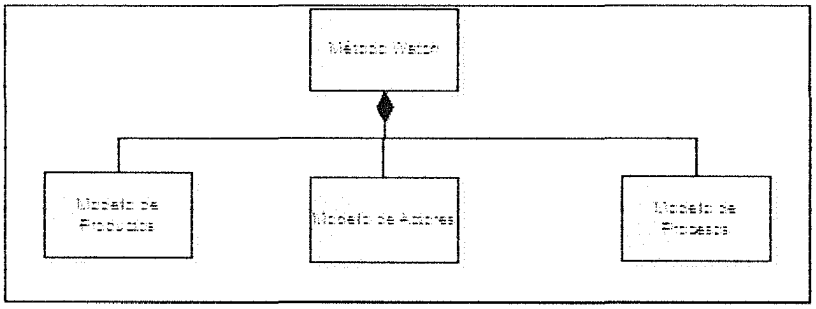


Figura 3.1: Modelos del método *White Watch*.

en todo proyecto de software, se limitan a prescribir las actividades básicas de control de cambios, de validación y de verificación de especificaciones técnicas y de productos parciales y finales. El modelo de procesos está organizado en dos grupos de procesos complementarios:

- Los procesos gerenciales, que incluyen los procesos de soporte.
- Los procesos técnicos de desarrollo del producto de software.

El modelo de procesos está inspirado en la metáfora del reloj de pulsera (*Watch* en Inglés), organiza los procesos técnicos en forma circular, en las posiciones del dial de un reloj, y ubica los procesos gerenciales en el centro (motor de control), de manera que éstos puedan controlar la ejecución de los procesos técnicos. Esta manera de estructurar el marco metodológico permite que la ejecución de los procesos de desarrollo sea cíclica, iterativa y controlada. En la Figura 3.2 se muestran los procesos del método los cuales se describen a continuación:

El Modelado del Negocio: agrupa a las actividades encargadas de caracterizar y entender el dominio de la aplicación, es decir, el sistema de negocios para el cual se desarrolla la aplicación. El Modelo del Negocio es el primer documento técnico que se produce durante la ejecución de los procesos técnicos del desarrollo de una aplicación (Ver Figura 3.3). Su objetivo es asegurar la obtención del conocimiento adecuado del dominio de la aplicación, de manera tal que se facilite, en los procesos siguientes, definir apropiadamente los requisitos de la aplicación. El dominio de una aplicación es el sistema funcional para el cual se elabora

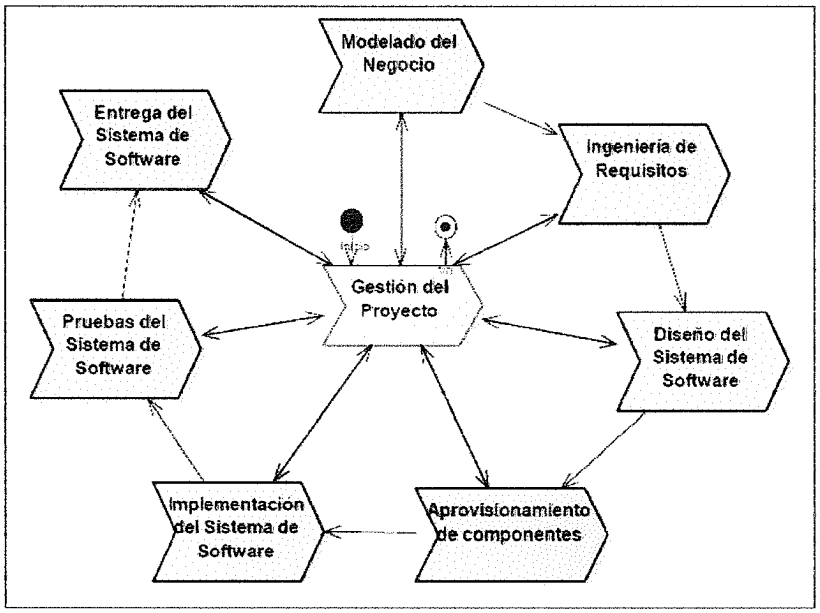


Figura 3.2: Modelo de Procesos del Método *White Wacht.*

dicha aplicación. Este sistema consiste en uno o más procesos de negocios que son ejecutados, con la finalidad de alcanzar objetivos predefinidos.

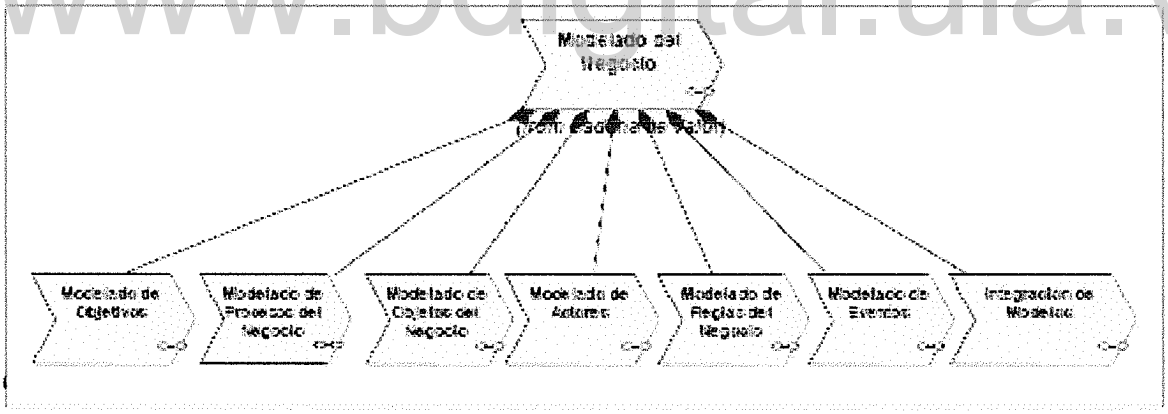


Figura 3.3: Modelo de Negocio

El Modelo de Dominio de la Aplicación es un documento técnico que describe:

- El sistema de negocios.
- Los objetivos del sistema de negocios.
- Cadena de Valor del proceso.

- Los procesos de negocio que permiten alcanzar dichos objetivos y los diagramas de actividad por subproceso de bajo nivel.
- Los actores que ejecutan estos procesos de negocio y las unidades a las cuales estos actores están adscritos.
- Los objetos de negocio que intervienen, en modo alguno, en la ejecución de los procesos de negocio.
- Modelo de clases de objetos del negocio.
- Lista de reglas de negocio.
- Los eventos que disparan o activan la ejecución de los procesos de negocio.

Ingeniería de Requisitos: el objetivo es identificar, describir, especificar y documentar cada uno de los requisitos funcionales y no funcionales que la aplicación debe satisfacer. El documento persigue dos objetivos complementarios. Por un lado, busca identificar y describir las necesidades de información y requisitos funcionales que los usuarios de la aplicación tienen; y por otro lado, el documento especifica técnicamente los requisitos funcionales y no funcionales que el equipo de desarrollo empleará para diseñar la aplicación. Los productos a obtener en esta fase del método son:

- Objetivos y alcance claramente definidos.
- Lista de actores clasificados.
- Modelos de casos de uso con sus respectivos escenarios.
- Lista de requisitos.
- Requisitos clasificados.
- Diagramas de caso de uso con sus descripciones textuales.

- Diagramas preliminar de clases.
- Diagramas de estados.

Diseño del sistema de *software*: abarca las actividades necesarias para especificar, diseñar y documentar la arquitectura de software que debe tener la aplicación, las cuales son especificadas a continuación.

- Definición de la estructura inicial de la aplicación: donde se origina el listado descriptivo de las metas de diseño, la estructura de la aplicación y la arquitectura de la aplicación.
- Diseño de la interfaz Usuario/Sistema: se lleva a cabo el diseño de las pantallas y el diagrama jerárquico de las mismas.
- Diseño de la Base de Datos: se realiza el modelo conceptual integrado de la base de datos, el esquema relacional o equivalente de la base de datos (integrado y verificado) y finalmente, el esquema físico de la Base de Datos.

- Diseño de componentes o módulos de software: se definen los componentes o módulos, se realiza la especificación de interfaces, y se crea la arquitectura de componentes.
- Especificación del diseño: se genera documento de diseño integrado y validado, y los procedimientos de administración de la base de datos.

Aprovisionamiento de componentes: organiza todas las actividades de diseño detallado de los componentes arquitectónicos, relacionados con la interfaz gráfica de la aplicación, sus componentes de software, su base de datos y su interacción con otras aplicaciones, de modo más detallado se realizan las siguientes actividades:

- Instalación de la plataforma de desarrollo.
- Adquisición de componentes requeridos.
- Adaptaciones de los componentes.

- Desarrollo de componentes: se desarrollan los componentes o módulos que no pudieron ser reutilizados o adquiridos, para ello se debe elaborar el diseño detallado de cada operación de interfaz, luego se codifica las operaciones de los componentes y finalmente se elabora la interfaz.

- Diseño de ejecución de pruebas de componentes: se realizan las especificaciones de caso de prueba y se depuran los errores encontrados durante las pruebas funcionales de cada componente.

Ensamblaje del sistema de *software* se llevan a cabo las siguientes actividades:

- Construcción de la interfaz U/S, se ensambla la capa de presentación con los componentes o elementos de software de la interfaz U/S, obteniendo como producto las especificaciones de casos de prueba y la interfaz U/S probada.

- Ensamblaje de componentes, se ensambla la capa lógica de negocio con los componentes que la integran y se ejecutan las especificaciones de casos de prueba correspondientes, y además se obtiene la capa lógica de negocios probada.

- Construcción de la base de datos.

- Pruebas de integración de las capas de la arquitectura.

Pruebas del sistema de *software*: ordena las actividades de pruebas de la aplicación como un todo, incluyendo las pruebas funcionales, no funcionales y de aceptación de la aplicación, se realizan las siguientes actividades:

- Realización de los mecanismos de prueba, especificándose cada uno de ellos, sus respectivos casos de prueba y se genera un informe de incidentes de prueba.

- Corrección de los errores.

Entrega del sistema de software: se estructura el conjunto de actividades que preceden a la puesta en producción de la aplicación. Incluye la capacitación de usuarios, la instalación

de la aplicación en su plataforma de producción u operación, las pruebas de instalación y la entrega final del producto. También, se elabora la documentación del sistema en forma de manuales o documentos de la aplicación; y se da el respectivo entrenamiento a los usuarios.

3.2. Herramientas de diseño

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modeling Language*) es un lenguaje de modelado de sistemas de software; está respaldado por el OMG (*Object Management Group*), y desde el año 1995, UML es un estándar aprobado por la ISO como ISO/IEC 19501:2005 *Information technology.* Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema [56]. UML ofrece un estándar para describir un "plano"del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. Las herramientas de diseño UML permiten a un diseñador crear los diagramas necesarios para construir un modelo de diseño completo. Este tipo de herramientas debe proporcionar una sintaxis y semántica sólida y de verificación, para esta investigación se utilizará Umbrello como un componente incorporado al IDE [29].

Umbrello es una herramienta de modelado UML, código abierto y su propósito principal es brindar soluciones en el análisis y diseño de sistemas. Permite realizar la transición entre el diseño y la implementación, este puede generar código fuente, en diferentes lenguajes de programación. Además, si desea utilizar UML en un proyecto C++ ya iniciado, Umbrello UML Modeller puede crear una maqueta del sistema, a partir del código fuente, realizando un análisis del código e importando las clases e instancías encontradas en él.

Este proyecto fue iniciado por Paul Hensgen, el cual se encargó de todo su desarrollo hasta finales del 2001 generando una versión 1.0 el nombre original de la aplicación era UML Modeller. Este programa continúa mejorándose y evolucionando, y es mantenido por un grupo de desarrolladores de diferentes lugares del mundo. Además, en septiembre de 2002, el

proyecto cambió el nombre de UML Modeller a Umbrello UML Modeller. El código generado consta de declaraciones de clases con sus métodos y atributos, de forma se pueda rellenar los espacios en blanco proporcionando la funcionalidad de las operaciones de sus clases. Umbrello UML Modeller 1.2 viene provisto con soporte para la generación de código en ActionScript, Ada, C++, CORBA IDL, Java, JavaScript, PHP, Perl, Python, SQL y XMLSchema.

Para la construcción del IDE se utilizará la versión más reciente (2.0) [31] en donde la interfaz de este programa esta implementada en Qt4/KDE4 y el componente lógico en C++.

3.3. Herramientas de desarrollo

Este tipo de herramientas le permiten al ingeniero de software crear una IGU, proporcionando acceso a componentes reutilizables y brindando un marco de trabajo con elementos básicos y avanzados para el diseño e implementación de la IGU. Para la implantación de la interfaz gráfica de EDISMA se utilizará QT su versión 4.7.4 que es una biblioteca multiplataforma de libre licencia, que permite desarrollar aplicaciones con una interfaz gráfica de usuario y el desarrollo de programas sin interfaz gráfica como herramientas para la línea de comandos y c/'onsolas para servidores.

QT [60] utiliza el lenguaje de programación C++ de forma nativa, adicionalmente puede ser utilizado en varios otros lenguajes de programación a través de *bindings* [1]. El API de la QT cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos, y ademas funcionalidades para el manejo de estructuras de datos tradicionales. Distribuida bajo los términos de GNU *General Public License*, es *software* libre y de código abierto, realizado la división de software QT de Nokia. QT Creator es un IDE creado por Trolltech para el desarrollo de aplicaciones,la cual requiere la versión 4.x de QT. Ver Figura 3.4

[1] Un binding es una ligadura o referencia a otro simbolo más largo y complicado, y que se usa frecuentemente [64]. Es una adaptación de una biblioteca para ser usada en un lenguaje de programación distinto de aquél en el que ha sido escrita.

Figura 3.4: QT Creator 4.7.4

Capítulo 4

Entorno de Desarrollo Integrado para la construcción de SMA

Este capítulo se describe el desarrollo de EDISMA, siguiendo el método *White-watch*, lo cual permite garantizar la uniformidad, consistencia, facilidad de integración y calidad de los distintos componentes arquitectónicos que forman parte del mismo, se inicia con el modelado de negocios, siguiendo con ingeniería de requisitos, diseño de software, aprovisionamiento de componentes, ensamblaje del sistema de *software* y finalmente las pruebas del sistema de *software*. Cada uno de los productos obtenidos en cada fase del método *White-watch*, se encuentran descritos en este capítulo y en el anexo digital.

4.1. Introducción

Al iniciar la creación de un SMA, es necesario tomar en cuenta, los aspectos importantes muy propios de ellos [13]: los agentes se asumen que son autónomos (capaces de tomar sus decisiones para satisfacer sus objetivos de diseño); se asume que los mecanismos de sincronización y de coordinación requeridas por un SMA están implantadas en la plataforma computacional que les da soporte; y los encuentros que ocurren entre los componentes del SMA es porque ellos están interesados y dichos encuentros están modelados en el modelo de coordinación del SMA.

Para la creación de un SMA con EDISMA se toman en cuenta los aspectos descritos anteriormente, es por ello que inicialmente se debe llevar a cabo el diseño con MASINA,

especificando los modelos de agente, tarea, comunicación y coordinación. En la Figura 4.1 se observa una infografía de la creación de SMA.

Inicialmente es necesario realizar el diseño de los agentes, utilizando MASINA, y luego éste es digitalizado por EDISMA, generando la documentación de los modelos para cada uno de los agentes del SMA en formato XML, obteniéndose un código intermedio que puede ser usado posteriormente como insumos, para diversas funcionalidades tales como verificaciones de modelos y creación de códigos fuentes en diversos lenguajes (siempre y cuando éstos provean compatibilidad con el MGS).

Luego se genera para cada uno de los agentes que conforman el SMA, los archivos en lenguaje C++, conformados por: un archivo cabecera extensión .h , un archivo cuerpo extensión .cpp y un archivo principal que contiene la función (main) que ejecuta al agente. Además para el SMA se genera un archivo makefile que contiene el script que permite compilar el SMA y un disparador que arranca la ejecución del SMA en el programa denominado "server".

Para llevar a cabo la instanciación del SMA, es necesario que los archivos que definen e implementan los métodos de los agentes, sean completados con el código faltante, por el programador. Finalmente se compilan y ejecutan los agentes en el MGS.

4.2. Modelado de Negocios

En esta primera fase se define el conjunto de objetos, conceptos y sus relaciones con el objetivo de expresar la lógica que contempla el funcionamiento de EDISMA.

4.2.1. Modelado de Objetivos del Sistema de Negocios, dominio de EDISMA

4.2.1.1. Alcance del Sistema

EDISMA podrá generar los modelos especificados en MASINA en lenguaje XML, obteniéndose así un código intermedio que puede ser usado posteriormente como insumos, para diversas funcionalidades tales como verificaciones de modelos y creación de códigos fuentes en diversos lenguajes (siempre y cuando éstos provean compatibilidad con el MGS). Posterior-

Figura 4.1: Creación de SMA.

mente se genera el código fuente en C++ de los agentes del SMA que contiene los aspectos de definición y especificación de los método que permiten implantar los agentes. Es decir, que para realizar la instanciación de los agentes es necesario completar con código fuente, los archivos generados por EDISMA que poseen la especificación de los métodos (archivo cuerpo) y los archivos que ejecutan a cada agente (archivo main), ya que el código generado consta de declaraciones de clases con sus métodos y atributos, de forma que el usuario/programador pueda rellenar los espacios en blanco proporcionando la funcionalidad de las operaciones de cada uno de los agentes, disminuyendo de tal forma a este el tiempo de desarrollo del SMA. Cabe destacar que el código generado por EDISMA esta adaptado unicamente a la biblioteca del MGS contruido en [11]. En la sección 5.4 se describe mediante un caso de estudio, las líneas de código que deben ser completadas y modificadas.

4.2.1.2. Subsistemas del Sistema de Negocios

Para identificar los subsistemas del negocio se construyó la arquitectura de EDISMA, que describe la estructura y la organización de los componentes, sus propiedades, y la conexión entre ellos (ver Figura 4.2), a continuación se describen cada uno de los componentes de EDISMA.

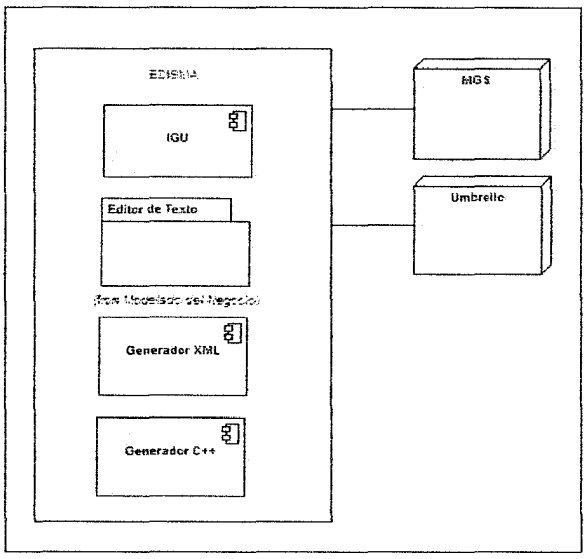


Figura 4.2: Arquitectura de EDISMA.

Interfaz Gráfica de Usuario (IGU)

Es el componente que provee una IGU que permite, inicialmente, recibir parámetros necesarios para la construcción de los agentes que forman parte de un SMA. La obtención de los datos se lleva a cabo mediante formularios que el usuario podrá llenar, suministrando secuencialmente los elementos necesarios para la construcción de los modelos de MASINA, en este caso en el siguiente orden: Modelo de Agente, Modelo de Tareas, Modelos de Comunicación y Modelo de Coordinación. Además, este componente permite invocar la inicialización de la aplicación Umbrello para generar los diagramas de secuencia y de actividades empleados en el modelo de comunicación y el modelo de tareas, respectivamente. Este componente representa de modo gráfico los siguientes elementos:

- Sistema Multiagente: viene representado por el espacio gráfico en donde es posible generar los distintos modelos.
- Agentes: se representan mediante una figura geométrica la cual permite la configuración de sus elementos gráficos.
- Relaciones entre los agentes: cuando se genera un modelo de conversación, se genera una línea que une a los agentes participantes.
- Acciones: se define como acción cualquier evento realizado en el espacio gráfico, dichas acciones son especificadas por orden de creación, dentro del entorno gráfico.
- Archivos: son los elementos generados por los componentes que se describen a continuación, y éstos pueden ser visualizados mediante un árbol de gestor de archivos integrado en el entorno gráfico.

Generador de XML

Extensible Markup Language (XML), es un sencillo formato basado en texto para la representación de información estructurada: documentos, datos, libros, transacciones, entre

otros. Fue derivado de un formato antiguo estándar llamado SGML (ISO 8879), con el fin de ser más adecuado para uso Web [47]. Algunas de las ventajas que ofrece XML son:

- Es extensible, una vez diseñado, es posible extender XML agregando nuevas etiquetas, de modo que se pueda continuar utilizando sin que ésto sea un obstáculo. Por ejemplo, si al modelo de agente en MASINA se incorporan nuevos parámetros, en la aplicación deben llevarse a cabo los cambios que esto implique y ésto puede incorporarse sin mayor problema.
- El analizador es un componente estándar, para cualquiera de las versiones.
- Facilita la reutilización ya que un archivo creado en XML, posee una estructura bien definida que permite procesar de un modo sencillo los elementos que en él se encuentran.
- Permite comunicar aplicaciones de distintas plataformas, sin que importe el origen de los datos.

Una vez obtenidos los parámetros, mediante la IGU, el Generador de XML, crea archivos extensión XML para cada uno de los modelos de especificados en MASINA y además un archivo índice para el SMA, el cual contiene asociado cada uno de los modelos indicados anteriormente, y es de gran utilidad ya que contiene todos los elementos que conforman el SMA y se utiliza para la verificación y consistencia del SMA. Se obtiene así un código intermedio que puede ser usado posteriormente como insumos, para diversas funcionalidades tales como verificaciones de modelos y creación de códigos fuentes en diversos lenguajes (siempre y cuando éstos provean compatibilidad con el MGS).

Editor de texto

Es el componente que permite crear y modificar archivos que serán generados por EDISMA, compuestos únicamente por texto sin formato, conocidos comúnmente como archivos de texto o texto plano. Este componente además de realizar las operaciones básicas, de edición que poseen editores de textos, tales como emacs y el block de notas [26,38], tambien permite darle formato a los archivos que contienen etiquetas especiales.

Generador de C++

El componente Generador C++ recibe como insumo los archivos XML, creados con el componente Generador de XML. Los archivos obtenidos en este componente implementan la estructura y los elementos que permiten integrarlos en el MGS, para cada uno de los agentes que forman parte del SMA, los archivos generados por este componente son:

- Archivo cabecera: es un archivo extensión .h y está conformado por una clase compuesta de atributos y procedimientos (solo las declaraciones) que representa al agente.
- Archivo del cuerpo: es un archivo extensión .cpp que está conformado por la implementación de cada uno de los procedimientos declarados en el archivo cabecera.
- Archivo principal: es un archivo extensión .cpp que está conformado por la implementación del archivo principal de cada uno de los agentes.

Es importante destacar que, tal y como se especificó en el alcance, es necesario que el usuario o programador culmine la implementación de los mismos, de manera particular lo referido a los métodos que implementan las tareas específicas de cada agente.

4.2.1.3. Diagrama de Objetivos

En la Figura 4.3, se especifica el objetivo general que es construir un IDE que permita crear SMA de manera genérica y a través de una IGU , y sus objetivos especifícos.

Cada uno de estos objetivos será logrado, con la ejecución de los procesos correspondientes que permiten obtenerlos, descritos a continuación:

- Desarrollar una IGU que provea una interfaz para modelas los agentes con MASINA , para ellos es necesario especificar inicialmente cada uno de los parámetros o elementos que conforman los modelos que forman para de MASINA.
- Implantar un componente que especifique los modelos de MASINA, para ellos es necesario tener previamente un componente que permite capturar los datos de la metodología para que estos puedan ser documentados.

- Implantar un componente que genera la estructura del sistema multiagente, se llevara cabo inicialmente estudiando la biblioteca del MGS que posee elementos que implantan aspectos de definición, comunicación y coordinación para armar el esqueleto de los agentes.

- Reutilizar un componente de editor de textos, para ellos se realizara una búsqueda de aquellos componentes que permitan reutilizarse para adaptarse a EDISMA.

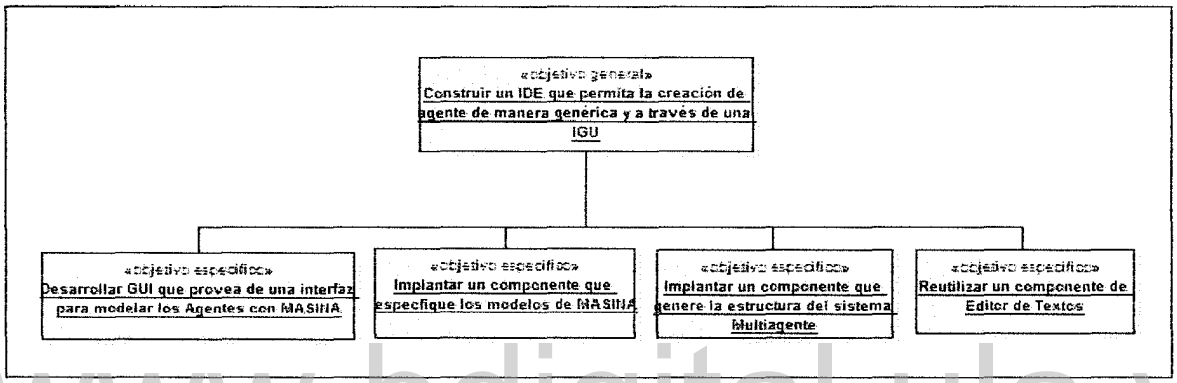


Figura 4.3: Objetivos general de EDISMA.

4.2.2. Modelo de Procesos del Negocio

En este modelo se representa el conjunto de procesos que se realizarán para la creación de EDISMA, y que conducen al logro de los objetivos del mismo. Como punto de partida se define la cadena de valor del Sistema de Negocios, la cual agrupa los procesos del negocio en dos grandes categorías: los procesos primarios y los procesos de apoyo.

4.2.2.1. Cadena de Valor

La cadena de valor de los procesos de EDISMA se puede observar en la Figura 4.4

4.2.2.2. Jerarquía de procesos

El proceso fundamental es que EDISMA permite la creación de agentes de manera genérica, a través de una IGU, para ello es necesario especificar los siguientes subprocesos.

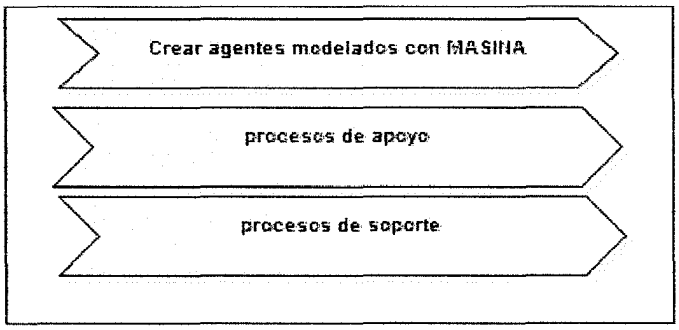


Figura 4.4: Cadena de valor de EDISMA.

- **P1**: Crear agentes modelados con MASINA.
  - **P1-1:** Especificar los datos de los Modelos de agente, tareas, conversación y comunicación del SMA.
  - **P1-2:** Crear la estructura del SMA para instanciarlos en el MGS.
  - **P1-3:** Editar los archivos que implementan el SMA.
- **P2:** Crear un componente que provea un editor de textos.
- **P3:** Crear un componente que analize diagramas UML de actividades y de secuencia.

Una vez ejecutados cada uno de los procesos se obtienen productos descritos en la sección 4.2.6.

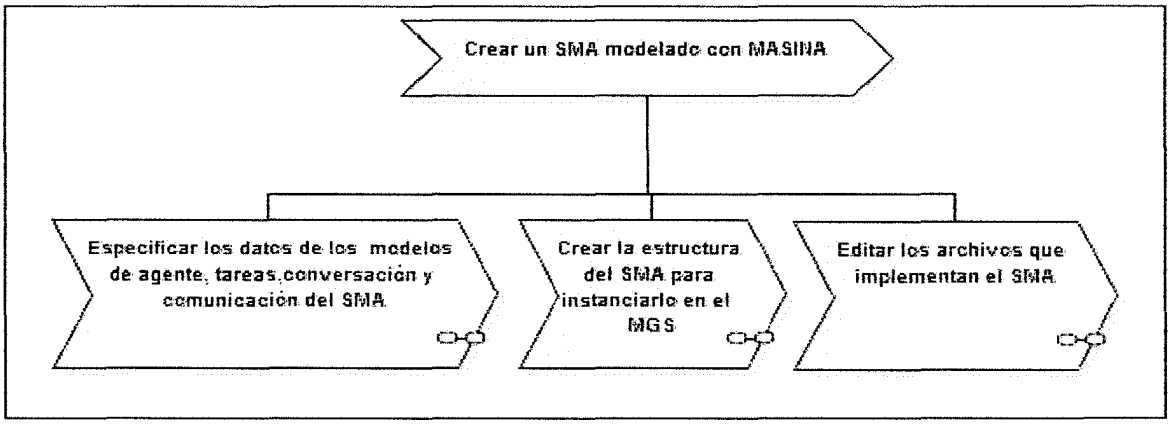


Figura 4.5: Diagrama de Proceso P1.

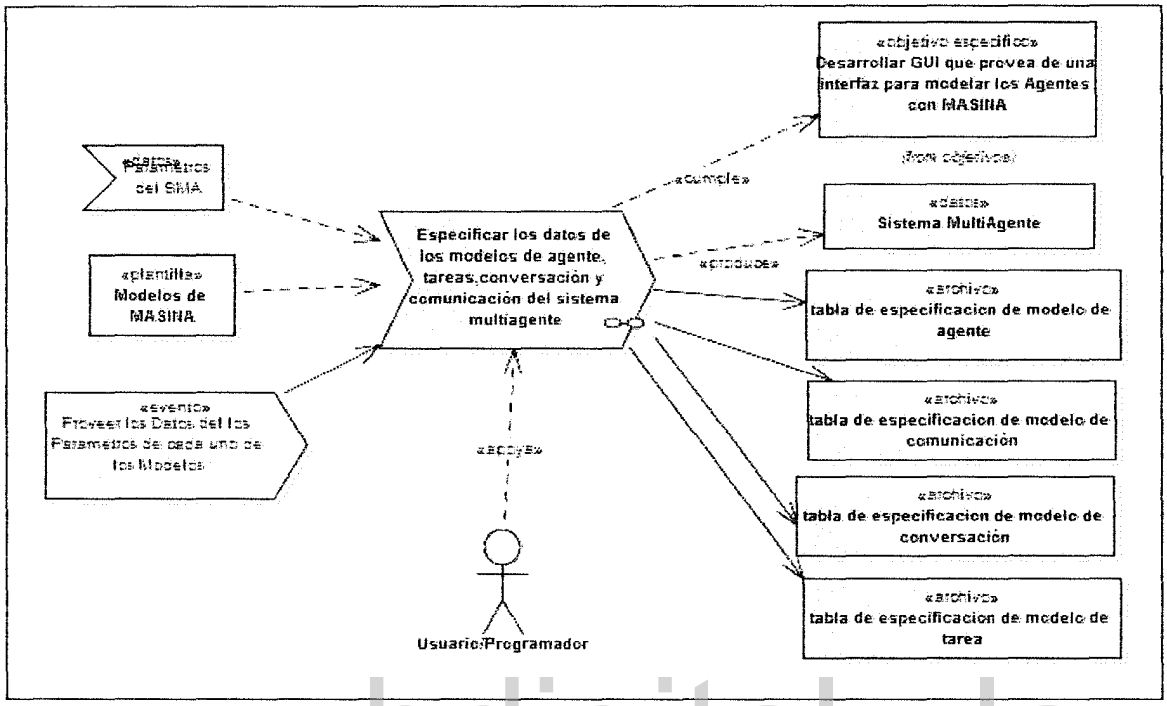


Figura 4.6: Diagrama de Proceso P1-1.

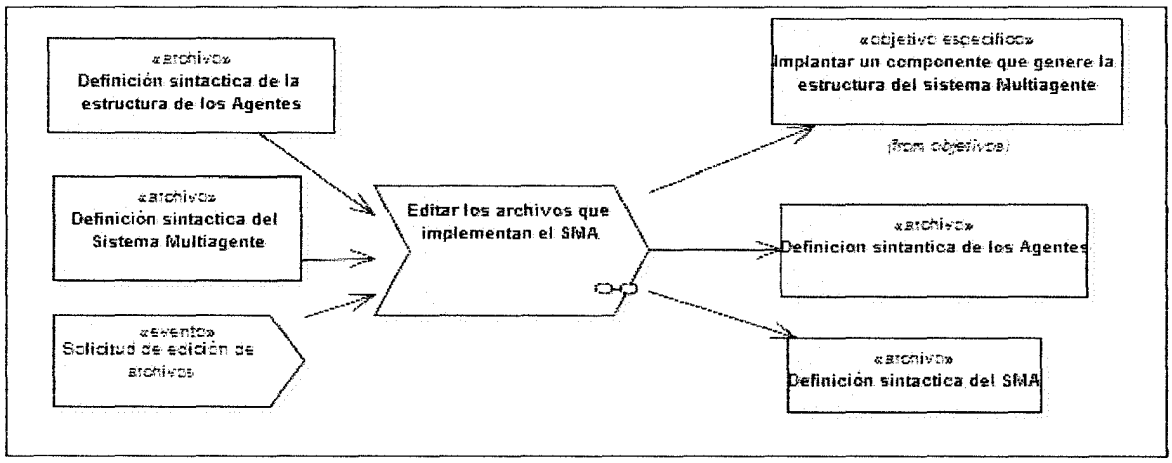


Figura 4.7: Diagrama de proceso P1-3.

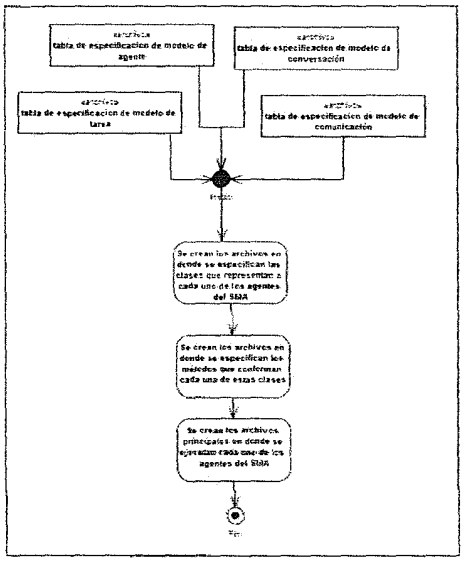


Figura 4.8: Diagrama de actividades del Proceso P1-2.

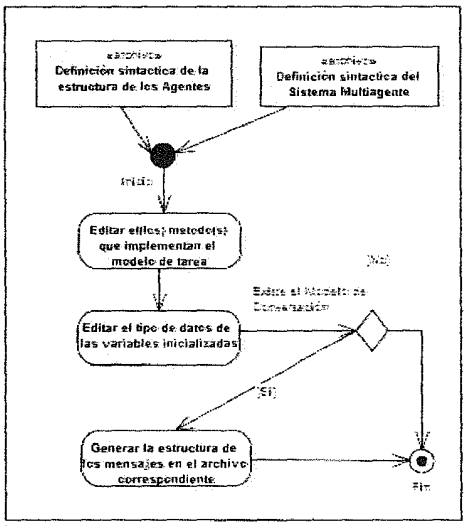


Figura 4.9: Diagrama de actividades del Proceso P1-3.

4.2.2.3. Descripción de Procesos

En esta etapa se especifican y validan los procesos anteriormente expuestos en la jerarquía de procesos, para ello se realizan los diagramas de procesos y de actividades para éstos.

P1: Proceso Crear Agentes Modelados con Masina

Este proceso se divide en tres subprocesos (Ver Figura 4.5):

- **P1-1**: Especificar los datos de los modelos de agente, tareas, conversación y comunicación del SMA, el diagrama de procesos se muestra en Figura 4.6, el cual es iniciado por un evento denominado "Proveer los datos de los parámetros de cada uno de los modelos", esto se refiere a que el usuario debe proporcionar los datos que conforman cada uno de los modelos de MASINA. Por ejemplo en el modelo de agente se deben proporcionar una serie de parámetros especificados en la Plantilla del modelo de agente de MASINA ver Tabla2.1. El diagrama de de actividades 4.8.

- **P1-2**:Crear la estructura de los agentes para instanciarlos sobre el MGS (ver Figura 4.10) y para el diagrama de actividades (ver Figura 4.11). Para llevar a cabo la ins-

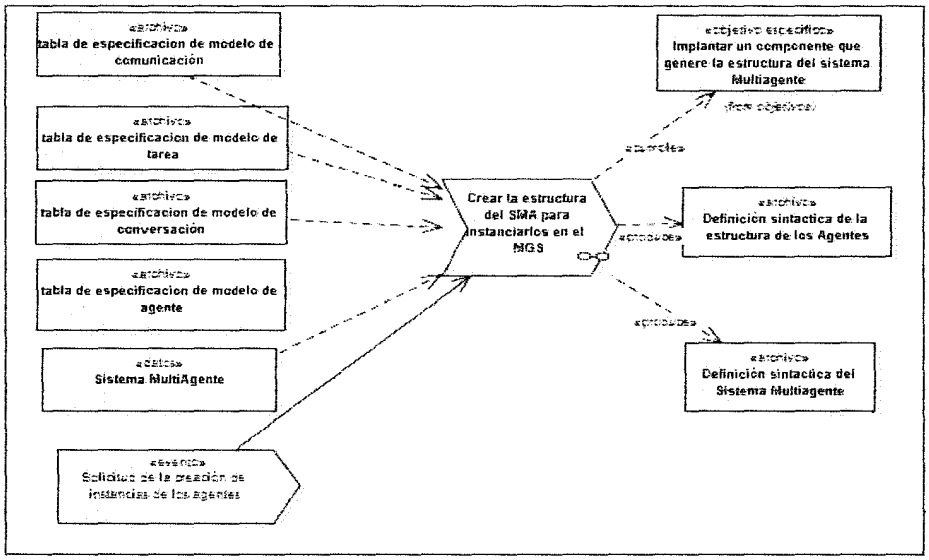


Figura 4.10: Diagrama de proceso P1-2.

tanciación es necesario llevar a cabo previamente la ejecución de los agentes, lo cual implica que debe realizarse un proceso de compilación que generan las instancias o codigo ejecutable, que es ejecutado sobre el MGS.

- **P1-3:**: Editar los archivos que implementan el SMA (ver Figura 4.7 y para el diagrama de actividades ver Figura 4.9).

Los diagramas de actividades del P2 y del P3 se encuentran en el Anexo digital. Es importante descatar que en el P3 se hace referencia a crear un componente que analize diagramas UML de actividades y de secuencia, el cual permite obtener parámetros que forman parte del modelo de conversación y de comunicación de MASINA.

4.2.3. Modelado de actores

Los únicos actores que interactuarán con EDISMA seran los usuarios ó programadores que desean crear agentes con la herramienta. Estos actores deben poseer conocimientos de la definición de un agente, además de conocer la metodología MASINA y tener dominio en C++ para poder realizar la implementación de los agentes.

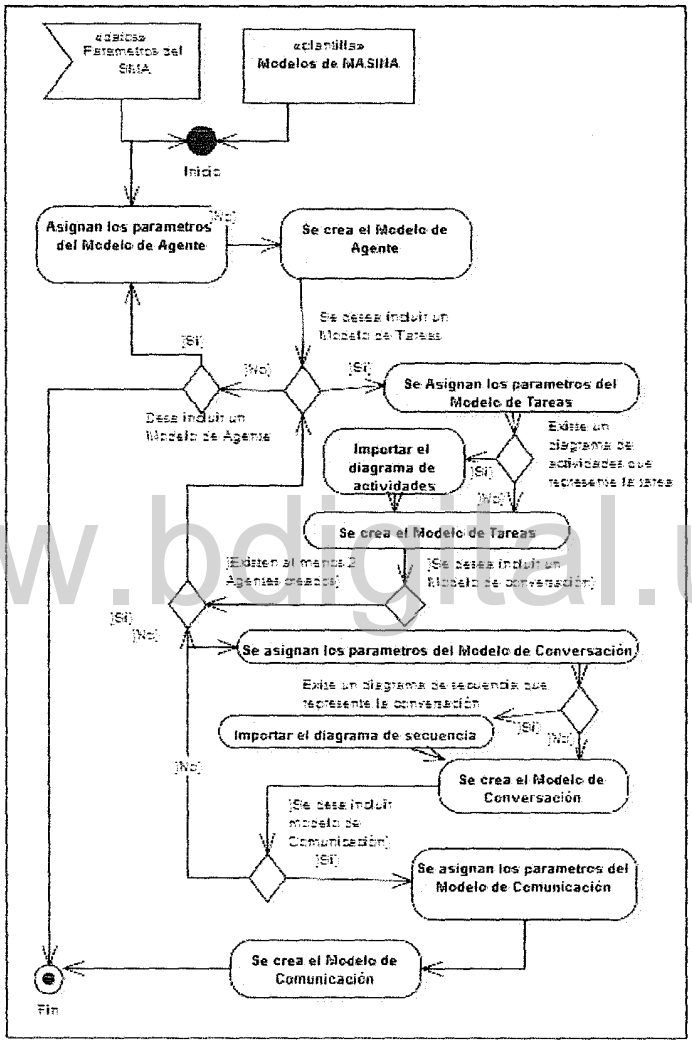


Figura 4.11: Diagrama de actividades del Proceso P1-1.

4.2.4. Modelo de objetos de negocio

Los objetos que forman parte de EDISMA vienen representados por cada uno de los modelos expuestos en MASINA, excepto por el modelo de inteligencia, siendo entonces los modelos de agente, tareas, conversación y comunicación. Cada uno de estos modelos, a su vez están conformados por otros objetos, todos éstos se muestran en la Figura 4.12, descritos conceptualmente en la sección 2.2. En este modelo, se incluyen los archivos que permiten implementar un SMA, descritos en la sección 4.2.1.

Además se especifica un objeto denominado disparador, que tiene la función de iniciar el proceso de instanciación de los agentes que forman parte del SMA. Debido a que es un elemento de software que crea a los agentes del SMA, es necesario especificar un diseño del mismo, adaptandolo a MASINA, el cual se encuentra en el anexo.

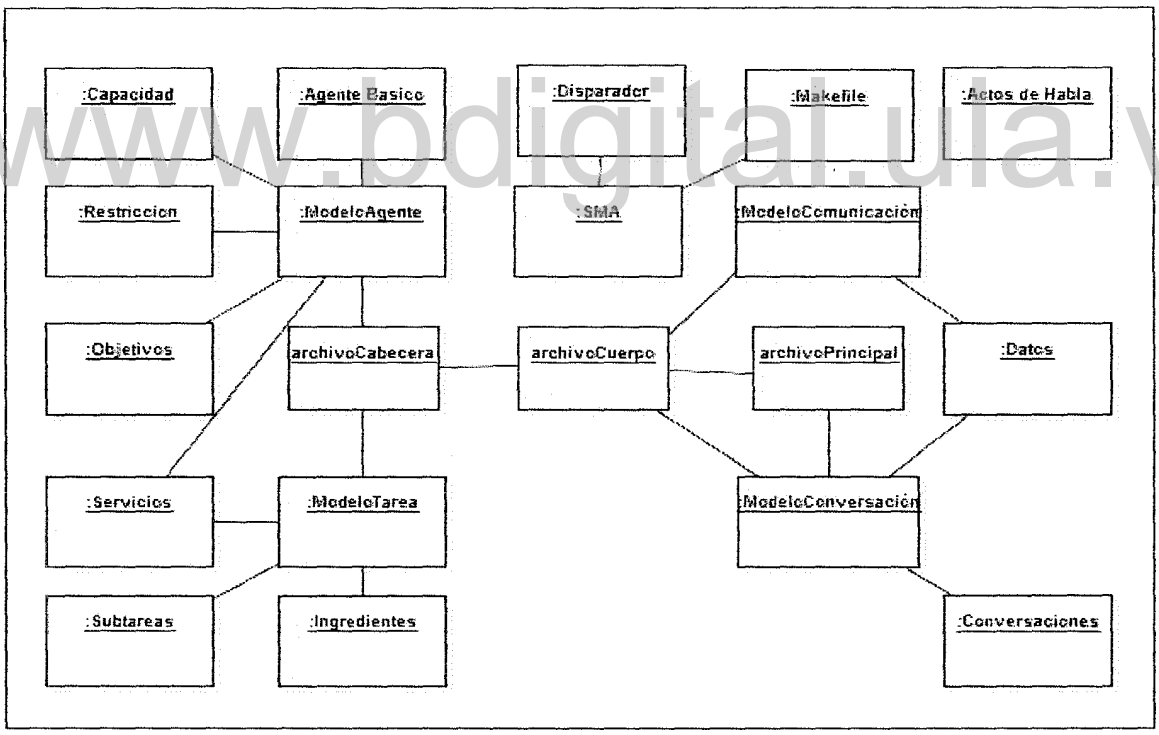


Figura 4.12: Diagrama de Objetos de EDISMA.

4.2.5. Identificación de reglas de negocio

Las reglas que debe cumplir EDISMA para su adecuado funcionamiento son:

1. La estructura de los agentes está fundamentada en la metodología MASINA.
2. Los archivos que implementan a los agentes hacen uso del MGS creado en [11], el cual está desarrollado en C++ por lo cual se utilizará el mismo lenguaje en esta primera versión.
3. Los agentes se comunican con procesos del MGS en cada nodo.

4.2.6. Modelado de Eventos

Los eventos del negocio son hechos cuya ocurrencia dispara la ejecución inmediata de un conjunto de acciones asociadas a los procesos del negocio. Los eventos se representan mediante un diagrama de eventos, que puede observarse en la Figura 4.13, en el cual se identifican y especifican las causas, fuentes ó insumos de origen, así como sus efectos o impactos en objetos y procesos del negocio.

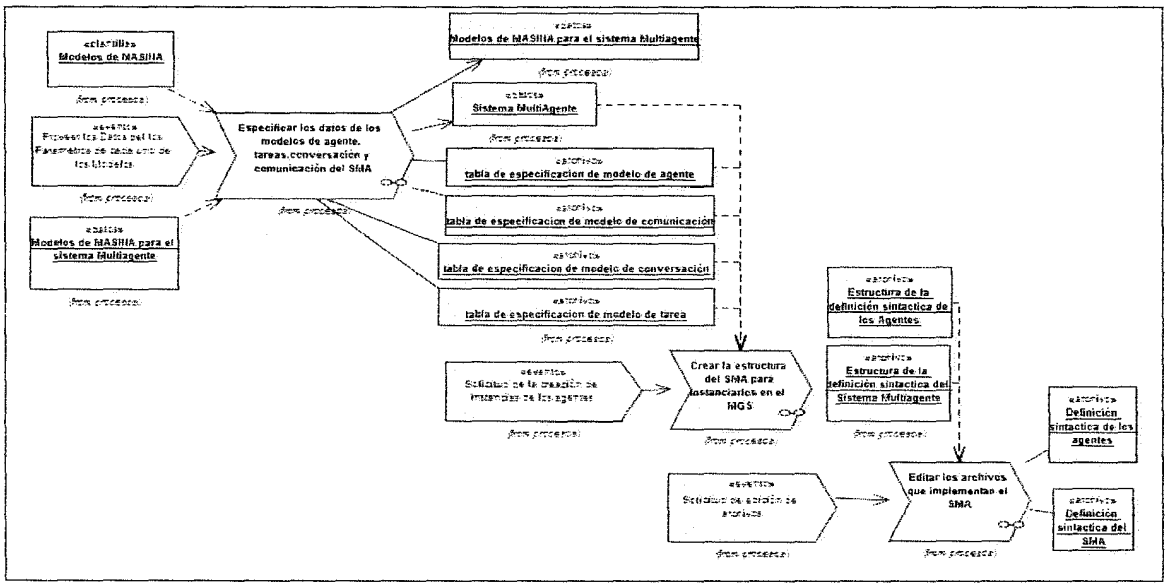


Figura 4.13: Diagrama de Eventos de EDISMA.

4.2.6.1. Matriz de Eventos vs. Procesos

La matriz de eventos vs procesos puede observarse en la Tabla 4.1, allí se utilizan las siguientes abreviaturas:

- **P1:** Especificar los datos de los Modelos de agente, tareas, conversación y comunicación del SMA.
- **P2:** Crear la estructura del SMA para instanciarlos en el MGS.
- **P3:** Editar los archivos que implementan el SMA.

Evento/Proceso	P1	P2	P3
Proveer los datos de cada uno de los párametros de los Modelo de Masina	X		
Solicitud de la creación de instancias de los agentes		X	
Solicitud de edición de archivos			X

Tabla 4.1: Matriz de Eventos vs. Procesos

4.2.6.2. Matriz Objetos vs. Procesos

La Tabla 4.2 muestra la matriz de Objetos de Negocio vs. Procesos de Negocio, que indica los objetos de negocio que son utilizados como recursos en cada uno de los procesos principales y de apoyo de alto nivel. Adicionalmente, en dicha matriz se señala cuando un objeto de negocio es creado (C), usado (U) o actualizado (A).

4.3. Ingeniería de Requisitos

En esta sección se realizará el descubrimiento, análisis y especificación de cada uno de los requisitos funcionales y no funcionales que EDISMA deberá reunir para cumplir con los objetivos descritos en el Modelado de Negocio.

4.3.1. Descubrimiento de Requisitos

En esta etapa se elabora una lista preliminar de los requisitos a cumplir:

Objeto	P1	P2	P3
Capacidad	C	U	
Restricción	C	U	
Objetivos	C	U	
Servicios	C	U	
Subtareas	C	U	
Datos	C	U	
Ingredientes	C	U	
ModeloAgente	C	U	U
ModeloTarea	C	U	U
Conversaciones	C	U	
AgenteBásico	C	U	
ModeloConversación	C	U	U
ModeloComunicación	C	U	U
ActoHabla	C	U	
archivoCabecera		C	U
archivoCuerpo		C	U
archivoMain		C	U
makefile		C	U
Disparador		C	U
SMA	C	U	U

Tabla 4.2: Objetos vs. procesos

- Los usuarios ó programadores deben poder crear un SMA a través de una IGU.
- EDISMA debe ser desarrollado con un lenguaje de programación de libre licencia que entre otras bondades permite que tanto el código fuente como los archivos binarios sean modificados.
- No utilizar máquina virtual, debido a que los procesos que éstas ejecutan están limitados por los recursos y abstracciones proporcionados por ellas.
- EDISMA debe permitir crear la estructura de los agentes en C++, haciendo uso de las librerias del MGS .
- Los usuarios ó programadores deben poder importar a EDISMA diagramas de actividades y de secuencia para especificar mediante éstos el modelo de tarea y conversación respectivamente.
- EDISMA debe documentar los Modelos de MASINA de cada uno de los agente que forman parte del SMA.

Para cada uno de los requisitos se utilizará la plantilla de especificación de requisitos Volere [53], la cual está creada para ser utilizada como una base de la especificaciones de requisitos. Para el requisito 01 que establece que los usuarios deben poder crear un SMA a través de una IGU (ver Tabla 4.3), las demás plantillas se detallan en el Anexo digital.

A continuación se describen algunos aspectos de la Planilla:

Grado de satisfacción del interesado, hace referencia al éxito del requisito. Escala del 1 al 5 indicando si fue poca su satisfacción o extremadamente satisfecho, respectivamente.

Prioridad, indica la importancia que posee el requisito por parte del interesado. Escala del 1 al 5 indicando poco o extremadamente importante, respectivamente.

Numero de Requisito	01	Tipo de Requisito	funcional	Caso de Uso/ Evento Relacionado	CU-01
Descripción	Los usuarios ó programadores deben poder crear un SMA a través de una IGU.				
Justificación	Es necesario una IGU que permita realizar la implementación y/o programación de los agentes debido a la carencia de la misma.				
Origen (Interesado)					
Criterio de Aceptación / Validación:	Despliegue de la interfaz necesaria para especificar cada uno de los parámetros del SMA				
Nivel de satisfacción del Interesado:					
Prioridad::	5		Requisitos en Conflicto:		
Material de Soporte					
última Modificación					

Tabla 4.3: PLanilla Volere que especifica el requisito 01

4.3.2. Análisis de Requisitos

En esta etapa se analizan los requisitos descritos anteriormente, para ello se subdividen en: **Requisitos Funcionales**, "los sujetos fundamentales o esenciales que constituyen la médula del producto. Ellos describen lo que el producto tiene que hacer o cuáles acciones de procesamiento debe tomar" [53]. Son aquellos requeridos por EDISMA para cumplir con los objetivos contemplados en el modelado del negocios. **Los Requisitos No funcionales**, "son las propiedades que las funciones deben tener, tales como desempeño y capacidad de uso" [53].

Requisitos Funcionales

- Los usuarios deben poder crear un SMA a través de una IGU.
- EDISMA debe permitir crear la estructura de los agentes en C++, haciendo uso de las librerias del MGS.
- EDISMA debe generar la especificación del SMA en XML.
- Los usuarios deben poder importar diagramas de actividades y de secuencia.
- EDISMA debe tener integrado un editor de textos.

Requisitos No Funcionales

- EDISMA debe documentar los Modelos de MASINA de cada uno de los agente que forman parte del SMA.
- EDISMA debe ser desarrollada con QT.

4.3.3. Especificación de Requisitos

Para especificar cada uno de los requisitos funcionales, se realizan los diagramas de caso de uso, con sus descripciones textuales, se inicia con el el caso de uso general de EDISMA, el cual se observa en la Figura 4.14, y luego se obtienen una serie de caso de usos, descritos a continuación:

- **CU-01:** Generar un SMA a través de una herramienta IGU(ver Figura 4.15 y descripción textual en la Tabla 4.4).
- **CU-02:** Crear la estructura de los agentes en C++ haciendo uso de la librerias del MGS.
- **CU-03:** Generar especificación de la estructura del agente.
- **CU-04:** Crear código fuente a partir de diagramas de actividades y de secuencia.
- **CU-05:** Crear los modelos del SMA en formato XML.

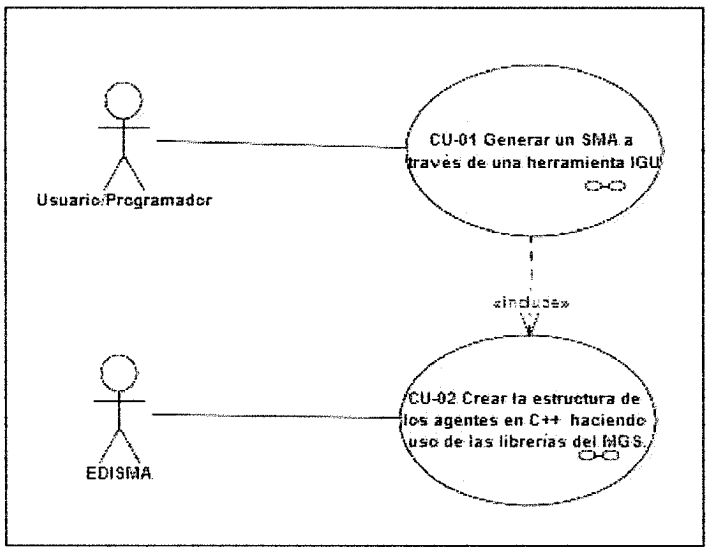


Figura 4.14: Caso de uso principal EDISMA.

En está sección solo se describe a modo de ejemplo el CU-01, los demas se encuentran especificados en el anexo.

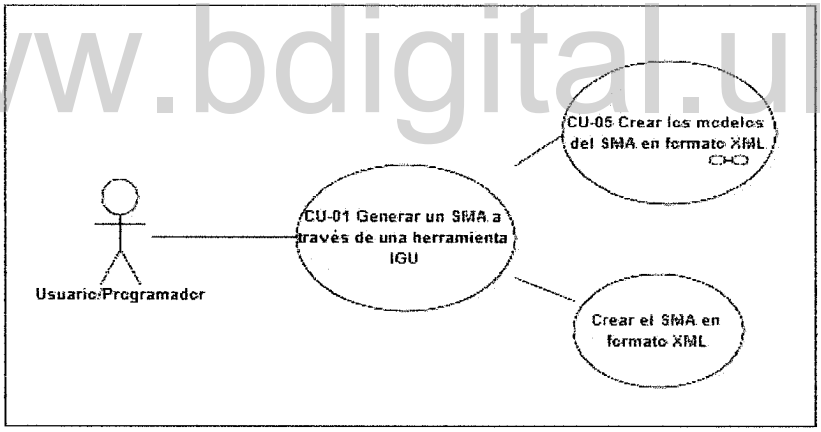


Figura 4.15: CU-01.

4.4. Diseño de *software*

En esta sección se describen los procesos de diseño necesarios para modelar los requisitos establecidos en la sección de ingeniería de requisitos. Este proceso es llevado a cabo en las siguientes actividades: definición de la estructura inicial de la aplicación, diseño de la interfaz usuario/sistema, diseño de la base de datos y diseño de componentes.

Caso de Uso	CU-01		
Versión	1	Fecha	15/11/2012
Autor	Paola Rivero		
Fuente			
Objetivo	Generar SMA a través de una herramienta IGU		
Precondición	Definición de los Modelos de los agentes en MASINA		
Secuencia Normal			
1	El usuario crea el entorno del SMA a través de la IGU		
2	El usuario solicita la creación de un Agente		
3	El usuario proporciona cada uno de los parámetros necesarios para generar el Modelo de Agente		
4	Si se proporcionaron los parámetros necesarios para la creación del Modelo de agente este es creado. De lo contrario no es creado Si se proporcionan los parámetros necesarios para la creación del Modelo de Tareas este es creado. De lo contrario no es creado el Modelo de Tareas Si existen más de dos agentes. Si se proporcionan los parámetros necesarios para la creación del Modelo de Conversación este es creado. De lo contrario no es creado. Si se proporcionan los parámetros necesarios para la creación del Modelo de Comunicación este es creado. De lo contrario no es creado.		

Tabla 4.4: CU-01

4.4.1. Definición de la estructura inicial de la aplicación

Este proceso es llevado a cabo con las actividades que se especifican en la Figura 4.16

4.4.1.1. Definición de metas de diseño y arquitectura de software

En el subproceso de definición de metas de diseño se realiza un nuevo análisis a los requisitos funcionales y no funcionales, especificados en la sección anterior, y se decide de acuerdo a éstos, el estilo arquitectónico que va a poseer EDISMA.

La arquitectura de software que se decidió para representar estos requisitos es el estilo basado en componentes (ver Figura 4.2) , y a su vez cada uno de los componentes posee una arquitectura de 3 capas: Modelo Vista Controlador (MVC), compuesto por la capa de presentación, de control y de datos. Este tipo de arquitectura fue escogida debido a que los componentes de EDISMA deben tener una base de datos para almacenar los distintos objetos que lo conforman, métodos que ejecutan la lógica de la aplicación e interfaces gráficas que

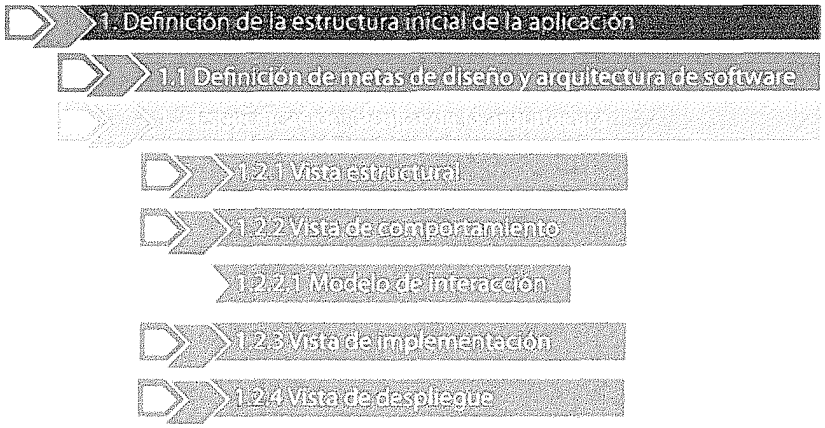


Figura 4.16: Esquema de actividades de la definición de la estructura inicial de la aplicación.

muestran ó capturan información. Además, este tipo de arquitectura satisface los requisitos no funcionales que expresan que el sistema debe ser robusto, confiable, eficiente; además de las otras características de calidad que EDISMA debe poseer.

Cada una de las tres capas que conforman este estilo arquitectural de los componentes de EDISMA tiene una función específica que se define a continuación:

- Capa de presentación: es la capa que presenta un formato adecuado para interactuar, usualmente la interfaz de usuario.
- Capa de control: es a donde llega el flujo de información recibida, ésta es procesada de acuerdo al tipo, obteniéndose la salida generada por diversos procedimientos programados para que EDISMA funcione de un modo adecuado.
- Capa de datos: en esta capa se encuentran elementos como :
  - Archivos que conforman un proyecto ó SMA.
  - Modelos de MASINA para cada uno de los agentes, así como los atributos de cada uno de éstos.

4.4.1.2. Especificación técnica de la arquitectura

Las descripciones arquitectónica de EDISMA se componen de múltiples vistas, en ellas se especifican diferentes aspectos que se requieren modelar, tales como: aspectos , dinámicos,

estructurales, implementación y de despliegue

4.4.1.3. Vista estructural

La vista estructural forma parte de la especificación técnica de la arquitectura, está compuesta por el conjunto de clases definidas para cada uno de los componentes que forman parte de la arquitectura de EDISMA. En la Figura 4.17 puede observarse el diagrama de clases de EDISMA.

4.4.1.4. Vista de comportamiento

Este proceso forma parte de la especificación técnica de la arquitectura, y se lleva a cabo organizando las clases de acuerdo al diseño arquitectónico de los componentes que conforman a EDISMA el cual es MVC. Las clases se organizan en entidad, borde y control, las cuales serán representadas mediante los estereotipos correspondientes, a continuación se realiza una descripción de cada uno de los tipos de clases (ver Figura 4.18):

- **Estereotipo entidad (*entity*)**: es empleado para los objetos que almacenan información respecto al estado interno de EDISMA, ó de los elementos que ella genera. Las clases perteneciente a este estereotipo son:

  - ***GeneradorMA ,GeneradorMT, GeneradorCon y GeneradorCom***: generan archivos extensión XML para documentar cada uno de los atributos de los Modelos de: Agente, Tareas, Conversación y Comunicación, respectivamente.
  - **GeneradorSMAXML**: genera un archivo extensión XML que contiene cada uno de los agentes que forman parte del SMA y los modelos que forman parte de los mismos.
  - **GeneradorSMA**: es la encargada de generar los archivos que permiten implementar cada uno de los agentes, recibe como insumos los modelos en extensión XML, que conforman al agente, y el archivo generado por la clase GeneradorSMAXML .

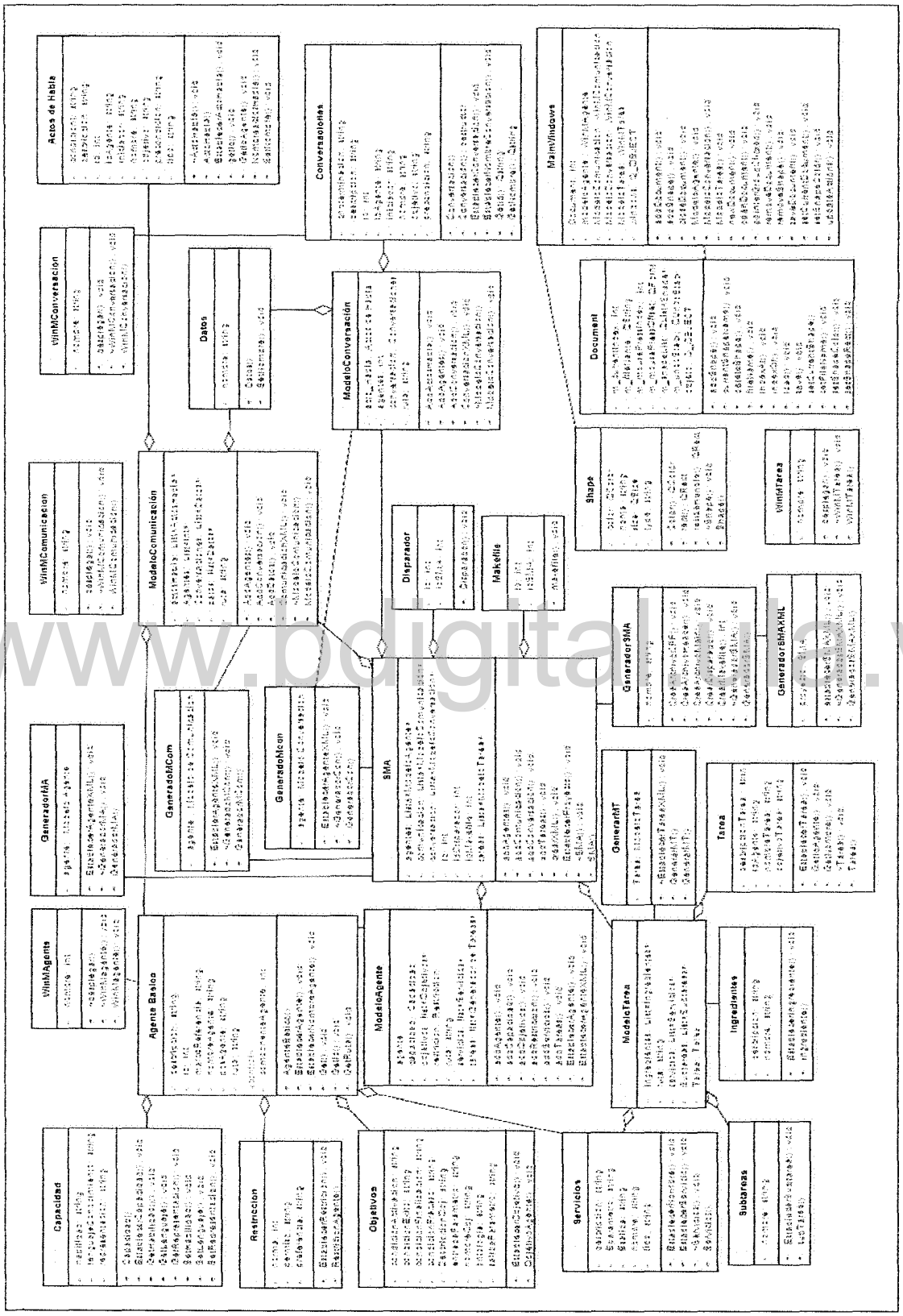


Figura 4.17: Vista estructural.

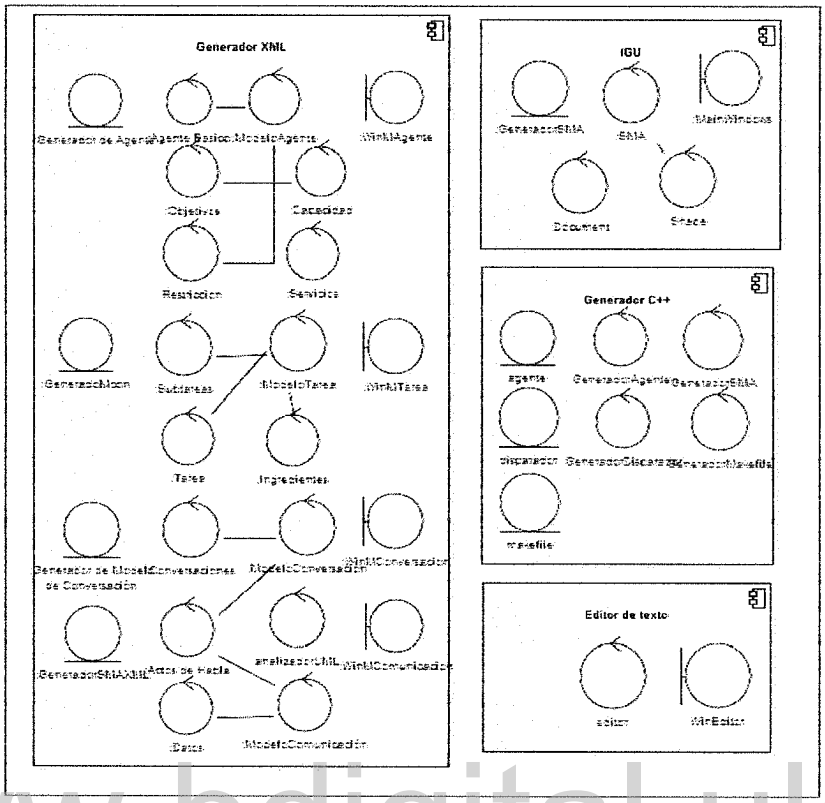


Figura 4.18: Vista de comportamiento de EDISMA.

- **Estereotipo borde(*boundary*)**: es utilizado para objetos que implementan las interfaces del sistema con el mundo externo. Las clases pertenecientes a este estereotipo son:

  - **MainWindows**: es la clase que contiene los aspectos gráficos de la pantalla principal de EDISMA, a partir de la instanciación de ella es posible invocar el resto de las clases del tipo *boundary*.

  - **Las clases WinMagente, WinMTarea, WinMcomunicacion, WinMConversacion**: permiten mostrar al usuario/programador los elementos que debe proporcionar para conformar los modelos de: agente, tareas, conversación y comunicación respectivamente, asociados a un agente especifíco.

- **Estereotipo control (*control*)**: se usa para objetos que implementan el comporta-

miento o control de la lógica de los casos de uso. Los objetos del tipo control modelan las funcionalidades de EDISMA, y éstas son:

- **Capacidad, Restricción, Objetivos, Subtareas, Servicios, AgenteBasico**: estas clases reciben parámetros de la clase WinMAgente, y permiten generar objetos que serán utilizados por la clase ModeloAgente.
- **ModeloAgente**: es la clase encargada de conformar el modelo de agente y suministrar los parámetros necesarios a la clase GeneradorMA. Esta clase recibe los datos de las clases Capacidad, Restricción, Objetivos, Subtareas, AgenteBasico.
- **Ingredientes y Tarea**: estas clases reciben parámetros de la clase WinMTarea y permiten generar objetos que serán utilizados por la clase ModeloTareas.
- **ModeloTareas**: es la clase encargada de conformar el Modelo de Tarea de un determinado agente y suministrar los parámetros necesarios a la clase GeneradorMT. Esta clase recibe los datos de las clases Tarea e Ingredientes.
- **Conversaciones**: esta clase recibe parámetros de la clase WinMConversacion y permiten generar objetos que serán utilizados por la clase Modeloconversacion.
- **Modeloconversacion**: es la clase encargada de conformar el Modelo de Conversación de un determinado agente y suministrar los parámetros necesarios a la clase GeneradorConversacion. Esta clase recibe los datos de la clase Conversaciones.
- **Datos y Servicios**: estas clases reciben parámetros de la clase WinMComunicacion y permiten generar objetos que serán utilizados por la clase ModeloComunicacion.
- **ModeloComunicacion**: es la clase encargada de conformar el Modelo de Comunicación de un determinado agente y suministrar los parámetros necesarios a la clase GeneradorCon. Esta clase recibe los datos de las clases Datos y Servicios.
- **SMA**: esta clase recibe parámetros de las clases WinMAgente, WinMTarea, WinM-

Conversacion, WinMComunicacion y MainWindows y permite enviar los parámetros necesarios a la clase GeneradorSMA y GeneradorSMAXML.

- **Document**: es la clase encargada de generar el espacio ó entorno gráfico donde se encuentran representados los agentes que conforman el SMA.
- **Shape**: permite gestionar los elementos gráficos que representan los agentes y sus componentes en el entorno gráfico, instanciado por la clase Document.

4.4.1.5. Modelo de interacción

Esta sección pertenece a la vista de comportamiento y esta conformada por los diagramas de secuencia que describen la dinámica o comportamiento de EDISMA. Estos diagramas permiten modelar la interacción entre los actores y el sistema, y ésta es representada por el pase de mensajes entre objetos, es decir por la activación de los métodos o procedimientos.

En la Figura 4.19 se muestra el diagrama de secuencia "crear un Modelo de Agente", en donde se puede observar cómo es la interacción entre las clases. Inicialmente el usuario realiza una petición de creación de un agente, dicha petición es realizada a la clase MainWindows, y está, luego se comunica con WinMAgente, que proporciona la IGU necesaria para recibir los parámetros de entrada que conforman el modelo de agente. Luego se hace una llamada a la clase ModeloAgente, que realiza las gestiones lógicas, y finalmente interviene la clase GeneradorMA, la cual almacena cada uno de los datos generados (en formato XML), para el modelo de agente. Una vez realizadas todas estas acciones, se envía al usuario la notificación correspondiente. El resto de lo diagramas de secuencia se especifícan en el anexo digital.

4.4.1.6. Vista de implementación

Es sección forma parte de la especificación técnica de la arquitectura, y especifica los detalles de la implementación de EDISMA, éstos se refieren a la plataforma, lenguaje y demás elementos estructurales necesarios para la implementación, descritas continuación

- Plataforma de sistema operativo: Windows 7, ya que es el sistema operativo que disponía instalado por defecto el equipo empleado para el desarrollo.

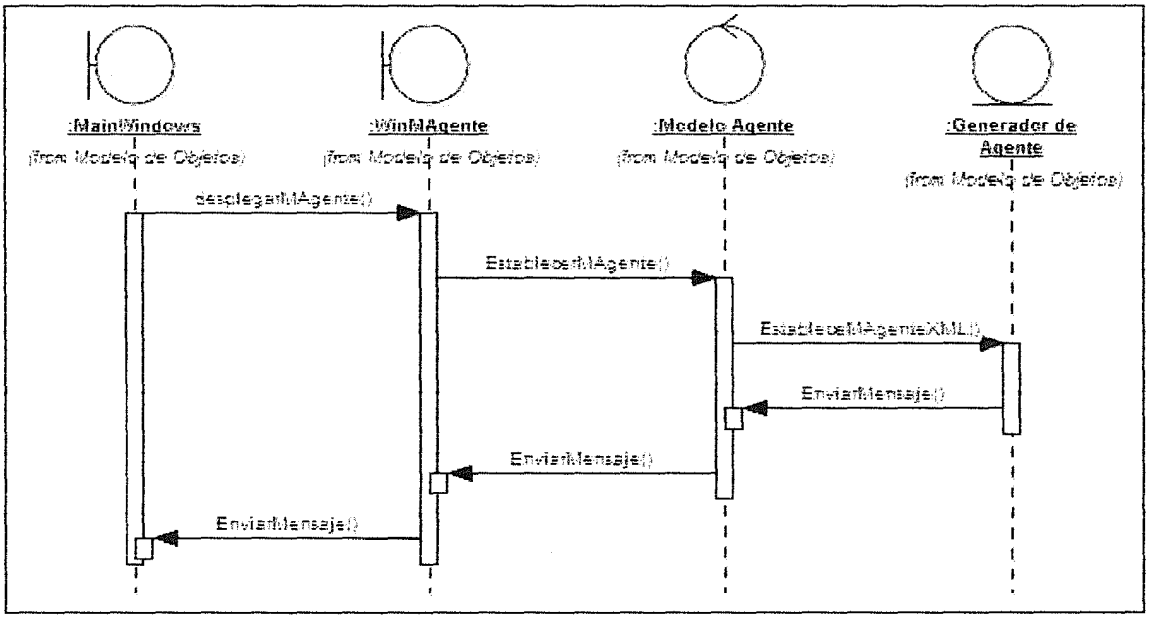


Figura 4.19: Diagrama de Secuencia Crear un Modelo de Agente.

- Lenguaje para el desarrollo: QT4, C++ y el conjunto de bibliotecas que conforman el MGS .
- Herramenta utilizada (No se encuentra integrada a EDISMA) para la creación de Diagramas UML: Umbrello.

4.4.1.7. Vista de despliegue.

En este proceso se especifican los detalles de instalación, despliegue y operación de EDISMA. La vista de despliegue explica cómo los componentes ejecutables (código objeto) y los elementos en tiempo de ejecución se deben instalar en la plataforma de operación. La Figura 4.20 muestra el diagrama de despliegue que describe la arquitectura física del sistema, durante la ejecución, en términos de:

- Nodos: Son objetos físicos que representan algún tipo de recurso computacional, y es donde se ejecutan o despliegan los componentes de software.
- Componentes de despliegue: Son piezas de software que conforman un sistema ejecutable.

Este diagrama permite identificar la plataforma de hardware necesaria para EDISMA y especificar la estructura de los componentes de *software* que la conforman, así como lo nodos (*hardware*) desplegados.

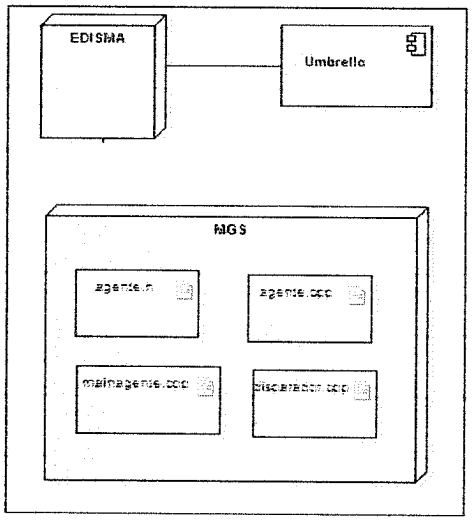
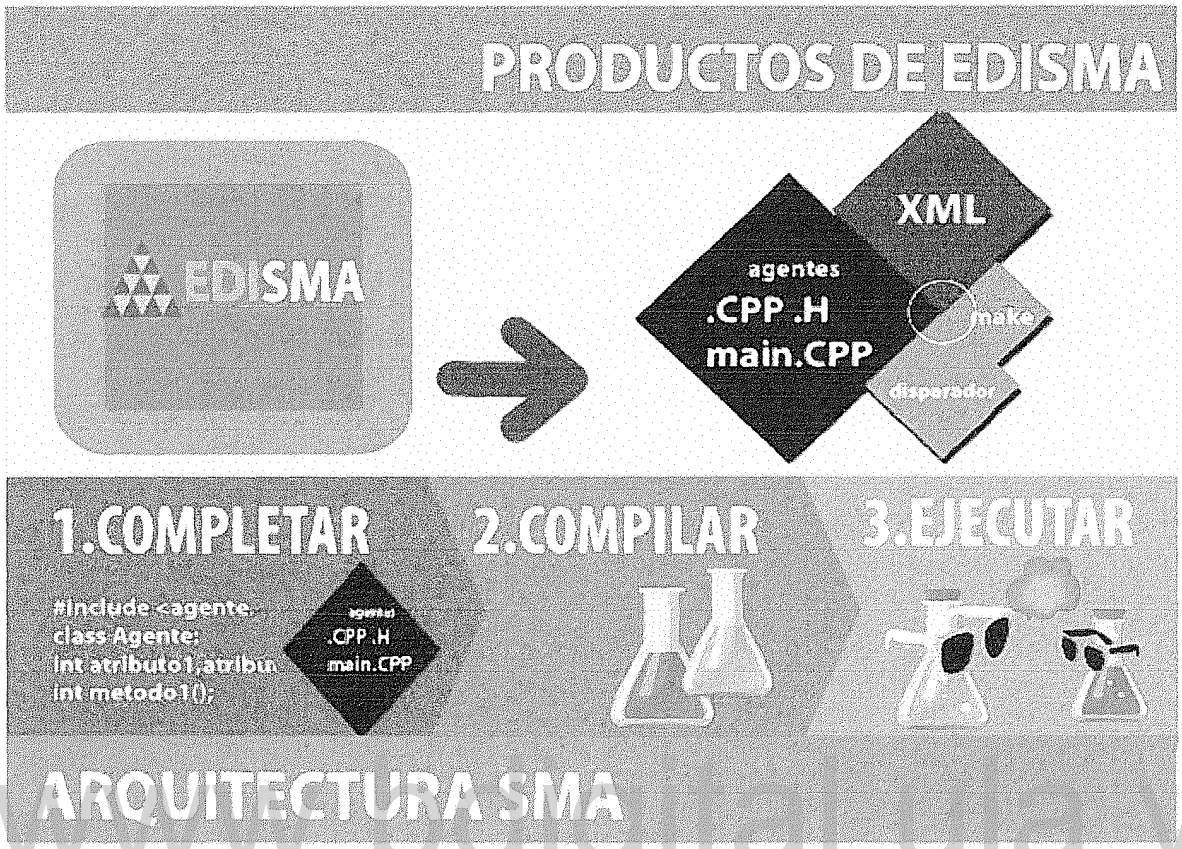


Figura 4.20: Diagrama de Despliegue de EDISMA.

También se realizó una infografía (ver Figura 4.21) que permite visualizar la relación que existe entre productos que genera EDISMA con la arquitectura del SMA. Los elementos que EDISMA genera están conformados por una serie de archivos, que al ser modificados pueden ser compilados utilizando la biblioteca del MGS. Posteriormente pueden ser ejecutados en el programa denominado server, que previamente debe estar instalado y configurado en cada uno de los nodos en los cuales .

4.4.2. Diseño de interfaz usuario/sistema

En esta actividad se establecen las caracteristicas estructurales y estéticas de la IGU de EDISMA para el único perfil que ésta posee para el usuario. También, describe el comportamiento de la interfaz, presentando además las pantallas de captura de datos para cada una de las funcionalidades contempladas en EDISMA.



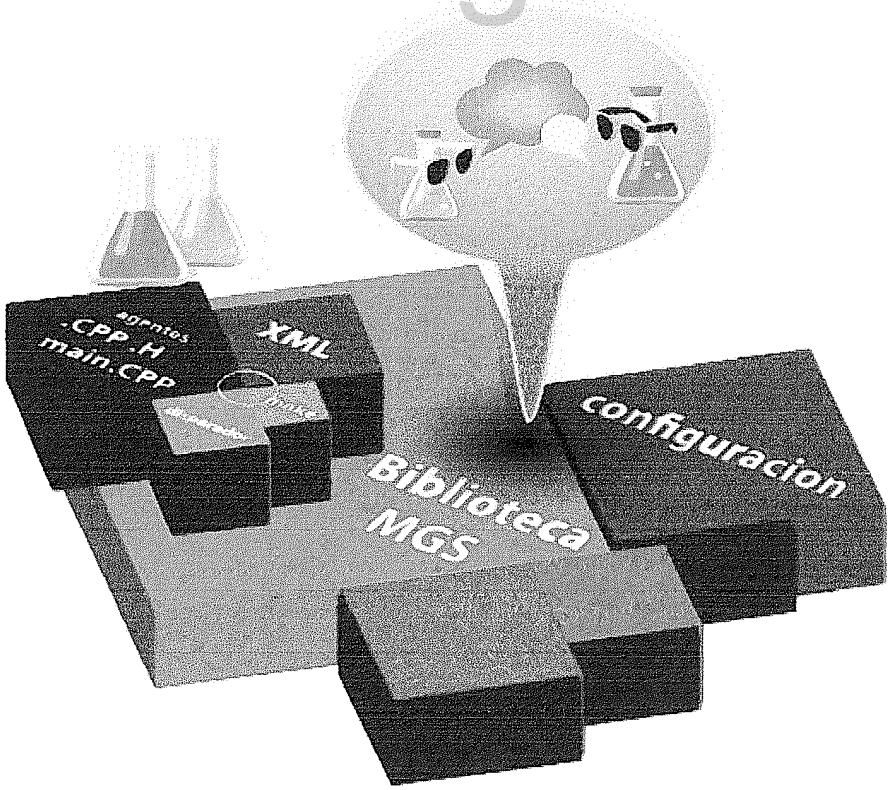


Figura 4.21: Productos de EDISMA y arquitectura del SMA.

4.4.2.1. Elaboración del contenido y servicios de las interfaces

En este subproceso se identifican cada uno de los servicios ó funcionalidades de EDISMA, que podrán ser accedidas mediante la IGU, los cuales son:

- Crear un SMA.
- Crear un agente.
- Crear un modelo de tarea a un agente.
- Crear un modelo de conversación entre agentes.
- Crear un modelo de comunicación entre agentes.
- Abrir un SMA elaborado creado previamente.
- Modificar posición, tamaño y color de los elementos gráficos que representan a los agentes.
- Crear los archivos que permiten la implantación de los agentes en el MGS.
- Editar los archivos creados por EDISMA.

4.4.2.2. Diseño de la interfaz del usuario

En este proceso se especifican los elementos que conforman la interfaz, es decir el modo en que ésta va ser estructurada.

La estructura de la interfaz

La interfaz de EDISMA está conformada por los siguientes elementos (Ver Figura 4.22):

- Área 1 - Menú Principal: es el área que se encuentra en la sección superior de la IGU de EDISMA, y permite llevar a cabo acciones sobre los proyectos.
- Área 2 - Menú Secundiario: es el área que se encuentra debajo de el Área 1 en la sección superior de la IGU de EDISMA, y permite llevar a cabo acciones básicas sobre los proyectos.

- Área 3 - Pila de acciones: es un elemento gráfico en donde se puede observar la serie de acciones llevadas a cabo en el entorno de EDISMA; por ejemplo sí se agrega un agente en dicha pila se mostrará un mensaje indicando la acción.

- Área 4 - Gestor de Archivos: es el área situada en la sección inferior derecha de la IGU, permite visualizar los archivos que se generan de los modelos de MASINA y/ó de la implentación de los agentes.

- Área 5 - Entorno Gráfico: esta conformado por el un espacio que permite realizar los elementos gráficos que conforman al SMA. Esta área se encuentra posicionada en el centro de la IGU, es posible gráficar los agentes y las relaciones entre éstos.

- Área 6 - Barra Lateral: es el área que se encuentra posicionada en la izquierda de la IGU principal y esta compuesta por una serie de iconos que representan las funcionalidades de : crear Agente, crear Modelo de Tarea, crear Modelo de Conversación, crear Modelo de Comunicación, eliminar un agente y manipular el color de los elementos gráficos que representan a los Agentes.

4.4.2.3. Prototipo de la interfaz

La IGU de EDISMA tiene la particularidad de desglosar la información de un modo progresivo, es decir inicialmente el usuario podrá acceder a funcionalidades básicas sobre las gestiones de proyectos, y en caso de iniciar uno nuevo, se presentará la opción de crear un agente, que es el proceso inicial cuando se crea un SMA.

La Figura 4.23 muestra la interfaz de EDISMA, al inicio de su ejecución, puede observarse que sólo se encuentran activas las opciones de generar un nuevo proyecto, abrir un proyecto, salir ó iniciar con la creación de un agente. Las opciones activas están a color y las inactivas en gris.

Al iniciar la creación de un agente se despliega una ventana que se muestra en la Figura 4.24, que permite al usuario proporcionar los elementos que conforman el Modelo de Agente,

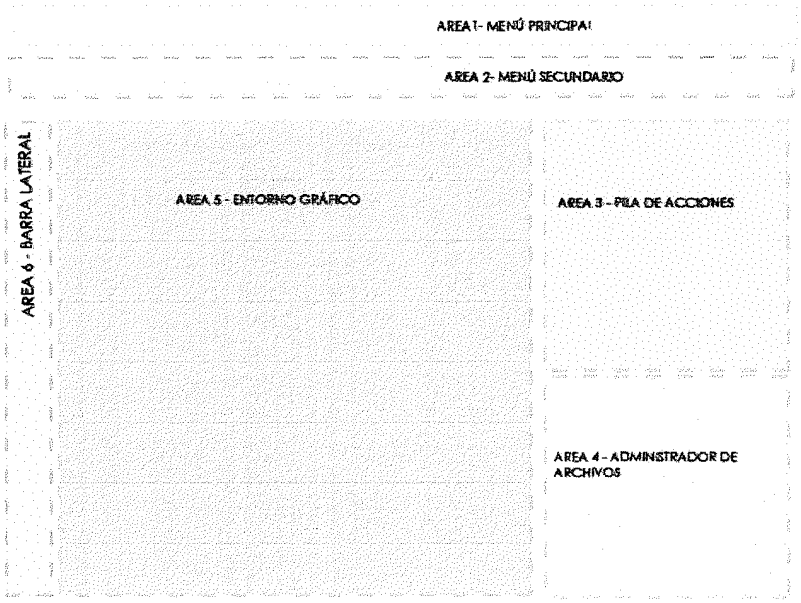


Figura 4.22: Estructura de la interfaz.

de un *modo secuencial*; inicialmente éste llenará un formulario con los aspectos básicos del agente, para luego continuar con los servicios, objetivos, restricción y capacidad (de acuerdo a MASINA).

El modelo de tarea puede especificarse a través de la opción Modelo de Tarea, que se encuentra en la barra izquierda y que despliega una ventana como la que se muestra en la Figura 4.25, en donde el usuario proporciona, a través de un formulario, cada uno de los parámetros necesarios para crear dicho modelo. La Figura 4.26 y 4.27 muestran las ventanas del modelo de conversación y comunicación respectivamente; en donde al igual que los demás modelos el usuario/programador puede introducir, mediante formularios, cada uno de los parámetros que los conforman.

4.4.2.4. Presentación del contenido

Las IGU de EDISMA ofrece un esquema consistente para representar la información que permite llevar a cabo cada uno de los procesos especificados en el Modelado del Negocio. En esta sección se especifican las características gráficas de los elementos que forman parte de

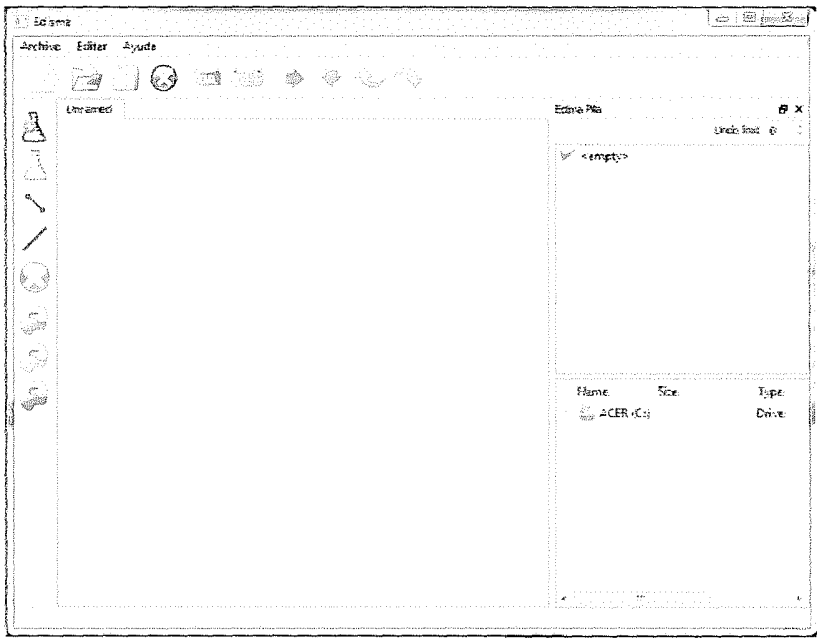


Figura 4.23: Interfaz principal de EDISMA.

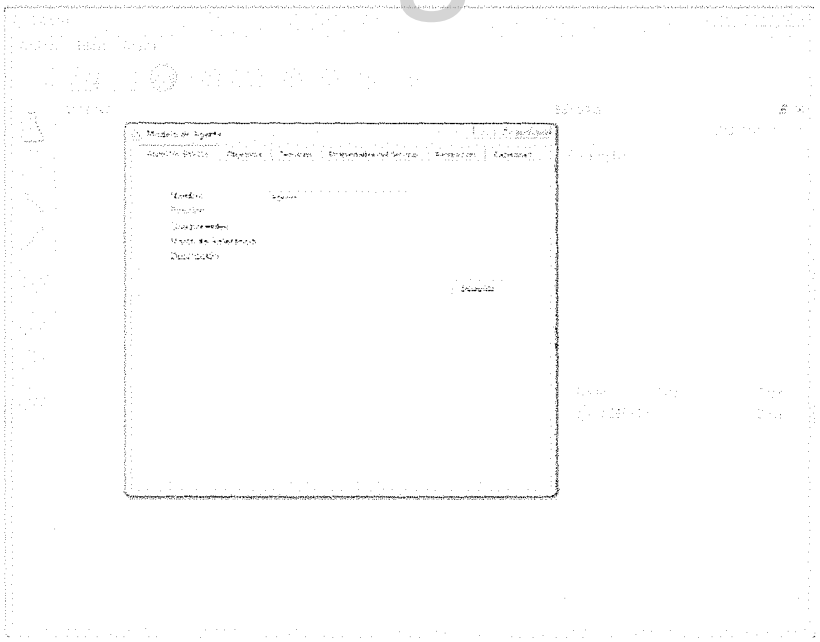


Figura 4.24: Interfaz para crear un Modelo de Agente.

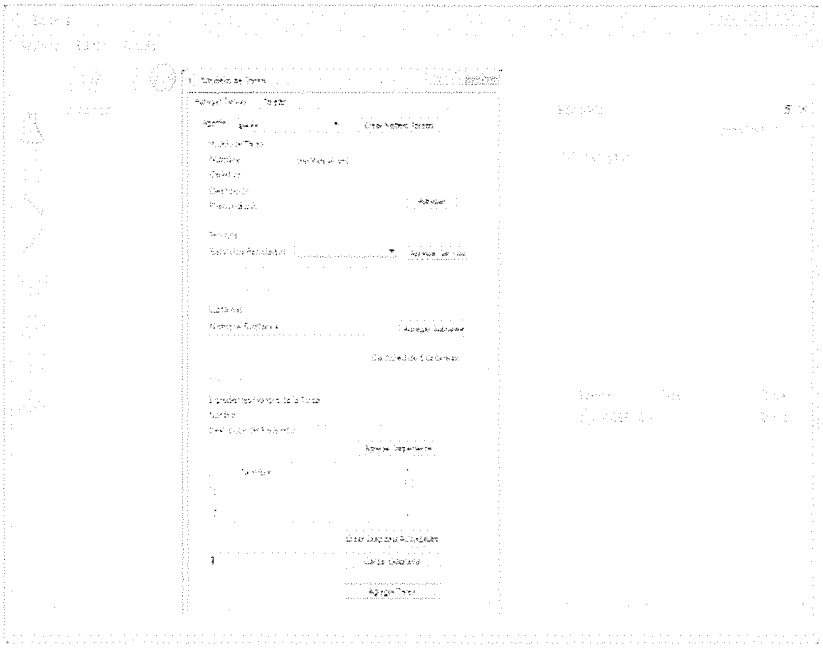


Figura 4.25: Interfaz para crear un Modelo de Tarea.

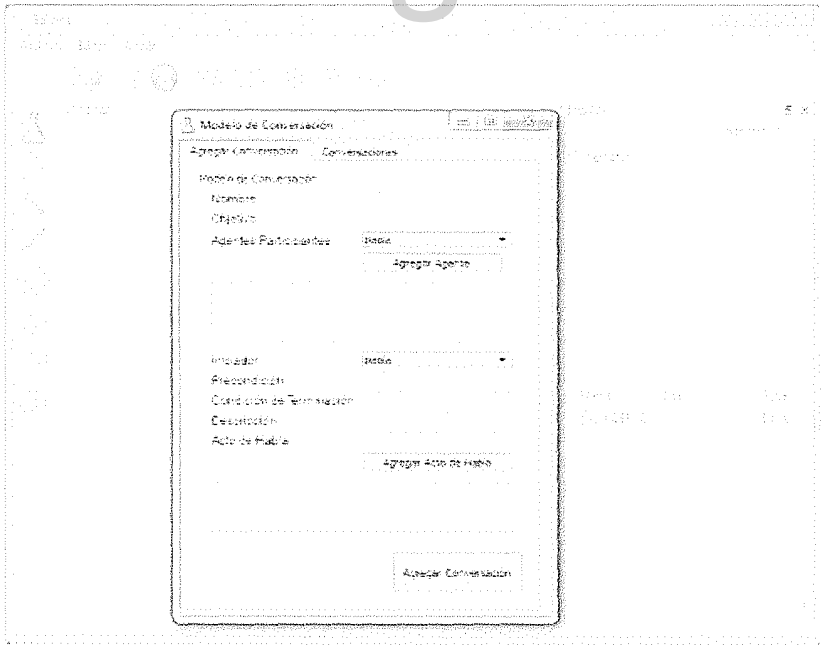


Figura 4.26: Interfaz para crear un Modelo de Conversación.

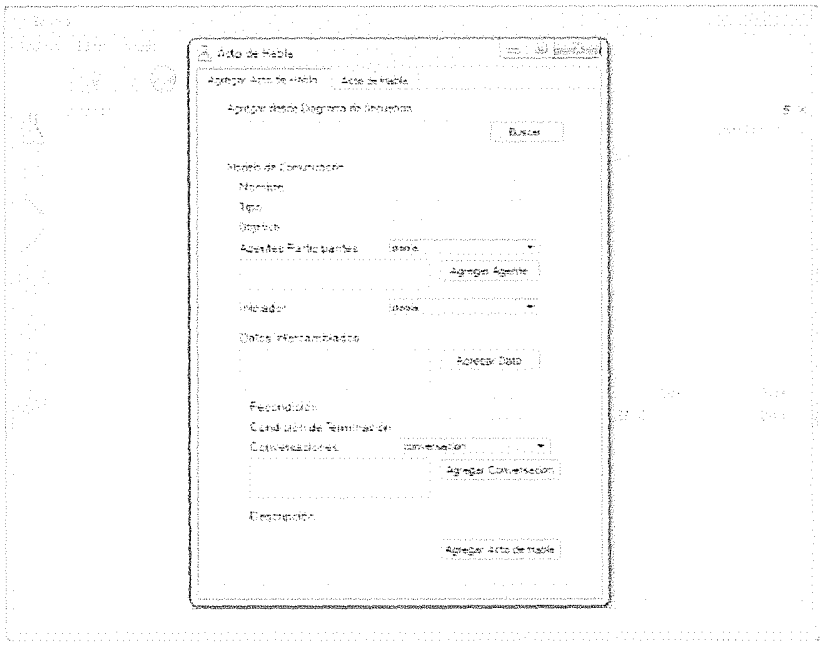


Figura 4.27: Interfaz para crear un Modelo de Comunicación.

la IGU:

- Los colores usados para EDISMA, son de un modo general tonalidades grises, ya que por las características neutrales de este color permiten ciertas bondades en materia de interacción humano–computador. Los iconos en cambio poseen colores vivos que permitan llamar la atención visual del usuario.

- El estilo de fuente usada es "*ms shell Font*", ésta es una fuente legible y facilita la lectura [32].

- Los botones, al igual que la IGU en general, son grises, pero en otra tonalidad para permitir hacer contrate con el fondo, ya que de este modo se mantiene el equilibrio entre los colores empleados.

- Los *grid* (rejillas) que permitan visualizar aquellos elementos que son agregados en mas de una unidad, poseen colores grises y las casillas seleccionadas, cambian su color a azul para realizar contraste, y permitir que el usuario pueda distinguir el elemento seleccionado.

4.4.3. Diseño de la Base de Datos

EDISMA a pesar de ser un sistema con diversas funcionalidades, no necesita un manejador de base de datos formalmente para la ejecución de sus tareas, sin embargo posee un repositorio de archivos que permitan llevar a cabo la implementación para cada uno de los agentes, descritos a continuación:

- archivoCabecera: proporciona la estructura que deben tener los archivos extensión .h que permiten realizar la definición de las clases.
- archivoCuerpo: provee la estructura que deben tener los archivos extensión .cpp que posee la especificación de los métodos definidos en el archivo cabecera.
- archivoPrincipal: proporciona la estructura que deben tener los archivos extensión .cpp pero que contienen el método main.

Para la implentación del makefile y del disparador al igual que para los agentes, se disponen de los archivos necesarios que poseen la estructura básica de los mismos.

4.4.4. Diseño de componentes o módulos de SW

EDISMA posee tres componenes (ver Figura 4.28) :

- IGU: permite obtener los parámetro ó datos suministrados por el usuario, así como enviar los mensajes correspondientes de, éxito ó fracaso, en el uso de las distintas funcionalidades.
- GeneradorModelos: es el componentes que recibe mediante la interfaz, los elementos necesarios para generar los modelos de agente, tarea, conversación y comunicación de MASINA.
- GeneradorInstancias: este componente recibe, mediante la interfaz, el proyecto ó SMA en un archivo extensión XML, para poder construir los archivos que permiten la im-

plementación de los agentes, con las restricciones que se describen en el alcance del Modelado de Negocios.

La ventaja de este diseño, orientado a componentes, es que es posible reutilizar a futuro estos componentes de un modo independiente; debido a que están diseñados bajo el principio de segregación de la interfaz: *es mejor tener muchas interfaces específicas del cliente que una interfaz de propósito general.*

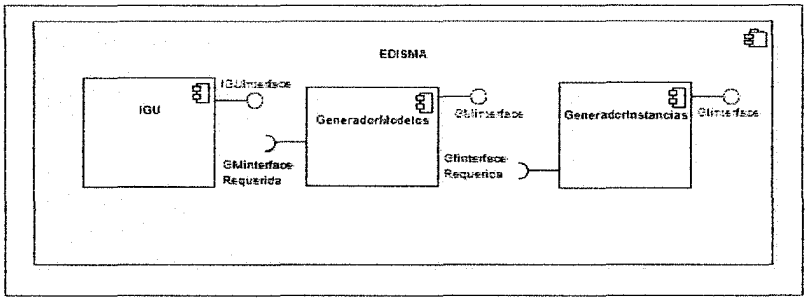


Figura 4.28: Diagrama de Componentes.

4.4.5. Proceso de especificación detallado de componentes

La interfaz se denomina como "la conexión física y funcional entre dos aparatos o sistemas independientes" [43], sin embargo, puede considerarse como un elemento de *software* que permite enviar y/o recibir información. En el "contrato de uso" se especifican detalladamente cada una de las interfaces de EDISMA, a continuación:

- IGUInterface: debido a que el usuario interactúa únicamente con EDISMA, a través de formularios, botones y demás elementos gráficos que permiten obtener y recibir información, esta interfaz es la encargada de suministrar los parámetros obtenidos por medio de la IGU a otros componentes, para mantener los principios de MVC.

- GMInterface Requerida: es la interfaz que recibe los parámetros suministrados por IGUInterface, para ser utilizados por el componente GeneradorModelos. Estos parámetros están conformados por cada uno de los atributos que conforman los modelos de MASINA.

- GMInterface: es la interfaz que permite suministrar los parámetros necesarios para que el componente GeneradorInstancias funcione de un modo adecuado. Esta interfaz se comunica con la IGUInterface para notificar al usuario los productos obtenidos por el componente GeneradorModelos

- GInterface Requerida: permite adquirir los datos necesarios para que el componente GeneradorInstancias pueda formar los archivos que permiten instanciar los agentes. Los datos son los modelos de MASINA en formato XML, por cada uno de los agentes.

- GInterface: al igual que la IGUInterface interactua con las interfaces gráficas para enviar mensajes, indicando los productos obtenidos por el componente GeneradorInstancias.

4.5. Aprovisionamiento de Componentes

En esta sección se procede a buscar, adquirir, adaptar o codificar los componentes de software que integran cada una de las tres capas de la arquitectura de EDISMA [39]. Los componentes que se puedan reutilizar se adquieren y se adaptan; mientras que los restantes se tienen que diseñar y codificar en su totalidad, definiendo previamente los elementos de *software* necesarios. Una vez que estos componentes se elaboren o se adapten, según sea el caso, se procede a probarlos separadamente usando estrategias, técnicas y herramientas de pruebas de unidades.

4.5.1. Instalación de la plataforma de Desarrollo

Para Desarrollar EDISMA es necesario instalar :

- Qt Creator: en su versión 2.4.1 basada en Qt 4.7.4 (32 bit) la cual se encuentra en un repositorio web. (http://qt-project.org/downloadsqt-creator).

- QtXml: este módulo ofrece un lector y escritor de documentos XML. Este modulo se encuentra disponible en el mismo repositorio web de Qt creator.

- El sistema operativo en el que se desarrollara EDISMA será Windows 7.

4.5.2. Adquisición de Componentes

Para el desarrollo de EDISMA no se adquirió ningún componente de software sin embargo fue necesario la adquisición de una aplicación que permitiera la creación y gestión de diagramas UML, para ello se utiliza Umbrello en su versión 2.0 la cual permite proporcionar insumos que son necesarios para la ejecución de los productos finales a obtener. Esta aplicación se encuentra en un repositorio web [30]. En el manual de instalación [31] se especifican los requerimientos y en el manual de usuario, [29] se especcifica desde los conceptos básicos de UML hasta como modelar con la herramienta.

4.5.3. Diseño y Ejecución de pruebas de componentes

Las pruebas del componentes se llevan a cabo integrándolo con EDISMA, es por ello que las misma se realizan en etapas posteriores.

4.6. Emsamblaje del Sistema de *Software*

En esta etapa se codifican los componentes que conforman a EDISMA, para ello se construye la capa de presentación ó interfaz de usuario, la capa lógica, la base de datos y finalmente se realizan las pruebas, a cada uno de estos componentes y las pruebas integración.

4.6.1. Construcción de la interfaz de usuario

La interfaz de usuario se construyó de acuerdo a los elementos de contenido y servicios especificados en la sección de Diseño. En esta etapa se ensambló la capa de presentación con los componentes de software especificados en la sección 4.3.3.

4.6.1.1. Planificación y realización de las pruebas de la interfaz de usuario

En esta actividad se busca determinar cuáles son las pruebas requeridas y los elementos que deben probarse, con el objetivo de corroborar que los resultados obtenidos en las pruebas sean los esperados. Las pruebas funcionales se realizaron de modo local en una máquina con

las especificaciones de *hardware* y *software* descritas en secciones anteriores. La Tabla 4.5 muestra la descripción de las pruebas de comportamiento realizadas para EDISMA.

4.6.2. Ensamblaje de Componentes de EDISMA

En esta etapa se codifican los componentes que forman parte de EDISMA, utilizando el paradigma orientado a objetos. A continuación se nombran los archivos creados, los cuales se en cuentran en el CD que será entregado. Para cada una de las clases se creó un archivo extensión cpp; en donde se describen los métodos, un archivo extensión .h, que contiene la definición de los métodos y de la clase con sus atributos, y un archivo .ui para cada una de las interfaces de usuario necesaria.

- **Archivos extensión .h**: ActosHabla.h, capacidad.h, conversación.h, ConstanteGenerador.h, datos.h, generadorAgente.h, generadorConversacion.h, generadorProyecto.h, generadorTarea.h, Ingrediente.h, objetivoAgente.h, preconversacion.h, subtarea.h, uiactoHabla.h , uitarea.h, uiconversacion.h, uiagente.h, commands.h, document.h, mainwindow.h.

- **Archivos extensión .cpp** : ActosHabla.cpp, capacidad.cpp, conversación.cpp, ConstanteGenerador.cpp, datos.cpp, generadorAgente.cpp, generadorConversacion.cpp, generadorProyecto.cpp, generadorTarea.cpp, Ingrediente.cpp, objetivoAgente.cpp, preconversacion.cpp, subtarea.cpp, uiactoHabla.cpp , uitarea.cpp, uiconversacion.cpp, uiagente.cpp, main.cpp, commands.cpp, document.cpp, mainwindow.cpp.

- **Archivos extensión .ui** :actoHabla.ui, conversación.ui, tarea.iu, , agente.ui,

4.6.2.1. Planificación y realización de las pruebas de componentes

En esta actividad se establecen las pruebas requeridas y los elementos que deben probarse, con el objetivo de verificar que los resultados obtenidos en las pruebas sean los esperados. Las pruebas funcionales se realizaron de modo local en una máquina con las especificaciones

de *hardware* y *software* descritas en secciones anteriores. La Tabla 4.6 muestra la descripción de las pruebas de comportamiento realizadas para EDISMA.

4.6.3. Construcción de la base de datos

En esta etapa se construyen los archivos que forman parte de la base de datos. Para ello inicialmente se define como estará conformada su estructura interna, y posteriormente se procede crear en extensión XML los siguientes archivos:

- archivoCabecera: el cual proporcionará la estructura que deben tener los archivos extensión .h que permiten realizar la definición de las clases.
- archivoCuerpo: proveerá la estructura que deben tener los archivos extensión .cpp que posee la especificación de los métodos definidos en el archivo cabecera.
- archivoPrincipal: proporcionará la estructura que deben tener los archivos extensión .cpp pero que contienen el método main.

Las pruebas a estos archivos que conforman la base de datos se realizarán en el capítulo 5, mediante el caso de estudio, para de este modo determinar si dichos archivos cumplen o no, con los objetivos especificados en el diagrama de objetivos 4.3.

4.6.4. Pruebas de integración de las capas de arquitectura de EDISMA

En esta etapa se realizan las pruebas generales de EDISMA, una vez llevado a cabo las pruebas de interfaz U/S y las pruebas de componentes especificadas en las secciones anteriores.(Ver Tabla 4.7)

Caso de prueba	Resultado Esperado	Resultado a obtener.
Pruebas de confiabilidad.	Se probaron los casos de uso mencionados en la seccion 4.2.3. Especificación de Requisitos.	Se ejecutaron correctamente los caso de uso de la sección 4.2
Pruebas de configuración.	Pruebas de instalación en dos maquinas distintas con los recursos de hardware y software especificados en la sección 4.5.1.	Se instaló correctamente en dos máquinas.

Tabla 4.7: Pruebas de confiabilidad y configuración.

4.7. Entrega del sistema de *software*

Este es el último proceso técnico que se realiza durante el desarrollo de EDISMA. Su objetivo es poner en operación la versión 1.0 de EDISMA que se elaboró en cada ciclo de desarrollo. La Figura 4.29 describe, de manera general, este proceso en función de sus entradas y salidas. Además se realiza en esta etapa la entrega de la aplicación en un archivo que posee un instalador, para que la misma pueda ser instalada y ejecutada.

4.7.1. Capacitación de Usuarios

A modo de ilustrar como se realiza la construcción de un SMA en EDISMA, se especifica a continuación el modo en que se introducen los modelos de MASINA y demás parámetros necesarios para poder obtener el código que permite la implantación del SMA, para ello se ejecutan los siguientes pasos:

1.Crear un SMA: al iniciar la creación de un SMA, se selecciona la opción de crear SMA.(ver Figura 4.30)

1.1 Crear un Agente: se selecciona la opción Agregar Modelo de Agente, representada por el matraz de color verde, o también se puede acceder por el menú superior con la opción editar, seleccionando Agregar Modelo de Agente (ver Figura 4.31). Luego se despliega una ventana en la cual se introducen cada uno de los parámetros especificados en el modelo de agente, ver Figura 4.32.

1.2 Crear una tarea: se selecciona la opción Modelo de Tarea representada por el matraz de color naranja, o también se puede acceder por el menú superior con la opción

Figura 4.29: Funcionamiento de EDISMA

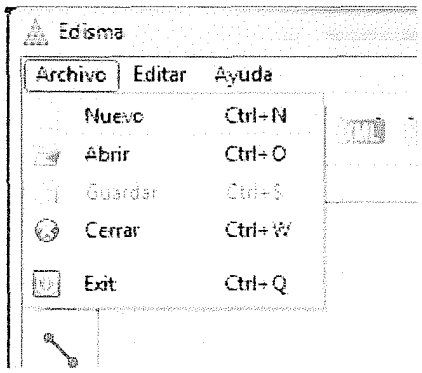


Figura 4.30: Crear un SMA en EDISMA

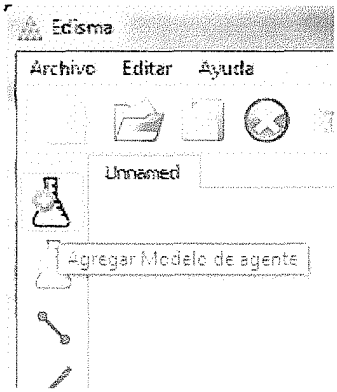


Figura 4.31: Crear un agente en un SMA en EDISMA.

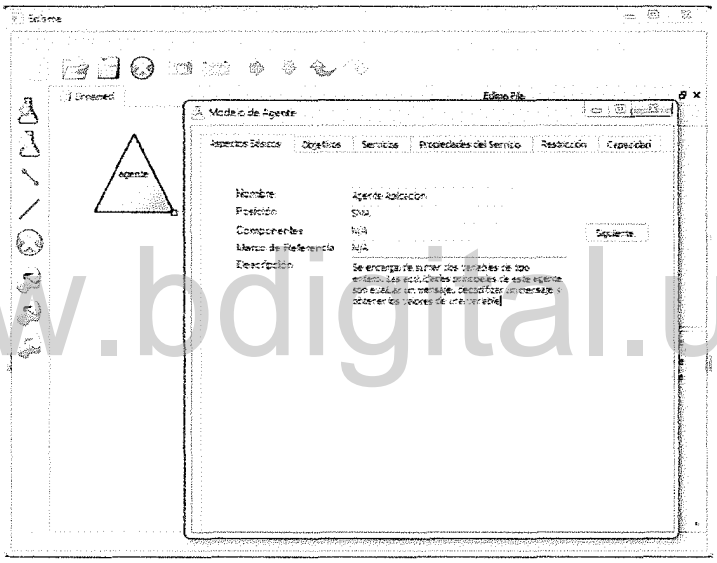


Figura 4.32: Modelo de Agente en EDISMA para el AA.

editar, seleccionando Agregar Modelo de Tarea (Ver Figura 4.33). Luego se despliega una ventana en la cual se introducen cada uno de los parámetros especificados en el Modelo de Tarea (ver Figura 4.35). En la Figura 4.35 se observa que existe un botón con la opción cargar diagrama que permite importar un diagrama de actividades creado en Umbrello que permite parametrizar el campo de subtareas y además aporta un *pseudo* algoritmo a la tarea, que posteriormente es utilizado por el generador C++.

1.3 Crear una conversación: una vez que se ha creado al menos dos agentes es posible generar una conversación. Para ello se selecciona la opción Modelo de Conversación,

representada gráficamente por una linea negra con circulos en sus dos extremos o también se puede acceder por el menú superior con la opción editar, seleccionando Agregar Modelo de Conversación (Ver Figura 4.34). Luego se despliega una ventana en la cual se introducen cada uno de los parámetros especificados en el Modelo de Conversación (ver Figura 4.36). En la Figura 4.36 se observa que existe un botón con la opción cargar diagrama que permite importar un diagrama de secuencia creado en Umbrello, que permite parametrizar los atributos de campo de habla, agentes participantes y agente iniciador.

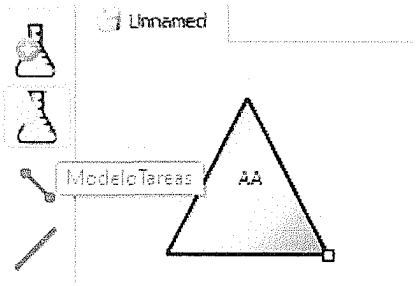


Figura 4.33: Crear una tarea en EDISMA.

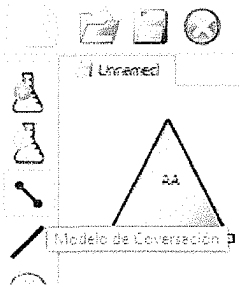


Figura 4.34: Crear una conversación en EDISMA.

1.4 Crear un acto de habla: una vez que se ha creado al menos una conversación es posible generar un acto de habla. Para ello se selecciona la opción Modelo de Comunicación, representada graficamente por una línea negra, o también se puede acceder por el menú superior con la opción editar, seleccionando Agregar Modelo de Comunicación (Ver Figura 4.38). Luego se despliega una ventana en la cual se introducen cada uno de los parámetros especificados en el Modelo de Comunicación (ver Figura 4.37).

2. Generar los Archivos C++: una vez que se han creado todos los modelos que forman parte del SMA, se procede a generar los archivos C++ que lo implantan. Para ello se selecciona la opción c++, representada gráficamente por un rectángulo verde ubicado en la barra superior de iconos (ver Figura 4.39). Luego se despliega en la sección del arbol de archivos de EDISMA los archivos generados, compuestos por archivos .cpp, .h con sus respectivos main.cpp (ver Figura 4.40).

3. Completar los archivos generados en C++ en el editor: una vez que se han

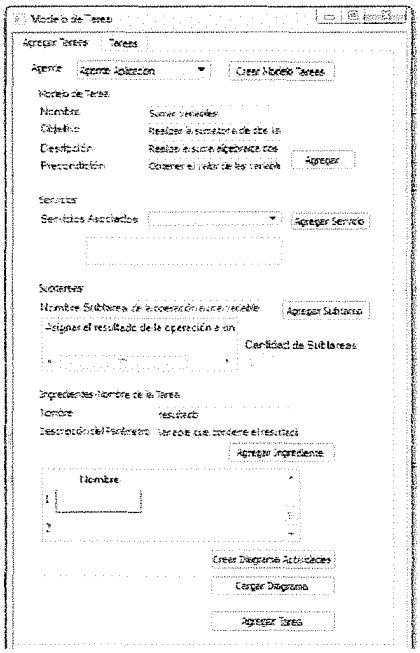


Figura 4.35: Modelo de Tarea en EDISMA para el AA.

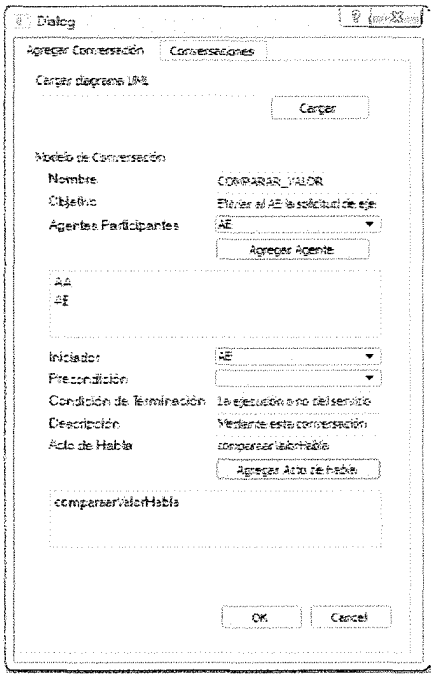


Figura 4.36: Modelo de Conversación en EDISMA.

creado los archivos c++ que permiten implantar el SMA, se procede a completar dichos archivos para que los mismos pueda ser compilados y posteriormente ejecutados. Para ello se selecciona el archivo que se desea completar, en el arbol de archivos que se muestra en la Figura 4.40, y con el evento doble clic se despliega dicho archivo en el editor (ver Figura 4.41). Luego una vez completado el código, se procede a guardar los cambios del mismo.(ver Figura 4.42

4. Generar Disparador: una vez que se han creado y completado los archivos c++ que permiten implantar el SMA, se procede a generar el disparador que permite crear las instancias de los agentes. Para ello se selecciona la opción Editar, Crear Disparador (ver Figura 4.43).

5. Generar MakeFile: una vez que se han creado el disparador, se procede a generar el makefile que permite mediante un *script* compilar los agentes del SMA y el Disparador. Para ello se selecciona la opción Editar, Crear MakeFile(ver Figura 4.43).

6. Reubicar los archivos obtenidos en el paso 3 y 4: una vez que se ha crea-

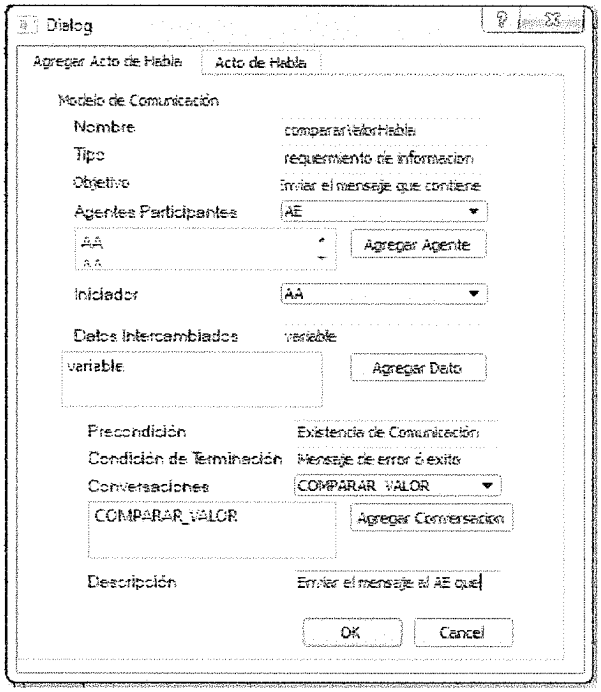


Figura 4.37: Modelo de Comunicación en EDISMA para la conversación compararValorHabla.

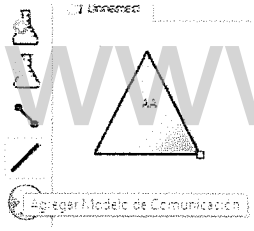


Figura 4.38: Crear un acto de habla en EDISMA.

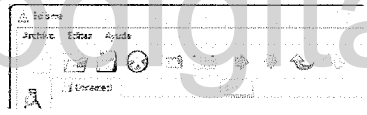


Figura 4.39: Crear archivos C++.

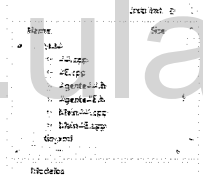


Figura 4.40: Archivos Generados.

do el MakeFile, se procede a ubicar los archivos .h, .cpp y main.cpp del SMA en la ruta MGS/interfaz/superior. Para ello se puede utilizar el navegador de archivos o mediante comandos en un *shell*.

7. Reubicar el MakeFile obtenido en el paso 5: una vez que se han reubicado los archivos c++ generados en EDISMA, se procede a reubicar el MakeFile en la siguiente ruta MGS/interfaz/. Para ello al igual que en el paso 6, se puede utilizar el navegador de archivos o mediante comandos en un *shell*. Luego se compila el SMA, usando el makefile que esta en la misma ruta donde fue reubicado, con el comando Make en un *shell*.

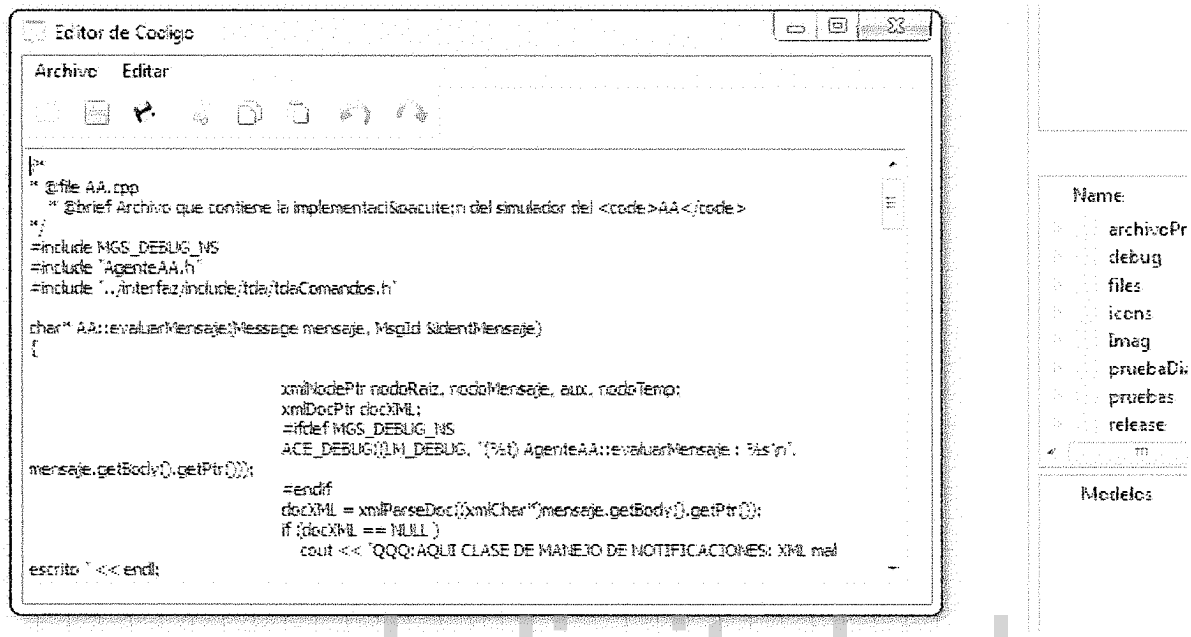


Figura 4.41: Editor de archivos C++ en EDISMA.

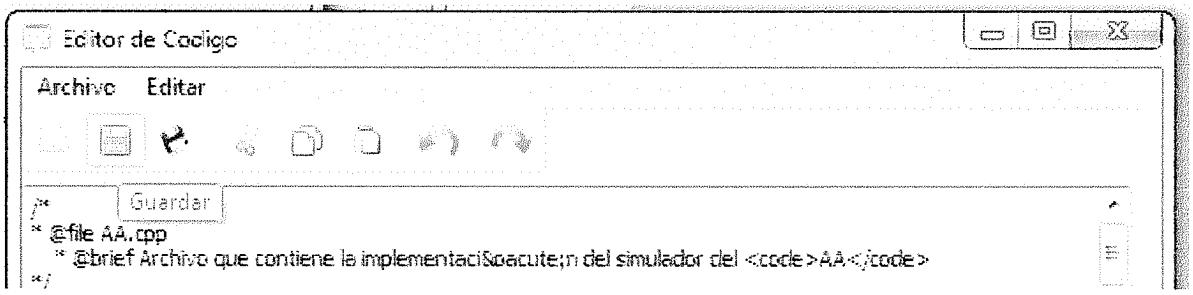


Figura 4.42: Funcionalidad guardar del editor de archivos C++.

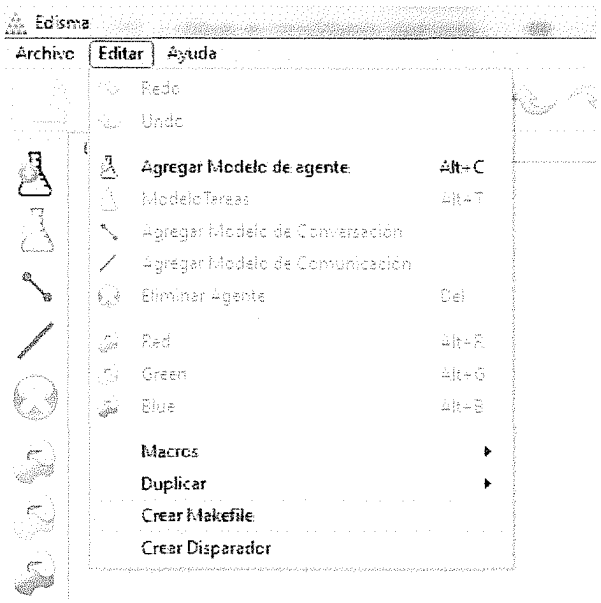


Figura 4.43: Funcionalidad crear Disparador y MakeFile.

8. Ejecutar el Disparador: una vez que se ha compilado el makeFile, el usuario debe garantizar que el MGS esté en ejecución en cada nodo, y que la configuración es correcta, para esto debe seguir los pasos del manual de instalación y pruebas del MGS [40]. Luego es necesario posicionarse en la siguiente ruta /etc/mgs/agente. Para ello, al igual que en el paso 6 y 7 ,se puede utilizar el navegador de archivos o mediante comandos en un *shell*. Luego se compila se ejecuta el disparador con el comando ./"nombredelDisparador".

Caso de prueba	Resultado exitoso	Resultados posibles
Crear un Entorno para un SMA	Creación del entorno.	No es posible crear el entorno.
Guardar un Entorno de un SMA	Se muestra un mensaje exitoso de creación.	Existencia previa de un entorno con el mismo nombre y se muestra un mensaje indicando que se suministre otro nombre para Guardarlo.
Abrir un Entorno SMA	Se despliega una ventana de gestión archivos que permite la búsqueda del archivo que contiene el SMA que se desea aperturar, y una vez seleccionado se envía un mensaje exitoso. Se despliega en el entorno gráfico los elementos gráficos que conforman el SMA.	Mensaje de Error indicando que el tipo de archivo no coincide con el que corresponde al formato de los SMA
Agregar un Modelo de Agente	Crear un modelo de agente a nivel gráfico en el entorno SMA, desplegando una Figura geométrica que lo representa con su identificación textual. Activación de la opción Modelo de Tarea. Se muestra un mensaje en la pila de acciones indicando la creación de un Modelo de Agente.	Mensaje de notificación indicando que el Modelo de Agente no pudo ser creado debido a que no se suministró los elementos necesarios.
Agregar un Modelo de Tarea	Se envía un mensaje de creación exitosa del Modelo de Tarea Activación de la opción Modelo de Conversación. Se muestra un mensaje en la pila de acciones indicando la creación de un Modelo de Tarea.	Mensaje de notificación indicando que el Modelo de Tarea no pudo ser creado debido a que no se suministró los elementos necesarios.
Agregar un Modelo de Conversación	Crear un modelo de tarea a nivel gráfico entre los agentes especificados. Activación de la opción Modelo de Comunicación. Se muestra un mensaje en la pila de acciones indicando la creación de un Modelo de Conversación.	Mensaje de notificación indicando que el Modelo de Conversación no pudo ser creado debido a que no se suministró los elementos necesarios.
Agregar un Modelo de Comunicación	Crear un modelo de comunicación a nivel gráfico entre los agentes especificados. Se muestra un mensaje en la pila de acciones indicando la creación de un Modelo de Comunicación	Mensaje de notificación indicando que el Modelo de Comunicación no pudo ser creado debido a que no se suministró los elementos necesarios.
XML	Envía un mensaje indicando la creación del SMA y la ruta de los archivos XML.	Envio de un mensaje indicando que no es posible crear los archivos
C++	Envía un mensaje envía un mensaje indicando el éxito de la operación y la ruta de los archivos generados.	Envia un mensaje indicando que no es posible la operación

Tabla 4.5: Pruebas funcionales de EDISMA.

Caso de prueba	Resultado exitoso	Resultado a obtener.
Crear un Entorno para un SMA.	Creación del entorno a nivel gráfico y creación de los archivos temporales en donde se almacenara la información que contenga dicho SMA.	No se crean los archivos temporales
Guardar un Entorno de un SMA.	Se generan dos archivo en donde se especifican los aspectos gráficos y lógicos del SMA y se muestra un mensaje exitoso de creación.	Existencia previa de un entorno con el mismo nombre.
Abrir un Entorno SMA.	Se realiza la lectura de los archivos que forman parte del SMA y se despliega en el entorno gráfico los elementos gráficos y lógicos que conforman el SMA.	Incompatibilidad de lectura al seleccionar un archivo que no tenga el formato adecuado para ser reconocido por EDISMA.
Agregar un Modelo de Agente.	Se documenta el modelo de agente en un archivo extensión XML. Se actualiza el archivo del SMA agregando la información respecto al Modelo de Agente creado.	Validación de existencia previa de archivos que contengan documentación de Modelo de Agente con el mismo nombre.
Agregar un Modelo de Tarea	Se documenta el Modelo de Tarea en un archivo extensión XML. Se actualiza el archivo del SMA agregando la información respecto al Modelo de Tarea creado.	Validación de existencia previa de archivos que contengan documentación de Modelo de Tarea con el mismo nombre.
Agregar un Modelo de Conversación.	Se documenta el Modelo de Conversación en un archivo extensión XML. Se actualiza el archivo del SMA agregando la información respecto al Modelo de Conversación creado.	Validación de existencia previa de archivos que contengan documentación de Modelo de Conversación con el mismo nombre.
Agregar un Modelo de Comunicación	Se documenta el Modelo de Comunicación en un archivo extensión XML. Se actualiza el archivo del SMA agregando la información respecto al Modelo de Tarea creado.	Mensaje de notificación indicando que el Modelo de Comunicación no pudo ser creado debido a que no se suministró los elementos necesarios.
XML.	Verifica la ruta ó el lugar en donde se encuentran los archivos XML que documentan al SMA bajo los principios de MASINA para notificarlo a través de un mensaje.	No es posible crear los archivos XML
C++.	Crea los archivos necesarios para instanciar los agentes y envía un mensaje envía un mensaje indicando el éxito de la operación.	No es posible llevar a cabo la operación

Tabla 4.6: Pruebas de componentes.

Capítulo 5

Caso de Estudio

5.1. Introducción

El caso de estudio consiste en generar un SMA con EDISMA, para ello es necesario previamente tener especificado formalmente lo Modelos de MASINA para cada uno de los agentes. El objetivo del caso de estudio es comprobar que lo archivos generados con EDISMA, implentan gran parte del SMA, dejando al programador solo la actividad de completar el código que que implementa una parte del modelo de tarea y la estructura de los mensajes.

5.2. Descripción General

Tomando en cuenta que el MGS posee una prueba de instanciación de un SMA, se ha realizado el proceso de ingeniería inversa sobre dicha prueba, que consiste en analizar los archivos que la conforman, para luego especificar el modelo del SMA y proceder a generarlo con EDISMA. El SMA está conformado por cuatro agentes:

- Agente Aplicación (AA): realiza la suma algebraica de dos variables cuyos valores son tipo entero y asigna dicho resultado a una variable.
- Agente Especializado (AE): realiza la comparación del valor de una variable con un valor fijo, para identificar si dicha variable posee un valor mayor, menor o igual.
- Agente Negocio (AN): solicita la búsqueda del Agente Repositorio y del AA, así como los servicios del mismo.

- Agente Repositorio (AR): se encarga de obtener el valor de tres variables, este agente presta su servicio al AA y al AE

A fín de ilustrar la especificación de los modelos de MASINA, a continuación se caracteriza el AA y el Disparador, los démas agentes se encuentran en el anexo.

5.3. Agente Aplicación

En esta sección se presenta los Modelos de Agente, Tarea, Comunicación y Coordinación especificados en MASINA [10], para el AA, los modelos para el AE,AN y AR se encuentran especificados en el anexo.

5.3.1. Modelo de Agente

La Tabla 5.1 muestra el Modelo de Agente para el AA, en donde se especifica cada uno de los detalles básicos que lo conforman.

5.3.2. Modelo de Tareas

En este caso, la tarea que ejecuta el AA, ver Tabla 5.2, se asocia al único servicio definido sumar dos variables.

5.3.3. Modelo de Coordinación

En función del objetivo y servicios del SMA, se definen una serie de conversaciones que permiten la ejecución de dichos servicios. En la Figura 5.1 se presenta el diagrama de interacción con los las conversaciones definidas para el SMA, las cuales son las siguientes:

- PrestarServicio (Ver Tabla 5.3)
- ObtenerVariables.
- SumarDosVariables.
- LocalizarAplicacion.

Tabla 5.1: Plantilla del Modelo de Agente del AA

Agente		
Nombre		Agente Aplicación
Posición		SMA
Componentes		No aplica
Marco de referencia		No aplica
Descripción del agente		Se encarga de sumar dos variables de tipo entero. Las actividades principales de este agente son evaluar un mensaje, decodificar un mensaje y obtener los valores de una variable.
Objetivos del Agente		
Nombre		Sumar dos variables
Descripción		Realizar la suma algebraica de los valores tipo enteros de dos variables.
Parámetro de entrada		Var1, Var2
Parámetro de salida		Resultado
Condición de activación		Recepción de solicitud
Condición de finalización		Sumatoria de las variables
Condición de éxito		Realizar la sumatoria de las variables
Condición de fracaso		Las condiciones pueden ser las siguientes: no se realiza la sumatoria, formato es incorrecto, se produjo un error de comunicación entre los actores ó los datos no existen.
Ontología		
Servicios del Agente		
Nombre		Sumar dos variables
Descripción del Servicio		Recibe la solicitud de sumatoria de dos variables así como los valores de dichas variables. El agente solicita el servicio del Agente Especializado para comparar el valor. El agente envía un mensaje al Agente Especializado para informar la obtención del resultado. Este servicio puede ser ejecutado bajo la modalidad de peticiones.
Tipo de Servicio		Clasificación (Interno, Externo o ambos: servicio dual)
Parámetros de entrada		Var1,Var2
Parámetros de salida		Resultado
Propiedades del Servicio		
Nombre	Valor	Descripción
Calidad	0-100	N/A
Auditable	1	N/A
Garantía	0-100	N/A
Capacidad	0-100	N/A
Confiabilidad	0-100	N/A
Capacidad del Agente		
Habilidades del agente		Gestionar el acceso a repositorio a datos y decodificar mensajes
Representación del Conocimiento		N/A
Lenguaje de Comunicación		C++
Restricción del Agente		
Normas		El acceso a los repositorios de datos depende de la existencia de conexión entre el Agente Gestor de Datos y el medio de almacenamiento
Preferencias		Gestión de solicitudes por orden de legada
Permisos		La solicitud de servicios son accedidos por AGS. Los datos son obtenidos a través del AGD

Tabla 5.2: Modelo de Tarea del AA

SUMAR VARIABLES	
Nombre	sumarDosVariables
Objetivo	Realizar la sumatoria de dos Variables
Descripción	Realiza la suma algebraica dos variables cuyos valores sean enteros.
Servicios asociados	Sumar dos variables
Precondición	Obtener el valor de las variables.
Sub-tareas	Asignar el resultado de la operación a una variable
INGREDIENTES-NOMBRE DE LA TAREA	
valor1	Variable que posee un valor entero
valor2	Variable que posee un valor entero
resultado	Variable que contiene el resultado de la suma algebraica de la variable1 y variable2

- ComparарValor.
- Informar, esta conversación es utilizada por varios agentes es por ello que a modo ilustrativo solo se especificará cuando es iniciada por el AE y enviada al AA.

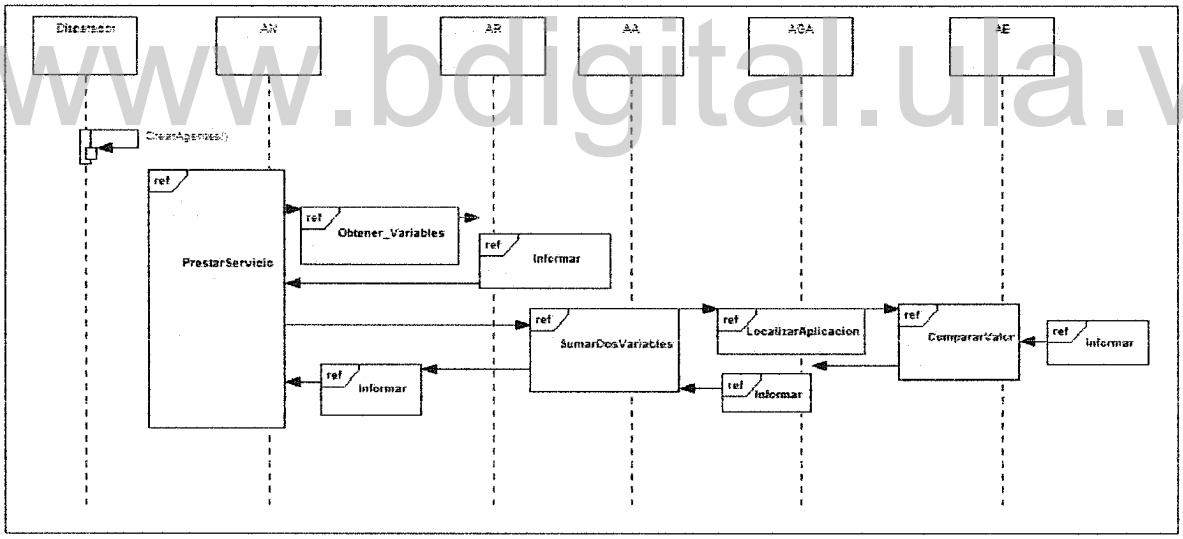


Figura 5.1: Diagrama de interacción del SMA.

A modo de ejemplo sólo se especificará en este capítulo la conversación PrestarServicio, el resto se encuentran en el anexo. En el diagrama de interacción (ver Figura 5.1), participa el disparador, los cuatro agentes definidos en la sección 5.2, y además el AGA que es un agente propio del MGS definido en 2.3. La conversación que inicia la ejecución es PrestarServicio, y

esta a su vez genera la invocación de las demás conversaciones.

Tabla 5.3: Modelo de Conversación PRESTAR_SERVICIO del SMA

CONVERSACIÓN: LOCALIZAR_APLICACION	
Objetivo	Enviar al AGA la solicitud de realizar la invocación de los servicios de localización.
Agentes participantes	AA y AGA
Iniciador	AA
Actos de habla	localizarAplicacionComparar localizarAplicacionCompararReturn
Precondición	sumarDosVariables.
Condición de terminación	La ejecución o no del servicio
Descripción	Mediante esta conversación, el AA inicia la conversación que permite al AGA la ejecución de su servicio de localización

5.3.4. Modelo de Comunicación

En esta sección, a modo se ejemplo, se realizará la especificación del acto de habla de la conversacion COMPARAR_VALOR, el cual es iniciado el por AA y recibido por el AE para enviar el mensaje que contiene la información respecto al servicio compararValor (Ver Tabla 5.4).

Tabla 5.4: Modelo de comunicación para acto de habla localizarAplicacionComparar del SMA

ACTO DE HABLA: localizarAplicacionComparar	
Nombre	localizarAplicacionComparar
Tipo	Requerimiento de información
Objetivo	Enviar el mensaje que contiene la información respecto al servicio Comparar-Valor al AGA.
Agentes participantes	AA y AGA.
Iniciador	AA
Datos intercambiados	Datos de entrada: Uid del AGA
Precondición	Existencia de comunicación entre el AA y AGA
Condición de terminación	Mensaje de error o de éxito del servicio
Conversaciones	LOCALIZAR_APLICACION
Descripción	Enviar el mensaje al AGA que para que localize el agente que posee el servicio compararValor.

5.4. Construcción del caso de estudio con EDISMA

A modo de ilustrar como se realizó la construcción del SMA del caso de estudio con EDISMA, siguiendo los pasos descritos en la sección 4.7.1, se especifica a continuación cómo se ejecutó la digitalización de los modelos de MASINA descritos en la sección 5.3 y en el anexo.

1.Crear el SMA: al iniciar la creación de un SMA, se selecciona la opción de crear SMA y se especifica el nombre y la ruta en la cual se creará el proyecto.(ver Figura 5.2)

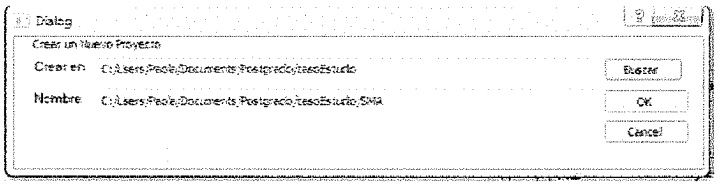


Figura 5.2: Crear el caso de estudio con EDISMA.

1.1 Crear los agentes: se selecciona la opción Agregar Modelo de Agente, y luego en la ventana desplegada se inicia especificando los parámetros para el AA descrito en la sección 5.3.1, ver Figuras 5.3, 5.4, 5.5 y 5.6. Luego se crea el AE, AR y AN siguiendo los modelos de agente descritos para cada uno de ellos en el anexo.

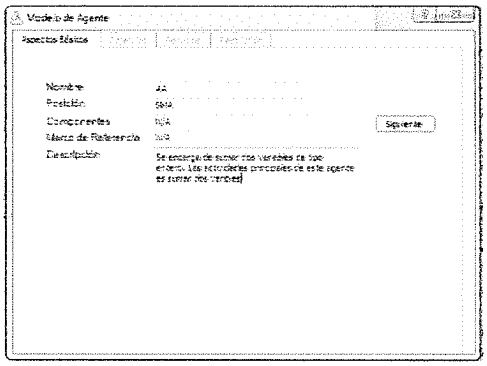


Figura 5.3: Crear AA, pestaña de aspectos básicos.

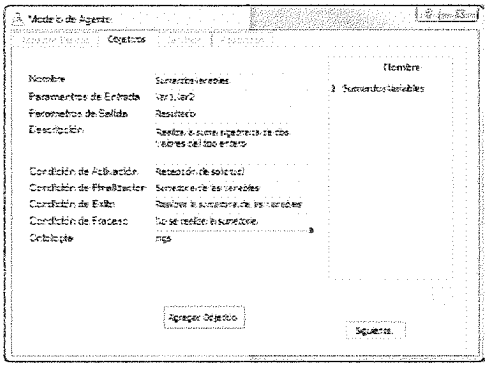


Figura 5.4: Crear AA, pestaña de objetivos.

1.2 Crear las tareas: se selecciona la opción Modelo de Tarea, luego en la ventana que se despliega, se introducen cada uno de los parámetros especificados en el Modelo de Tarea del AA (ver Tabla 5.2). Luego se importa el diagrama de actividades (ver Figura 5.7) que

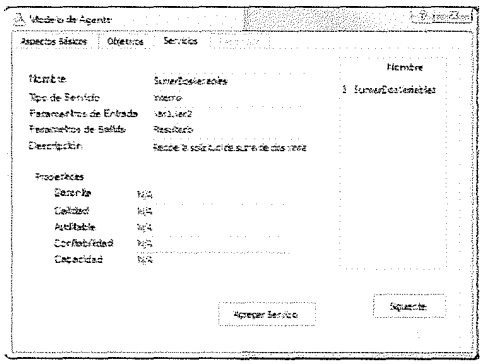


Figura 5.5: Crear AA, pestaña de servicios.

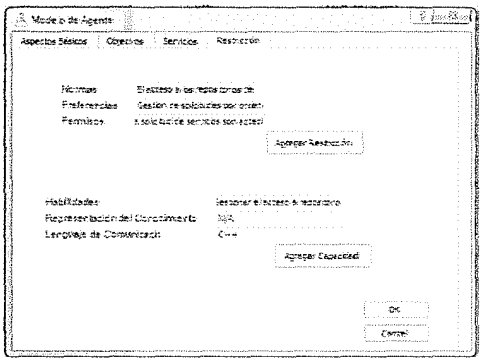


Figura 5.6: Crear AA, pestaña de restricciones.

permite parametrizar el campo de subtareas y además aporta un *pseudo* algoritmo en el código fuente que implanta dicha tarea.

1.3 Crear las conversaciones: para ello se selecciona la opción Modelo de Conversación y se despliega una ventana en la cual se introducen cada uno de los parámetros especificados en el Modelo de Conversación LOCALIZAR_APLICACION de la Tabla **??** (ver Figura 5.10). En la Figura 5.8 se observa el diagrama de interacción elaborado en Umbrello y que fue importado con EDISMA.

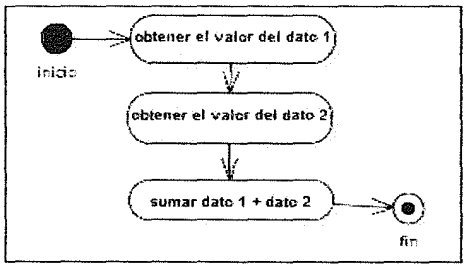


Figura 5.7: Diagrama de actividades para la tarea SumarDosVariables.

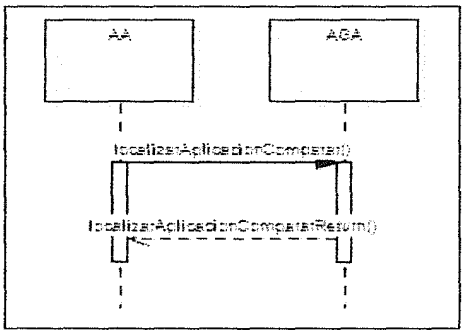


Figura 5.8: Diagrama de interacción para la conversacion LocalizarAplicacion.

1.4 Crear actos de habla: los actos de habla de la conversación LocalizarAplicacion, se crearón al momento de importar el diagrama de interacción de la misma.

2. Generar los Archivos C++: se selecciona la opción c++, y posteriormente se despliega en la sección del arbol de archivos de EDISMA los archivos generados, compuestos

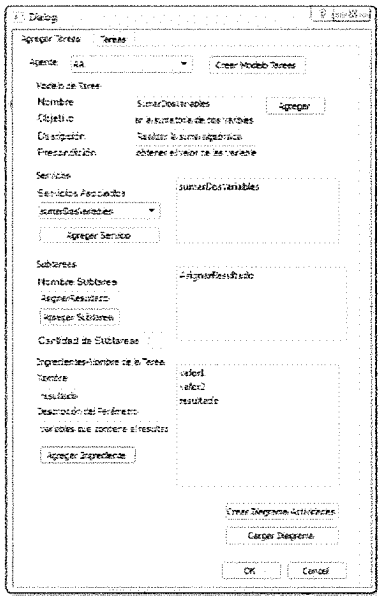


Figura 5.9: Modelo de tarea en EDISMA para el AA.

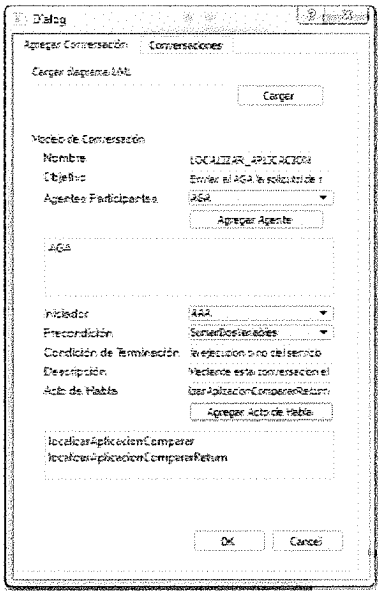


Figura 5.10: Modelo de conversación en EDISMA para el SMA.

por archivos .cpp, .h con sus respectivos main.cpp (ver Figura 5.11).

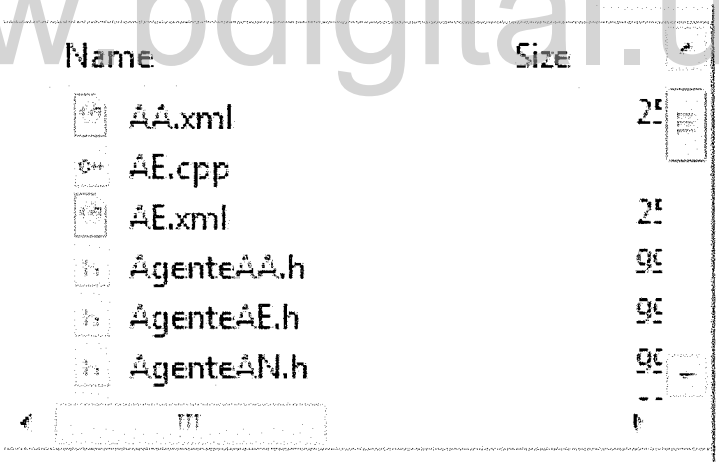


Figura 5.11: Archivos generados para el SMA.

3. Completar los archivos generados en C++ en el editor: una vez que se han creado los archivos que permiten implantar el SMA, se procede a completarlos para que los mismos pueda ser compilados y posteriormente ejecutados. A modo de ejemplo se selecciona el archivo AA.cpp, en el arbol de archivos y este se despliega en el editor (ver Figura 5.12), donde es posible observar en color amarillo, el lugar donde debe completarse el código, para

poder implementar el AA, se procede a guardar los cambios del mismo (ver Figura ??).

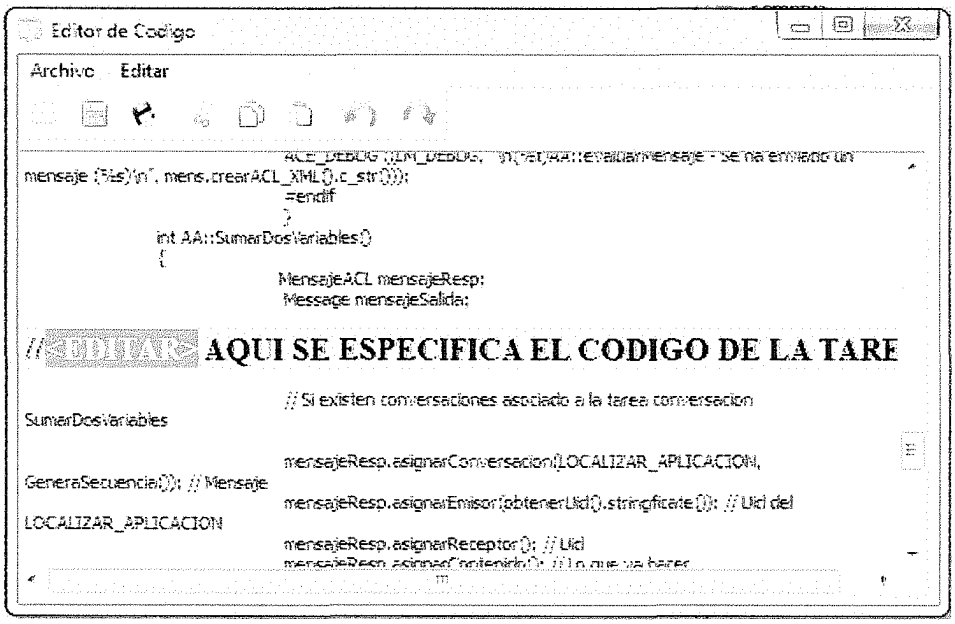


Figura 5.12: Editor de archivos C++ en EDISMA, AA.cpp sin edición

4. Generar Disparador: se selecciona la opción Editar, Crear Disparador.

5. Generar MakeFile: se selecciona la opción Editar, Crear MakeFile.

6. Reubicar los archivos obtenidos en el paso 3 y 4: una vez que se ha creado el MakeFile, se procede a ubicar los archivos .h, .cpp y main.cpp del SMA en la ruta MGS/interfaz/superior. Para ello se puede utilizar el navegador de archivos o mediante comandos en un *shell.*

7. Reubicar el MakeFile obtenido en el paso 5: una vez que se han reubicado los archivos C++ generados en EDISMA, se procede a reubicar el MakeFile en la siguiente ruta MGS/interfaz/. Para ello al igual que en el paso 6,se puede utilizar el navegador de archivos o mediante comandos en un *shell.* Luego se compila el SMA, usando el makeFile que esta en la misma ruta donde fue reubicado, con el comando Make en un *shell.*

8. Ejecutar el Disparador: una vez que se ha compilado el makeFile, el usuario debe garantizar que el MGS esté en ejecución en cada nodo, y que la configuración es correcta, para esto debe seguir los pasos del manual de instalación y pruebas del MGS [40] es necesario

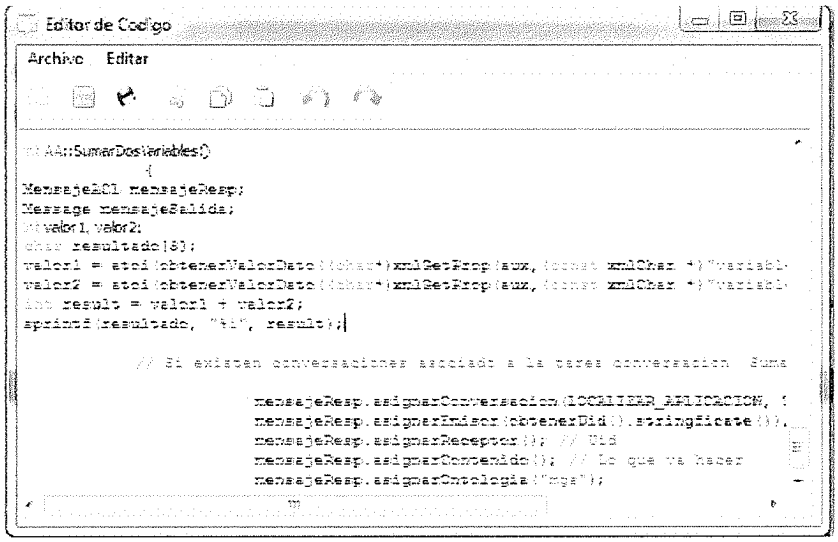


Figura 5.13: Editor de archivos C++ en EDISMA, AA.cpp con edición.

posicionarse en la siguiente ruta /etc/mgs/agente. Para ello, al igual que en el paso 6 y 7 ,se puede utilizar el navegador de archivos o mediante comandos en un *shell*. Luego se compila se ejecuta el disparador con el comando ./"nombredelDisparador".

5.5. Resultados Obtenidos

Una vez diseñado el SMA del caso de estudio descrito al inicio del capítulo 5, diseñado en la sección 5.3, e implementado siguiendo los pasos de la sección 5.4 , finalmente se obtuvo el SMA construido con EDISMA.

EDISMA generó la estructura de los agentes del SMA y además los aspectos de comunicación y coordinación, de modo que el programador se encargó de los detalles que implantan las tareas de los agentes, adicionando el código necesario en cada uno de los archivos cuerpo de cada agente. Tambien es necesario, que el programador especifique la estructura de los mensajes, que son utilizados en el modelo de comunicación, en el archivo tdaMensajeACLNS.h.

Posteriormente fue posible ejecutarlo en el MGS obteniendo exitosamente los resultados de cada una de las tareas asignadas descritas en la sección 5.2.

Capítulo 6

Conclusiones y Recomendaciones

6.1. Conclusiones

En este trabajo se ha construido un IDE llamado EDISMA, que permite la creación de Sistemas MultiAgentes. EDISMA tiene la versatilidad de generar la estructura de archivos que permite implantar agentes modelados en MASINA, que pueden ser instanciados en un MGS y con las siguientes caracterícas generales:

1. Es un entorno amigable.
2. No requiere de uso de una máquina virtual.
3. Facilita al programador la tarea de codificar los aspectos de definición de los agentes, aspectos de comunicación y coordinación, que son implementados utilizando los métodos de la biblioteca del MGS.
4. Posee integrado un editor de textos, que permite editar internamente los archivos y ademas indica al programador donde debe incluir código fuente para completar los mismos.

Para modelar el Sistemas MultiAgente se emplea la fase de análisis de MASINA, en la cual se lleva a cabo la especificación detallada de todos los elementos propuestos en la fase de conceptualización, así como la especificación de las comunicaciones y coordinación entre

el SMA, detallando por medio de plantillas los modelos de agente, tareas, coordinación y comunicación, excepto el de inteligencia debido a que actualmente no está diseñado.

EDISMA está compuesta por un entorno gráfico, un editor de textos y un analizador de diagramas de UML creados con Umbrello. El entorno gráfico está compuesto por el área de visualización de los agentes creados, área de acceso a las funcionalidades, área de gestión de archivos y área de modelos creados.

Las funcionalidades de EDISMA abarcan los aspectos de especificación de los modelos diseñados del SMA y los aspectos de implementación de los mismos, a través de una IGU la cual permite parametrizar cada uno de los atributos que conforman las plantillas de MASINA, además importar los diagramas UML que modelan algunos aspectos del modelo de tarea y de coordinación.

El producto inicial de EDISMA es generar una documentación en archivos XML de cada uno de los modelos, para tener un código intermedio. Debido a que es XML es extensible, posee analizadores estándar, amplia compatibilidad, entre otras particularidades, este primer producto permitirá que a EDISMA puedan incorporarse nuevas funcionalidades ó modificaciones tales como verificaciones de modelos y creación de códigos fuentes en diversos lenguajes de alto nivel (siempre y cuando estos provean compatibilidad con el MGS).

Para implementar el SMA se genera inicialmente una serie de archivos en C++, conformados por un archivo cabecera, que realiza las definiciones de los agentes y de los métodos, un archivo cuerpo que posee la especificación de los métodos definidos y un archivo principal que permite la instanciación de los agentes. Estos archivos deben completarse en el editor de textos, para obtener el código que permite compilar los agentes.

Finalmente, el caso de estudio presentado muestra las bondades de EDISMA para llevar a cabo la implantación de un SMA conformado por cuatro agentes que conversan entre ellos y ejecutan diversos servicios.

6.2. Recomendaciones

La primera versión de EDISMA cumple con la mayoría de los requerimientos especificados, sin embargo, se recomiendan los siguientes aspectos para posteriores versiones :

- Desarrollar ó reutilizar un componente para la creación de diagramas de actividades y de secuencia, que para el mismo pueda ser integrado en EDISMA, ya que en la versión actual se dispone de un componente que analiza los diagramas UML generados en la aplicación Umbrello.
- Desarrollar un componente que permita la generación de la estructura de los mensajes que son utilizados en las conversaciones y actos de habla del SMA, los cuales poseen características propias de acuerdo a los servicios que los mismos están asociados, pero que pudiese estandarizarse.
- Incorporar un simulador a EDISMA, que permita proyectar el comportamiento de cada uno de los agentes que forma parte del SMA, sin ejecutarlos directamente en el MGS.
- Desarrollar una versión, de EDISMA, que genere códigos en otros lenguajes de programación que puedan ser de utilidad para implementar casos de estudio de SMA, tomando en cuenta que MGS se encuentra en C++.

Referencias

[1] Inc. Acronymics. An integrated toolkit for constructing intelligent software agents reference manual. Versión 1.4, Acronymics, Inc, Arkansas, EEUU, Junio 2004.

[2] H. Afsaneh. *Communication and Cooperation in Agent Systems: A Pragmatic Theory.* Springer Verlag, 1996.

[3] J. Aguilar, I. Bessembel, M. Cerrada, F. Hidrobo, and F. Narciso. Una metodología para el modelado de sistemas de ingeniería orientado a agentes inteligencia artificial. In *Revista Iberoamericana de Inteligencia Artificial*, number 38, pages 39–60, 2008.

[4] J. Aguilar, C. Bravo, and F. Rivas. Diseño de una arquitectura de automatización industrial basada en sistemas multiagentes. *Revista Ciencia e Ingeniería, Facultad de Ingeniería*, 25(2):75–88, Julio 2004.

[5] J. Aguilar, V. Bravo, M. Cerrada, F. Hidrobo, G. Mousalli, and F. Rivas. Especificación detallada de los agentes del scdia. Informe técnico del 3er año, proyecto agenda petróleo, ULA-FONACIT, Venezuela, Diciembre 2003.

[6] J. Aguilar and R. Castellanos. Modelo ontológico de verificación de sistemas multiagentes diseñados bajo masina. *Revista Avances en Sistemas e Informática*, 5:163–176, 2008.

[7] J. Aguilar, M. Cerrada, and F. Hidrobo. A methodology to specify multiagent systems. In *Agent and Multi-Agent Systems: Technologies and Applications*, pages 92– 101, Berlin / Heidelberg, Springer 2007.

[8] J. Aguilar, M. Cerrada, G. Mousalli, F. Rivas, and F. Hidrobo. A multiagent model for intelligent distributed control systems. *Lecture Notes in Artificial Intelligence*, 3681:191–197, 2005.

[9] J. Aguilar, M Cerrada, A. Ríos, and F. Hidrobo. Diseño y Construcción de una Plataforma para el desarrollo e implantación de sistemas multiagentes. Technical report, ULA, CDCHT, 07 2010.

[10] J. Aguilar, F. Hidrobo, and M. Cerrada. A methodology to specify multiagent systems. *Lecture Notes in Artificial Intelligence*, 4496:92–101, 2007.

[11] J. Aguilar, L. Leon, A. Ríos, F. Hidrobo, J. BarRíos, and O. Teran. Reporte de validacion de pruebas de integracion del invel interfaz y base. In *Seguimiento, control y evaluacion de la fase de desarrollo del Medio de Gestión de Servicios de la plataforma Net-DAS 2.0*, pages 1–9, Merida, Diciembre 2007.

[12] J. Aguilar, G. Mousalli, V. Bravo, and H. Díaz. Agentes de control y de gestión de servicio para el modelo de referencia scdia. In *II Simposio Internacional de Automatización y Nuevas Tecnologías, TECNO2002*, pages 45–50, Mérida, Venezuela, Noviembre 2002.

[13] J. Aguilar, A. Ríos, F. Hidrobo, and M. Cerrada. *Sistemas MultiAgentes y sus aplicaciones en Automatización Industrial.* Universidad de Los Andes, Mérida, Venezuela, 1 edition, 2012.

[14] J. Aguilar, A. Ríos, F. Hidrobo, and L. Leon. An architecture for industrial automation based on intelligent agents. *WSEAS Transactions on Computers*, 4(12):1808–1815, 2005.

[15] J. Aguilar and W. Zayas. Sistema multi-agente para crear agentes de control para el scdia. *Avances en Sistemas e Informática*, 6(2):47–58, 2009.

[16] J. Barrios and J. Montilva. Método white_watch para el desarrollo de proyectos pequeños de software. Versión 1.2, Universidad de Los Andes, Venezuela, Enero 2010.

[17] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. Software framework for developing multi-agent applications. *Information and Software Technology*, 50:10–21, 2008.

[18] R. Bravo. Sistema generador de codigo para la metodología masina. Tesis de pregrado, Universidad de Los Andes, Mérida, Venezuela, 2009.

[19] V. Bravo, J. Aguilar, F. Rivas, and M. Cerrada. Diseño de un medio de gestión de servicios para sistemas multiagentes. In *XXX Conferencia Latinoamericana de Informática*, pages 431–439, Arequipa, Perú, Octubre 2004.

[20] M. Cerrada, J. Cardillo, J. Aguilar, and R. Faneite. Agents-based reference design for fault management systems in industrial processes. *Computers in Industry*, 58(4):313–328, 2007.

[21] O. Etzioni, M. Cafarella, D. Downey, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Unsupervised named-entity extraction from the web an experimental study. *Artificial Intelligence*, 165(1):91–134, 2005.

[22] J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.* Addison-Wesley Professional, 1999.

[23] Foundation for Intelligent Physical Agent. Specification. http://www.fipa.org, 2004.

[24] S. Franklin and A. Graesser. Is it an agent, or just a program : A taxonomy for autonomous agents. *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag*, 1(1):1, 1996.

[25] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Third International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35. Springer-Verlag, 1996.

[26] Inc Free Software Foundation. Tgnu emacs manual. http://www.gnu.org/software/emacs/manual/text/emacs.txt, 1993-2012.

[27] V. De Freitas, O. Viloria, G. Alvarez, and W. Blanco. Factores tecnológicos que inciden en la adopción de las herramientas case en las organizaciones venezolanas. *ESPACIOS. REVISTA VENEZOLANA DE GESTIN TECNOLGICA*, 20(3):43–50, 1999.

[28] Olivier Gutknecht and Jacques Ferber. The madkit agent platform architecture. In *Agents Workshop on Infrastructure for Multi-Agent Systems*, pages 48–55, 2000.

[29] P .Hensgen. Manual de umbrello uml modeller. Umbrello UML Modeller Autores, 2003.

[30] P .Hensgen. Manual de umbrello uml modeller. http://docs.kde.org/stable/es/kdesdk/umbrello/umbrello.pdf, 2013.

[31] P .Hensgen. Umbrello 2.0. http://uml.sourceforge.net/download.php, 2013.

[32] K .Hornbaek. Current practice in measuring usability: Challenges to usability studies and research. *International Journal of Human-Computer Studies*, (64):79–102, 2006.

[33] C. Iglesias, M. Garijo, Gonzalez J, and J. Velazco. Analysis and design of multi-agent systems using mas-commonkads. In M.P. Singh, A.S. Rao, and M. Wooldridge, editors, *Intelligent Agents IV. Agents Theories, Architectures and Language*, pages 1–16. Springer, LNAI 1365, 1999.

[34] 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*, pages 317–330, London, UK, 1999. Springer-Verlag.

[35] S. Luke, C. Cioffi-Revilla, K. Sullivan L. Panait, and G. Balan. A multi-agent simulation environment. *In Simulation: Transactions of the society for Modeling and Simulation International*, 7:517–527, 2005.

[36] D. Martin, A. Cheyer, and D. Moran. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence: An International Journal*, 13:91–128, 1999.

[37] A. Mas. *Agentes software y sistemas multiagente, Conceptos, arquitecturas y aplicaciones.* Pearson, 2004.

[38] Microsoft. Block de notas. http://support.microsoft.com/kb/260563/es, 2013.

[39] J. Montilva. Desarrollo de aplicaciones empresariales. Informe técnico, Grupo de Investigación en Ingeniería de Datos y Conocimiento. Universidad de Los Andes, Mérida, Venezuela, 2004.

[40] H. Morillo. Reporte de instalación y pruebas del medio de gestión de servicio. Informe técnico, Universidad de Los Andes, Venezuela, Junio 2012.

[41] F. Narciso and I. Besembel. Técnicas de desarrollo de sistemas de objetos (tdso). In *XXII Seminario Interuniversitario de Investigación en Ciencias Matemáticas*, pages 440–450, Ponce, Puerto Rico, Febrero 2007.

[42] N. Nilsson. *Inteligencia Artificial: Nueva Síntesis.* McGraw-Hill, 1964.

[43] Real Academia Espa nola. *Diccionario de la lengua española.* Espasa, 22 edition, 2001.

[44] S. Nwana, T. Divine, C. Lyndon, C. Lee, and J. Collis. Zeus: A toolkit for building distributed multi-agent. *1*, 1:1, 2005.

[45] P. OBrien and M. Agents of change in business process management. *Software Agents and Soft Computing: Concepts and Applications, Lecture Notes in Artificial Intelligence Series*, 1198:132–145, 1997.

[46] G. OHare and N. Jennings. *Foundations of Distributed Artificial Intelligence.* Wiley, 1996.

[47] W3 org. Extensible markup language. http://www.w3.org/XML/, 2013.

[48] C. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, 11(6):24–29, 1996.

[49] G. Premkumar and M. Potter. Adoption of computer aided software engineering (case) technology: An innovation adoption perspective. *DataBase Advances*, 26(1):105–123, 1995.

[50] R. Pressman. *Ingenieria del Software.* Mc Graw Hill, 2006.

[51] M. Rincón, J. Aguilar, and F. Hidrobo. Generación automática de código a partir de máquinas de estado finito. *Computación y Sistemas*, 1:163–176, 2011.

[52] A. Ríos, F. Hidrobo, J. Aguilar, and M. Cerrada. Control and supervisión sytem development with intelligent agents. *WSEAS Transactions on Systems*, 6(1):141–148, 2007.

[53] J .Robertson and S .Robertson. Volere requirements specification template. 2012.

[54] S. Russell and P. Norvig. *Inteligencia Artificial: Un Enfoque Moderno.* Prentice Hall Hispanoamericana, Madrid, 2 ed. edition, 2004.

[55] Stuart Russell. Rationality and intelligence. *Artificial Intelligence*, 94:57–77, 1995.

[56] S. Schach. *Análisis y diseño orientado a objetos con UML y el proceso unificado.* Mc Graw Hill, 01 edition, 2005.

[57] I. Seilonen, T. Pirttioja, P. Appelqvist, A. Halme, and K. Koskinen. Distributed planning agents for intelligent process automation. In *IEEE InternationalSymposium on Computational Intelligence in Robotics and Automation*, pages 614 – 619, Finland, July 2003.

[58] Y. Shoham and K. Leyton-Brown. *Multiagent systems: algorithmic, game-theoretic, and logical foundations.* Cambridge University Press, 2009.

[59] Yoav Shoham. Agent-oriented programming. *Artif. Intell.*, 60(1):51–92, 1993.

[60] Trolltech. Qt4. http://qt-project.org/downloads, 2013.

[61] J. Vidal. Fundamentals of multiagent systems with netlogo examples. http://www.damas.ift.ulaval.ca/~coursMAS/ComplementsH10/mas-Vidal.pdf, 2010.

[62] N. Vlassis. A concise introduction to multiagent systems and distributed artificial intelligence. In R. Brachman and T. Dietterich, editors, *Synthesis lectures on artificial intelligence and machine learning*. Morgan and Claypool Publishers, 2008.

[63] G. Weiss. *Multi-agent System: a modern approach to distributed artificial intelligence*. MIT Press, 1999.

[64] Wikipedia. Binding. http://en.wikipedia.org/wiki/Binding, 2004.

[65] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley and Sons, 2009.

[66] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.