UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO

Instituto de Ciencias Básicas e Ingeniería

Centro de Investigación en Tecnologías de Información y Sistemas

TESIS

Ajuste genético basado en series de tiempo de modelos de Leslie

que para obtener el grado de
Maestro en Ciencias Computacionales, presenta el

Lic. José Antonio García Mejía

Director:

M. en C. Ramón Soto de la Cruz

Dr. Gustavo Núñez Esquer

Julio 2004

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO
INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍA
CENTRO DE INVESTIGACIÓN EN TECNOLOGÍAS DE INFORMACIÓN Y SISTEMAS

Oficio No. CITIS-033O/2OO4
Pachuca de Soto. Hidalgo, 05 de agosto de 2004

Lic. José Antonio García Mejía Presente

Por este **conducto** le **comunico** que el Jurado asignado para la revisión de trabajo de tesis titulado **"Ajuste genético basado en series de tiempo de modelos de Leslie",** que para obtener el grado de Maestro en ciencias **Computacionales** fue presentado por usted, ha tenido a bien en reunión de sinodales, autorizarlo para impresión.

A continuación se integran las **firmas de conformidad** de los integrantes del jurado.

PRESIDENTE:	Dr. Cherif Ben-Youssef Brants
PRIMER VOCAL:	Dr. Virgilio López Morales
SECRETARIO:	M. en C. Omar A. Domínguez Ramírez
PRIMER SUPLENTE:	Dr. Joel Suárez Cansino
SEGUNDO SUPLENTE:	Dr. Julio Waissman Vilanova

ATENTAMENTE
"AMOR, ORDEN Y PROGRESO"

Dr. Roberto A. Hernández Gómez
Coordinador

c.c.p. M. en D. Adolfo Pontigo Loyola.- Director de Control Escolar
c.c.p. M. en C. Raúl García Rubio.- Director del ICBI
c. c. p. Minutario /api

Carretera Pachuca-Tulancingo Km. l 5
Ciudad Universitaria C.P. 42084
Pachuca de Soto, Hidalgo

Tel. (01771)71-72000 ext. 6738. 6735
Fax. (01771)71-72000 ext 6732
email: citis@uaeh.reduaeh.mx

Agradecimientos

En principio deseo agradecer a Dios el Señor que me permitió obtener este objetivo en mi vida, a él sea la gloria.

Quiero expresar mi agradecimiento especial a Ramón Soto C. por ayudarme a desarrollar esta tesis y aguantarme todo este tiempo, ya que pensé que en algún momento me mandaría a volar. Al Dr. Gustavo Núñez que siempre demostró un interés por este tema y que finalmente se ve concretado con esta tesis. Quiero agradecer también a los profesores en general del CITIS que aportaron gran parte de mi formación en este grado de maestría durante estos años. Mis agradecimientos al tribunal por las observaciones hechas y que pulieron este trabajo. Al Dr. Virgilio por sus correcciones que me ayudaron mucho, a Ornar Domínguez que desde que lo conozco ha sido de apoyo y de ánimo para mi formación académica, a Julio Waisman que siempre nos dio ánimo para terminar, y especialmente a Sherif por su tiempo, observaciones y correcciones detalladas que me hicieron darme cuenta de cuan importante es este tipo de documento de tesis. A Mariano Pozas por las pequeñas platicas ocasionales que me dieron ánimo de terminar. A mis compañeros de generación por la compañía de todos estos años y el apoyo mutuo que nos dimos. A don Maree que limpió mi lugar de trabajo estos años. Al personal administrativo que facilitaron las cosas para llevar a cabo este trámite. Quiero expresar además que mi vida se conforma del apoyo moral y que gran parte de ese apoyo se lo debo a Genoveva mi mamá, que siempre estuvo apurándome a terminar y por recordame todas las mañanas de mi compromiso. A Fernel mi papá que siempre fue de apoyo y sustento. A mi esposa amada que ha estado todo este tiempo sintiendo lo mismo que yo cada día. Especialmente a Noe por apoyarme en el momento que más lo necesite. A Euler por su consejos que siempre me fueron de edificación.

Todo tiene su tiempo, y todo lo que se
quiere debajo del cielo tiene su hora.
Ec. 3:1

Este trabajo está dedicado a mi esposa Lupita.
Su paciente comprensión ha permitido que llegue
al final de esta etapa de mi vida.

Resumen

En este trabajo se presenta un sistema multiagente llamado *Genetic Collective Leslie (GECOLE),* para la predicción de dinámicas de poblaciones. Para la implementación de *GECOLE* se desarrolló un modelo predictor, el cual tiene como componentes principales un algoritmo genético, codificación genética, evaluación de aptitud y agentes predictores. Los agentes utilizan modelos de Leslie como mecanismo de razonamiento.

En el transcurso de los años 40's se descubrió que la dinámica de una población, podría ser representada en rangos de edades tal que aportara resultados importantes; representando la información en rangos vitales contenidos en una *tabla de vida* y representando esa información por una matriz. En la actualidad, los modelos basados en esta representación son comúnmente referidos como *modelos de matriz de Leslie.*

Un problema importante al utilizar los modelos de Leslie representados en matriz es la elección de los parámetros que definen su comportamiento. Estos modelos describen una población estructurada por edades en una matriz, que incluye factores como la sobre vivencia y fertilidad para estimar los cambios existentes en una población.

En este trabajo se presenta una técnica que utiliza un algoritmo genético conducido por series de tiempo de una población, a fin de obtener los parámetros de un conjunto de matrices de Leslie. Se espera que este conjunto de matrices modele adecuadamente la población en estudio de manera colectiva.

Las mejores matrices obtenidas por el algoritmo genético son empleadas como mecanismo de razonamiento en bs agentes. Cada agente realizará su tarea de manera independiente, aportando una solución individual. Finalmente los agentes realizan predicciones individuales del comportamiento del sistema, las cuales son empleadas para obtener una solución global. La solución final se presenta al tomar la media aritmética de los resultados de los agentes predictores.

El sistema *GECOLE* se desarrolló utilizando la plataforma MultiAgent Development Kit (MadKit) y el modelo organizacional Aalaadin.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

La dinámica de poblaciones es un área de investigación que se enfoca al modelado y simulación de sistemas poblacionales, analizados como sistemas dinámicos.

Las aproximaciones que existen para realizar el modelado y la simulación de sistemas dinámicos son variadas, dependiendo, de la complejidad del sistema en estudio. La complejidad del sistema está determinada por diversos factores como son: un número grande de variables que afectan al sistema, información que en ocasiones es difícil de obtener por medios comunes (es decir la observación, muestras de la población, hábitos alimenticios), existencia de no linealidades, y altos niveles de ruido en las mediciones de las variables que describen al sistema.

Los sistemas poblacionales son considerados como sistemas dinámicos complejos. Una aproximación que ha proporcionado buenos resultados para la simulación de poblaciones es el uso de los llamados Modelos de Leslie. Estos modelos describen una población estructurada por edades en una matriz, que incluye factores como la sobre vivencia y la fertilidad para estimar los cambios existentes en una población.

Sin embargo, un problema de la técnica de Leslie es que, a fin de sintonizar los parámetros de los modelos, se requiere mucha información y experiencia, tanto en lo referente a su utilización como al objetivo que se desea modelar.

En este trabajo, se presenta una técnica que utiliza un algoritmo genético conducido por series de tiempo que representan los cambios de una población, a fin de obtener los parámetros de un conjunto de matrices de Leslie. Se espera que estas matrices modelen adecuadamente la población en estudio. Las mejores matrices obtenidas por el algoritmo genético son empleadas como mecanismos de razonamiento en agentes.

La aportación principal del presente trabajo es el desarrollo de un sistema multiagente llamado *Genetic Collective Leslie (GECOLE),* para la predicción de una población en estudio.

El sistema tiene como componentes principales un algoritmo genético, codificación genética, evaluación de aptitud y un equipo de agentes predictores. Los agentes utilizan modelos de Leslie como mecanismo de razonamiento.

Finalmente los agentes realizan predicciones individuales del comportamiento del sistema, las cuales son empleadas para obtener una solución global mediante un proceso simple de agregación, mediante la media aritmética de los resultados de los agentes predictores.

1.1. Objetivos

Objetivo general

Desarrollar una técnica que permita ajustar por medio de un algoritmo genético diferentes modelos de Leslie mediante series de tiempo, para emplearlos como mecanismo de predicción de agentes en un sistema multiagentes.

Objetivos específicos

- Desarrollar un algoritmo genético para ajuste de modelos de Leslie basado en series de tiempo de la población global.
- Desarrollar un Sistema Multiagentes para obtener la predicción colectiva de una serie de tiempo.
- Desarrollar una clase de agente predictor basada en el modelo de Leslie.

1.2. Justificación

En muchos casos, al estudiar la dinámica de un sistema se carece de información suficiente, inclusive acerca de las dependencias entre las diversas variables que intervienen en el problema, de tal manera que toda la información con la que se dispone, se limita a un conjunto de observaciones presentada de forma gráfica, la cual se define como una serie de tiempo. Una serie de tiempo es una sucesión de observaciones que se recoge de forma periódica y homogénea a lo largo del tiempo. Las series de tiempo concentran toda la información del sistema y en muchos casos es la única fuente de información disponible.

Al disponer sólo de tal información el proceso de modelado consiste en representar estas series de tiempo en un modelo, tradicionalmente discreto, el cual se espera que proporcione mayor información que los mismos datos [Morrison, 1991].

Los algoritmos genéticos ofrecen un método de búsqueda subóptima en espacios de soluciones grandes. Estos algoritmos son guiados evolutivamente hacia las mejores áreas en el espacio de búsqueda y evalúan cada solución del conjunto de soluciones para determinar la mejor solución.

En este trabajo se utiliza la aproximación de algoritmos genéticos, para ajustar los parámetros sobre el conjunto de posibles soluciones representado por varios modelos de Leslie, y se busca obtener modelos apropiados que se ajusten a un comportamiento observado en una serie de tiempo.

Los Sistemas Multiagentes (SMA) presentan la característica de distribuir tareas que son llevadas a cabo por agentes individuales. A estos agentes se les asignan procesos para obtener soluciones individuales del problema, y de esta manera se evalúan modelos complejos por medio de modelos parciales, esperando obtener un mejor resultado al obtener una solución global.

La aproximación que se presenta es una alternativa intermedia entre los modelos cualitativos (como las ecuaciones de Lotka-Volterra o las matrices de Leslie convencionales) y los métodos numéricos (como las redes neuronales, los algoritmos genéticos, o la solución numérica de ecuaciones).

1.3. Antecedentes

El trabajo que se presenta aquí, es parte de un proyecto de investigación apoyado por Conacyt denominado *"Simulación de sistemas dinámicos basada en sistemas genéticos multiagentes: aplicación a la predicción de demanda de energía eléctrica".* En este proyecto se desarrolla una metodología para la predicción de series de tiempo provenientes de sistemas complejos. El objetivo de este proyecto es el desarrollo de la metodología citada y su aplicación a la predicción de la demanda de energía eléctrica.

Como otros trabajos relacionados con el proyecto mencionado y actualmente en desarrollo en el Centro de Investigación en Tecnologías de Información y Sistemas (CITIS), se pueden citar:

- Desarrollo de un sistema multiagente llamado (SiCAE) Sistema Cooperativo de Agentes Expertos [Palacios *et.* a/,2002]. Este sistema esta formado por sistemas expertos, basado en un modelo de cooperación con el propósito de resolver colectivamente problemas complejos.

Los agentes expertos tienen una base de conocimientos independientes, de acuerdo a su área de experiencia. La base de conocimiento de cada agente experto se desarrolló en una máquina de inferencia para aportar soluciones inteligentes.

Como caso de estudio se planteó el desarrollo de agentes en el área de medicina. Esta investigación corresponde a una tesis de maestría elaborada por Ana Leticia Palacios Coyoli.

¦ Sintonización Genética de un PID Predictivo Difuso. En este trabajo se busca fuzificar las partes proporcional e integral del controlador fpPID.

Se diseña además un algoritmo de sintonización genética para el controlador resultante del fpPID. Se pretende mejorar el desempeño del controlador para mediciones con ruido y en sistemas complejos, mediante la fuzificación de las partes proporcional e integral y sintonizando el controlador difuso por medio de programación genética. Este trabajo se está realizando como tesis de maestría en ciencias computacionales por Luis Heriberto García Islas.

¦ Predicción del mercado de divisas mediante sistemas multiagentes heterogéneos. En este proyecto se presenta una metodología para la simulación de sistemas dinámicos débilmente estructurados[1] basada en sistemas genéticos multiagentes. Esta metodología ha sido designada como *ill* —Σ

El sistema *ill* —Σ emplea un sistema multiagente que simula a un conjunto de expertos organizados con el fin de analizar un sistema débilmente estructurado. Cada agente en el sistema analiza el problema en estudio desde su propia perspectiva y experiencia [Soto & Núñez, 2002].

La solución se obtiene como resultado de una discusión entre los especialistas. Los expertos que no aportan información útil durante un cierto periodo de tiempo son reemplazados por nuevos agentes. La aptitud de los individuos para obtener la solución de un problema dado es evaluada en términos del impacto de cada agente individual en el desempeño global del sistema, más que en su habilidad individual [Soto & Núñez, 2003].

[1] Proyecto Conacyt 3827-A **Simulación de Sistemas Dinámicos Basada en Sistemas Genéticos Multiagentes:** *Aplicación a la Predicción de demanda de energía eléctrica.*

1.4. Estructura del trabajo

El presente trabajo está estructurado de la siguiente manera:

- **Capítulo 2, Dinámica de poblaciones.** En este capítulo se da un panorama sobre las aproximaciones realizadas para la solución de problemas dinámicos en poblaciones, además se enfatiza el funcionamiento del modelo de Leslie y su importancia. También se abordan las extensiones que se han estudiado acerca de este modelo de estimación enfocado al comportamiento de una población.

- **Capítulo 3. Programación evolutiva de sistemas multiagentes.** Presenta un panorama general sobre la evolución de los paradigmas de programación, así como de los sistemas multiagentes, mencionando sus orígenes y tipos de agentes, enfatizando su organización; finalmente se presenta un esbozo acerca de los agentes evolutivos.

- **Capítulo 4. Modelo predictor.** Consiste en una descripción detallada del modelo que se propone basado en el modelo de Leslie. Para realizar el proceso genético, se describen las partes que componen al algoritmo genético predictor que realiza el tratamiento de las series de tiempo.
- **Capítulo 5. Desarrollo del sistema.** Describe el análisis y diseño de la aplicación desarrollada, así como los principales componentes que la integran.
- **Capítulo 6. Análisis de resultados.** Se presentan y analizan los resultados obtenidos a partir de las pruebas efectuadas del modelo predictor propuesto.
- **Capítulo 7. Conclusiones y trabajo futuro.** Se describen las conclusiones que se obtuvieron en este trabajo de tesis y las posibles perspectivas.

Capítulo 2

Dinámica de poblaciones

En este capítulo se presenta un panorama sobre los trabajos realizados para la obtención de solución de la dinámica de poblaciones. El estudio de la dinámica de un sistema se conoce por presentar un comportamiento complejo, y la información con que se dispone se limita a un conjunto de observaciones. Para la solución de estos sistemas, se mencionan las técnicas que existen para la predicción de series de tiempo. En esta sección se aborda también el estudio sobre el modelo de Leslie, iniciando en su estructura básica, y presentando ejemplos de su funcionalidad.

Se abordan también las extensiones que existen sobre el modelo de Leslie, tomando en cuenta las diferentes extensiones del modelo de Leslie, como las extensiones estocásticas. Se presentan las simulaciones para el modelo de Leslie generadas de manera estocástica. El resultado de estas simulaciones es generar series de tiempo de poblaciones hipotéticas, con comportamientos de *fluctuaciones ambientales.*

2.1. Aproximaciones clásicas

El estudio de poblaciones se interesa en estudiar los cambios existentes en las poblaciones, así como los factores que conllevan a su dispersión, migración, extinción, etc. La ecología de poblaciones se encarga de estudiar estos factores y de conocer además los procesos que afectan la distribución y abundancia de las poblaciones animales y vegetales [Odum, 1983].

Una primera contribución significativa al estudio de poblaciones fue hecha por Thomas Malthus [Malthus, 1798], quien hizo notar que la demanda de recursos naturales en una población que aumenta, en determinado momento tiene que exceder el suministro; esto conlleva al aumento de competencia por los recursos como alimentos y refugios. Este concepto es denominado *"lucha por la existencia".*

Posteriormente Harry Smith (1935), propuso la distinción de factores de mortalidad dependiente e independiente de la densidad. Los factores dependientes son de efecto facultativo, es decir que los individuos dependen de sus aptitudes y capacidades para adaptarse al medio que los rodea, estos individuos son los que tienen mayor probabilidad de sobre vivencia. Estos factores también son llamados de mortalidad facultativa, donde en una población sobrevive el *"mejor adaptado",* teoría propuesta primeramente por Charles Darwin (1859) en "El origen de las especies".

Los factores independientes de la densidad son los que tienen efecto catastrófico o mortalidad catastrófica, como el clima, insecticidas, etc. El estudio de estos factores son importantes, sin embargo, dado que no se pueden predecir las catástrofes, estos factores no son tomados en cuenta en el contexto del presente trabajo.

Un factor de mortalidad dependiente de la densidad es identificado como aquel que causa grados de mortalidad variables en la población sujeto, y en la cual el grado de mortalidad causado es una función relacionada con la densidad de la población afectada, y comúnmente puede involucrar un efecto retardado. Un ejemplo se puede observar en la Figura 2.1.

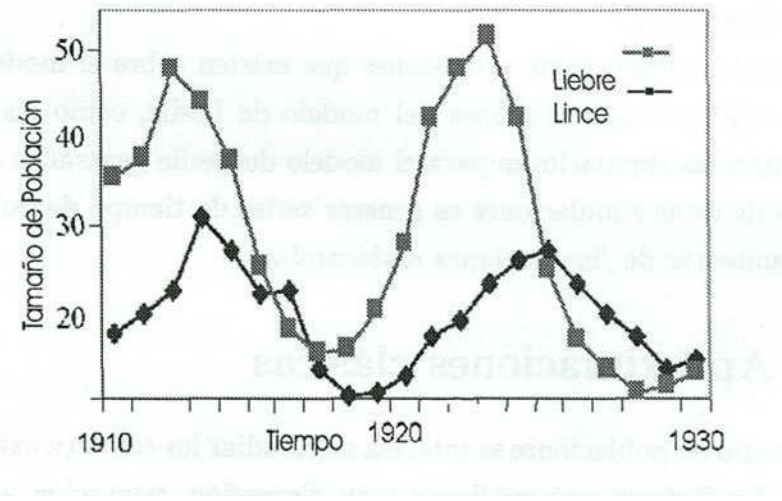

Figura 2.1: *Dinámica de poblaciones que representa a la liebre pata de nieve y su depredador el lince canadiense [MacLulich, 1937].*

En la Figura 2.1, se pueden apreciar los cambios que existen en la interacción de las dos poblaciones, en donde por ejemplo la población de linces disminuye cuando a su vez disminuye el número de presas (liebres), pero los efectos aparecen de manera retrasada.

A.J. Lotka (1925) y V. Volterra (1926), en el transcurso de la década de 1920, presentaron

una aproximación de modelos matemáticos que representa la interacción depredador/presa. Este fue el primer intento de representar matemáticamente un modelo de población que llegara a un balance cíclico con un promedio (característico) de la media de la *densidad* de la población en estudio, aplicado a lograr un equilibrio dinámico [Lotka,1925].

El modelo de Lotka-Volterra asume que la eliminación de la presa está en función de la densidad de esta, en relación con la oportunidad de encuentro y no sólo con el número de los enemigos naturales. De acuerdo con este modelo, las poblaciones de la presa y del depredador fluctúan de una manera regular, comportamiento que Volterra llamó *la ley de ciclos periódicos* [Abramson & Zanette,1998]. Los trabajos de Lotka-Volterra sentaron la base para otros investigadores quienes propusieron que los factores de mortalidad dependientes de la densidad mantienen una regulación en la población.

Las investigaciones sobre la regulación de poblaciones basadas en los factores de mortalidad dependientes de la densidad presentadas por Finklestein [Finklestein & Carson, 1985], establecen que una población tiende a tener un balance natural. Esta idea se considera como la esencia de la teoría del *"Balance de la Naturaleza"*, que inicialmente fue propuesta por A.J. Nicholson [Nicholson & Bailey, 1935].

Investigadores como Berryman [Berryman,1981], comentan que como en la investigación que W.R. Thompson sostuvo, no era necesario proponer tal mecanismo de regulación de la población, ya que el medio ambiente nunca se mantenía continuamente favorable o desfavorable para cualquier especie, porque si tal fuera el caso, la población inevitablemente se? volvería infinita o disminuiría hasta la extinción. Thompson y sus colegas sostenían que era más correcto decir que las poblaciones realmente siempre estaban en un estado de *"equilibrio dinámico"* con sus medios ambientes [Thompson, 1956].

Todas las aportaciones que inicialmente se hicieron sobre los estudios de poblaciones, fueron de carácter teórico y desarrolladas de manera deductiva a partir de experimentos controlados en laboratorio u observaciones de campo, resultando en una tendencia a la sobre simplificación.

Los estudios de estimaciones del crecimiento teórico global de una población basado en estructuras de edades, fue una contribución hecha por Leslie, presentando, entre otros resultados, su crecimiento interno, su descripción de la distribución de una población clasificado por edades en el tiempo [Leslie, 1945].

2.2. Predicción de series de tiempo

Cuando se realiza el estudio de la dinámica de un sistema, se encuentra que en muchos casos se carece incluso de información acerca de las dependencias entre las diversas variables que intervienen en el problema. En un ecosistema suele ocurrir que toda la información de la que se dispone, se limita a un conjunto de observaciones recogida de forma regular y homogénea a lo largo del tiempo, denominada serie de tiempo.

Al disponer sólo de tal información, el proceso de modelado consiste en mapear estas series de tiempo en un modelo, típicamente continuo, el cual se espera que proporcione mayor información que los mismos datos [Morrison, 1991].

A finales de los años 20 's, Yule inventó la llamada técnica auto regresiva para pronosticar el número anual de manchas solares [Yule, 1927], la cual dio como inicio a lo que se llamó *predicción moderna* [Diez *et. al,* 1999].

Al principio de las investigaciones acerca de la predicción de series de tiempo, se quería obtener un comportamiento interesante sobre sistemas lineales. Era necesario asumir la intervención de un factor externo como ruido, que afectaba al sistema lineal. De esta manera las series de tiempo eran generadas por sistemas lineales afectados por ruido, y estas investigaciones culminaron con la metodología ARIMA de Box y Jenkins [Box & Jenkins, 1976]. Ahora, estos métodos son conocidos de manera genérica como procesos ARIMA *(AutoRe-gresive Integrated Moving Average)* [Makridakis *et.* al, 1983].

Sin embargo se presentaron casos simples de estudio para los que esta última metodología es poco adecuada. Por una parte el hecho de que series temporales aparentemente complicadas puedan ser generadas por ecuaciones muy simples, pone de manifiesto la necesidad de realizar estudios teóricos mucho más generales para el análisis y predicción de series de tiempo; por otra parte estos modelos no incluyen, en general; una descripción causal entre las diferentes variables. De esta manera fueron surgiendo trabajos en los que se trataban series no estacionarias o no lineales, con nuevos métodos: modelos bilineales, biespectrales, de umbral, entre otros [Tong, 1990]; [Priestley, 1981].

En la actualidad, las capacidades de las computadoras y el desarrollo de técnicas de aprendizaje automático que surgieron dentro de la Inteligencia Artificial (IA) han permitido nuevas áreas de investigación que se han utilizado para el estudio de sistemas dinámicos no lineales, y que han dado resultados favorables en distintos casos de estudio. Entre estas técnicas se encuentran las Redes Neuronales Artificiales (RNAs), la Lógica Difusa (LD), el Razonamiento Basado en Casos (RBC), y recientemente los Algoritmos Genéticos (AG) y los Sistemas Multiagentes (SMA).

Entre los trabajos que se han hecho sobre estas técnicas se encuentran los resultados que han aportado las redes neuronales artificiales (RNAs), las cuales son modelos matemáticos inspirados en la organización y el funcionamiento de las neuronas biológicas. Existen numerosas variantes de RNAs que están relacionadas con la naturaleza de la tarea que se ha asignado. De la misma manera existen también distintas variaciones sobre cómo modelar la neurona; en algunos casos se asemeja mucho a las neuronas biológicas mientras que en otros, los modelos son muy diferentes.

Los trabajos desarrollados enfocados a estas aproximaciones, se reconocen como nuevas aplicaciones en el área de series de tiempo. Por mencionar algunos trabajos como el de Pino Diez y Priore Moreno, que presentan la aplicación de RNAs al cálculo de previsiones a corto plazo en el mercado eléctrico [Diez *et.* a/,1999], presentando resultados favorables y una comparación con los métodos tradicionales. El trabajo realizado por Weigend y sus colaboradores, es aplicado a la predicción de manchas solares; otro trabajo que realizaron fue la predicción de los cambios en el mercado de divisas con RNAs conectadas [Weigend *et.* al, 1999].

Sobre la técnica de lógica difusa se encuentran aproximaciones realizadas, por mencionar algunas, la de Min-Soo Kim y Seong-Gon Kong [Min-Soo & Seong-Gon, 2001], y el de Soto y Núñez con el trabajo de conjuntos difusos dinámicos para predicción de series de tiempo [Soto & Núñez, 2001].

2.3. La matriz de Leslie

Los estudios recientes de poblaciones estructurados por edades comenzaron con las investigaciones de la *"tabla de vida",* realizados por Graunt y Halley en el siglo XVII.

La tabla de vida manifestaba un programa de mortalidad en cada cohorte (grupo de determinada edad en la población o generación), esta representación aportaba más información, debido a que se estructuraban las etapas de vida de la población e indicaban los cambios que ocurrían principalmente en la densidad de la población afectada. Estas ideas fueron principios de los modelos estructurados por edades [Cullen, 1985].

En el transcurso de los años 40's se descubrió que la dinámica de una población en rangos de edades podría ser representada en forma tal que aportara resultados importantes, mostrando la información en rangos vitales contenidos en una *tabla de vida* y representando esa información por una matriz. En la actualidad, los modelos basados en esta representación son comúnmente referidos como *modelos de matriz de Leslie* [Leslie, 1945].

El modelo de Leslie pertenece a la categoría de los modelos dependientes de la densidad, I el modelo fue diseñado fundamentalmente para describir la distribución de individuos en una población en rangos de edades [Cullen, 1985].

Se considera que la aportación de un modelo de población se presenta cuando hace una estimación (generación tras generación) de los cambios en la abundancia o para explicar por qué ocurren cambios cuando las poblaciones llegan a determinadas densidades.

2.4. Estructura del modelo de Leslie

2.4.1. Notación

El modelo de Leslie pretende describir los diversos procesos ecológicos que afectan a la mayoría de las poblaciones, entre los cuales se pueden mencionar: el desarrollo o crecimiento de la población a través de un ciclo de vida, la mortalidad y la reproducción en una edad específica.

Las variables y parámetros que se toman en cuenta para el modelo de Leslie son:

- $\vec{N}_{x,t}$= número de organismos de la edad *x* en el tiempo *t.*
- $\vec{S}_x$ = denota la supervivencia de los organismos de la edad *x* en el intervalo de edad *x* a *x*+1; este elemento describe la transición de una clase a otra, a través de su maduración o su metamorfosis.
- $\vec{m}_x$ = denota el número de la media de la descendencia producidos por una hembra que se encuentra en el rango de edad de *x* a *x* + 1

El modelo de Leslie toma en cuenta dos ecuaciones:

$$\vec{N}_{x+1,t+1} = \vec{N}_{x,t}\vec{S}_x \tag{2.1}$$

$$\vec{N}_{o,t+1} = \sum_{x=0}^{n} \vec{N}_{x,t}\vec{m}_x \tag{2.2}$$

donde la Ec.(2.1) representa el desarrollo y la mortalidad, mientras que la Ec.(2.2) representa la reproducción de los individuos en estudio especificando el número de individuos en el primer rango de edad. El número de descendencia es sumado sobre todos los rangos de edad de las hembras.

2.4.2. Representación matricial del modelo de Leslie

Al combinar la Ec.(2.1) con la Ec.(2.2) el modelo de Leslie es descrito mediante una matriz. Esta representación describe el crecimiento, mortalidad y reproducción de los organismos (individuos) de una población, estructurada por edades.

La estructura del modelo de Leslie de forma matricial esta dada por la Ec.(2.3):

$$\tilde{N}_{t+1} = \hat{A}\tilde{N}_t \tag{2.3}$$

donde

$$\tilde{N}t = (N_{0,t}, N_{ht}, ..-, N_{n,t})^T \tag{2.4}$$

es el vector transpuesto que representa la distribución de edades de la población en el tiempo *t,* y contiene el número de organismos (individuos) de la población en el rango de edad *x* a *x* + 1 en el tiempo *t* (la edad es cuantificada en la misma unidad que el tiempo).

Usualmente, sólo las hembras son consideradas y los machos son ignorados, porque como regla, el número de machos no afecta el crecimiento de la población.

La matriz $\hat{A}$ contiene los coeficientes dados por los porcentajes de descendencia y supervivencia. La matriz $\hat{A}$ se define en la Ec.(2.5),

$$\begin{pmatrix} m_0 & m1 & ... & m_{n-1} & m_n \\ S_0 & 0 & ... & 0 & 0 \\ 0 & S_1 & ... & 0 & 0 \\ \vdots & ... & ... & ... & ... \\ 0 & 0 & ... & S_{n-1} & 0 \end{pmatrix}_{nxn}, \hat{A} \in R^{nxn} \tag{2.5}$$

donde cada coeficiente de la matriz de transición pertenece al conjunto *R,* de los números reales y representan la contribución de cada clase en la población, el vector N_t, se define por contener coeficientes que pertenecen al conjunto de los números reales *R:*

$$\tilde{N}_t \in R^{nxl} \tag{2.6}$$

y representan los factores que determinan el comportamiento de cada individuo de la población en la matriz.

2.4.3. Clasificación de la información para el modelo de Leslie

Para modelar el comportamiento de alguna población, la información en estudio es re presentada generalmente en una tabla. Esta información debe ser estructurada en el modele De acuerdo a la información de la población que se obtiene mediante observación, se clasifica en dos categorías.

La primera categoría se presenta cuando la información se categoriza uniformemente. E estado *x* de los individuos en la población esta clasificado por edad de la misma longitud (como se muestra en la Cuadro 2.1).

Edad	m_x	sT
0	*OM*	
1	0,04	0,82
2	0,39	0,79
3	0,47	0,75
4	0,48	0,69

Cuadro 2.1: *Información censada anualmente de una población. Clasificación por edad.*

El modelo se define de forma *"clasificado por edad"* al asignar la información en el modelo d Leslie. Esta información presenta una matriz estructurada como la matriz *B* en la Ec.(2.7) que tiene elementos diferentes de cero en la primera fila y en la primera subdiagonal.

$$B = \begin{pmatrix} 0,84 & 0,04 & 0,39 & 0,47 & 0,48 \\ 0,82 & 0,0 & 0,0 & 0,0 & 0,0 \\ 0,0 & 0,79 & 0,0 & 0,0 & 0,0 \\ 0,0 & 0,0 & 0,75 & 0,0 & 0,0 \\ 0,0 & 0,0 & 0,0 & 0,69 & 0,0 \end{pmatrix} \quad (2.7)$$

La información que se presenta en el Cuadro 2.1 es parte de la información de una poblado] de ovejas que fue censada cada año después de la estación de cría. El vector rh_x en el Cuadro 2.1 define datos porcentuales del número de descendencia, y el vector S_x representa datos porcentuales de supervivencia.

Generalmente este tipo de clasificación se usa en el modelo de Leslie, pero existen diferentes modelos que se han propuesto, los cuales presentan modificaciones en la estructura básica.

La segunda categoría se puede apreciar cuando la información se clasifica de manera no uniforme como se observa en el Cuadro 2.2. El estado *x* representa una clasificación de edades diferentes de una etapa del ciclo de vida.

Edad	m_x	S_x
0,7	0,84	
1,5	0,64	0,22
1,9	0,89	0,69
2,3	1,47	0,25
4,7	0,78	0,19

Cuadro 2.2: *Información acerca de una población clasificada por etapa.*

El modelo es *"clasificado por etapa",* cuando la información de la población en estudio se estructura en la matriz y se definen los elementos en cualquier posición en la matriz.

$$C= \begin{pmatrix} 1,6 & 0,2 & 0,5 & 0,4 & 0,2 \\ 0,3 & 0,6 & 0,0 & 0,0 & 0,0 \\ 0,0 & 0,4 & 0,5 & 0,0 & 0,0 \\ 0,0 & 0,0 & 0,3 & 0,6 & 0,0 \\ 0,0 & 0,0 & 0,0 & 0,3 & 0,5 \end{pmatrix} \quad (2.8)$$

Sin embargo se ha considerado en general que un modelo es *clasificado por etapa,* si los elementos con valores diferentes de cero asignados en la matriz están en la primera fila, la diagonal principal, y la primera subdiagonal [Caswell, 1989] como se muestra en la matriz *C* de la Ec.(2.8) con la correspondiente estructura.

2.4.4. Ejemplos del modelo de Leslie

Se presentan a continuación algunos ejemplos desarrollados con el modelo de Leslie que describen diferentes comportamientos de acuerdo a la elección de los parámetros. En las gráficas resultantes se toma al eje de las ordenadas como un factor de tiempo; y al eje de las absisas, como un porcentaje del tamaño de la población en estudio.

En una primera simulación, se definen en la matriz de la Ec.(2.9) los valores de una población. Los valores fueron tomados arbitrariamente para ejemplificar los diferentes comportamientos que pueden suceder al elegir diversos parámetros en el modelo de Leslie.

Cada columna representa una fase de vida de los organismos (individuos) en un estado específico. El número en la intersección de una columna *i* y un renglón *j* indica cuantos organismos en el estado *j* son producidos por un organismo en el estado *i,* Es importante mencionar que en el modelo de Leslie, el estado de un organismo está definido sólo por la edad.

Para el caso de la matriz dada por (2.9), por ejemplo la segunda columna corresponde a la edad *a* = 2, la hembra de edad 2 produce m_2 = 5,0 descendencia de edad *a*=1 (primera celda en la columna), y tendría una transición a la clase 2, con una probabilidad S_2=0,6 (celda de la subdiagonal).

a=	1	2	3	4

$$M_1 = \begin{pmatrix} 0,5 & 5,0 & 15,0 & 10,0 \\ 0,1 & 0,0 & 0,0 & 0,0 \\ 0,0 & 0,6 & 0,0 & 0,0 \\ 0,0 & 0,0 & 0,3 & 0,0 \end{pmatrix}$$

En la gráfica de la Figura 2.2 se aprecia el resultado obtenido generando la serie de tiempo a partir de la matriz M_1. Se observa por ejemplo, al ver la transición en el tiempo que la población al principio se mantiene sin crecer, hasta que llega a *t* = 40 presenta un crecimiento exponencial con tendencia a estabilizarse en t = 85. Este comportamiento es resultado de la elección particular de los parámetros definidos inicialmente en la matriz dada por (2.9).

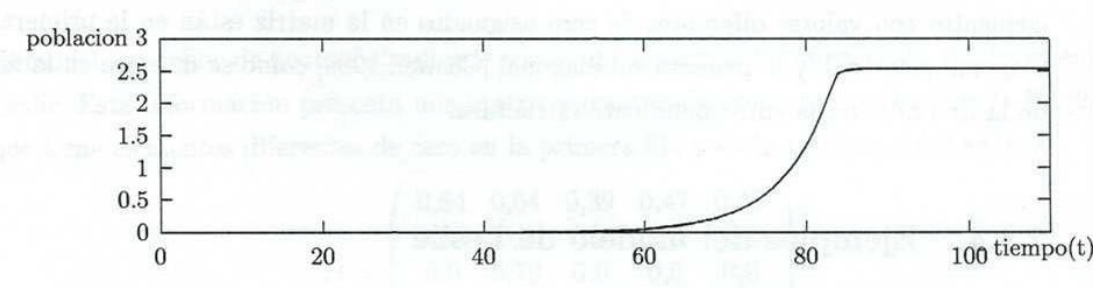

Figura 2.2: *Crecimiento de la población con la matriz de Leslie M_1.*

En la realización de una segunda simulación, se define en la matriz dada por (2.10) la información de otra población. Los datos para esta segunda simulación se tomaron arbitrariamente.

$$M_2 = \begin{pmatrix} 6,2 & 0,3 & 0,2 & 13,1 \\ 0,1 & 0,0 & 0,0 & 0,0 \\ 0,0 & 0,6 & 0,0 & 0,0 \\ 0,0 & 0,0 & 0,3 & 0,0 \end{pmatrix} \qquad (2.10)$$

En la gráfica de la Figura 2.3 se observa el resultado obtenido. Se puede apreciar para este caso, que la población crece exponencialmente, iniciando en *t* = 7 hasta *t* = 15. Esta gráfica nos demuestra que con los valores definidos inicialmente, la población tendrá un crecimiento

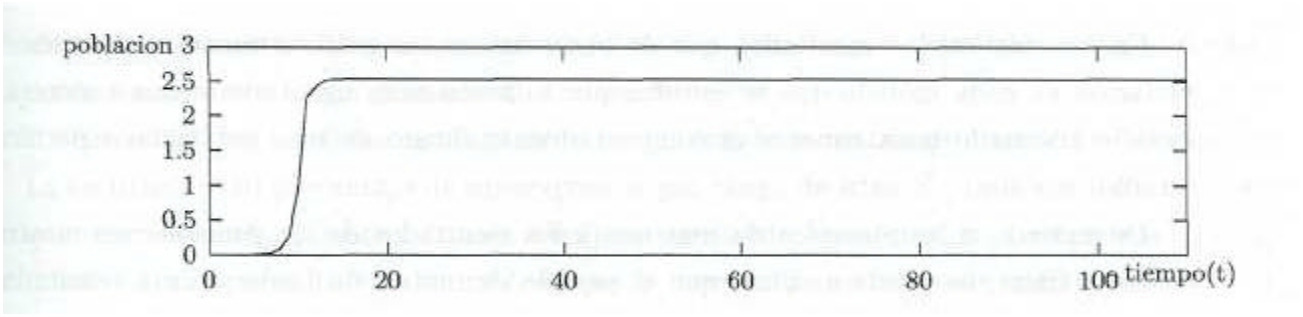

Figura 2.3: *Crecimiento de la población con la matriz de Leslie M_2*

muy rápido, y una estabilización en el tiempo restante. Los valores críticos representan los cambios importantes en la población, producidos por los datos definidos inicialmente en la matriz, y que representan, por mencionar un ejemplo, factores internos como enfermedad que afectan a la población.

En la realización de la simulación basada en la matriz dada (2.11), se definen valores arbitrarios de una población. Los valores asignados para el factor de descendencia son más pequeños que los valores de sobrevivencia para cada rango de edad, de esta manera las hembras producen poca descendencia. Estos factores hacen denotar que la tendencia de la población en estudio, para este caso, se extinga.

$$M_3 = \begin{pmatrix} 0,2 & 0,1 & 0,5 & 0,9 \\ 0,1 & 0,0 & 0,0 & 0,0 \\ 0,0 & 0,6 & 0,0 & 0,0 \\ 0,0 & 0,0 & 0,3 & 0,0 \end{pmatrix} \tag{2.11}$$

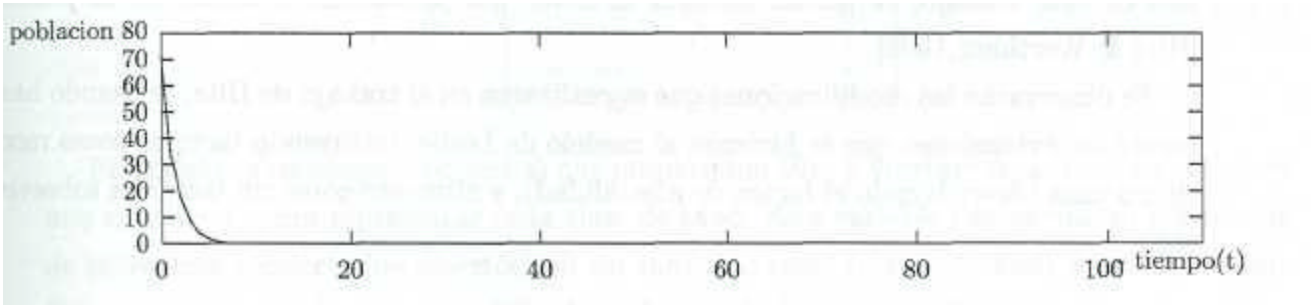

Figura 2.4: *Simulación de una población con la matriz de Leslie con tendencia a la extinción* M_3.

En la gráfica de la Figura 2.4 se aprecia la simulación obtenida. Se observa que la población no tuvo oportunidad de crecimiento, debido a los valores asignados en el factor de supervivencia.

En conclusión, los resultados que se observan en las gráficas mencionadas describen variación en cada modelo que se estudia, por lo tanto sería una tarea ardua encontrar un I modelo adecuado para conocer el comportamiento futuro de una población específica en estudio.

De acuerdo a los ejemplos de matrices y los resultados de las simulaciones mostrados en las gráficas, se puede analizar que el modelo de matriz de Leslie aporta resultados de acuerdo a la especificación de los valores propuestos inicialmente en los parámetros de las matrices. Esta definición presenta un crecimiento o extinción de la población. Esto depende de la población específica en estudio. Al final se logra la estimación de un comportamiento intrínseco de la población global.

2.5. Extensiones del modelo de Leslie

El modelo de Leslie describe los cambios que existen en la población, basándose sólo en factores de supervivencia y fecundidad en una distribución interna de la población.

Sin embargo, existen otros factores que pueden afectar a las poblaciones tales como: la temperatura ambiente, diversas características de su hábitat, depredadores naturales, comida, etc. Para representar estas tendencias, los elementos del modelo de Leslie pueden ser remplazados por parámetros que especifiquen rangos de supervivencia y reproducción en función de varios factores.

En el trabajo de Hitz y Werthener se observa una modificación del modelo de Leslie. Este trabajo presenta una extensión del modelo básico de Leslie. La principal característica de este trabajo, es incluir factores externos que modifican el estado de la población [Hitz & Werthner, 1996].

Se observarán las modificaciones que se realizaron en el trabajo de Hitz, revisando básicamente las extensiones que se hicieron al modelo de Leslie, incluyendo factores como recolección o caza (describiendo el factor de mortalidad), y alimento como un factor de sobrevivencia.

2.5.1. Extensiones de Hitz y Werthener

Hitz y Werthener destacaban que en el modelo original de Leslie, la transición del último rango de edad, de la población no era afectada por el factor de mortalidad. Entonces, plantearon la necesidad de extender el modelo de Leslie introduciendo dos nuevos componentes: un modelo que representa una política de recolección (cosecha) o sesgo y un modelo de

alimento representado por un crecimiento dinámico. Estos dos factores alimentan al modelo poblacional dando mayor realismo al modelo de Leslie.

Estos factores implican cambios en la estructura de la matriz de transición del modelo. La sustitución del porcentaje de supervivencia por rango de edad S_x, (que son los individuos que se mueven al siguiente rango de edad), es sustituido ahora por dos parámetros b_x y a_x, donde b_x es el porcentaje de individuos que se mueven de un rango de edad x a otro $x + 1$ (de una clase de edad a otra), y $a_x = 1 — b_x$ es el porcentaje de individuos que se quedan en el rango de edad x.

Los factores representan el hecho de que no todos los miembros de un rango de edad pueden moverse al siguiente rango, representando así una fragmentación más fina en el rango de edad. Esta sustitución es representada en la matriz B de la Ec.(2.12):

$$\hat{B} = \begin{pmatrix} a_0 & 0 & \dots & 0 & 0 \\ b_0 & a_1 & \dots & 0 & 0 \\ 0 & b_1 & \dots & 0 & 0 \\ \vdots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & b_{n-1} & a_0 = 1 \end{pmatrix} \qquad (2.12)$$

La descendencia de los organismos representada por m_x en la matriz de la Ec.(2.13) queda igual (esto significa que individuos de diferente edad pueden mostrar características semejantes, por ejemplo para la procreación), y tiene que tomar lugar antes de que exista una cosecha (cierto porcentaje de muertes) en la población.

$$\hat{M} = \begin{pmatrix} m_0 & m_1 & \dots & m_{n-1} & m_n \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \dots & \dots & & \vdots \\ 0 & 0 & \dots & 0 & 0 \end{pmatrix} \qquad (2.13)$$

El modelo de recolección (cosecha) que propusieron Hitz y Werthener, se basa en proponer una variable h_x para representar cada clase de edad. Esta variable representa un porcentaje de individuos recolectados (muertos) en un rango de edad (clase de edad) en cada tiempo. Este modelo de recolección esta definido en la matriz H de la Ec.(2.14):

$$\hat{H} = \begin{pmatrix} h_0 & 0 & \dots & 0 & 0 \\ 0 & h_1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & .. \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & h_n \end{pmatrix} \qquad (2.14)$$

El modelo de crecimiento de alimento está basado en un modelo logístico de la forma:

$$z(t + 1) = z(t) + rz\{t)(l - z(t)/K) \qquad (2.15)$$

donde z es el volumen de alimento, r es el rango de reproducción y K es la capacidad de acarreo. Este modelo se simplifica asumiendo que el alimento disponible es proporcional a la probabilidad de sobrevivir de un individuo en base al alimento en su habitat. Si v_x corresponde a una probabilidad de sobreviencia, entonces se tiene:

$$v_x = f_x z \qquad (2.16)$$

donde f_x es un valor de probabilidad. Esta consideración se representa en la matriz V dada por (2.17). Se toma también en cuenta que un individuo de un rango de edad x (clase de edad) depende de la comida disponible.

$$\hat{V} = \begin{pmatrix} f_0 z & 0 & \dots & 0 & 0 \\ 0 & f_1 z & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0\ 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & f_n z \end{pmatrix}$$

Finalmente la extensión que se realiza al modelo de Leslie original $\vec{N}_{t+1} = \hat{A}\vec{N}_t$, queda de la siguiente manera:

$$\vec{N}_{t+1} = (\hat{B}\hat{V}\left[\hat{I} - \hat{H}\right] + \hat{M}\left[\hat{I} - \hat{H}\right])\vec{N}_t$$

donde $\hat{I} \in R^{(n+1)x(n+1)}$ representa la matriz identidad.

De esta manera en cada paso de tiempo, después de que se lleve a cabo la fecundidad, la población es menguada por la política de cosecha, los sobrevivientes son calculados de acuerdo al alimento disponible, y finalmente la generación actual pasa al siguiente rango de edad. Sin embargo contrario al modelo de Leslie, donde todos los miembros del último rango de edad mueren, en este modelo los miembros de la población de un rango de edad x pueden sobrevivir con un probabilidad f_x.

De acuerdo a Hitz y Werthener, esto aporta más realismo al método de Leslie. Pero estos trabajos realizados aún cuando extienden el modelo de Leslie, observan una limitación principal. Para aplicar este método a una población en estudio, se tienen que elegir igualmente los parámetros que rigen el comportamiento de la población para obtener los resultados deseados.

2.5.2. Ilustración

En el Cuadro 2.3 se muestran datos para obtener la gráfica logística de *z(t* + 1), de la Ec. (2.15), que sirven para especificarlos en la matriz *V* de sobre vivencia de la matriz dada por (2.17). La proporción de comida disponible comporta en un hábitat se representa en la gráfica de la Figura 2.5.

z	r	K	z(t+l)
12	0.5	95	17.24210526
25			34.21052632
38			49.4
45			56.84210526
56			67.49473684
69			78.44210526
78			84.97894737
86			90.07368421
92			93.45263158
131			106.1789474
151			106.4947368

Cuadro 2.3: *Información para el modelo logístico del crecimiento del alimento.*

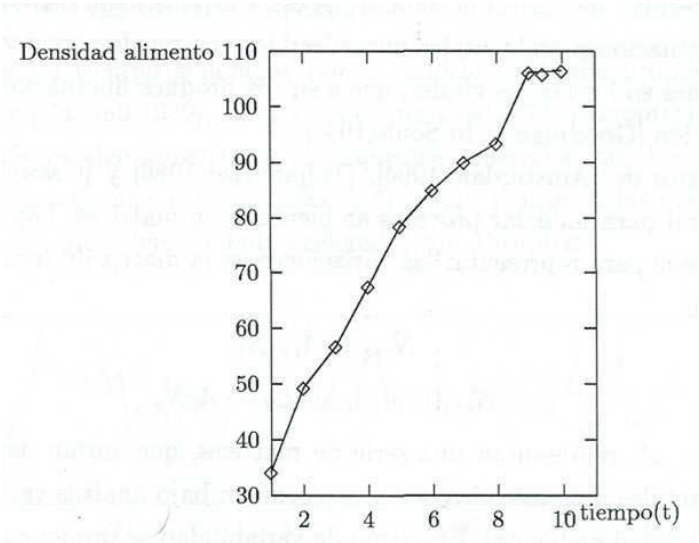

Figura 2.5: *Ejemplo de la gráfica logística del crecimiento de alimento.*

2.6. Extensión estocástica del modelo de Leslie

En esta sección se analizará las extensiones estocásticas sobre el modelo de Leslie. El estudio de estas aproximaciones consideran factores ambientales y demográficos con efectos estocásticos, para presentar la simulación del comportamiento de una población en estudio. Se presenta la forma de introducir factores estocásticos representados por factores ambientales en la matriz de Leslie. Esto sirve de base para generar series de tiempo hipotéticas, que simulen un comportamiento con variaciones temporales dinámicas en la población.

2.6.1. Efectos estocásticos

Los efectos estocásticos son típicamente incorporados dentro de los modelos de población en dos formas, que corresponden a dos diferentes recursos de variaciones estocásticas en las poblaciones naturales.

El primer efecto es la *estocasticidad demográfica,* que muestra un comportamiento que surge debido al hecho de que los nacimientos y muertes por unidad de tiempo son impredecibles, causando que el número de individuos en una población cambie constantemente.

Los modelos que incluyen este tipo de efecto tratan los rangos vitales como probabilidades, de esta manera el número de nacimientos y muertes es representado por variables aleatorias.

El segundo recurso de variación estocástica es la *estocasticidad ambiental,* la cual surge debido a las fluctuaciones ambientales impredecibles que pueden conducir a la población a tener fluctuaciones en los rangos vitales, que a su vez produce fluctuaciones imprevisibles en la propia población [Goodman & In Soule, 1987].

En los trabajos de [Amsterdam, 1994]; [Tuljapurkar, 1986] y [Caswell, 1989] se presenta una forma general para modelar procesos ambientales en matrices. Específicamente, toman al modelo de Leslie para representar las variaciones en la matriz de transición. Este modelo esta definido por:

$$\tilde{N}_{t+1} = \acute{A}_{t+1}\tilde{N}_t \tag{2.19}$$

$$\vec{N}_{t+1} = \hat{A}_t\hat{A}_{t-1}\hat{A}_{t-2}\ldots\hat{A}_0\vec{N}_o \tag{2.20}$$

donde $\hat{A}_0, \hat{A}_1 \ldots, \hat{A}_t$ representan una serie de matrices, que varían de acuerdo a las fluctuaciones ambientales. Los métodos que se encuentran bajo análisis varían dependiendo del patrón de variabilidad ambiental. El patrón de variabilidad se propone con métodos aleatorios.

La secuencia de matrices puede ser constante *($\hat{A}_t = \hat{A}$)* o periódica *($\hat{A}_k = \hat{A}_{2k}$ — $\hat{A}_{3k}$* —para algún periodo dado *k)* en el caso de ambientes determinísticos. En ambientes estocásticos,

la secuencia de matrices es generada por un proceso estocástico, y este proceso es clasificado en ambientes homogéneos y ambientes no homogéneos. Si la probabilidad de transición que dirige los cambios en el ambiente son invariantes en el tiempo, se dice que el proceso estocástico ambiental es homogéneo. El proceso estocástico clasificado como ambiente no homogéneo es conocido si la probabilidad de transición cambia en el tiempo [Caswell, 1989]. Los ambientes también son clasificados por la presencia o ausencia de auto correlación entre sucesivos ambientes.

En base al planteamiento de la Ec.(2.19), se generaron las simulaciones de series de tiempo hipotéticas, que representan el comportamiento de una población con factores ambientales.

El tamaño de la población global en el tiempo *t,* se define por la Ec.(2.21):

$$P = \sum_{x=0}^{n} \vec{N}_{x,t} \tag{2.21}$$

donde $\vec{N}_{x,t}$ representa el vector de la distribución de edades en la población. La población global denotada por P, se puede apreciar por las series de tiempo generadas de manera hipotética. La secuencia de matrices fue generada por un proceso estocástico, que representa básicamente un proceso ambiental [Nakaoka, 1993].

2.6.2. Series de tiempo sintéticas

Las series de tiempo generadas por el modelo de Leslie propuesto en la Ec.(2.19) y Ec.(2.20) por [Caswell, 1989], proporcionan resultados de comportamientos de poblaciones hipotéticas. Se pueden apreciar en los esquemas generados, que las series de tiempo tienen fluctuaciones temporales pronunciadas, y en algún tiempo *t,* las caídas (disminución de la población) o los picos (crecimiento acelerado) son abruptos.

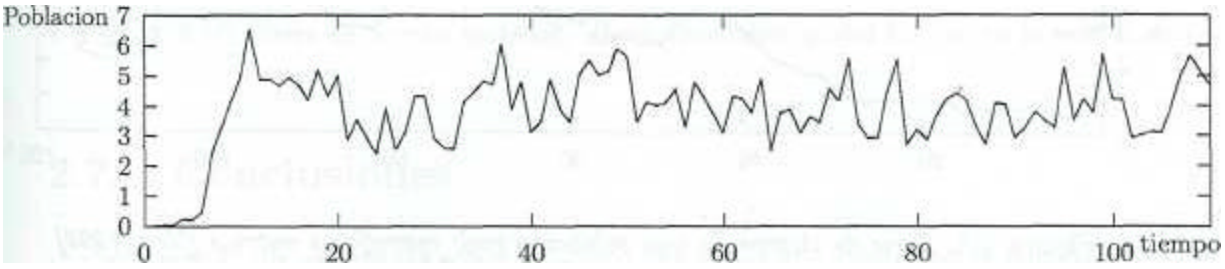

Figura 2.6: *Serie de tiempo sintética, de una población con la matriz de Leslie.*

En una primera simulación generada en la Figura 2.6, las variaciones temporales que se aprecian en el esquema, presentan fluctuaciones dinámicas parecidas a ciertos tipos de especies de animales, entre las que se pueden mencionar algunas como los insectos o los crustáceos, un ejemplo del comportamiento de una población de estas especies se puede apreciar en la Figura 2.7, que muestra el comportamiento de una población de moscas de la fruta [Nerc, 1999]. Estas especies en momentos de crecimiento acelerado se presentan como plagas, pero de la misma forma tienden a la eliminación de ciertas porciones de la población en tiempos muy cortos.

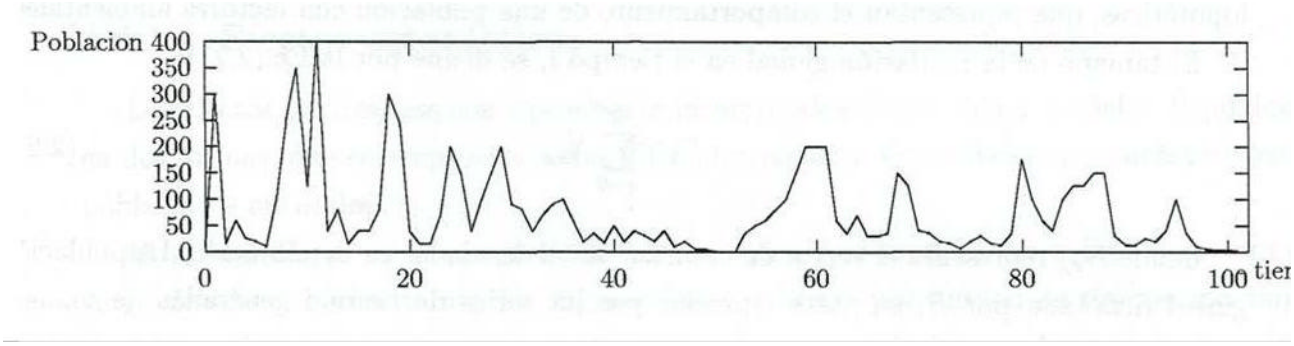

Figura 2.7: *Serie de tiempo de una población real: mosca de la fruta [Nerc, 1999]*

Se presenta una serie de datos real en la Figura 2.8. Esta serie de tiempo proviene de información recolectada de forma regular de la especie de las Nutrias [Nerc, 1999], aunque esta serie de tiempo presenta un comportamiento más estable, y no tenga mucho parecido con las series de tiempo generadas de manera sintética, se puede apreciar que en una población real existen variaciones temporales con un comportamiento estocástico.

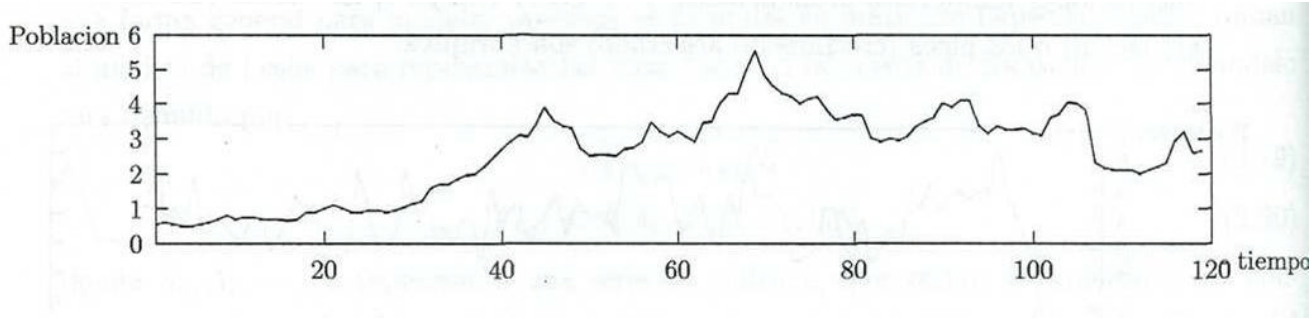

Figura 2.8: *Serie de tiempo de una población real: mamíferos nutrias [Nerc, 1999]*

Se generaron otras simulaciones de series de tiempo sintéticas. Estas series de tiempo presentan las mismas características de la simulación anterior mostrada en la Figura 2.6.

Estas aproximaciones se observan en las gráficas de la Figura 2.9 y la gráfica de la Figura 2.10. Se pretende que al generar la simulación, se haga una comparación de los resultados obtenidos del comportamiento de la población hipotética con el comportamiento de algunas poblaciones que presentan *fluctuaciones ambientales* en la población [Nakaoka, 1993].

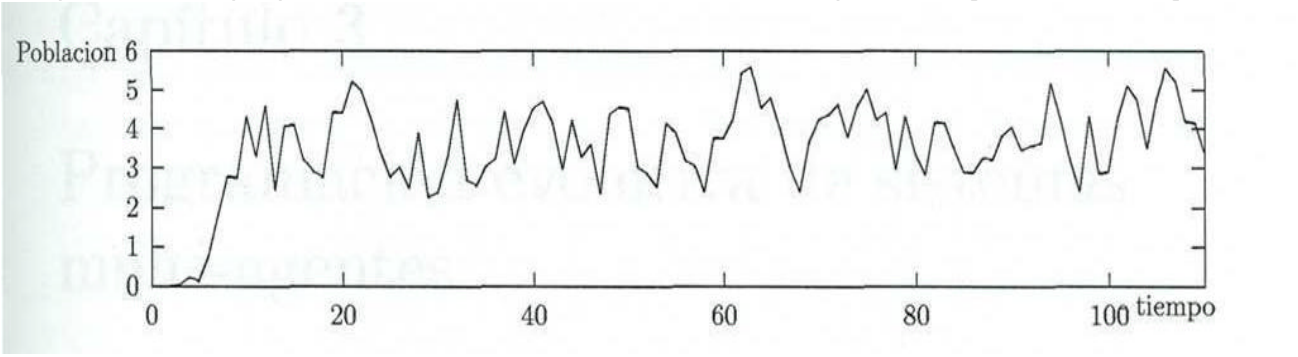

Figura 2.9: *Serie de tiempo sintética de una población global.*

Estas series se tomaron como base para probar el modelo predictor que se propone. Es importante mencionar que los resultados de las series de tiempo presentan un comportamiento estacionario, pero con variaciones temporales que es característica de los sistemas poblacionales reales.

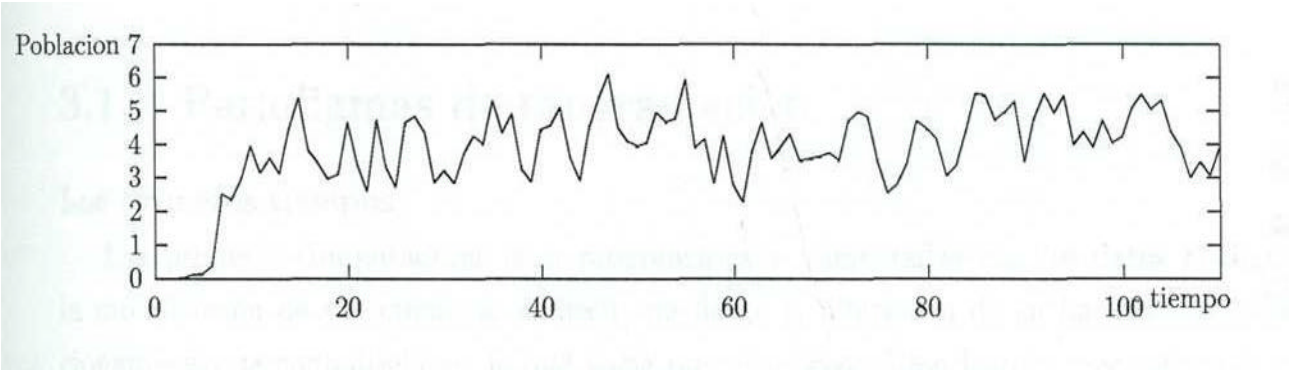

Figura 2.10: *Serie de tiempo sintética, de una población global basada en la matriz de Leslie.*

2.7. Conclusiones

Se presentó un panorama sobre las aproximaciones que existen en la solución de la dinámica de poblaciones. Se presentó el modelo de Leslie, su estructura básica y extensiones. Se presentaron ejemplos de simulación del modelo de Leslie y se observa que de acuerdo a la elección de los parámetros en el modelo, proporciona un comportamiento diferente. Se

desarrollaron simulaciones de poblaciones hipotéticas basadas en la extensión estócastica del modelo de Leslie. Estas series de tiempo se proponen para hacer una predicción basada con modelos de Leslie en el sistema multiagentes.

Capítulo 3

Programación evolutiva de sistemas multi-agentes

En el presente capítulo se describe un panorama de la evolución en los paradigmas de programación. Se describen también los tópicos referentes a la teoría evolutiva, enfatizando los conceptos de los algoritmos genéticos. Se aborda el concepto de sistemas multiagentes, mencionando sus orígenes y tipos de agentes, enfatizando las características de organización, así como las plataformas de desarrollo. Finalmente se presenta un esbozo acerca de los agentes evolutivos.

3.1. Paradigmas de programación

Los primeros tiempos

Las primeras computadoras eran programadas y alimentadas con los datos mediante la modificación de sus circuitos, es decir, mediante la alteración de su hardware o el accionamiento de conmutadores, lo que hacía que el proceso fuese lento y costoso y que no permitiese muchas opciones de trabajo [Larousse, 1991].

Posteriormente aparece el concepto de *programa almacenado* y a continuación los primeros lenguajes de programación de computadoras, los cuales intentaban aprovechar los pocos recursos que en ese entonces existían. Así, apareció el lenguaje de bajo nivel denominado ensamblador, que aunque es un lenguaje simbólico, depende mucho de la máquina en la que se desea ejecutar el programa. Además se introdujeron las pseudo instrucciones para ejecutar alguna acción. Con esto nace el concepto de instrucciones y posteriormente aparecen los lenguajes de alto nivel.

Enfoque procedural

El siguiente paso en el avance del campo de la programación consistió en el desarrollo de los lenguajes de alto nivel. La principal característica de estos lenguajes es que son más independientes de la computadora, específicamente del hardware. Uno de los aspectos que caracteriza a este planteamiento al diseñar un programa, es visualizar su secuencia de ejecución, con un organigrama o diagrama de flujo, que mostraba esquemáticamente el control de la ejecución [Larousse, 1991].

Enfoque estructurado

A finales de los años sesenta, se propuso una forma de programar sin utilizar casi en ninguna forma la instrucción *goto,* y se sustituyó por otra forma más comprensible; esta forma I estaba basada en demostrar que todo programa se podía escribir utilizando únicamente tres instrucciones de control: un bloque secuencial de instrucciones ejecutados sucesivamente, una **I** instrucción condicional *if-then-else,* y un bucle condicional *while-do, until-do, for-to, switch* I y *case/break.*

Programación orientada a objetos

La programación orientada a objetos, proporciona una nueva forma de programar, te-1 niendo como principales aspectos: la abstracción de la realidad, y considerar al programa como un conjunto de objetos que interaccionan entre sí.

Este tipo de programación fue tomado de la percepción de la realidad, facilitando a los programadores la creación de programas complejos. Los sistemas operativos visuales propiciaron que las aplicaciones se desarrollaran basándose en esta forma de programar, creando interfaces visuales que emplean botones, ventanas y menús flotantes.

Los principales conceptos que se utilizan en este tipo de programación son: objetos, clases, herencia, polimorfismo, encapsulación, método, mensaje, propiedad, atributo o variable.

Programación orientada a agentes

La conjunción de la programación orientada a objetos, más los aspectos que caracterizan al concepto de agente, dan como resultado la *programación orientada a agentes (POA)* [Guessoum & Briot, 1999].

Los mecanismos de uniformidad de los objetos proveen las facilidades para implementar la comunicación entre agentes. El concepto de encapsulación de objetos habilita la combinación de granularidad de varios agentes, y el mecanismo de herencia hace posible la especialización

de conocimiento.

Los programas como agentes presentan un comportamiento autónomo, y una característica que les permite interactuar de manera social con otros agentes. Los agentes pueden cooperar con otros agentes a través de un lenguaje común para realizar tareas de manera cooperativa.

Los investigadores presentaron un nuevo esquema, que consiste en construir sistemas computacionales como sociedades de agentes. Estos agentes deberían de contener ciertos aspectos para que pudieran funcionar como tal, los cuales deben ser [Shoham, 1991]:

- Autónomos, que ejecutan procesos concurrentemente en computadoras.
- Sistemas cognitivos, programados en términos de creencias, deseos e intenciones.
- Sistemas con razonamiento, especificados en términos de lógica, tomados de la psicología humana.
- Sistemas que se comunican pasándose diferentes tipos de mensajes.

Una definición comúnmente aceptada de *agente* es la de un sistema que actúa en un ambiente complejo y dinámico con un sentido autónomo para realizar sus objetivos; en este sistema cada agente tiene sus propios sensores, desempeñando sus propias tareas designadas [Wooldrige, 1997].

En [Shoham, 1993] se propuso que un sistema completo en programación orientada a agentes debería tener los siguientes componentes:

- Un *sistema lógico* para especificar el estado *mental*[2] y el comportamiento de un agente.
- Un *lenguaje de programación interpretado* para programar agentes usando versiones simples del sistema lógico.
- Un *proceso de agentificación* para generar agentes ejecutables por medio de especificaciones.

Shoham presentó un diseño lógico y un lenguaje de programación llamado *AGENT-0.* Este lenguaje presenta una forma de programar agentes implementando los elementos cognitivos, sin embargo las simplificaciones incorporadas al *AGENT-0* son tan extremas que el

[2] Aún sujeto a discusión por la comunidad.

lenguaje puede ser considerado como poco interesante. Existen ahora herramientas variadas que han surgido paulatinamente, presentando formas diferentes de programar a los agentes.

Sistemas multiagentes

Un *sistema multi-agentes* es un conjunto de *agentes* que interactúan y cohabitan en su medio para lograr los objetivos propuestos, mostrando características como: flexibilidad, escalabilidad y capacidad de resolver en el mundo real una serie de problemas en diversos ámbitos actuando de manera social y colaborativa [Shoham, 1993].

3.2. Programación automática

Esta programación surgió cuando se presentó la necesidad de desarrollar sistemas de forma más rápida. Esta dio inicio a un tipo de lenguajes de programación que permitía a los programadores generar el código automáticamente. Con este aspecto inicio la aparición de los lenguajes visuales, que facilitaban en gran medida la programación. Una forma particular de programación automática es la programación evolutiva.

3.2.1. Computación evolutiva

Las técnicas de computación evolutiva fueron desarrolladas haciendo una analogía con la naturaleza. Estas técnicas usan la metáfora de reproducción y adaptación como los organismos vivos que existen en ecosistemas.

Este enfoque utiliza los conceptos de selección natural y reproducción sexual, que conjuntamente dan como resultado la adaptación para el desarrollo de sistemas.

La selección natural determina los miembros de la población que sobrevivirán hasta reproducirse. Cuando un organismo falla en una prueba de aptitud, como el reconocimiento y huida del depredador, perece. La reproducción sexual garantiza la mezcla y recombinación de sus genes entre la descendencia [Ándelo & Ríos, 1997].

El problema principal de las técnicas evolutivas es construir, un *código genético* capaz de representar la estructura de distintos programas, en forma parecida como el *ADN* representa la estructura genética de algún ser vivo; a esto se le conoce como el *ADN artificial.* Es decir que el ADN artificial es la estructuración de cadenas para describir las posibles soluciones del problema, donde los genes son las variables que cambian en el transcurso del proceso evolutivo.

Estas técnicas permiten buscar y encontrar soluciones a diversos problemas de optimización y de diseño mediante un enfoque análogo con los procesos evolutivos de la naturaleza.

En su funcionamiento, las técnicas evolutivas buscan la solución mediante la exploración de espacios que son llenados de posibles soluciones, y un punto determinado en ese espacio es la solución óptima [Bentley, 1999].

A partir de estas técnicas se desarrollan algoritmos que son guiados evolutivamente hacia las mejores áreas en el espacio de búsqueda y evalúan cada solución del conjunto de soluciones para determinar la mejor; de esta manera las soluciones pueden ser comparadas con estructuras de cadenas como individuos *(adaptados)* representados en el espacio de búsqueda.

En estas técnicas existen operadores como el de *mutación* que transforma algún *gen* de la estructura del *ADN artificial.* Otro operador es la *selección* de individuos.

3.2.2. Técnicas de computación evolutiva

Las principales aproximaciones en computación evolutiva son: *programación evolutiva* (PE) creada por Lawrence Fogel (1966) [Fogel *et. al, 1966]* y desarrollada posteriormente por David Fogel (1992). *Estrategias evolutivas* (EE) creadas por Lugo Rechenber (1973) y promovidas por Thomas Báck (1996) [Báck *et.* al, 1996]; la *programación genética* (PG) desarrollada por John Koza (1992) [Koza, 1992]; y los *algoritmos genéticos* (AG) creados por John Holland (1973, 1975) y desarrollados posteriormente por David Golberg (1989) [Goldberg, 1999].

Se comenzará por proporcionar un panorama general del funcionamiento de las técnicas mencionadas, así como la comparación de las mismas:

a.- *Programación evolutiva*

La programación evolutiva fue creada originalmente por Lawrence J. Fogel, como una estrategia de optimización estocástica similar a los algoritmos genéticos, pero en lugar de hacer énfasis en la conexión del comportamiento de los padres y su descendencia, genera estrategias que son soluciones candidatas para un problema dado, y son activadas a evolucionar para encontrar la mejor estrategia.

Es asumido para la programación evolutiva, que un panorama de un problema puede ser caracterizado en términos de variables, y que hay una solución, o en su caso una solución múltiple, en términos de esas variables [Heitkótter & Beasley, 2000].

El método básico de programación evolutiva involucra tres pasos:

1. Elegir una población inicial de soluciones generadas aleatoriamente. El número de soluciones en una población es adecuado a la alta velocidad de optimización, las solucionesadecuadas para activar dependen totalmente del planteamiento del problema; así se puede evitar no malgastar las soluciones.

2. Cada posible solución *(individuo)* se reproduce en un nuevo individuo y cada uno de los individuos es transformado de acuerdo al operador de mutación antes definido; después la mutación es evaluada en base al cambio funcional impuesto por los padres.

3. Cada individuo de la descendencia es evaluado en el valor de aptitud. Comúnmente uní torneo *estocástico* se sostiene para determinar *n* individuos, después retenidos para la población de individuos, aunque algunas veces esto es llevado a cabo *deterministicamente.*

Es importante mencionar que la programación evolutiva utiliza el operador *crossover* ocasionalmente y que su principal operador es *mutación* para la determinación de descendencia.

Existen dos importantes diferencias, entre la *programación evolutiva* y los *algoritmos genéticos:*

El algoritmo genético utiliza como base el operador cruzamiento *crossover* para reproducción, y el operador de *mutación* para generar una mejor solución. Mientras que la programación evolutiva utiliza el operador de *mutación* solamente para cambiar aspectos de las posibles soluciones de acuerdo a una distribución estadística.

El algoritmo genético clásico estructura las posibles soluciones del problema en vectores representativos al problema dado. Mientras que en la programación evolutiva la representación se hace del planteamiento de un problema, y el operador de mutación no exige una estructuración lineal [Fogel *et. al,* 1966].

Para mostrar el funcionamiento de la programación evolutiva a continuación se muestra el seudocódigo básico:

```
INICIO
    t=0;                          // comienza con un tiempo inicial
    initpopulation P(t);          // población generada aleatoriamente
    evalúate P(t);                // evalúa el valor de aptitud inicial
    WHILE NOT done DO
```

```
    P(t)= mutate P(t);                    //perturba la población estocásticamente
     evalúate P(t);                       //evaluar la nueva aptitud
                                          //selecciona estocásticamente
     P(t+l)=survive P(t),P(t);            // el sobreviviente
                                         //del valor de aptitud actual
         t=t+l;                          //incrementa el contador de tiempo
   END
END
```

b.- Estrategias evolutivas

Las estrategias evolutivas fueron desarrolladas por Rechember en 1973. En esta técnica evolucionan no sólo las variables de un problema dado, sino también los parámetros mismos de la técnica utilizada (es decir, las desviaciones estándar). A esto se le llama *auto-adaptación.*

Los operadores de recombinación de las estrategias evolutivas son:

- *Sexuales:* el operador actúa sobre 2 individuos elegidos aleatoriamente de la población de padres.

- *Panmíticos:* se elige un solo padre al azar, y se mantiene fijo mientras se elige al azar un segundo padre.

Las estrategias evolutivas simulan el proceso evolutivo al nivel de los individuos, por lo que la recombinación es posible.

c- Programación genética

Esta técnica añade una importante ventaja, al estructurar un *ADN* artificial, asignando en cada *gen* una estructura de posibles soluciones representadas por árboles jerárquicos. Para finalizar la técnica obtiene un *individuo* que comprende a la solución óptima [Koza, 1992].

Los objetos que constituyen el conjunto de soluciones, no son cadenas de caracteres de una longitud ajustada, sino son programas. Cuando es ejecutado el algoritmo, los posibles candidatos de solución para el problema son buscados. El cromosoma estructurado codifica las posibles soluciones de un problema dado.

Estos programas son expresados en programación genética como árboles jerárquicos analizados sintácticamente y gramaticalmente. Los programas en el conjunto de soluciones están

compuestos por los elementos: conjunto de función *(function set)* y conjunto terminal *(terminal set),* estos son fijados por conjunto de símbolos seleccionados apropiadamente ala solución del problema de interés.

En programación genética el operador de cruza *(crossover)* identificado como el operador de reproducción, es implementado tomando aleatoriamente subárboles seleccionados como individuos en la población de acuerdo a un operador de evaluación *(fitness)* [Heitkótter & Beasley, 2000].

d.- Algoritmos genéticos

Debido a que el presente trabajo esta enfocado sobre la utilización de los algoritmos genéticos, se presenta un acercamiento a su funcionamiento general.

Los algoritmos genéticos (AG) son definidos como una de las técnicas evolutivas; esta técnica ha tenido la mayor parte de su desarrollo en aplicaciones relacionadas con problemas de optimización.

Los AG tuvieron sus inicios con el matemático John H. Holland, profesor de la Universidad de Michigan. Su funcionamiento y su estructura original fue dada a conocer con la publicación de su libro *Adaptation in Natural and Artificial Systems* [Holland, 1992]. La investigación desarrollada por Holland, sirvió de fundamento para el desarrollo de los algoritmos genéticos modernos, siendo David E. Goldberg el personaje más importante para su difusión y aplicación en ingeniería [Goldberg, 1999].

El desarrollo del algoritmo genético fue en las tres décadas pasadas, comenzando con un simple algoritmo genético. El ejemplo desarrollado fue la selección de la rueda de la ruleta, un solo punto de cruzamiento y mutaciones aleatorias directas.

Para desarrollar un algoritmo genético más rápido y más robusto se han adicionado operaciones como: *valor esperado* y *selección por torneo, cruzamiento multipunto* y *uniforme, mutaciones en escalada, compartición elitismo, micro algoritmo genético,* y los últimos trabajos desarrollados son *algoritmos genéticos difusos* [Agudelo & Ríos, 1997].

El algoritmo genético es una técnica de optimización basada en conceptos de selección natural y genética. Dentro de este método las variables son representadas como genes en un cromosoma artificial; es decir se estructuran los genes como variables dentro de una cadena de la posible solución.

Por selección natural [Pinero, 1998] y operadores genéticos, mutación y recombinación, los cromosomas con mejores aptitudes son considerados. Así se garantiza que los cromosomas con mejores aptitudes se propagarán en las futuras generaciones.

Usando el operador de recombinación, el algoritmo genético combina genes de dos cromosomas parientes para formar dos nuevos cromosomas *(hijos)* que tienen una alta probabilidad de poseer mejores aptitudes que sus parientes antecesores. La mutación permite nuevas áreas para ser exploradas en la región de soluciones. Estos algoritmos genéticos ofrecen generaciones mejoradas basadas en la función de aptitud de los cromosomas.

Los conceptos que usualmente son atendidos por esta técnica son presentados en la parte siguiente:

1. ***Cromosoma*** *(chromosome):* representación de las variables como genes en una cadena o vector. También llamada posible solución codificada o individuo en una población.
2. ***Convergencia*** *(convergence):* un gen se dice haber convergido cuando el 95% de la población comparte el mismo valor de aptitud [DeJong, 1975].
3. ***Cruzamiento*** *(crossover):* combina los genes o variables de los cromosomas para formar dos nuevos cromosomas también llamados hijos.
4. ***Valor de capacidad*** *(fitness valué):* función objetivo de un individuo, en otras palabras, es el valor que será evaluado por cada uno de los individuos dentro de la población.
5. ***Gen*** *(gen):* es la asignación de las variables en la estructura de un problema planteado.
6. ***Alelos*** *(alíeles):* estos son los valores que pueden tomar cada gen en la estructura del ADN artificial, es decir los valores que toman las variables dentro de la estructura del cromosoma.
7. ***Genotipo*** *(genotype):* representación codificada dentro del cromosoma.
8. ***Mutación*** *(mutation):* modifica aleatoriamente algún gen de la estructura del cromosoma en un cierto tiempo.
9. ***Fenotipo*** *(phenotype):* expresión física de las estructuras, llamado el conjunto decodificado de parámetros. Es el resultado que observamos en el planteamiento del problema.
10. ***Proporción de reemplazo*** *(replacement vate):* este valor especifica la fracción de la población a ser remplazada por cadenas aleatoriamente generadas.

El procedimiento que efectúan los algoritmos genéticos es el siguiente: primero una población de tamaño *n* es creada de una selección aleatoria de parámetros; cada conjunto de parámetros representa un cromosoma de un individuo.

A cada individuo le es asignado un valor de aptitud. Este valor sirve para identificar a cada cromosoma y representa al mejor individuo en su ambiente. Entonces se llevan a cabo las tres operaciones internas para crear la próxima generación: a) selección, b) cruzamiento y c) mutación.

Los individuos más aptos son seleccionados para cruzarse, mientras los individuos débiles mueren. Al aplicar el operador de cruzamiento en los *padres* se genera un nuevo cromosoma (hijo), este es incluido con el conjunto de cromosomas de la población.

Para mostrar el funcionamiento de los algoritmos genéticos a continuación se muestra un programa básico en seudo código [Agudelo & Ríos, 1997]:

```
WHILE NOT Termine DO
    FOR TamPoblación/2 DO
        Seleccione 2 individuos de la generación para aparear
        Crear 2 nuevos hijos
        Calcular el valor de aptitud de los dos hijos
        Seleccionar una nueva generación
    END
    IF Población ha convergido THEN Termine = True END
```

3.3. Agentes

3.3.1. Conceptos básicos

Como agente se puede expresar a todas aquellas aplicaciones de software que muestren un comportamiento *"inteligente".* A través del desarrollo de esta área de investigación se encuentran definiciones como las siguientes:

- Shoham define a un agente como una entidad autónoma de software la cual funciona continuamente en un ambiente particular, a menudo habitada por otros agentes y procesos [Shoham, 1993].

- Weiss define a un agente como una entidad computacional, como un programa de software o un robot, que puede ser visto como un ente dotado de perceptores y actuadotes en un ambiente y que es autónomo en su comportamiento [Weiss, 1999].

- Stuart Russel define a un agente como una entidad que percibe su entorno a través de sensores y actúa sobre ese entorno a través de efectores. Un agente es racional cuando realiza la mejor acción posible considerando sus objetivos y metas [Russell & Norvig, 1995].

- Brenner especifica a un agente de software inteligente como un programa de software que puede realizar tareas específicas para un usuario y que posee un grado de inteligencia suficiente para ejecutar parte de sus tareas de forma autónoma y para interactuar con su entorno de forma útil [Brenner *et. a*/, 1998].

- Maes define a un agente como un sistema que trata de completar unos objetivos en un entorno complejo y dinámico. Un agente es un sistema computacional que tiene una larga vida, tiene objetivos, sensores y efectores y decide autónomamente qué acciones realizar en la situación actual para maximizar el avance hacia sus objetivos [Maes, 1997].

- Wooldrige define a un agente como un sistema de software que actúa en forma autónoma para realizar sus objetivos, en un ambiente complejo y dinámico. Cada agente tiene sensores propios y desempeña las tareas que le fueron asignadas [Wooldrige, 1995].

Los agentes tienen que tener características para ser percibidos como tales, y por lo menos existen cuatro aspectos básicos que tienen que expresar [Wooldrige & Jennings, 1995]: autónomos, cooperativos, perceptivos y pro-activos.

En el presente trabajo se adopta la definición de Russel y Maes, de acuerdo al sistema multiagente desarrollado.

3.3.2. Estandarización de agentes

El esfuerzo para estandarizar la aplicación de agentes y asegurarse de su interoperabilidad continua. Estos acercamientos para la estandarización de agentes los presenta: FIPA *(Foundation for Intelligent Physical Agents),* OMG *(Object Managent Group)* y *Agent Working Group.*

La Fundación para los Agentes Físicos Inteligentes (FIPA) se formó en 1996 para producir normas de software para agentes heterogéneos, agentes que interactúan entre ellos y sistemas basados en agentes. Para la elaboración de estas normas, FIPA requiere de la colaboración de sus miembros y de la investigación de agentes en general, para construir especificaciones que puedan usarse para lograr interacción entre los sistemas basados en agentes desarrollados por diversas compañías y organizaciones.

FIPA está organizado y estructurado en dos grupos: aquellos que están involucrados en la producción y desarrollo de las normas, y los que están involucrados en el mecanismo de apoyo de FIPA.

Las especificaciones de FIPA representan el primer paso hacia el estándar de agentes. Ellos no pretenden describir la arquitectura interna o la forma de implementar los agentes, sino que especifican la interfaz necesaria para soportar la interoperabilidad de los sistemas de agentes.

Cuatro áreas de estandarización han sido identificadas por FIPA:

- **Comunicación entre agentes:** facilita la comunicación entre agentes, soporta la interacción entre ellos como negociación, cooperación y el intercambio de información. Un lenguaje de comunicación entre agentes ha sido especificado por FIPA el cual es FILA ACL, para soportar esta interfaz.
- **Administrador entre agentes:** incluye las interfaces necesarias para soportar la creación y localización de los agentes, la comunicación entre agentes así como facilitar la movilidad y seguridad.
- **Interacción de software:** interface que soporta la integración entre un agente de software y otro que no lo es.
- **Interacción de agente-hombre:** los agentes pueden interactuar con usuarios humanos y/o otros agentes.

3.3.3. Arquitecturas principales

Los agentes de software como aplicaciones tienen reacciones de acuerdo al entorno en donde se encuentran. En ese entorno pueden actuar de una forma que les permitan alcanzar sus objetivos planteados, para llevar a cabo sus metas que comúnmente son las tareas que el usuario tendría que hacer.

Existe una clasificación que se puede tomar como un acercamiento de los agentes de software, y que de forma general se toman como arquitecturas establecidas para la identificación de los mismos.

Arquitecturas de agentes.

- ***Agentes reactivos:*** la decisión tomada se lleva a cabo en forma directa de la situación a la acción.

Agentes basados en lógica: la decisión tomada es realizada a través de deducción lógica.

- ***Agentes*** **BDI(Creencias-Deseos-Intenciones):** la decisión tomada depende de la manipulación de estructuras de los datos que representan las creencias, deseos e intenciones del agente.
- ***Arquitecturas en capas:*** la decisión tomada es realizada a través de varias capas del software, cada uno de los cuales razona sobre el ambiente en diferentes niveles de abstracción.

3.3.4. Noción de agencia *(agency)*

El concepto de agencia[3] se define como el lugar en donde el agente vive, o como el lugar para que el agente gestione ciertas tareas, presentando de cierto modo servicios al usuario.

Desde el punto de vista descriptivo, existen dos usos comunes para expresar el concepto de agente: la noción blanda y la noción dura [Wooldrige, 1995]. Se describen a continuación las características que muestran a cada una de las nociones mencionadas:

Agencia blanda: esta expresión presenta al agente con ciertas propiedades para poder actuar de acuerdo a su entorno. La capacidad de comunicarse con otros *agentes* intercambiando mensajes en un lenguaje de comunicación expresivo [Wooldrige, 1995]. Estas propiedades son mencionadas a continuación:

- *Autonomía:* los agentes realizan sus actividades, tareas, etc., sin la intervención directa del usuario y tienen cierto control sobre sus acciones y su estado interno.
- *Habilidad social:* los agentes tienen capacidad para interactuar con otros agentes, mediante algún tipo de lenguaje de comunicación entre agentes, también tienen capacidad de interactuar con el usuario.
- *Reactividad:* los agentes perciben su ambiente y responden de una forma oportuna a cambios que ocurren en él, tomando en cuenta que su ambiente puede ser una interfaz con un usuario, otros agentes, etc.
- *Pro-actividad:* los agentes son capaces de mostrar comportamiento dirigido a metas, al tomar de forma autónoma la iniciativa.

[3] Segun el diccionario Vox-Larousse. Oficina o despacho del agente; dedicada a gestionar asuntos o prestar determinados servicios.

Agencia dura: para tener una apreciación sobre la noción de agencia dura, se han implementado conceptos que son utilizados muy comunmente por los humanos. Un sistema computacional tiene las características mencionadas anteriormente y además implementa nociones como que los agentes tengan creencias, intenciones y obligaciones.

Pero además existen otras propiedades que adicionalmente a las mencionadas anteriormente complementan el comportamiento de los agentes, entre las que se pueden mencionar:

- *Movilidad:* esta característica capacita al agente para movilizarse dentro de una red ***(LAN,WAM).***
- *Veracidad:* los agentes no pueden transmitir información que no sea verdadera.
- *Benevolencia:* esta característica asume que el agente siempre actúa de una forma benévola y que no realiza acciones que causen conflictos.
- *Racionalidad:* para este efecto se asume que el agente debe actuar de acuerdo a alcanzar sus objetivos, en la medida que sus creencias lo permitan.

3.3.5. Ambientes de desarrollo

Después de tener una apreciación más completa sobre los sistemas multiagentes, es importante conocer ahora los ambientes que existen para observar sus características.

Actualmente existe una diversidad de ambientes de trabajo que sirven de soporte como herramientas y que permiten desarrollar aplicaciones basadas en agentes. Todas estas herramientas tienen características propias que mantienen un interés específico en común que es la comunicación entre agentes dentro de su medio. Esta comunicación debe ser empleada mediante un lenguaje expresivo enfocado a agentes, donde se desarrolle la *habilidad social.*

A continuación se mencionan los ambientes de trabajo de mayor importancia.

El ambiente de trabajo *Zeus* presenta un ambiente integrado para la construcción de aplicaciones de agentes cooperativos en forma rápida. Esta herramienta pone énfasis en el aspecto metodológico de *Zeus* que se basa en su portabilidad y el soporte de multihilo. Esta metodología usa una descomposición de cuatro partes para el desarrollo de agentes: análisis diseño, desarrollo y soporte [Collis & Ndumu, 1999].

Existen tres grupos de clases en *Zeus,* una librería de los componentes de un agente un conjunto de herramientas visuales, y el software para construir agentes. Un agente *Zeui*

está compuesto por tres capas: *una capa de definición, una capa organizacional y la capa de coordinación.*

- La capa de definición representa las capacidades del agente BDI *(Belief-Desire-Intentions)*
- La capa de organización define las relaciones con otros agentes
- La capa de coordinación modela cada agente como una entidad

El ambiente de desarrollo fue hecho por *Agent Research Programme of the British Telecom Intelligent System Research Laboratory.*

El ambiente ***Agent Builder*** incorpora en su funcionamiento principal el desarrollo de sistemas multiagentes basándose en el modelo Agento y la arquitectura BDI. Esta herramienta proporciona interfaces gráficas para el diseño y desarrollo de sistemas multiagentes y es conocida por su alta calidad de software. Está fundamentada en el lenguaje Java y una de sus principales ventajas con respecto a la comunicación es el uso de KQML *(Knowledge Query and Manipulation Language),* que es uno de los principales estándares de comunicación en agentes.

Proporciona además librerías para el análisis del dominio del problema, herramientas para definir una agencia (colección de agentes inteligentes), integra y esta capacitado para usar librerías de Java, C y C++. [AgentBuilder, 1999].

El ambiente de trabajo ***Jack*** tiene como parte fundamental una metodología para el desarrollo de sistemas multiagentes que consiste en realizar un análisis y diseño del sistema. Este ambiente se fundamenta en el empleo del modelo BDI y dMARS y el uso del lenguaje Java. Es conocido por presentar una herramienta visual para construir de manera rápida sistemas multiagentes. Fue desarrollado por *Agent Oriented Software Pty. Ltd.,* en una compañía comercial de Australia, *Australian Artificial Intelligence Institute(AAII)* [Bussetta *et.* al, 1999].

El ambiente de trabajo ***JADE*** *(Java Agent Development Framework)* cumple con las características que exigen los estándares internacionales tanto en desarrollo como en comunicación. Este ambiente de trabajo *(framework)* fue implementado en *java* y tiene como principal ventaja desarrollar los sistemas multiagentes de acuerdo a las especificaciones que ha establecido FIPA *(Foundation for Intelligent Physical Agents). JADE* usa el lenguaje de comunicación de agentes de *FIPA* y fundamenta el desarrollo de agentes al utilizar una combinación de *sockets, RMI,* y *CORBA,* que son características básicas del lenguaje *Java* [Bellifemine, 2001].

El ambiente ***MadKit*** se fundamenta sobre el modelo organizacional llamado *Aalaadin,* este modelo presenta la conceptualización de *grupo, rol,* y *agente.* Una de sus principales ventajas es que no toma una estructura rígida para construir los agentes. Otra de sus ventajas es que se pueden desarrollar sistemas multi-multiagentes debido a su estructura organizacional en la cual se puede dividir a los agentes en grupos, regidos por un rol, y sus actividades de I acuerdo a sus objetivos interactúan en diferentes grupos [Gutknecht & Ferber, 1998].

Contrariamente a otras plataformas, MadKit es principalmente una máquina en línea de MAS *(Multi Agent System),* al usar un agente micro-kernel [Ferber & Gutknecht, 1998], Sin embargo una de sus principales desventajas es que la plataforma de desarrollo tiene que estar en línea para que los sistemas funcionen, cosa que no es muy útil si pensamos en un sistema multiagente que administre los correos electrónicos, puesto que si no están activos los agentes no pueden funcionar. Madkit se presenta como una plataforma que es flexible en su I desarrollo. Fue desarrollado por Oliver Gutnecht y Jacques Ferber en LIRMM *(Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier I).*

El ambiente ***JaTeLite:*** presenta una característica interesante, que es el desarrollo de agentes que se comunican robustamente a través de *Internet.* También proporciona una infraestructura básica basada en un agente planificador (ruteador de mensajes) el cual se encarga de actividades como administrador al realizar entre otras cosas: enviar y recibir mensajes, transferencia de archivos, invocación a otros programas entre computadoras.

Esta herramienta mantiene una plantilla para la construcción de agentes que utilizan un protocolo y un lenguaje común de alto nivel, es decir que es entendible entre los usuarios. Esta plantilla proporciona al usuario numerosas clases de Java predefinidas que facilitan la construcción de un sistema multi-agente. Las clases se proporcionan en paquetes, para que el desarrollador pueda decidir que clases se necesitan para el desarrollo de un sistema.

3.4. Agentes evolutivos

Este nuevo campo trata de unir el paradigma de *Agentes Inteligentes* con la *Computación Evolutiva,* y lo llaman *Agentes Inteligentes Evolutivos* [Cristea *et.* al, 2000]. Esta tendencia trata de aportar la suma de características de las dos áreas, al proporcionar una forma de sistemas *autoadaptables* a su entorno.

Las características propias de los *agentes* junto con los rasgos del paradigma evolutivo proporcionan a los *agentes evolutivos* la habilidad de evolucionar y adaptarse a su medio de trabajo.

Los agentes evolucionan al operar sobre la población incluyendo los mecanismos de cambios genéticos que ocurren en los agentes sobre las generaciones [Pereira & Costa, 2001].

Los trabajos que se han realizado sobre esta nueva área de investigación destaca el que realizó Sinclair llamado *Evolving Simple Software Agents: Comparing Genetic Algorithm and Genetic Programming Performance* [Sinclair &: Shami, 1997].

Sinclair presenta la eficiencia relativa de los algoritmos genéticos y la programación genética aplicada en los agentes de software. El dominio del problema consiste en un agente recolector de comida colocado en un área de cientos de células con unidades de comida extendidas uniformemente. En este trabajo se realiza una comparación del desempeño de la programación genética y los algoritmos genéticos.

Cada agente se representa por un *genotipo* en su estructura interna. Esta estructura determina su comportamiento. Parte de los recursos de conocimiento de un agente son genéticamente heredados de sus padres. Esto tiene como consecuencia que los agentes a través de su funcionamiento *aprendan,* mejorando cada vez su comportamiento inicial.

El trabajo de Grefenstette llamado *Predictive Models Using Fitness Distributions of Genetic Algorithms,* expone la construcción de modelos predictivos basada en las capacidades de los algoritmos genéticos. A los modelos se les asigna una representación, una función de aptitud y un conjunto de operadores genéticos [Grefenstette, 1995].

Una aproximación desarrollada de agentes evolutivos se realizó con el trabajo de Soto y Nuñez llamado *simulación de sistemas débilmente estructurados basada en sistemas genéticos multiagentes.* En este trabajo se expone una característica que hace que los agentes evolucionen tanto su estructura interna que determina su comportamiento como su estructura externa basada en la población de los diferentes agentes como predictores [Soto & Núñez, 2002].

3.5. Conclusiones

Se presentó inicialmente un esbozo sobre las técnicas de programación clásicas y se hizo una introducción a las nuevas áreas de programación. Se abordó un panorama amplio en los diferentes paradigmas evolutivos y sus características. Se hizo énfasis en el funcionamiento de los algoritmos genéticos.

Fue necesario incluir el estudio sobre *agentes,* siendo el tema principal para el desarrollo del sistema GECOLE en el presente trabajo. Se mencionó las características de los sistemas multiagentes, los alcances y las diferentes plataformas que existen para desarrollar sistemas basados en agentes. Se finaliza esta unidad con una introducción a los agentes evolutivos.

Capítulo 4

Modelo predictor

En este capítulo se presenta el modelo de agente predictor desarrollado en este trabajo. Así mismo, se presenta el algoritmo genético empleado por el agente encargado de ajustar los parámetros en los modelos de Leslie.

Los datos disponibles acerca del estudio de una población pueden dar lugar a grandes conjuntos de modelos, dependiendo de las variables de entrada y salida consideradas. Encontrar los mejores modelos para un sistema dinámico, donde la información disponible es incompleta, es una tarea difícil de llevar a cabo. Entonces, es pertinente tomar en cuenta la técnica de los algoritmos genéticos [Michalewicz, 1992], que resuelve el problema de explosión combinatoria en la búsqueda de tales modelos.

En este trabajo se utiliza un algoritmo genético para encontrar los mejores modelos de Leslie de una serie de tiempo. Estos modelos estructurados generados por el proceso evolutivo describen el comportamiento parcial del sistema.

4.1. Estructura del modelo

Las cadenas en el algoritmo genético son modelos parciales generados aleatoriamente. El modelo completo se obtiene mediante la reunión de un conjunto de los modelos, al finalizar el proceso genético.

Estos modelos reflejan el comportamiento del sistema bajo estudio en un determinado periodo de tiempo. Con la finalidad de generar modelos adecuados los cuales se ajusten a los datos observados, el proceso evolutivo es desarrollado partiendo de una población inicial.

El modelo predictor está determinado por un modelo de Leslie. En la estructura del modelo es modificado el vector que contiene el número de organismos de la población. En seguida, se capturan los datos de la serie de tiempo en la nueva definición de la estructura que fue tomada para su evaluación.

El modelo de Leslie tiene la siguiente estructura:

$$\vec{N}_{t+1} = \hat{A}\vec{N}_t \tag{4.1}$$

donde *A* representa la matriz de coeficientes de los porcentajes de descendencia y supervivencia, que representa un modelo parcial. El vector traspuesto

$$\tilde{N}_t = (N_{0,u} N_{1,u} \text{-,}_i N_{n,t})^T \tag{4.2}$$

contiene el número de organismos de la población por cohorte (generación), es decir N_t es el número de organismos en la cohorte *x* en el tiempo *t.*

El vector N_t es modificado en cada posición en $\tilde{N}_{n,t} = fW_t$ por

$$\tilde{N}_{x,t} = fW_t \tag{4.3}$$

donde f es una variable que representa un número fraccionario entre cero y uno, también demuestra una fracción de la población en algún rango de edad y W_t representa el número total de individuos en la población al tiempo *t* de la serie. Entonces N_x¿, es la fracción de individuos en el rango de edad *n,* de tal manera que

$$\sum_{X=0}^{n} (\vec{N}_{x,t+1} W_{t+1}) \tag{4.4}$$

A fin de evaluar el error producido por algún modelo de Leslie individual M¸ se calcula el total de individuos en la población

$$\hat{c} = \sum_{x=0}^{n} (N_{x,t+1} W_{t+1}) \tag{4.}$$

y se compara con el valor correspondiente en la serie de tiempo.

Este modelo carga la base de datos de la serie. Los datos de la serie de tiempo son procesados en la estructura mencionada. El modelo en la Ec.(4.1) tiene una etapa de entrenamiento.

Este entrenamiento presenta un tipo de aprendizaje supervisado [Hernández, 1998], debido a que los datos en la matriz de transición serán ajustados a los datos observados del sistema, mediante el proceso evolutivo. Se espera que con este proceso de corrección y ajuste los resultados tiendan hacia un pequeño margen de error, tomando en cuenta la función de ajuste.

Se tiene por consiguiente un proceso de prueba y error, donde los datos obtenidos son probados por el modelo. Los valores finales, que son generados por los modelos con menor error son tomados para generar la serie de tiempo que se ajustará a los datos observados.

4.2. Algoritmo genético predictor

Una parte fundamental del trabajo se presenta en esta sección. Como se mencionó en la sección 3.2.1, los algoritmos genéticos han sido desarrollados para realizar búsquedas grandes y extensas. Estos algoritmos realizan una búsqueda heurística y utilizan una función de ajuste para validar la calidad de las posibles soluciones.

De acuerdo al funcionamiento de los algoritmos genéticos es indispensable que exista una función objetivo que sirva para evaluar los cromosomas o cadena codificada y produzca un valor de bondad, también llamado valor de ajuste. Este valor es usado para jerarquizar un conjunto de cadenas (también llamados individuos en la población de posibles soluciones[4]), siendo las mejores posibles soluciones aquellas cadenas con mejor valor de ajuste. Como es conocido, las mejores soluciones serán seleccionadas para pasar a la siguiente generación seguido de un nuevo entrenamiento y reproducción, de esta manera mejorar la solución buscada.

Las nuevas soluciones son generadas mediante diversas operaciones que emulan los mecanismos de la naturaleza y se aplican al proceso de producir nuevas cadenas en el conjunto de cadenas, siendo las operaciones más frecuentes la mutación y la cruza.

Estas operaciones son usadas para modificar las cadenas seleccionadas, en un esfuerzo para mejorar el resultado de ajuste y acercarse mucho más al objetivo o la solución óptima [Goldberg, 1999].

4.3. Codificación genética

La representación del cromosoma (posible solución) se presenta cuando cada posible solución es codificada por los parámetros representativos de la matriz de Leslie; esto es, los parámetros de supervivencia y fertilidad. Estos parámetros desempeñan el papel de los genes en un cromosoma, es decir la cadena estructurada con los valores correspondientes. El cromosoma está compuesto en la primera parte por los factores de supervivencia y en la

[4] En este trabajo, se emplea el término **cadena** en lugar de **individuo, y** el **de conjunto de cadenas, en** vez **de población,** a fin de no confundir con *individuo* y *población* del modelo poblacional.

segunda parte por los factores de fertilidad (descendencia), como se muestra en la Figura 4.1.

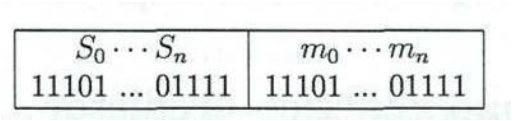

Figura 4.1: *Codificación del cromosoma.*

Los genes son referenciados por tener los valores que codifican al factor de sobrevivencia y son representados por $S_{0...}S_n$, *Si* e [0,1]. Los valores de fertilidad son representados por $m_{0..}m_n$,m_i G [0,1]. Es importante mencionar que existen varias formas para representa! los valores *(alelos)* de las variables y de la estructura del cromosoma *(genotipo).* En este caso se tomó un alfabeto binario.

La codificación adoptada del cromosoma se presenta por una longitud fija, la cual es segmentada para cada parámetro dentro de la estructura. Se tomó una longitud de veinticuatro bits para los parámetros de supervivencia. Cada segmento de seis bits representa cada uno de los parámetros de supervivencia, es decir se asigna al modelo cuatro variables.

La otra parte de la codificación consiste en tomar una longitud de veinticuatro bits para representar los parámetros de fertilidad, cada segmento se representa por seis bits para cada parámetro y se define también cuatro variables para representar los parámetros de fertilidad.

La decodificación del cromosoma para cada parámetro del modelo de la matriz de Leslie consiste en realizar la conversión a valores decimales. Los valores que se asignan a las variables pueden ser las combinaciones posibles binarias producidas por el proceso genético.

4.4. Evaluación de aptitud

Una parte importante en este trabajo es la evaluación que se realiza dentro del proceso evolutivo. Para ello, se debe tomar un factor de ajuste de cada posible solución (o individuo en la población de las posibles soluciones). Para realizar esto, el ajuste de cada individuo se evalúa en términos de los datos en la serie de tiempo y el resultado es evaluado en términos del error cuadrático medio normalizado *É,* NMSE *(Normalized Mean Squared Error)* [Simon, 2002], dado por:

$$\hat{E} = \sqrt{\frac{\sum (x_i)}{\quad}} \tag{4.6}$$

donde n es el conjunto de valores, *x* la media aritmética de *n,* x¿ el valor de la serie en *t = i, x_i* la predicción de x_i.

En el proceso de evaluación se calculan los nuevos valores de x_i con el modelo propuesto, se valida el valor resultante de la predicción con el valor propuesto por la serie de tiempo (el valor observado), en base al error generado. Esta evaluación se hace sobre el tamaño de la serie que se estimará, para cada cadena estructurada del conjunto de cadenas.

La aptitud es medida para cada uno de los modelos tomando en cuenta al final de cada generación los modelos con mayor error, esto es, los menos adecuados de acuerdo al valor de ajuste serán remplazados por nuevos modelos.

La construcción de los nuevos modelos parciales se realiza teniendo en cuenta los siguientes aspectos: el tamaño total del conjunto de soluciones posibles es de 1000 individuos. El tamaño representa 1000 codificaciones de cromosomas y cada cromosoma es evaluado con la función de aptitud en cada generación. Los nuevos modelos son generados mediante operaciones de cruza y mutación. En cada generación se reemplazan la mitad de la población por los mejores modelos resultantes.

La generación de las nuevas posibles soluciones se hará de acuerdo a los siguientes pasos:

1 Se seleccionan al azar dos cadenas del conjunto completo de soluciones posibles (ver Figura 4.2)

2 Se generan los vectores que describen los modelos parciales.

3 Se elige aleatoriamente un punto de cruzamiento y se intercambian los segmentos.

4 Se decide, para cada cadena, si se realiza una mutación en uno de sus genes seleccionado aleatoriamente (ver Figura 4.3).

5 Se seleccionan las cadenas que serán remplazadas.

6 Se inserta en la población global los nuevos individuos (modelos parciales) de la población.

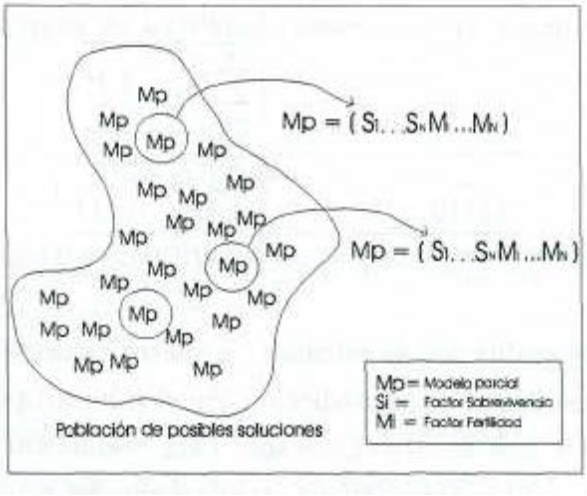

Figura 4.2: *Selección de soluciones parciales*

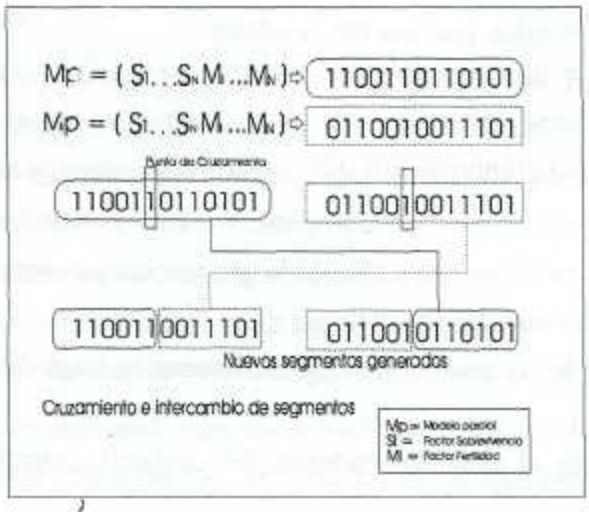

Figura 4.3: *Cruza de segmentos*

7. El total de nuevos modelos parciales que se generan y se reemplazan es la mitad de la población que se tomó en cuenta.

Este proceso se repite hasta que los modelos con mayor ajuste son obtenidos después de un número de generaciones predeterminado.

Las series de tiempo son de mucha importancia en el funcionamiento total del modelo presentado. Los modelos parciales generados para ser evaluados son conducidos por la serie de tiempo asignada como patrón de comportamiento. Cada modelo parcial se presenta como

posible solución y los resultados son evaluados con la serie de tiempo propuesta. Al final se presentan los mejores resultados con los de menor error generado. Y se realiza la predicción de la serie propuesta.

4.5. Agentes predictores

Los sistemas multiagentes presentan características apropiadas para el desarrollo del sistema en estudio. Es adecuado pensar que agentes individuales realizan el trabajo de manera independiente, distribuyendo el trabajo, de esta manera cada agente aporta una solución particular e independiente para resolver el problema de manera colectiva.

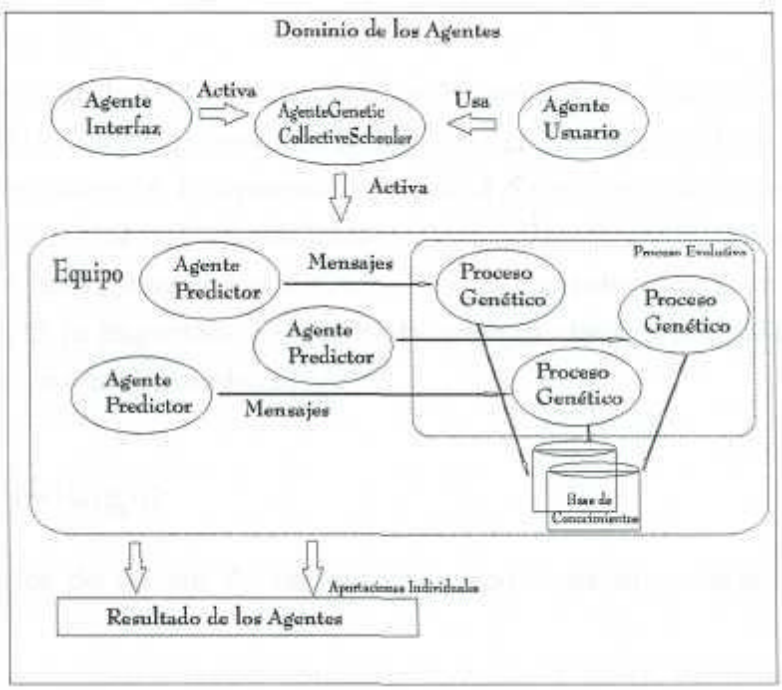

Figura 4.4: *Diagrama GECOLE*

En este trabajo, el agente *Genetic Collective Scheduler* es el agente principal que administra los agentes predictores. Cada agente tiene un mecanismo de predicción, consistiendo en un modelo parcial de Leslie, obtenido mediante el proceso de ajuste genético.

El esquema de la Figura 4.4 muestra de manera general el funcionamiento del Sistema GECOLE. En el siguiente capítulo se verá el funcionamiento con detalle y se describirá la distribución de las tareas de manera específica.

4.6. Conclusiones

Se presentó la estructura modificada del modelo Leslie. Este modelo toma en cuenta las series de tiempo que conducirán al ajuste de los modelos generados genéticamente. Se presentó también las partes del algoritmo genético que es utilizado en este trabajo.

Se presentó el procedimiento que realiza el algoritmo genético para la generación, evaluación, cruza y mutación de las soluciones parciales. Se mostró el esquema del funcionamiento general de los agentes predictores en el sistema multiagente.

Capítulo 5

Desarrollo del sistema Genetic Collective Leslie (GECOLE)

En el presente capítulo se describen las fases del análisis y el diseño del sistema *Genetic Collective Leslie* (GECOLE). Se presenta también el esquema desarrollado para el diseño del presente sistema multiagente. La aportación principal de este trabajo de tesis es el desarrollo del sistema GECOLE y los agentes predictores. De la misma forma se desarrolló una interfaz de usuario para la interacción con el sistema. En este capítulo se ve a detalle este sistema, además de remarcar la importancia de la plataforma de desarrollo Madkit, en la cual fue desarrollado el sistema multiagente (SMA).

5.1. Metodología

La representación del modelo del sistema nos proporciona una perspectiva de la sociedad de agentes.

Los diagramas de UML(Unified Modelling Language) [UML, 1999] inicialmente fueron proyectados para soportar la descripción de los diferentes aspectos de los sistemas orientados a objetos [Rumbaugh et. al, 1999]. Sin embargo, como se ha mencionado anteriormente, el concepto de agente se presenta como la extensión de la programación orientada, a objetos. Estas características nos permiten explorar ciertos lenguajes orientados a objetos con el objetivo de adaptarlos a las características que presenta el paradigma orientado a agentes. En la actualidad existan plataformas que nos permitan aplicar todos los aspectos que son referidos a los agentes.

El modelado de sistemas basados en agentes se suele realizar mediante el lenguaje de modelado unificado (UML), este modelado propone extensiones propias cubriendo los requisitos

más importantes para el modelado de agentes y SMA (Sistemas Multi-Agentes), de esta I manera se presenta AUML (Agent UML), como una solución para el modelado de sistemas basado en agentes.

Un propósito inicial de AUML es representar protocolos de interacción de agentes AIP[5]. Los diagramas de AUML y los flujos de control e información son considerados *actos* de *comunicación* (mensajes). El protocolo es representado como entidad colocando un diagrama que represente el flujo de mensajes entre los agentes. Los mensajes son representados por los diagramas de secuencia, de colaboración, de actividad y de estados. Estos diagramas son agrupados en paquetes, constituidos como una agregación conceptual de secuencias e interacción [Juchem & Bastos, 2001].

Los diagramas de interacción son representados como modelos estructurales de interacción entre los agentes. Los diagramas de secuencia y los diagramas de colaboración son subtipos de diagramas de interacción, siendo semánticamente equivalentes. La disposición de los elementos gráficos de los diagramas de secuencia enfatiza una secuencia cronológica de comunicación. En cuanto a los diagramas de colaboración, estos demuestran las asociaciones entre agentes, donde una secuencia de interacción es representada a través de numeraciones de los mensajes.

Los diagramas de actividad y los diagramas de estado dan enfoque al flujo de proceso. Los diagramas de actividad representan las operaciones y los eventos que activan estos. La diferencia con los diagramas de interacción es la representación de una manera explícita líneas de ejecución de flujos de control, lo que es particularmente útil para los protocolos de interacción, compuesto por un proceso paralelo. Los diagramas de estado se centran principalmente en los estados. Este tipo de diagramas tiene una mejor aplicación como un mecanismo de restricciones para los protocolos, siendo incorporados a los agentes con las restricciones conocidas.

5.2. Modelo Aalaadin

La arquitectura del sistema multiagente que se presenta en este trabajo, está basada en el modelo de sistemas multiagentes Aalaadin [Gutknecht & Ferber, 1997]. El modelo Aalaadin considera un modelo organizacional que presenta los conceptos de *Grupo, Agente,* y *Rol.* De esta manera, el agente se conceptualiza como una entidad capaz de comunicarse con otros

[5] AIP (Agent Interaction Protocol) describe un patrón de comunicación, el cual propone una secuencia de mensajes permitida entre agentes así como sus restricciones sobre el contenido de estos mensajes.

agentes en el medio ambiente y desempeñar roles específicos, de acuerdo a sus objetivos dentro de los grupos. Los agentes que tienen un interés común o con características comunes, forman los grupos.

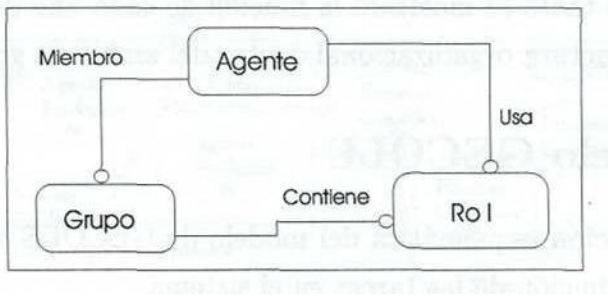

Figura 5.1: *Modelo Aalaadin*

La plataforma de desarrollo implementada a partir del modelo Aalaadin es Madkit (Multi Agent Development Kit)[Gutknecht & Ferber,1998]. Esta plataforma aporta las características necesarias para el desarrollo de los agentes, y es el ambiente en el cual se desarrolló el sistema que se presenta y en donde se desempeñará el Sistema GECOLE.

- **Agente**
  Un *agente* es especificado solamente como una entidad comunicativa activa que tiene *roles* dentro de los grupos existentes, sin adoptar alguna arquitectura específica.

- **Grupo**
  Los grupos son definidos como conjuntos atómicos de agregación de agentes. Cada agente puede ser parte de uno o más grupos. En su forma más básica, el grupo es sólo una forma de etiquetar a un conjunto de agentes, y en una forma más desarrollada, en conjunción con la definición del rol, el grupo permite representar cualquier sistema multi-agente.

- **Rol**
  El rol es una representación abstracta de la función, servicio o identificación de un agente dentro de un grupo. Cada agente puede manejar uno o más roles, y cada rol puede ser manejado por un agente local o un grupo.

Para la comunicación y el paso de mensajes, MadKit proporciona clases y métodos definidos, lo que permite la comunicación interactiva entre agentes existentes en el ambiente

La importancia de establecer una categorización establecida por el modelo organizacional de Aalaadin, coloca a los agentes en su lugar predeterminado, estableciendo los grupos adecuados para cada uno de los agentes que en particular se desarrollaron para el Sistema GECOLE. Por lo tanto se mostrará la función de cada uno de los agentes que componen al sistema y su estructura organizacional dentro del ambiente general.

5.3. Modelo GECOLE

La representación esquemática del modelo de GECOLE se presenta con la finalidad de describir la distribución de las tareas en el sistema.

La distribución de las tareas se puede hacer de forma espacial, basada en el esquema de las fuentes de información o de forma funcional de acuerdo al área de especialización de los agentes [Weiss, 1999].

El modelo de GECOLE propuesto utiliza la distribución de tareas de forma funcional y el mecanismo de estructura organizacional.

La asignación funcional de cada uno de los agentes que representa el esquema queda de la siguiente manera:

- Pg_i: proceso genético que se lleva a cabo en cada uno de los agentes predictores.
- *Ap_i.* conjunto de agentes que tiene asignado la tarea de predicción del modelo en estudio. Los agentes activados en su dominio tienen asignado un *Pg_i* que buscará el ajuste del modelo de Leslie en el conjunto de soluciones.
- *Aín:* agente interfaz que tiene la tarea de mantener la comunicación entre el usuario y los demás elementos del sistema.
- *Aad:* agente principal que tiene la tarea de activar al conjunto de *Ap_i*, y de mantener la comunicación entre ellos.
- *Au:* agente usuario que inicia a los agentes.

En el diagrama de la Figura 5.2 se presenta un panorama general del funcionamiento del sistema. Este funcionamiento es el siguiente:

- Una vez que el agente *Au* ha inicializado al agente *Ain,* se tienen tres partes principales. La primera parte es el control del conjunto de *Ap_i,* llevada a cabo por el agente *Aad,* que es responsable de la activación de los agentes.

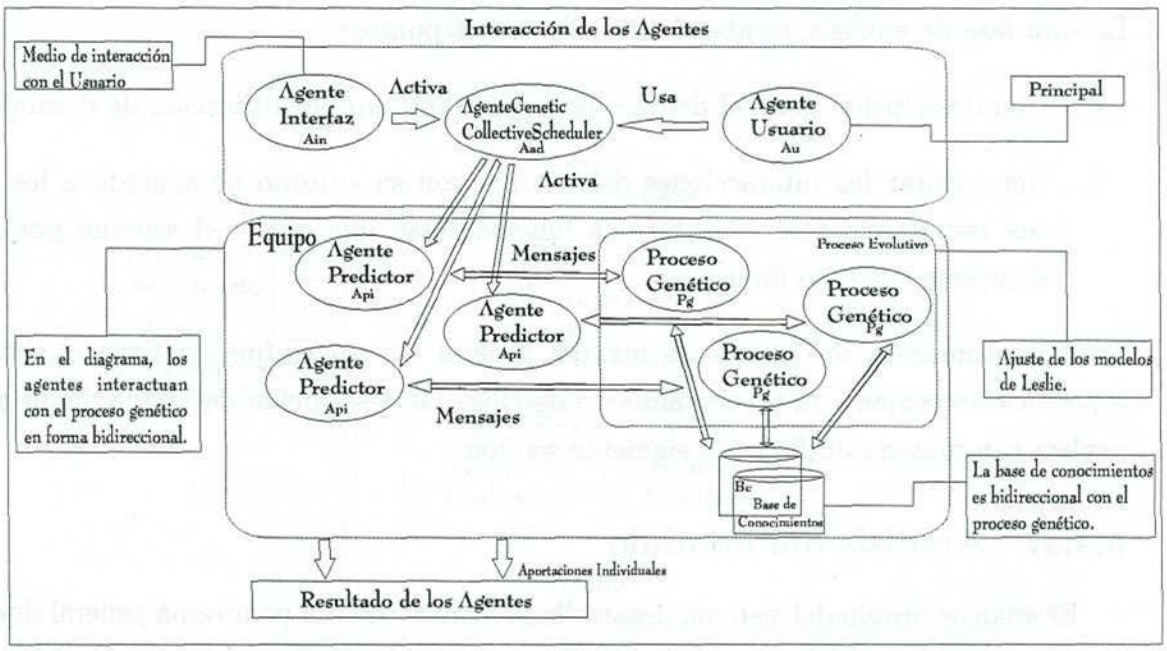

Figura 5.2: *Modelo de GECOLE*

- En la activación inicia la segunda parte. La distribución de las tareas a cada agente del equipo la cual consiste en activar a cada *Api* con el correspondiente $P_{gi,}$ para el ajuste los modelos de Leslie generados como conjunto de posibles soluciones. El $P_{gi,}$ toma de la base de conocimientos que es la serie de datos que sirve para conducir al modelo predictor al ajuste de los modelos parciales.
- Cada uno de los agentes *Api* obtendrá su conclusión individual con base en la predicción de cada uno de ellos. Los resultados individuales son mostrados al usuario en la interfaz de los *Api.*
- La tercera parte consiste en conjuntar los resultados del equipo de *Api,* para obtener la media aritmética de los resultados individuales y presentar una solución global.

5.4. Análisis de GECOLE

En esta sección se hace una descripción del escenario general, identificando los dominios de cada agente, así como los actores y roles. Es importante mencionar que el sistema GECOLE desarrollado en este trabajo de tesis, presenta como principal aportación el estudio sobre el comportamiento de poblaciones en sistemas multiagentes.

En esta fase de análisis, se abordan los siguientes puntos:

1. Una descripción general del escenario general y una identificación de dominios.
2. Representar las interacciones del sistema con su entorno de acuerdo a los actores y sus respectivos roles, también la funcionalidad que ofrece el sistema por medio de diagramas de caso de uso.

A continuación se observa de manera general las partes que contiene el sistema y se especifica brevemente el funcionamiento de ellas. La descripción detallada de los agentes se explica con mayor detalle en la siguiente sección.

5.4.1. Análisis del dominio

El análisis propio del sistema desarrollado introduce a un panorama general donde existe la interacción de todos los elementos del sistema, también se identifican los dominios y en base a esos dominios se identifican los actores y roles pertenecientes a cada agente, determinando sus funciones y sus alcances.

Escenario general

El funcionamiento sobre el escenario entre el usuario y los agentes del sistema GECOLE es el siguiente:

Al inicializar el agente principal, el usuario tiene varias opciones que le permite ejecutar, a los diferentes agentes que se clasifican de acuerdo a su funcionamiento en el sistema (ver Figura 5.3).

A continuación se explica el funcionamiento del diagrama:

1. El usuario tiene las opciones de elegir al agente que funge como la interfaz principal.
2. El Agente Interfaz se inicializa con las opciones disponibles:

    a) Agentes Predictores.
    b) Agente Editor.
    c) Agente Help.

3. Dependiendo de la elección del usuario, el Agente Interfaz inicializa a los agentes predictores.

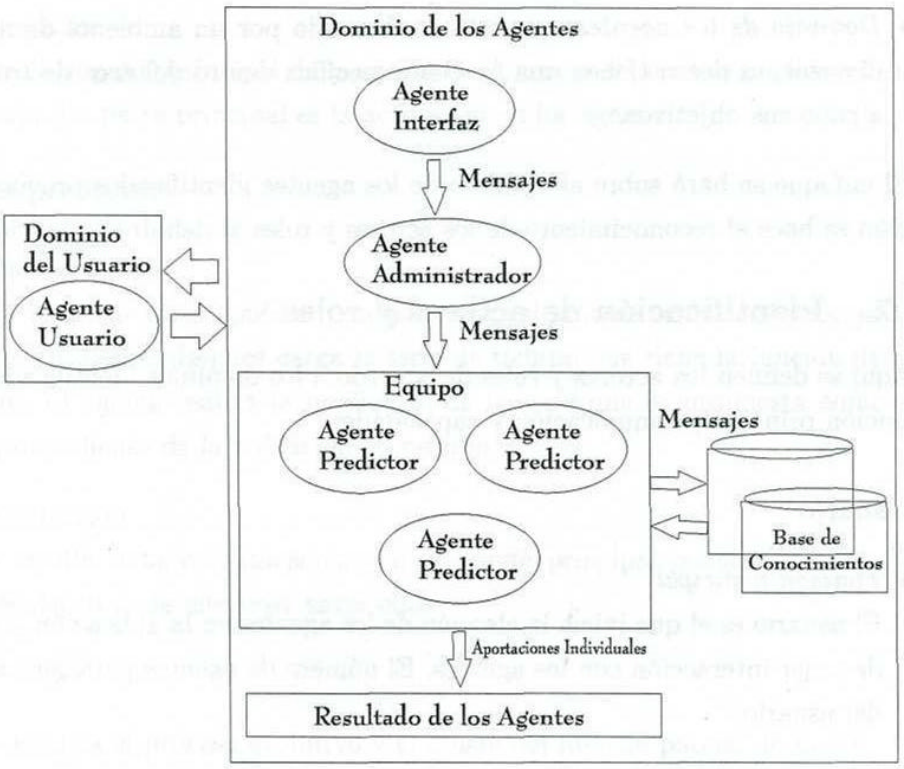

Figura 5.3: *Dominio general de la aplicación*

4. El usuario elige la opción de activar los agentes predictores, del Agente Interfaz.
5. El Agente Interfaz inicializa a los agentes predictores para conformar el equipo y realizar el proceso evolutivo. Internamente cargan la serie de tiempo y comienza el proceso.
6. En el momento que cada agente predictor es inicializado carga la base de conocimientos, que es la serie de tiempo especificada y que será procesada para el ajuste de los modelos de Leslie.
7. Los agentes predictores muestran los resultados que se van obteniendo a través del procedimiento genético y los resultados parciales de los modelos.

Identificación de dominios

De acuerdo al planteamiento anterior se puede observar la existencia de dos dominios:

¦ *Dominio del usuario,* que se encarga de tener interacción directa con los agentes.

¦ *Dominio de los agentes,* que está conformado por un ambiente de agentes con role diversos, es decir, tienen una función específica dentro del área de trabajo para lleva a cabo sus objetivos.

El enfoque se hará sobre el dominio de los agentes identificados previamente y a continuación se hace el reconocimiento de los actores y roles al definir sus funciones principales.

5.4.2. Identificación de actores y roles

Aquí se definen los actores y roles de acuerdo a los dominios identificados, estableciendo su función principal, comunicación y capacidades.

Usuario

¦ *Función principal*

El usuario es el que inicia la elección de los agentes en la aplicación y es el responsable de tener interacción con los agentes. El número de agentes para ser activados depende del usuario.

¦ *Comunicación*

La interfaz visual proporciona la comunicación entre el agente principal y el usuario.

¦ *Capacidades*

- Iniciar la aplicación
- Inicializar a los agentes para procesar la información y obtener resultados.
- Puede interactuar con los agentes generados.

Los actores que se consideran en el dominio de los agentes son los siguientes.

Agente de interfaz

¦ *Función principal*

Este agente es el responsable de manejar el enlace entre todos los componentes de la aplicación.

¦ *Comunicación*

Este agente tiene una comunicación estrecha entre el sistema, el usuario y el resto de I los agentes, el agente *Help* y el agente predictor.

¦ *Capacidades*

Controla el panel principal de la aplicación a través del cual se inicializa la Interfaz de Usuario. La parte principal es la activación de los agentes.

Agente predictor

¦ *Función principal*

El agente predictor después ser lanzado activa el proceso evolutivo que ajusta al modelo parcial de Leslie, después carga la serie de tiempo que tiene la función de conducir el ajuste. El agente realiza la predicción de la serie que es propuesta como patrón del comportamiento de la población en estudio.

¦ *Comunicación*

Este agente tiene comunicación con el agente principal *geneticCollectiveScheduler* a través del paso de mensajes entre ellos.

¦ *Capacidades*

- Realiza el proceso evolutivo y el ajuste del modelo parcial de Leslie.
- Carga la base de datos para su procesamiento.
- Realiza un monitoreo de los resultados obtenidos con los modelos generados.

Agente editor

¦ *Función principal*

Proporciona una interfaz cuyo propósito es visualizar el contenido de archivos, así como generar archivos nuevos. En esencia se utiliza este agente para observar los archivos generados que contienen los resultados del agente predictor.

¦ *Comunicación*

Este agente se comunica con el agente principal *geneticCollectiveScheduler* a través de mensajes.

¦ *Capacidades*

Mostrar, visualizar y construir archivos, presenta la posibilidad de generar archivos y editar los archivos creados. Tiene las propiedades básicas de un editor de texto como: copiar texto, pegarlo y cortarlo.

Agente Help

¦ *Función principal*

El objetivo principal es proporcionar una ayuda al usuario para conocer el funcionamiento general del sistema, así como el funcionamiento específico de cada uno de los agentes. En términos generales provee una ayuda al usuario.

¦ *Comunicación*

Este agente mantiene una comunicación con el agente principal *genetic Collectivt Scheduler.*

¦ *Capacidades*

Mostrar de manera activa una ayuda al usuario así como un panorama general del funcionamiento de la aplicación.

Todos los elementos establecidos en ésta sección se consideraron para el desarrollo del sistema GECOLE.

5.5. Diseño de GECOLE

En esta sección se diseñan los modelos que ayudarán a la implementación de la aplicación. Los modelos son diseñados partiendo de los elementos establecidos en la fase de análisis. Los modelos que se utilizan en esta fase son los: diagramas de caso de uso, diagrama de clases, diagrama de estados, diagrama de secuencias y diagrama de colaboración, los cuales se detallan en las secciones siguientes.

5.5.1. Diagrama de casos de uso

En esta sección se presentan los diagramas que muestran la interacción general y el funcionamiento de cada uno de los agentes.

Se observa el diagrama de Casos de Uso en la Figura 5.4 para detallar las acciones de los actores.

El diagrama de casos de uso se describe de la siguiente manera:

1. El agente usuario inicia la interacción mediante el agente interfaz que en este caso es *geneticCollectiveScheduler.*

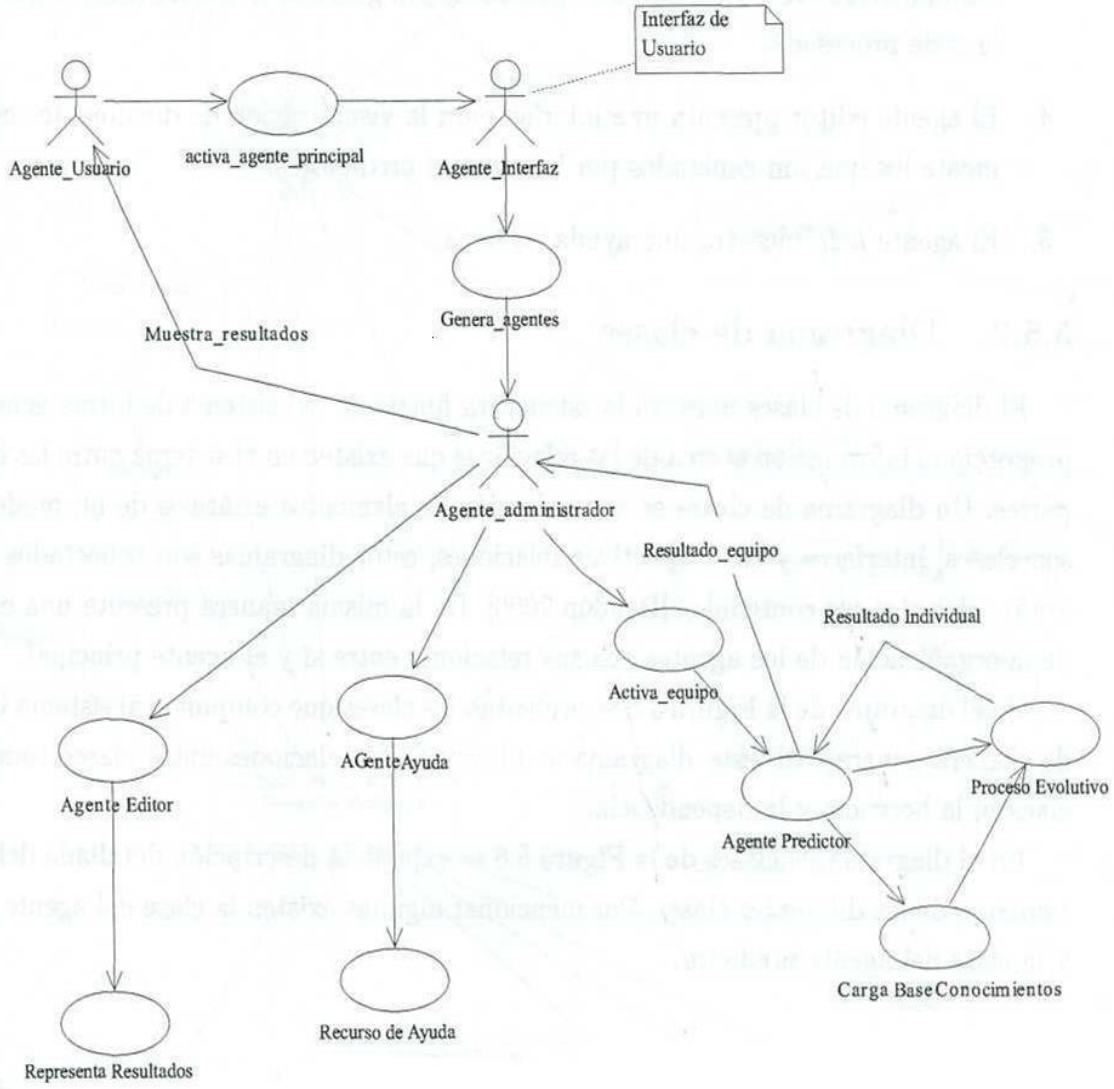

Figura 5.4: *Diagrama de Casos de Uso del sistema*

2. A través del agente *geneticCollectiveScheduler* se generarán los agentes correspondientes dependiendo la elección del agente usuario. Se puede inicializar el agente predictor, el agente editor o el agent *help.*

3. El agente predictor carga la base de conocimientos (serie de tiempo) la procesa en el módulo evolutivo y los resultados generados son guardados en otro archivo que contiene la serie procesada.

4. El agente editor presenta una interfaz para la visualización de documentos específicamente los que son generados por los agentes predictores.

5. El agente *help* muestra una ayuda en línea.

5.5.2. Diagrama de clases

El diagrama de clases muestra la estructura funcional del sistema de forma general. Este proporciona información acerca de las relaciones que existen en el sistema entre las diferentes partes. Un diagrama de clases es una colección de elementos estáticos de un modelo, como son clases, interfaces y sus respectivas relaciones, estos diagramas son conectados como un grafo entre sí y sus contenidos [Rendón, 2000]. De la misma manera presenta una estructura de la organización de los agentes con sus relaciones entre sí y el agente principal.

En el diagrama de la Figura 5.5 se presentan las clases que componen al sistema GECOLE de manera general. En este diagrama se observan las relaciones entre clases como la asociación, la herencia y la dependencia.

En el diagrama de clases de la Figura 5.6 se expone la descripción detallada del comportamiento de las diferentes clases. Por mencionar algunas existen la clase del agente principal y la clase del agente predictor.

Abstract Agent

Esta clase es propia de las libr-erías de MadKit. Es la clase principal en Madkit. Esta clase aporta el soporte necesario para que un agente presente un ciclo de vida, pase mensajes entre otros agentes y para otros agentes, además de administrar los grupos y los roles en el dominio de agentes, también puede disponer una interfaz gráfica para los agentes.

Los agentes que heredan el comportamiento de esta clase se identifica por definir a los agentes con una interfaz. El agente principal *genetic Collective Scheduler* proporciona una interfaz visual para interactuar con el usuario.

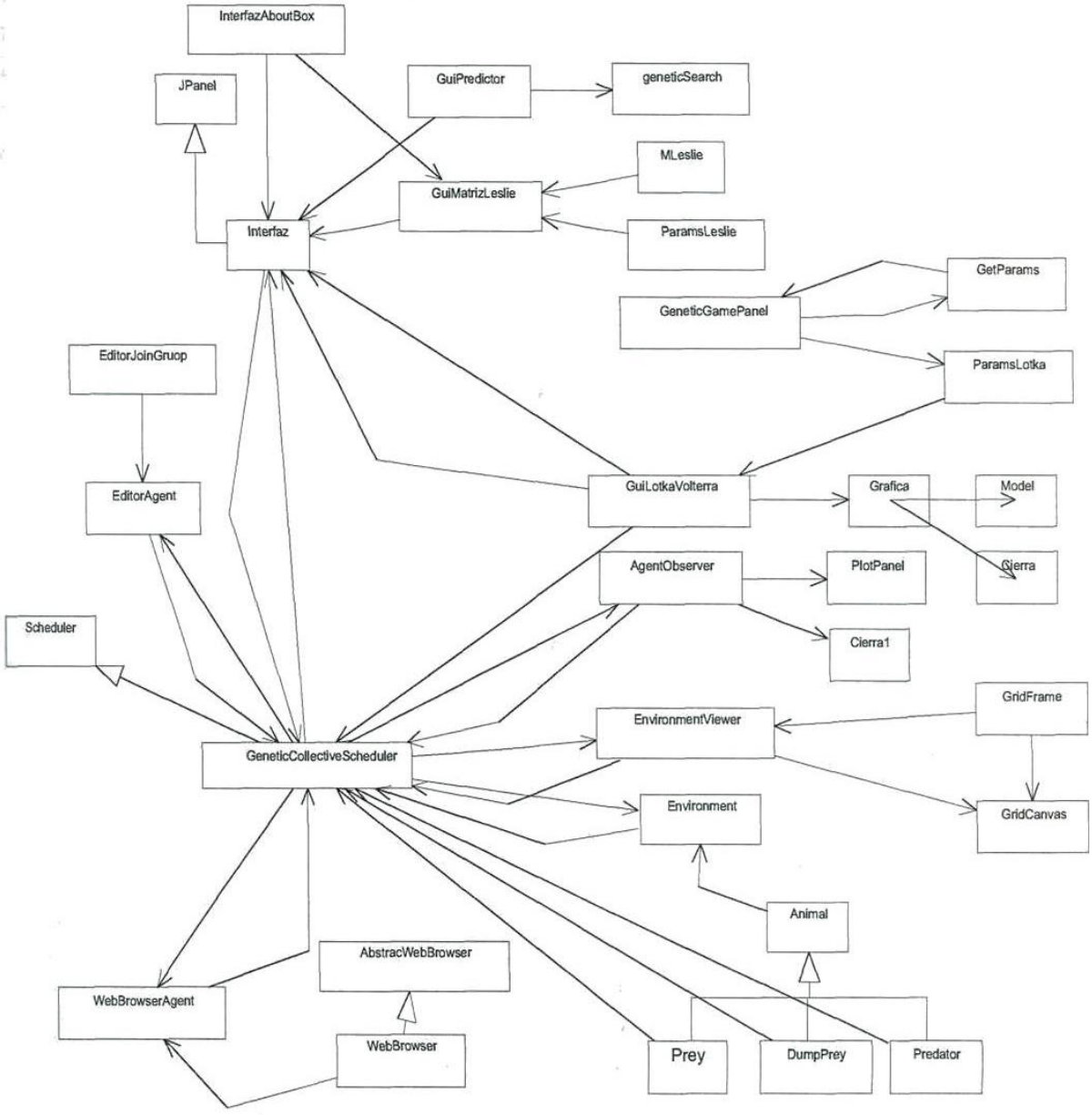

Fig. 5.5: *Diagrama de clase genético del sistema GECOLE*

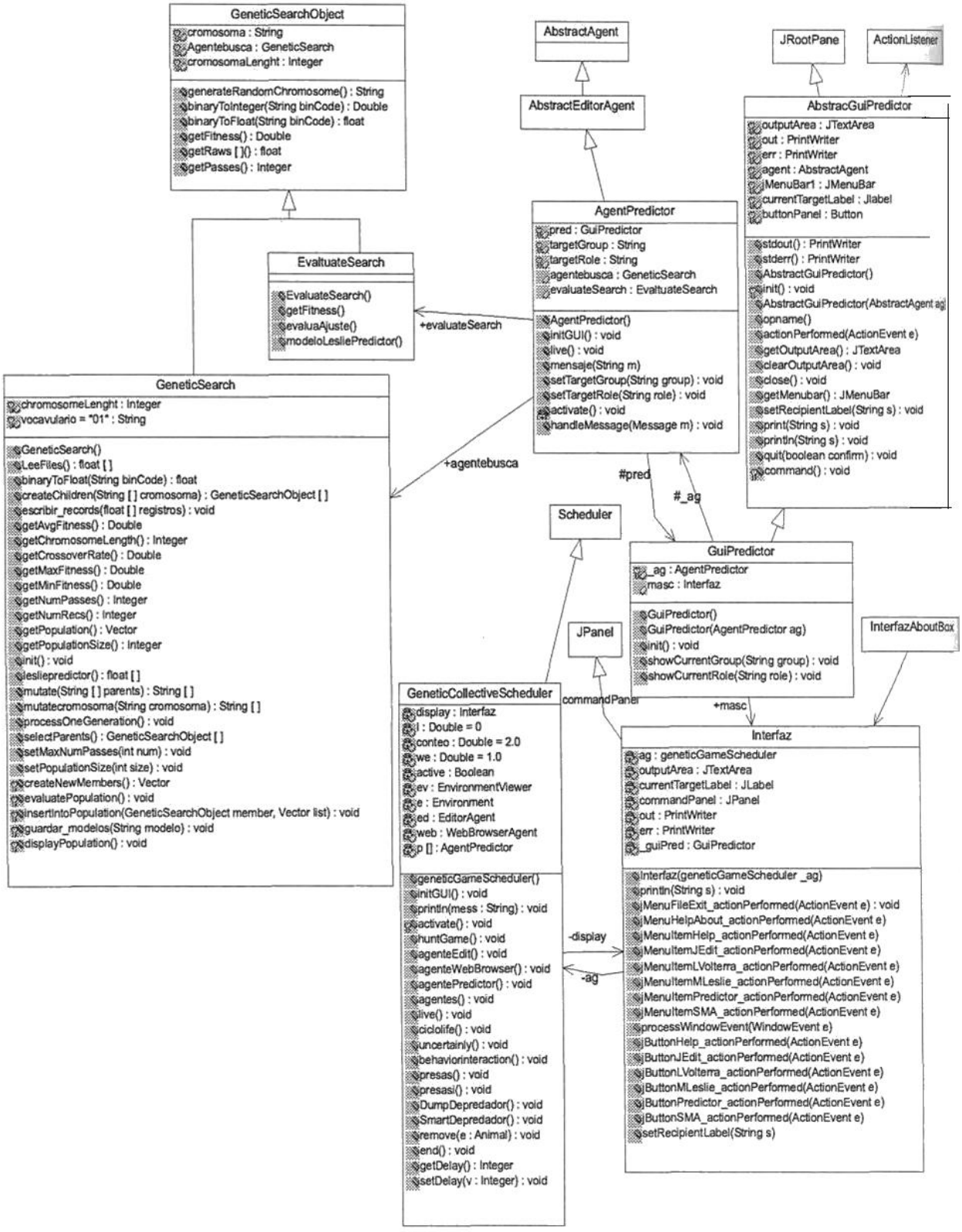

Figura 5.6: *Diagrama de clases del modelo predictor*

Agent Predictor

Esta clase fue desarrollada para generar el comportamiento de cada uno de los agentes predictores. En esta clase se disponen de métodos que permiten tener comunicación con el agente principal a través de mensajes. Se desarrolló métodos para interactuar con las clases que desarrollan el proceso genético.

Nombre Atributos	**Tipo**	**Descripción**
pred	GuiPredictor	Se referencia a esta clase para utilizar sus métodos y atributos
targetGroup	String	Presenta el estado del agente según su grupo.
targetRole	String	Mantiene el valor del rol que interpreta el agente.
agentebusca	GeneticSearch	Hace referencia a la clase *Genetic* para el modelo predictor.
evaluateSearch	EvaJuateSearch	Mantiene el *vínculo con la clase de* evaluación de los modelos parciales en la población.

Operaciones:

- *InitGUI():* Inicializa las operaciones de interfaz e inicializa los componentes del panel del agente.
- *Activate():* En este método se proporciona el comportamiento de activar e iniciar al agente. Envía mensajes al agente principal de su activación e inicio.
- *Live():* En este método se envia un mensaje para mostrar el estado del agente, activa todo su comportamiento y se pone activo para recibir mensajes.
- *setTargetGroup(String group):* Este método se generó para establecer el estado del grupo al que pertenece el agente.
- *setTargetRole(String role):* Este método establece un estado del rol que juega el agente en ese momento.
- *mensaje(String m):* El comportamiento de cada agente y la comunicación entre ellos se realiza lanzando mensajes a través de este método.
- *handleMessage(Message m):* Toma los mensajes recibidos de otros agentes.

- *stderr():* Expone al usuario un mensaje de error cuando éste llega a existir.

GeneticSearchObject

Esta clase define la abstracción del comportamiento genético y define métodos que son genéricos heredados por la clase *GeneticSearch.* **GeneticSearch**

Esta clase contiene todo el funcionamiento del proceso genético. La clase permitirá generar el conjunto de posibles soluciones y generar en cada generación los modelos parciales de solución. Obtiene los modelos finales que se ajusten a la población en estudio.

Nombre Atributos	**Tipo**	**Descripción**
population	Vector	Guarda todo el conjunto de *cromosomas* generado.
chromosomeLength	int	El valor se utiliza para proporcionar el tamaño del *cromosoma*
maxNumPasses	int	Selecciona el tamaño máximo de generaciones
populationSize	int	Proporciona el tamaño máximo de la población
numPasses	int	Lleva el conteo de las generaciones procesadas
vocabulary	int	Contiene el alfabeto que se utiliza en los cromosomas, para este caso es binario
resultados	double	Guarda los resultados obtenidos de la estimación de la serie en un arreglo

.

Operaciones:

- *GeneticSearch():* El constructor genera la inicialización de los componentes necesarios.
- *LeeFiles():* Carga los valores de la base de conocimiento que es la serie de tiempo propuesta.
- *binaryToFloat(String binCode):* Convierte el *genotipo* a valores que se puedan interpretar.
- *createChildren(String[ ] cromosoma):* Genera una solución parcial que es un *cromosoma* nuevo a partir de una *cruza* de sus *parientes.*
- *escribir_records(float[ ] registros):* Guarda los resultados finales que se obtuvieron al finalizar el proceso.

- *getAvgFitness():* Toma el valor de un promedio de ajuste para procesar los valores y evaluarlos.

- *getChromosomeLength():* Recibe la longitud del cromosoma que se tomará en cuenta para estructurar los parámetros en el modelo de Leslie.

- *getNumPasses():* Se le asigna un valor que será interpretado por el número de generaciones que se evalúan en el proceso genético.

- *getNumRecs():* Proporciona el número total de registros que contiene la serie de tiempo.

- *getPopulation():* Retoma la población total original cuando es generada.

- *init():* Inicializa todos los procedimientos de la clase.

- *lesliepredictor():* Recibe los modelos parciales que son enviados del método *modeloLesliePredictor()* de la clase de evaluación que es *EvaluateSearch.*

- *mutate(String[ ] parents):* Contiene el conjunto de *cromosomas* que se eligieron para su *mutación.*

- *mutatecromosoma(String cromosoma):* Recibe cada *cromosoma* para su proceso de *mutación.*

- *processOneGeneration():* Se encarga de procesar cada uno de los componentes del proceso genético y de llevar el conteo de cada una de las generaciones evaluadas.

- *selectParents():* Selecciona el mejor *cromosoma* de la población de acuerdo a la evaluación hecha en la clase *EvaluateSearch.*

- *setMaxNumPasses(int num):* Activa el número máximo de generaciones que se necesitan para completar el proceso.

- *setPopulationSize(int size):* Activa el tamaño total de la población, este valor será tomado en cuenta cuando se generan los *cromosomas* en la población de las soluciones parciales.

- *CreateNewMembers():* Genera a partir de la cruza y mutación los nuevos miembros que serán evaluados y son reingresados a la población.

- *insertIntoPopulation(GeneticSearchObject member, Vector list):* Después de procesar la *cruza* y en su debido caso alguna *mutación,* este método se encarga de anexarlo al conjunto de soluciones que es la población activa.

- *guardar_modelos(String modelo):* Este método retoma los modelos generados en cada generación y los guarda en un archivo para su visualización.

- *displayPopulation():* Se encarga de enviar los resultados procesados al *agente predidor* para que los visualice y los muestre al usuario.

EvaluateSearch

Esta clase tiene como función principal evaluar los modelos parciales que son generados, en el conjuto de soluciones por el proceso genético de la clase *GeneticSearch.* En cada generación se evalúan cada uno de los modelos parciales que se prueban para el ajuste de la serie de tiempo. Todos los modelos resultantes de cada generación son evaluados en esta clase y enviados a la clase *GeneticSearch.* El modelo final es enviado al agente predictor.

Operaciones:

- *EvaluateSearch():* Como constructor inicializa los parámetros correspondientes como el tamaño del cromosoma, el tipo de vocabulario que es el alfabeto utilizado, además inicia el proceso de evaluación de los modelos parciales.

- *getFitness:* Toma el valor de ajuste para cada modelo parcial, este es referido a la clase de *GeneticSearch.*

- *evalúa A juste ():* Esta operación retoma el modelo parcial que es evaluado para el ajuste de la serie. Se utiliza como función de evaluación (fitness) del proceso genético el cálculo del error cuadrático medio normalizado.

- *modeloLesliePredictor():* Los modelos parciales son procesados en el modelo predictor propuesto y evaluados por el método anterior, este proceso es conducido por la serie de tiempo propuesta.

Scheduler

Esta clase que proporciona las librerías de Madkit es de tipo abstracta. Tiene como principal función el de proporcionar las bases para la generación y activación de los agentes. Estos agentes siempre son activados dentro del ambiente multiagente.

Esta clase define al agente *geneticCollectiveScheduler* como un agente genérico que administra a todos los agentes. Sostiene una colección de activadores. Estos activadores usan los funcionamientos y operaciones privadas del agente para conseguir acceso a las referencias de las entidades que es la activación de los agentes.

GeneticCollectiveScheduler

Esta clase se define como el agente principal que tiene interacción con los agentes en el ambiente de los agentes. Como esta clase hereda de *Scheduler* tiene el comportamiento de ser el agente genérico que activa a los agentes. Presenta la característica de ser el agente administrador. Este agente permite generar y activar a los agentes predictores que formarán el equipo de trabajo.

Nombre Atributos	Tipo	Descripción
display	Interfaz	Se referencia a esta clase para utilizar sus métodos y atributos.
P	AgentPredictor	Hace referencia a la clase para activar el agente.
e	Environment	Utiliza las características, métodos y atributos para activar el agente.
ed	EditorAgent	Hace referencia a la clase de este agente para utilizarlo.
web	WebBrowserAgent	Hace referencia a esta clase para proporcionar la información que sirve de Ayuda.

Operaciones:

- *activate():* Presenta la inicialización del agente *Scheduler* enviando mensajes de inicio de cada agente a la interfaz principal que es mostrada al usuario.
- *agentePredictor():* Inicializa a los agentes predictores que formarán el equipo de trabajo para el ajuste de la serie de tiempo.
- *agenteEdit():* Este método activa a un agente *Editor.*
- *agente We6Browser():* Inicializa al agente *Help* que presenta una ayuda en línea del sistema.
- *initGUI():* Inicializa todos los componentes de la clase *Interfaz.*

- *Live():* Activa la operación principal de inicio y activación del agente *Scheduler* en el *Kernel* de Madkit.
- *println(String mess):* Muestra los mensajes enviados por los agentes.
- *end():* Termina y libera la memoria para cada uno de los agentes, es decir elimina los procesos.

AbstractGUIPredictor

En esta clase se definen los métodos que son abstractos para proporcionar el comportamiento de la interfaz en un agente predictor.

Nombre Atributos	**Tipo**	**Descripción**
Agent	AbstractAgent	Se referencia a esta clase para utilizar sus métodos y atributos
buttonPanel	JPanel	Monitorea el estado del agente predictor en una etiqueta de la interfaz.
outputArea	JTextArea	Visualiza los resultados en el área de texto.
currentTargetLabel	JLabel	Establece la etiqueta que muestra el estado.
Out	PrintWriter	Crea y establece un canal para el flujo e impresión de los datos
Err	PrintWriter	Tiene el mismo comportamiento del atributo anterior.

Operaciones:

- *AbstractGuiPredictor():* Este método como constructor hace un llamado a todos los componentes de la clase que hereda además de activar al método *init().*
- *AbstractGuiPredictor(AbstractAgent ag):* Toma la referencia del agente *AbstractAgent* inicializando el comportamiento de este. Otra operación que hace es la inicialización del área de texto donde son mostrados los resultados de los agentes.
- *Init():* Inicializa todos los componentes que contiene la interfaz del agente predictor.
- *actionPerformed(ActionEvent e):* Crea y adiciona el comportamiento de escuchar los eventos en los componentes de la interfaz.
- *clearOutputArea():* Limpia el área de despliegue de resultados.

- *Close():* Cierra la ventana del agente.
- *println(String s):* Presenta los resultados en el área de texto.
- *Stdout():* Muestra el estado del agente en ese momento.
- *Stderr():* Establece y muestra un mensaje de error si es que existe en un panel dentro de la interfaz.

GUIPredictor

Esta clase proporciona una interfaz gráfica para los agentes predictores. En esta clase se observan los resultados que se van generando por el proceso de cada agente. Estos resultados son mostrados para cada generación del proceso genético, además esta clase muestra los modelos parciales y los resultados finales de cada uno de los agentes.

Nombre Atributos	Tipo	Descripción
_ag	AgentPredictor	Se referencia a esta clase para utilizar sus métodos y atributos
currentTargetLabel	JLabel	Monitorea el estado del agente predictor en una etiqueta de la interfaz.
outputArea	JTextArea	Visualiza los resultados en el área de texto.

Constructor:

GuiPredictor(AgentPredictor ag): se utiliza para recibir el comportamiento que tiene el agente predictor y para hacer referencia a esta clase. En esta parte se envía también un mensaje de activación.

Operaciones:

- *initQ:* Inicializa la interfaz y el comportamiento del agente.
- *showCurrentGroup(String group):* Muestra a través de este método al grupo en que pertenece este agente.
- *showCurrentRole(String role):* Expone también el rol que esta jugando este agente.

Interfaz

Esta clase genera la interfaz principal del agente *geneticCollectiveScheduler* y muestra un panel que presenta las opciones para activar a los agentes, opciones para salir y para mostrar una ayuda.

Nombre Atributos	Tipo	Descripción
ag	geneticCollective Scheduler	Se referencia a esta clase para utilizar sus métodos y atributos
currentTargetLabel	JLabel	Monitorea el estado del agente principal Scheduler en una etiqueta de la interfaz.
outputArea	JTextArea	Visualiza los resultados en el área de texto.
out	PrintWriter	Crea y establece el conducto para el flujo e impresión de los datos
err	PrintWriter	Tiene el mismo comportamiento del atributo anterior.

Constructor:

Interfaz (GeneticCollectiveScheduler _ag): se utiliza para recibir el comportamiento del agente principal. Esta clase hace referencia al agente *Scheduler.* De esta manera todo el comportamiento de los componentes los maneja el agente principal.

Operaciones:

- *setRecipientLabel(String s):* Muestra en una etiqueta el estado del agente en ese momento. Los estados son mostrados como mensajes entre los cuales son: activado, enviando mensaje.

- *println(String s):* Despliega los mensajes en el área de texto principal del agente Scheduler.

- *jMenuFileExit_actionPerformed(ActionEvent e):* Crea y acciona la opción de menú Salir *(Exii)* incluida en el menú de Archivo.

- *jMenuHelpAbout_actionPerformed(ActionEvent e):* Crea y acciona la opción de menú Acerca de..., incluida en el menú de Ayuda *(Help).*

- *jMenuItemHelp_actionPerforrned(ActionEvent e):* Crea y acciona la opción de menú Ayuda *(Help)* incluida en el menú de Ayuda.

- *jMenuIternJEdit_actionPerformed(ActionEvent e):* Crea y acciona la opción de menú del agente Editor incluida en el menú de Agentes.

- *jMenuItemPredictor_actionPerformed(ActionEvent e):* Crea y acciona la opción de menú del agente predictor, incluida en el menú de agentes.

- *processWindowEvent(WindowEvent e):* Procesa los eventos que ocurren dentro de la interfaz.

- *jButtonHelp_actionPerformed(ActionEvent e):* Adiciona la acción correspondiente al botón Ayuda *(Help)* para que se visualice el agente ayuda.

- *jButtonJEdit_actionPerformed(ActionEvent e):* Este método permite llamar a través de la opción de menú al agente Editor. Que puede mostrar los archivos generados por el sistema.

- *jButtonPredictor_actionPerformed(ActionEvent e):* Método que proporciona al botón predictor la activación de los agentes predictores para su proceso.

5.5.3. Diagramas de estados

Este diagrama muestra una secuencia de eventos es decir cambios y estados que ocurren en cada uno de los agentes (clases) que participan en el sistema. En seguida se presentan los diagramas de estados de las clases del sistema.

Diagrama de estados de la clase Interfaz (Figura 5.7).

Mediante este diagrama se presenta el comportamiento de la clase *Interfaz,* su función es crear el panel principal del sistema GECOLE. Esta interfaz contiene varios elementos como son la barra de menú, la barra de botones, y el área de monitoreo.

Diagrama de estados de la clase GeneticCollectiveScheduler (Figura 5.8).

La secuencia del comportamiento de esta clase se presenta en el diagrama de la figura 5.8. Esta clase tiene la función de administrar la secuencia de activación de los agentes entre ellos al equipo de agentes predictores. En el diagrama se describe la secuencia que presenta su funcionalidad.

Diagrama de estados de la clase GeneticSearch (Figura 5.9).

La clase *GeneticSearch* expone el funcionamiento genético del agente predictor así como los estados en que se evalúa los modelos parciales generados para el ajuste del modelo. Esta clase presenta el comportamiento principal de la clase de genética. Los procedimientos que se generan siempre están coordinados con el agente principal.

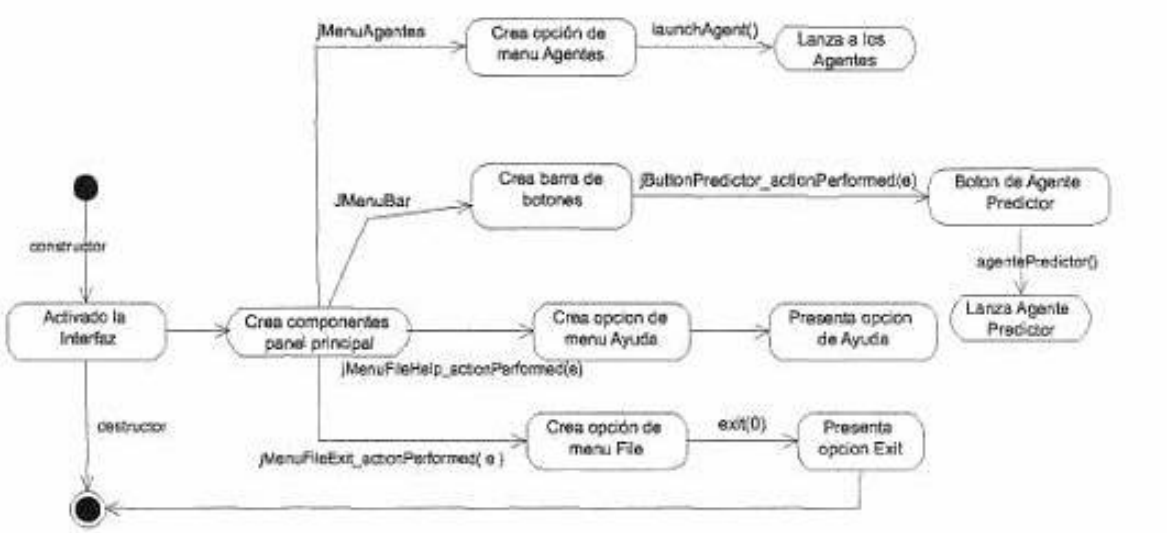

Figura 5.7: *Diagrama de estados de la clase Interfaz*

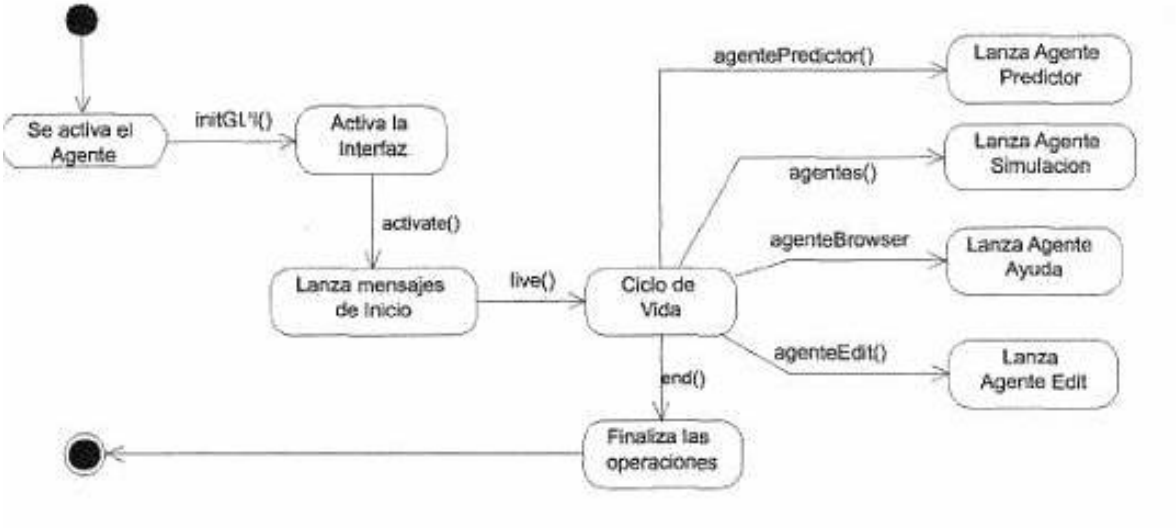

Figura 5.8: *Diagrama de estados de la clase GECOLE*

Diagrama de estados de la clase AgentPredictor (Figura 5.10).

Esta clase obtiene el funcionamiento evolutivo de la clase *GeneticSearch* y en conjunto con los demás agentes predictores llevan a cabo la evaluación de los modelos parciales generados. También mantiene la comunicación con el agente principal y finalmente los resultados son mostrados en el área de monitoreo que son procesados por los mensajes.

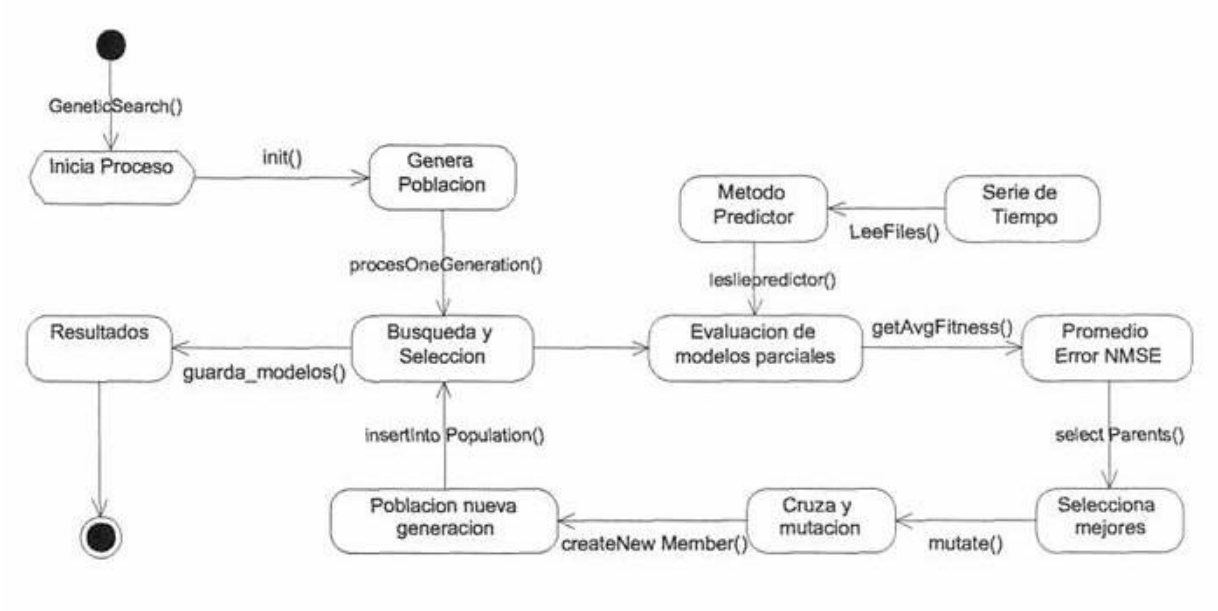

Figura 5.9: *Diagrama de estados de la clase GeneticSearch*

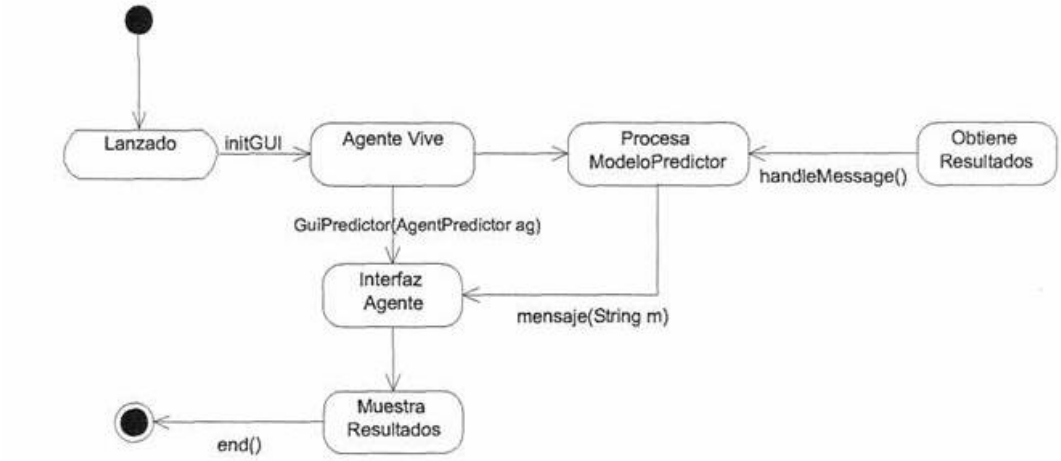

Figura5.10: Diagrama de estados de la clase AgentPredictor

5.5.4 Diagrama de secuencia

El diagrama de secuencia contribuye a la descripción de la dinámica que existe en el sistema, es decir muestra la interacción que existe entre los distintos agentes del sistema. De acuerdo a este planteamiento esta interacción se lleva a cabo mediante mensajes [Rendón, 2000] descritos por el comportamiento de los métodos en las clases.

A continuación se presenta el diagrama de secuencia en la Figura 5.11.

Descripción del diagrama de secuencia:

1. El usuario inicializa al agente principal *geneticCollectiveScheduler* que es mostrado mediante la interfaz que se presenta al usuario y que tiene control sobre los otros agentes.

2. Se activan los agentes predictores que formarán al equipo de agentes. Estos realizarán las tareas de ajuste genético. Los agentes predictores muestran una pequeña interfaz. En esta interfaz son mostrados los resultados que generan. Esta interfaz se activa a través del método *init()* de la clase *GuiPredictor.* Esta clase tiene relación directa con la clase *AgentPredictor* la cual activa el método *initGUI()* y el método *live().*

3. Cada agente activa el proceso genético dentro del cual es generado el conjunto de posibles soluciones. Estas soluciones son los modelos parciales del modelo de Leslie para su ajuste. Este procedimiento es realizado por el método *generateRandomChromosome().* El proceso genético lo realiza la clase *GeneticSearch.*

4. Los modelos generados son evaluados por la clase llamada *EvaluateSearch.* Esta clase evalúa cada uno de los modelos parciales generados mediante el método *evaluaAjuste().* En esta etapa es tomada la base de conocimiento que es la serie de datos propuesta en el comportamiento del método *modeloLesliePredictor().* La lectura de la base de conocimientos es realizada por el método *LeeFiles().*

5. Los mejores modelos generados de acuerdo a la evaluación son devueltos mediante el proceso genético a cada uno de los agentes. En esta etapa los resultados son guardados en archivos. Este procedimiento lo realiza el método *guardar_modelos().*

6. Los resultados que se van generando como son los modelos parciales son mostrados en la interfaz de cada agente para que finalmente los observe el usuario. Esto se hace a través de mensajes entre los agentes en la clase *AgentPredictor* mediante el método mensaje().

5.5.5. Diagrama de colaboración

El diagrama de colaboración muestra los mensajes a través de los cuales se produce la secuencia e interacción entre los agentes, como el diagrama de interacción física los enlaces

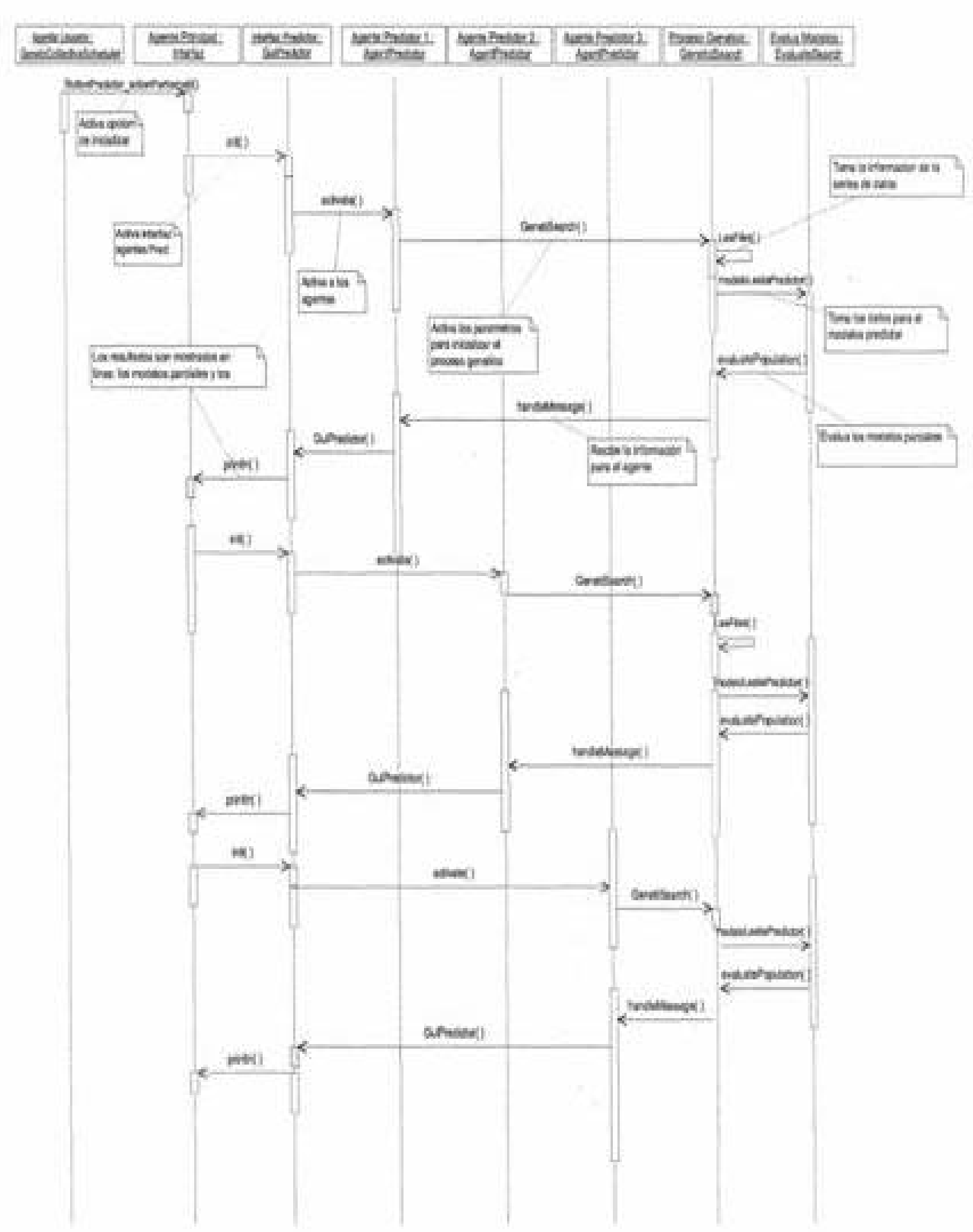

Figura 5.11: *Diagrama de secuencia del modelo Predictor*

entre las clases. En el diagrama de colaboración [Rendón, 2000] muestra su énfasis en la estructura complementada por los agentes y sus relaciones. En la Figura 5.12 se muestra el diagrama de colaboración del sistema en estudio.

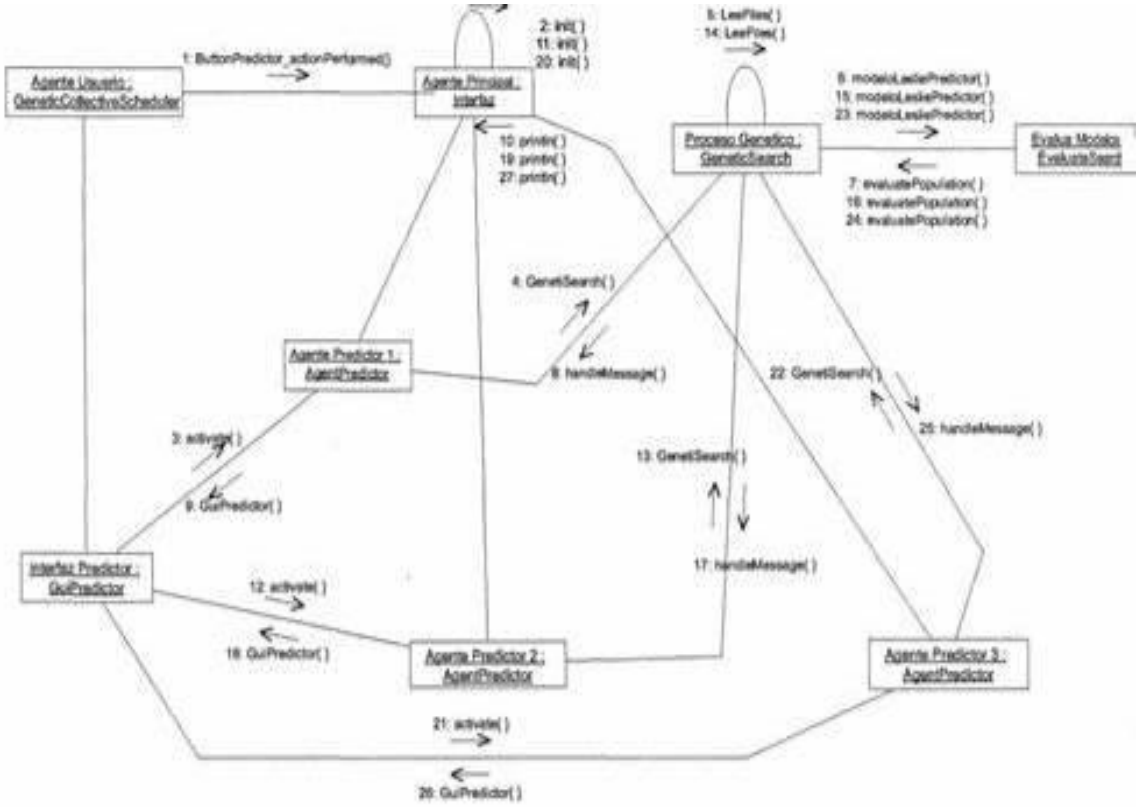

Figura 5.12: *Diagrama de colaboración del sistema GECOLE*

Descripción del diagrama de colaboración:

Los mensajes entre los agentes se observa con una numeración llevada secuencialmente, lo que permite visualizar la interacción general de todos los agentes. La descripción de los mensajes se explican en seguida:

¦ El agente usuario a través del método **1:ButtonPredictor_act¡onPerfbrmed()** activa al agente interfaz.

¦ El agente interfaz mediante el método **2:init()** activa la interfaz de los agentes predictores mediante GuiPredictor, el equipo de agentes predictores es activado mediante el

método 3:activate(), sucesivamente por 12:activate(), y por 21:activate(), para inicializar a los agentes.

- Cada agente generado tiene un enlace directo con la clase GeneticSearch donde bs agentes establecen su tarea de búsqueda y ajuste. Este enlace lo hace el método 4 GeneticSearch () para el primer agente predictor. Este enlace lo hace cada uno de los agentes con: 13:GeneticSearch(), y 22:GeneticSearch().

- En la clase GeneticSearch se inicia todo el proceso genético pero también es cargada la base de conocimientos con el método 5:LeeFiles() identificada por la serie de datos que se necesita para conducir al modelo predictor.

- En la clase EvaluateSearch son evaluados cada uno de los modelos parciales que son generados con el método 6:modeloLesliePredictor() que es el que permite el enlace con la clase mencionada.

- Los resultados de los modelos que se evalúan son regresados a la clase GeneticSearch mediante el método 7:evaluatePopulation(). Cada proceso es realizado por cada uno de los agentes y los resultados son enviados a cada agente por medio del método 8:handleMessage().

- Los agentes presentan sus resultados al usuario a través de su interfaz. Cada resultado es mostrado por el método 9:GuiPredictor() También es enviado un dato de estado al agente principal con el método 10:println(). Los agentes guardan los resultados generados de las estimaciones del modelo a través de la clase GeneticSearch.

5.6. Desarrollo e implementación de GECOLE

El sistema GECOLE se activa dentro del ambiente *Madkit.* Este ambiente es presentado por la interfaz gráfica principal *G-Box* que funciona como un agente administrador de conjuntos de agentes. El sistema GECOLE desarrolla todas sus actividades en *G-Box.* GECOLE presenta una interfaz gráfica que sirve de enlace para el usuario. El usuario interactua con GECOLE y activa a los agentes.

Además se muestran las opciones para activar a los agentes entre los cuales está el agente predictor.

Cabe mencionar que de acuerdo a las características de heterogeneidad, modularidad, entre otras, el lenguaje de programación *Java* es la plataforma fundamental para la arquitectura en la cual se desarrolló la aplicación.

A continuación se presenta en la Figura 5.13 el ambiente multiagente representado por la interfaz gráfica de G-Box. Se puede observar al sistema GECOLE dentro de este ambiente junto con todos sus componentes. Esta imagen muestra el panel principal del sistema GECOLE con las siguientes opciones:

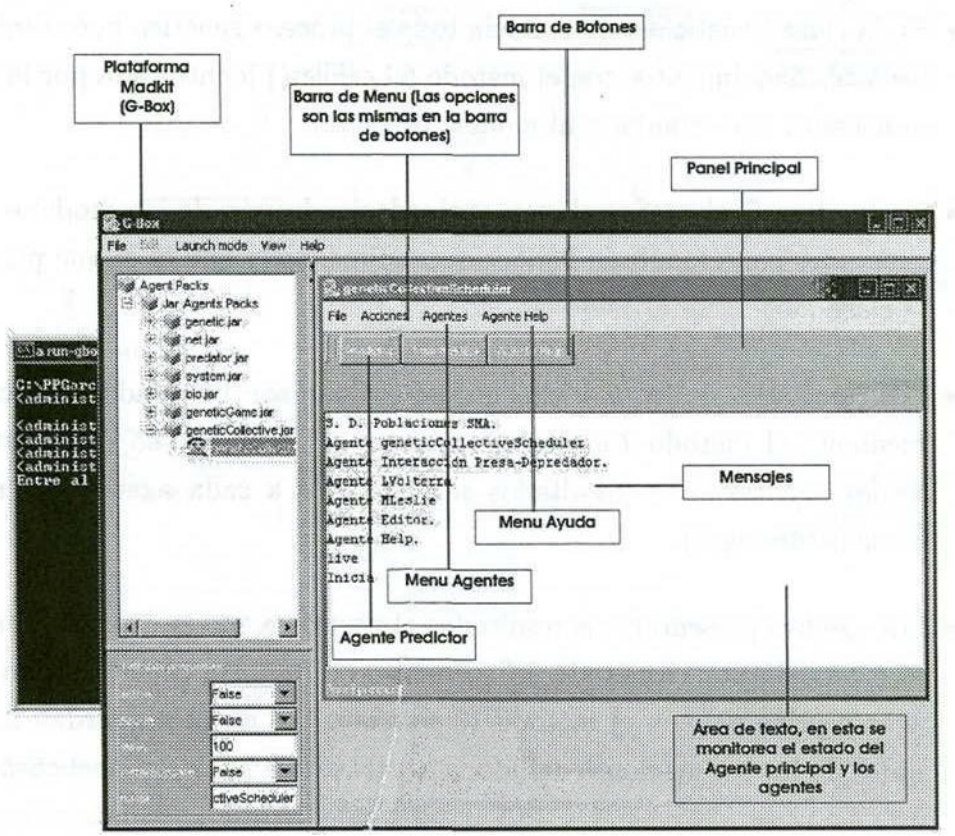

Figura 5.13: *Ambiente GECOLE*

1. Agente principal *geneticCollectiveScheduler:* Este se muestra como la interfaz inicial donde el usuario puede interactuar además cumple como administrador entre los agentes.

2. Barra de activación de agentes: Esta opción permite al usuario la posibilidad de activar cualquiera de los agentes que están disponibles entre los cuales está el agente predictor.

3. Área de presentación: Esta área muestra el monitoreo y los resultados que proporcionan los agentes.

4. Ambiente *G-Box:* Contiene al agente principal que se encuentra dentro de lo que es *Madkit* que es el medio donde se desarrolla.

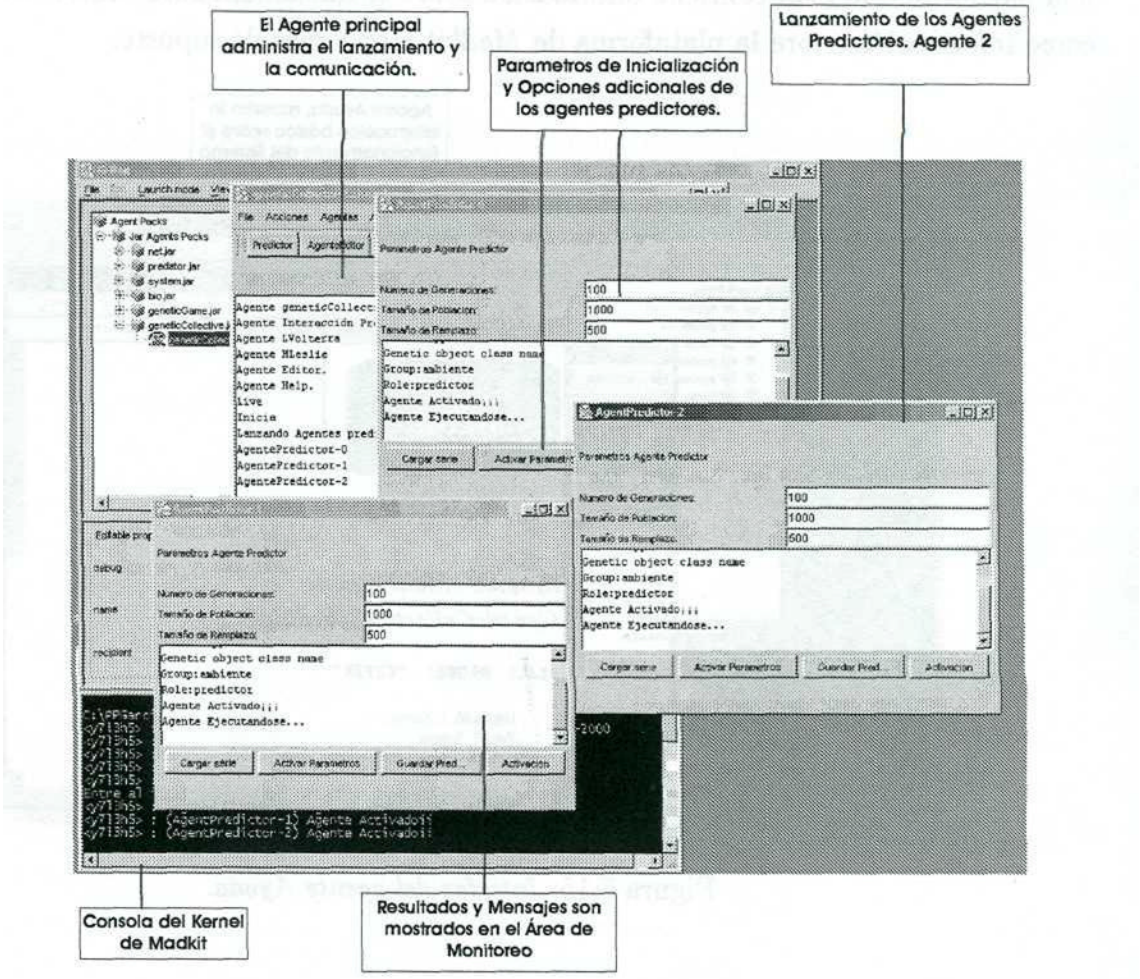

Figura 5.14: *Activación de los agentes predictores en el sistema GECOLE*

En la Figura 5.14 se presenta el sistema principal junto con los agentes predictores, se observa los agentes predictores son activados e inicializados. Cada agente predictor expone una interfaz gráfica individual que sirve para exponer mensajes y resultados.

1. Agente principal: inicializa la activación de los agentes a través de la elección sobre el panel de botones.

2. Agentes predictores: son representados cada uno de ellos por una pequeña ventana, la cual muestran los mensajes que se obtienen sobre los resultados y monitorea en el área de texto.

En la Figura 5.15 se observa al agente que juega el rol de ayuda. Este agente muestra una página web la cual contiene información sobre el funcionamiento básico del sistema así como información sobre la plataforma de *Madkit* y opciones de soporte.

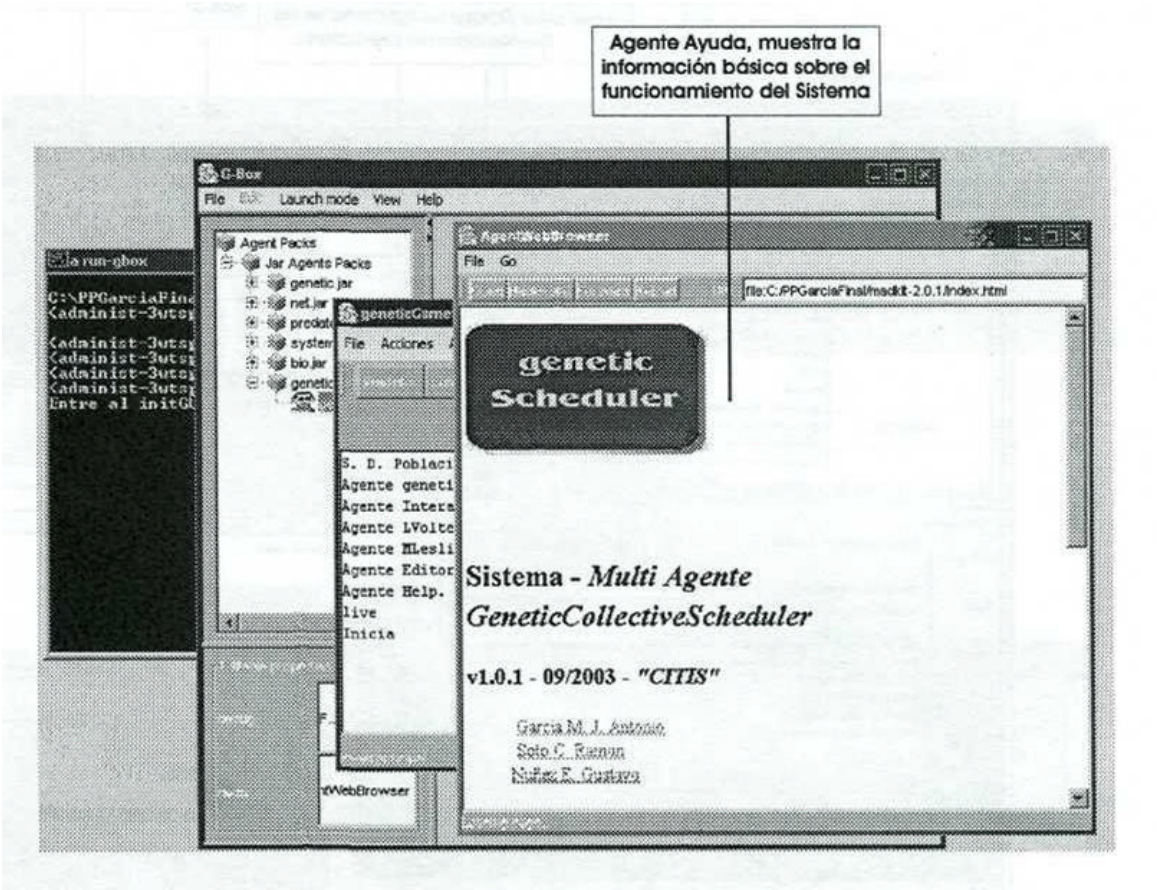

Figura 5.15: *Interfaz del agente Ayuda.*

5.7. Conclusiones

Se presentó inicialmente el análisis del sistema GECOLE. Se mostró un panorama general de la plataforma en la cual fue desarrollado el sistema. Esta plataforma esta basada en el modelo Aalaadin. Se detalla en el dominio de los agentes del sistema GECOLE los diferentes diagramas

que hacen referencia a la metodología UML para el diseño del sistema. Se detalló el funcionamiento de cada clase que participa en el sistema. Al finalizar se presentó la interfaz gráfica de usuario, basada en la plataforma *Madkit.* Se describieron sus partes y la función que realiza cada una de ellas.

Capítulo 6

Resultados

En este capítulo se exponen los resultados obtenidos en el desarrollo del proceso genético y la predicción del equipo de agentes en el sistema GECOLE. La validación de la técnica propuesta se realizó mediante la predicción de series de tiempo sintéticas y reales. En esta sección se presentan los resultados obtenidos en las pruebas.

6.1. Predicción de series de tiempo sintéticas

En la gráfica de la Figura 6.1 se muestra los resultados al predecir una serie de tiempo generada mediante la matriz dada por (6.1). Los modelos empleados en la predicción, obtenidos mediante el algoritmo genético, se muestran en la matriz dada por (6.2), la matriz dada por (6.3), la matriz dada por (6.4), la matriz dada por (6.5), la matriz dada por (6.6).

En este trabajo se optó por designar a $n = 3$ en la Ec. (4.4) de la sección 4.1 Estructura del modelo, donde se define la modificación de la estructura del modelo de Leslie. Las estructuras de las matrices se tomaron de A^{4*4}, y las pruebas respectivas fueron hechas en base a ese tamaño. Los modelos que se presentan en esta sección son modelos que son obtenidos como los mejores modelos parciales.

$$MA = \begin{pmatrix} 1{,}5671772 & 0{,}8549327 & 1{,}2495324 & 0{,}6003273 \\ 0{,}4918966 & 0{,}0 & 0{,}0 & 0{,}0 \\ 0{,}0 & 0{,}7408286 & 0{,}0 & 0{,}0 \\ 0{,}0 & 0{,}0 & 0{,}9463014 & 0{,}0 \end{pmatrix}$$

Para el proceso genético se designó que trabaje durante 100 generaciones, los modelos que se observan son obtenidos a partir de la generación 95 y para cada generación se obtiene el mejor modelo. El desarrollo del proceso de predicción se realiza al obtener los modelos que mejor se ajusten de acuerdo a la función de aptitud, entonces los modelos son

utilizado para predicción. La aptitud es evaluada por el error NMSE que se genera en cada modelo.

$$M_{p1} = \begin{pmatrix} 2.7737045 & 0.0639194 & 1.9171426 & 0.8446208 \\ 0.137879 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.162765 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.905946 & 0.0 \end{pmatrix}$$

En la Figura 6.1 se observa la serie de tiempo de una población global hipotética. La serie de tiempo resultante de la Figura 6.1 muestra la predicción por la línea con diamantes (<0>). Los resultados de la predicción final se presentan en gráfica de la Figura 6.1.

$$M_{p2} = \begin{pmatrix} 0.7836312 & 0.4140358 & 0.796738 & 0.1118836 \\ 0.093788 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.154698 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.273358 & 0.0 \end{pmatrix}$$

$$M_{p3} = \begin{pmatrix} 0.7811233 & 0.1612789 & 0.2273863 & 0.0802114 \\ 0.7181358 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.021882 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.1637694 & 0.0 \end{pmatrix}$$

$$M_{p4} = \begin{pmatrix} 1.722623 & 0.14199102 & 2.6999716 & 1.6135916 \\ 0.7356721 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.7112471 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.3489157 & 0.0 \end{pmatrix}$$

$$M_{p5} = \begin{pmatrix} 2.0878495 & 0.6412660 & 1.2795399 & 1.3818403 \\ 0.0924885 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.1374054 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.581231 & 0.0 \end{pmatrix}$$

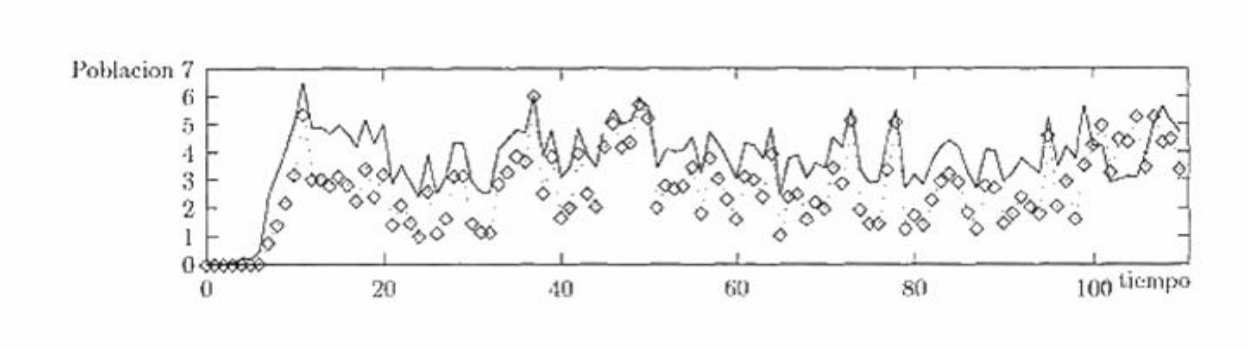

Figura 6.1: *Resultados del modelo predictor. Primera Prueba.*

El proceso genético presenta los mejores modelos parciales generados. En esta prueba se tomaron 600 datos para entrenamiento y 110 para predicción. La función de aptitud arroja un valor de NMSE para cada modelo parcial evaluado como la posible solución, dentro del conjunto de soluciones.

El NMSE nos indica el margen de error y con este margen proporciona al mejor modelo en cada generación. De esta manera los mejores modelos se toman para la predicción.

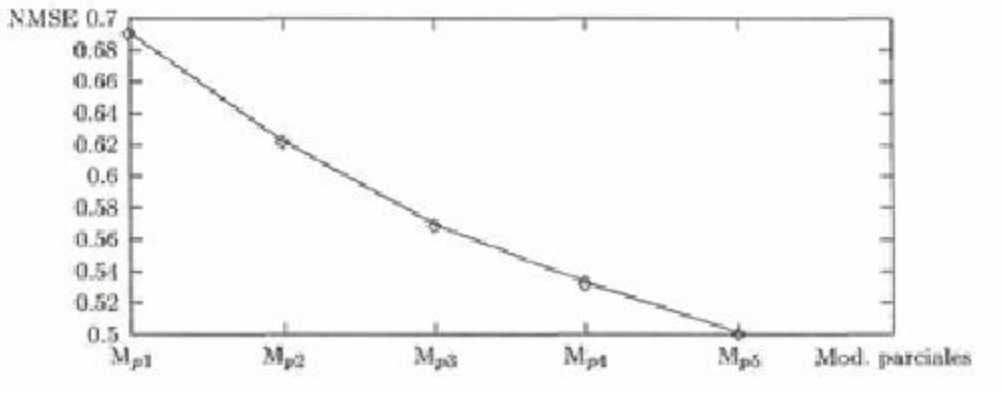

Figura 6.2: *Errores generados en la función de aptitud al evaluar los modelos parciales.*

En la Figura 6.2 se muestran los errores NMSE generados al evaluar los modelos parciales. Se muestran errores resultantes para las últimas generaciones, estos modelos son empleados para predecir la serie de tiempo evaluada como primera prueba.

Resultados de una segunda evaluación.

Se realizó la evaluación de diferentes series de tiempo generadas de manera sintética. El comportamiento que presentan en cada una de estas poblaciones hipotéticas es diferente.

Los modelos que se emplearon en la predicción de la serie que se aprecia en la gráfica de la Figura 6.3, fueron obtenidos mediante el algoritmo genético. Estos modelos se muestran en la matriz dada por (6.8), la matriz dada por (6.9), la matriz dada por (6.10), la matriz dada por (6.11), la matriz dada por (6.12).

$$MB = \begin{pmatrix} 1.5671772 & 1.2495324 & 6.8549327 & 0.6003273 \\ 0.4918966 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.7408286 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.9463014 & 0.0 \end{pmatrix}$$

El modelo con el cual se generó la serie sintética tomada como patrón es el modelo de la matriz dada por (6.7). Cada uno de los modelos obtenidos presentan un margen de error aceptable, al ser evaluados con la función de aptitud en el algoritmo genético.

$$M_{p1} = \begin{pmatrix} 1.8247459 & 0.81392986 & 2.3452148 & 1.5996855 \\ 0.6043759 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.437281 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.751667 & 0.0 \end{pmatrix}$$

$$M_{p2} = \begin{pmatrix} 1.9304523 & 0.4782556 & 2.8076911 & 1.30458987 \\ 0.8585742 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.3512602 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.7585647 & 0.0 \end{pmatrix}$$

$$M_{p3} = \begin{pmatrix} 1.8632386 & 0.30696613 & 2.42559713 & 1.72838485 \\ 0.1409052 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.2437123 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.6020474 & 0.0 \end{pmatrix}$$

$$M_{p4} = \begin{pmatrix} 2.367148 & 1.6379271 & 0.31220794 & 2.1374989 \\ 0.09746217 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.281437 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.3594663 & 0.0 \end{pmatrix}$$

$$M_{p5} = \begin{pmatrix} 1.33563197 & 0.8517022 & 2.7428562 & 0.9820777 \\ 0.54204997 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.2313311 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.18048456 & 0.0 \end{pmatrix}$$

Los resultados que se muestran en esta segunda evaluación, fueron obtenidos al tomar los modelos finales a partir de la generación 95.

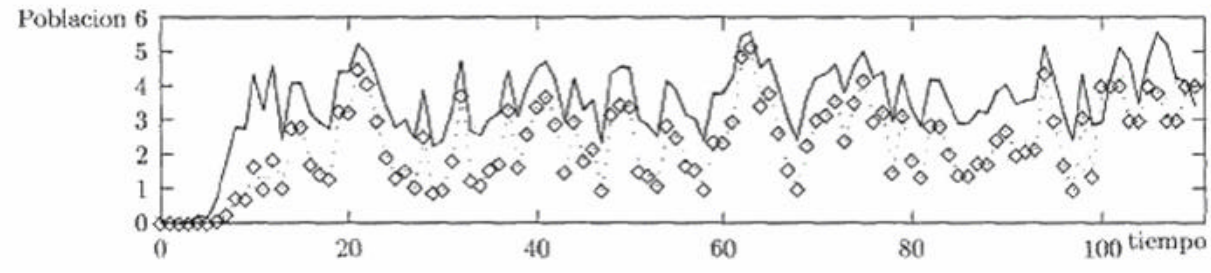

Figura 6.3 *Resultados de la segunda prueba obtenidos por el modelo predictor*

Se observa por ejemplo en la gráfica de la Figura 6.3, los resultados que se obtuvieron al predecir esta segunda serie de tiempo, generada mediante la matriz dada por (6.7). En esta evaluación se tomó el ochenta por ciento de los datos de la serie para entrenamiento y el veinte por ciento para predicción.

Los errores arrojados mediante la función de aptitud de los modelos obtenidos para esta serie son mostrados en la Figura 6.4.

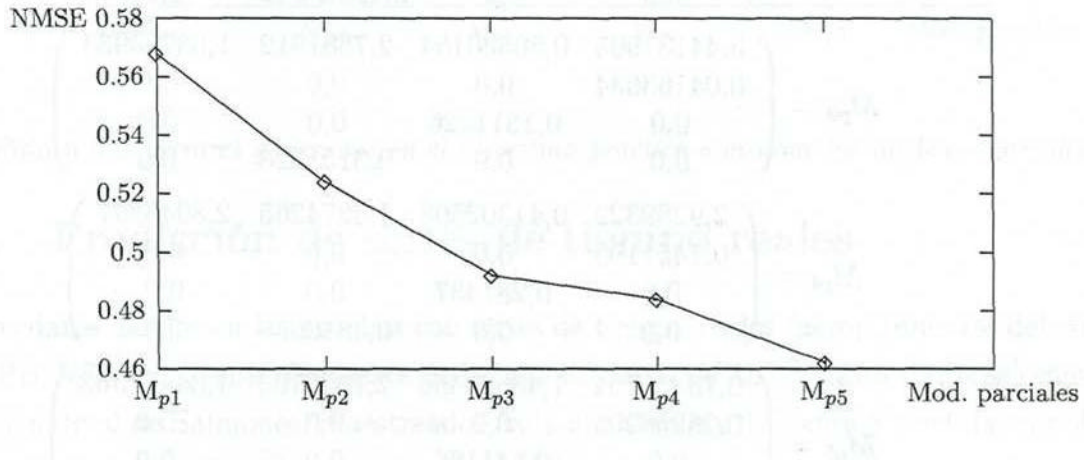

Figura 6.4: *Errores generados al evaluar los modelos parciales.*

Resultados de una tercera evaluación.

A continuación se presentan los resultados que se obtuvieron al predecir la serie de la gráfica mostrada en la Figura 6.5, generada mediante la matriz dada por (6.13).

Los modelos generados por el proceso genético son los modelos que se presentan en la matriz dada por (6.14), la matriz dada por (6.15), la matriz dada por (6.16), la matriz dada por (6.17), y la matriz dada por (6.18).

$$MC = \begin{pmatrix} 0.1806193 & 1.8464298 & 2.4466215 & 4.3492003 \\ 0.6039637 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.7891141 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.9281571 & 0.0 \end{pmatrix} \tag{6.13}$$

Las matrices con las que se generan las series sintéticas son presentadas como procesos ambientales estocásticos, sin embargo los modelos que se generan con el proceso genético, arrojan valores con errores pequeños.

$$M_{p1} = \begin{pmatrix} 1.59720933 & 1.6922444 & 2.8232385 & 4.45523065 \\ 0.18771183 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.1638844 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.735812 & 0.0 \end{pmatrix} \quad (6.14)$$

$$M_{p2} = \begin{pmatrix} 1.5655227 & 2.9864434 & 0.11860597 & 3.9585214 \\ 0.235370 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.1654114 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.43169852 & 0.0 \end{pmatrix} \quad (6.15)$$

$$M_{p3} = \begin{pmatrix} 3.44131905 & 0.50530154 & 2.7581912 & 1.63793933 \\ 0.04163634 & 0.0 & 0.0 & 0.0 \\ 0.0 & .01811426 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.3151228 & 0.0 \end{pmatrix} \quad (6.16)$$

$$M_{p4} = \begin{pmatrix} 2.9289322 & 0.41302508 & 1.5274265 & 2.8946597 \\ 0.1451193 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.281437 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.4810989 & 0.0 \end{pmatrix} \quad (6.16)$$

$$M_{p5} = \begin{pmatrix} 0.73339754 & 1.43603456 & 2.7058763 & 1.34477365 \\ 0.26256205 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.641166 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.223158 & 0.0 \end{pmatrix} \quad (6.17)$$

La gráfica resultante de la Figura 6.5 muestra los resultados obtenidos de una tercera evaluación.

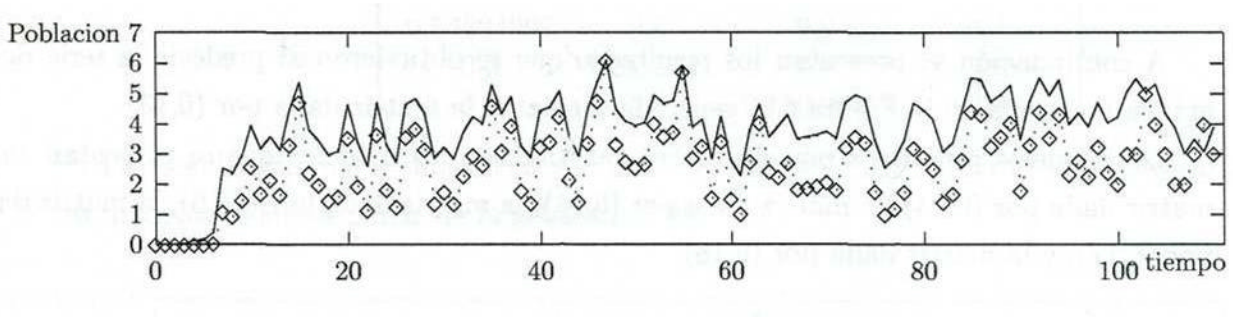

Figura 6.5: *Resultado de la tercera prueba por el modelo predictor.*

Se puede observar la serie que muestra la predicción con una línea con diamantes (?). Al inicio no presenta un ajuste preciso, pero al finalizar, se observa como los valores estimados son cercanos a los de la serie propuesta como patrón a seguir.

En la Figura 6.6, se muestra los errores arrojados para ésta última serie de tiempo

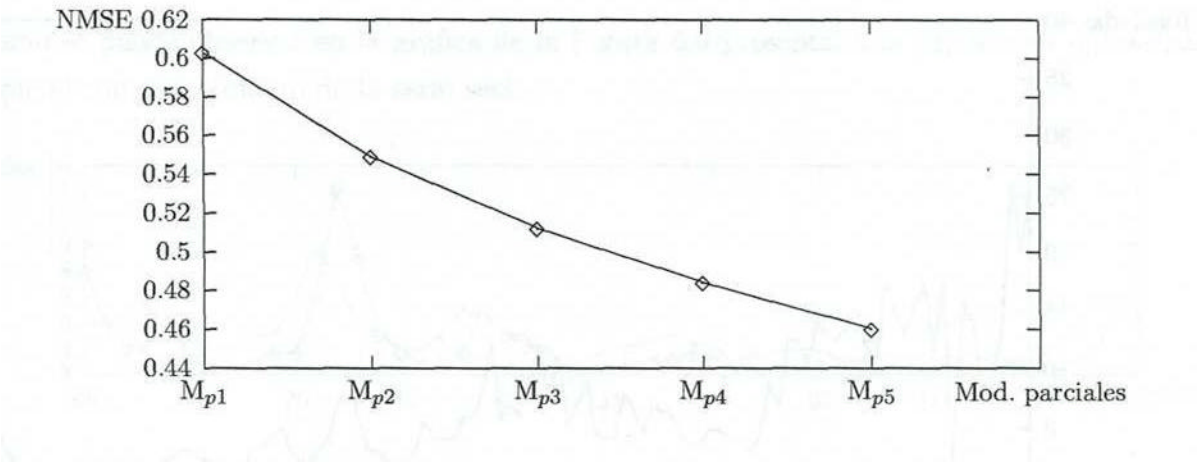

Figura 6.6: *Errores generados en el algoritmo genético a evaluar los modelos parciales.*

6.2. Predicción de series de tiempo reales

Los datos usados en las pruebas con series de tiempo reales fueron tomados del sitio web del CPB/NCEAS[6], específicamente de la base de datos *GPDD*[7]. Estos datos se refieren a la especie animal de Salmones, muestreados cada año durante 110 años y medida su población en toneladas estimadas usando estadisticas de captura, conteo de trampas y marcado y recaptura.

La especie de salmón tomada en cuenta es *Salmosalar* (Salmón Atlántico). Esta base de datos contiene 110 registros y comprende del periodo de 1876 a 1986, estos datos se obtuvieron del rio Drammenselv de la provincia de Buskerud, Noruega.

La Figura 6.7 muestra la serie de datos que representa el comportamiento de la especie de Salmones. Esta serie se tomó como patrón para ser evaluado por el sistema GECOLE y se realice el procedimiento adecuado y los agentes hagan la predicción.

Se toma una parte de los datos de la serie para el entrenamiento y ajuste de los modelos parciales en los agentes. Particularmente se utilizan 60 datos para entrenamiento y ajuste. La predicción se realiza con 50 datos.

Los resultados obtenidos de la predicción se muestran en la gráfica de la Figura 6.9. La línea marcada con diamantes representan los valores estimados y la línea continua los valores objetivos. Los datos que se utilizaron para entrenamiento y ajuste de los modelos parciales para este caso fueron 60 datos. Durante este proceso los agentes predictores obtuvieron

[6] CPB/NCEAS - Center for Population Biology Imperial College / National Center for Ecological Analysis and Synthesis
[7] GPDD - The Global Population Dynamics Database(1999)

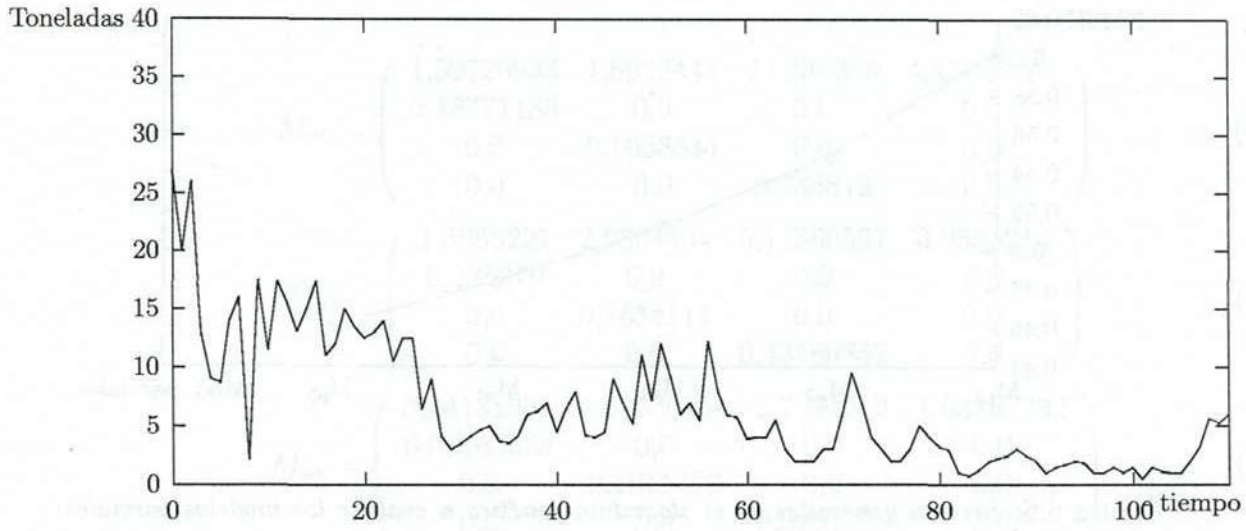

Figura 6.7: *Serie de tiempo de información de Salmones.*

un modelo de Leslie ajustado y que ha sido conducido por la serie de tiempo propuesta. En la gráfica de la Figura 6.8 se observan los comportamientos de los agentes predictores en el proceso de entrenamiento. La gráfica de la Figura 6.8 muestra el entrenamiento del agente predictor Ag_1 representado por la línea (—), Ag_2 representado por la línea *(?),* Ag_2 representado por la línea (+).

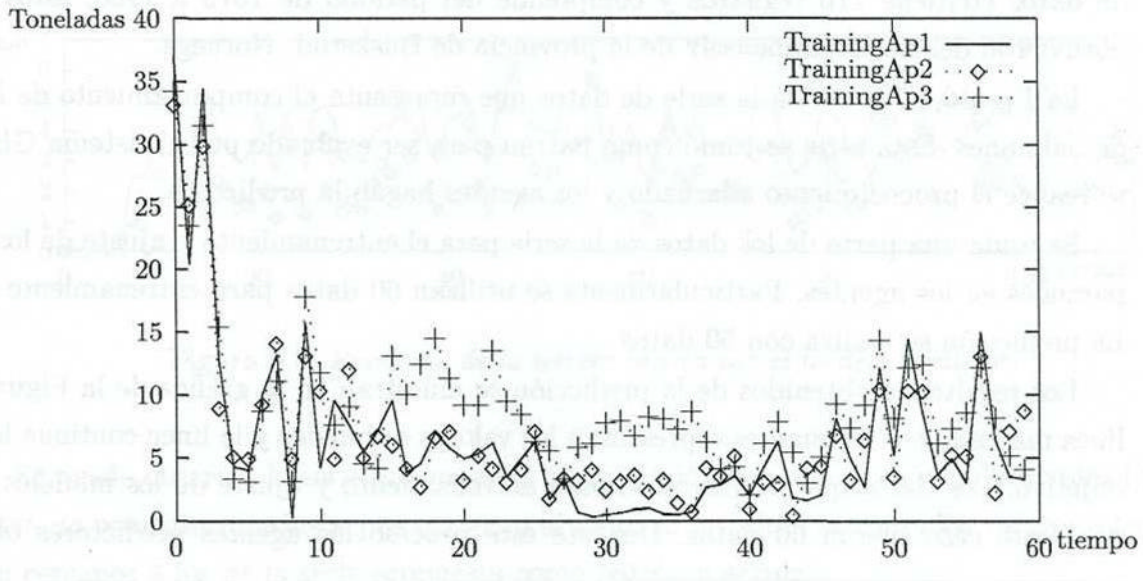

Figura 6.8: *Datos de entrenamiento con los modelos de Leslie obtenidos por los agentes Api*

Como se puede observar en la gráfica de la Figura 6.9 presentada la predicción que se hace sigue el comportamiento de la serie real

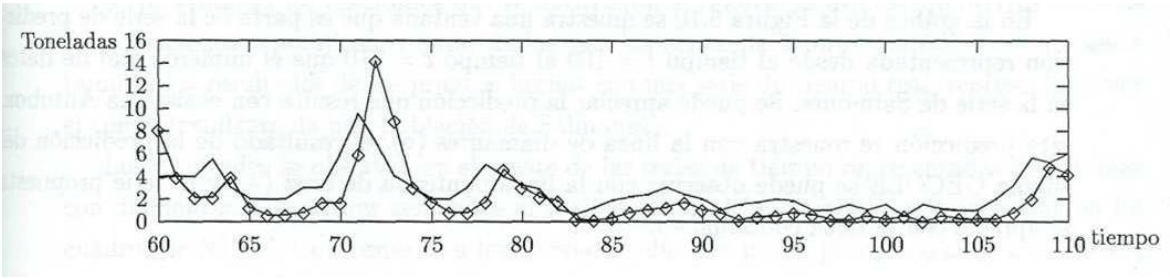

Figura 6.9: *Resultados de predicción hechas por el modelo predictor en series de tiempo de Salmones.*

Se realizó la comparación con otro método de predicción clásico, mencionado en la sección 2.2 como procesos ARIMA, este método realiza la predicción basada en regresión de los datos. El software que se utilizó para realizar la predicción de la serie mostrada en la gráfica de la Figura 6.7 se llama *AutoBox For Windows,* de *Automatic Forecastic System Inc.* versión 5.0. Este software se basa en el método de ARIMA para hacer una predicción de la serie. Este software puede hacer una predicción de 10 elementos de la serie propuesta como patrón.

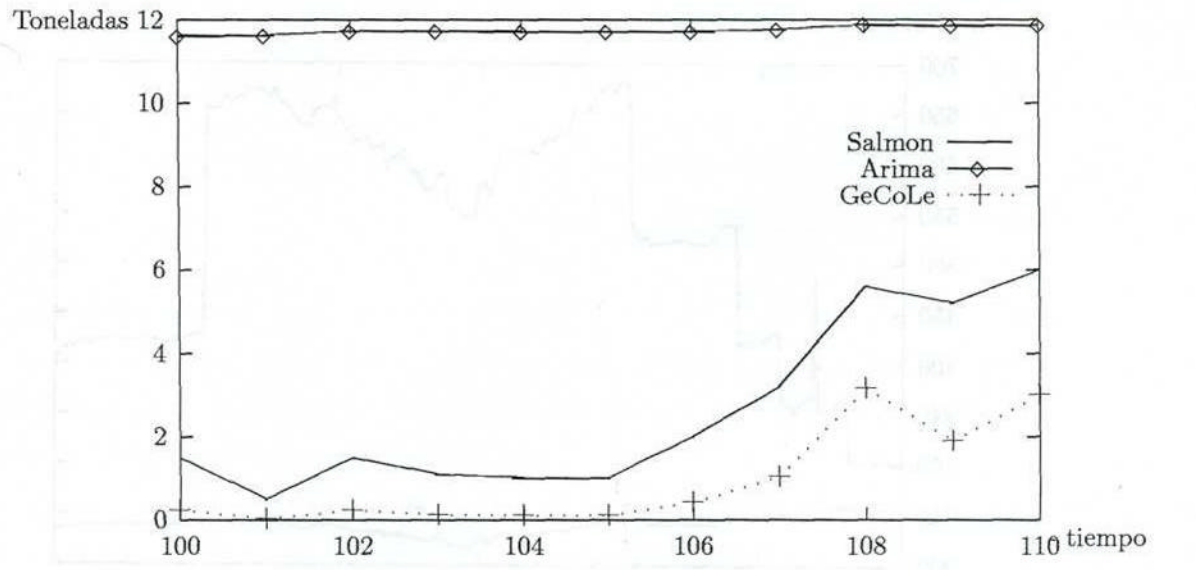

Figura 6.10: *Comparación con el método clásico (ARIMA) y el método predictor propuesto.*

Los resultados comparativos de la predicción de la serie de Salmones con el método del

sistem GECOLE y el sistema Autobox se muestran en la gráfica de la Figura 6.10. Se puede observar las predicciones generada por ambos sistemas.

En la gráfica de la Figura 6.10 se muestra una ventana que es parte de la serie de predicción representada desde el tiempo *t* = 100 al tiempo *t* = 110 que el numero total de datos en la serie de Salmones. Se puede apreciar la predicción que resulta con el sistema Autobox, esta predicción se muestra con la línea de diamantes *(o),* el resultado de la predicción del sistema GECOLE se puede observar con la línea punteada de cruz (+) y la serie propuesta se aprecia con la línea continua(—).

6.3. Caso disfuncional

El sistema GECOLE presenta un comportamiento no funcional en la evaluación de algunas series de tiempo. Es decir las series de tiempo representadas por valores grandes arrojan resultados no favorables en la predicción de dichas series. A continuación se presenta el resultado que se obtuvo de cargar una serie de tiempo al sistema GECOLE, esta serie contiene valores grandes de la serie.

Los datos que se obtienen no predicen adecuadamente la serie. Este resultado se observa en la gráfica de la Figura 6.11.

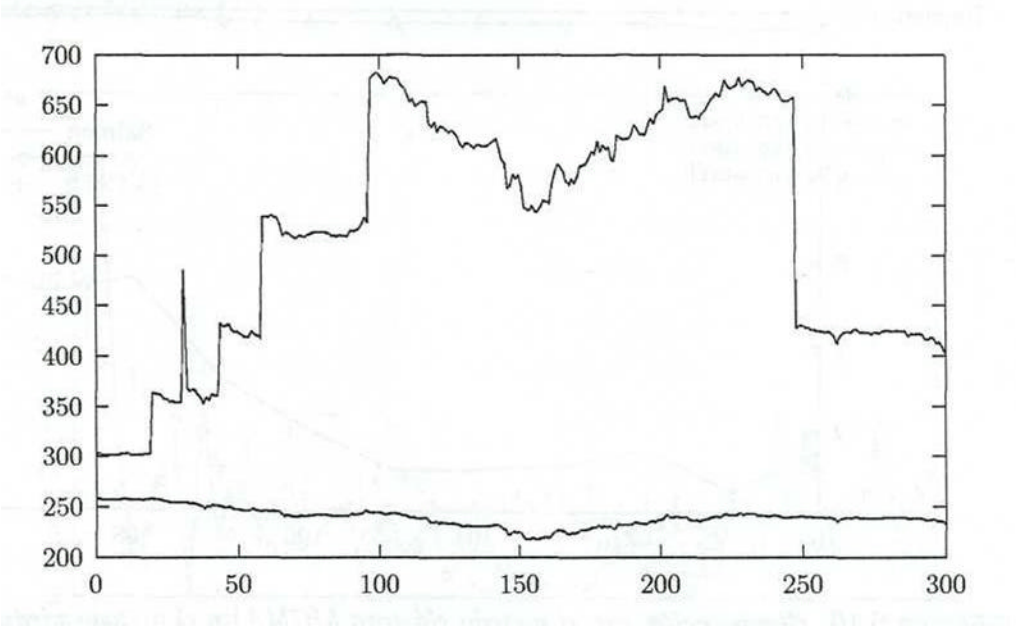

Figura 6.11: *Resultados no favorables de la predicción de una serie de tiempo con valores grandes*

6.4. Conclusiones

Se presentaron los resultados que se obtuvieron al realizar la evaluación de tres modelos de poblaciones hipotéticas, representadas por las series de tiempo sintéticas. Se presentó también los resultados de las pruebas hechas con una serie de tiempo real, representada por el comportamiento de una población de Salmones.

Los resultados se observan en el ajuste de las series de tiempo representadas por la línea con diamantes. Los errores generados al evaluar los modelos parciales se representan en los cuadros de NMSE. Con respecto a los errores producidos por el proceso genético, se aprecia que el NMSE disminuye conforme pasan las generaciones, y los modelos parciales encuentran los valores que más se acercan a la serie de tiempo propuesta.

Capítulo 7

Conclusiones y perspectivas

7.1. Conclusiones

En este trabajo se desarrolló un sistema multiagente llamado sistema *Genetic Collective Leslie (GECOLE),* para la predicción de sistemas poblacionales. Los sistemas poblacionales son un área importante de estudio, en sistemas dinámicos.

El sistema (GECOLE) genera un equipo de agentes que obtienen la predicción colectiva, de una serie de tiempo de una población en estudio, mediante modelos de Leslie.

Los agentes generados utilizan modelos de Leslie como una forma de razonamiento. El modelo de Leslie describe el comportamiento de una población. Un problema importante al utilizar los modelos de Leslie es la elección de los parámetros en las matrices que describen el crecimiento, mortalidad y reproducción de los organismos.

En este trabajo, se desarrolló una técnica que utiliza un algoritmo genético conducido por series de tiempo de una población, para obtener los parámetros de un conjunto de matrices de Leslie.

Se espera que este conjunto de matrices modelen adecuadamente la población en estudio de manera colectiva. Las mejores matrices, obtenidas por el algoritmo genético, son empleadas como mecanismo de razonamiento en los agentes. Estos modelos se obtienen para observar el ajuste que se obtiene en la predicción de la serie de tiempo.

Una vez realizado lo anterior, los agentes realizan predicciones individuales del comportamiento del sistema, las cuales son empleadas para obtener una solución global mediante un proceso simple de media aritmética.

Para el sistema desarrollado se empleó una metodología para el desarrollo de SMA, y la plataforma MultiAgent Development Kit (MadKit).Esta plataforma presenta características adecuadas para el desarrollo del sistema como son:

- Esta plataforma está basada en el modelo organizacional Aalaadin, que presenta a un agente como una entidad que juega un rol y pertenece a un grupo específico.
- No tiene una estructura rígida para la elaboración de agentes.
- Existe librerías adecuadas para implementar la comunicación.

La principal aportación de este trabajo fue presentar una técnica propuesta para buscar los modelos apropiados de matrices de Leslie que se ajusten a un modelo en estudio de una población global. Estos modelos son buscados mediante un algoritmo genético que es conducido por una serie de tiempo que se propone como patrón de una población en estudio. Un sistema multiagente se desarrolla para generar un equipo de agentes predictores independientes y encontrar una solución global.

7.2. Publicaciones

Otros resultados obtenidos en este trabajo fueron las siguientes publicaciones:

- Evolución de Poblaciones Mediante Agentes. García M. J. Antonio. Soto C. Ramón, Núñez E. Gustavo. Publicación en extenso de las memorias: XXIV Congreso Internacional de Ingeniería Electrónica, Instituto Tecnológico de Chihuahua, Chih. México. Vol. XXIV, ISSN: 1405-2172, pp. 339-344.
- Evolución de Poblaciones Mediante Agentes. García M. J. Antonio. Soto C. Ramón., Núñez E. Gustavo. Publicación en el libro: Temas Selectos de Computación 2001-2002, editado por la Universidad Autónoma del Estado de Hidalgo, 2003. ISBN: 9686340-920, págs. 246-257.
- Ajuste Genético de Modelos Colectivos de Leslie Basados en Series de Tiempo. García M. J. Antonio. Soto C. Ramón., Núñez E. Gustavo. Publicación en extenso de las memorias: XXV Congreso Internacional de Ingeniería Electrónica, Instituto Tecnológico de Chihuahua, Chih. México. Vol. XXV, ISSN: 1405-2172.

7.3. Limitaciones

La aproximación que se desarrolló presenta resultados favorables de acuerdo a las pruebas hechas y los resultados obtenidos con las bases de conocimientos antes mencionadas. Como

primer caso se tomaron series de tiempo sintéticas específicamente series distintas para probar el planteamiento propuesto. Como segundo caso se considero una serie de tiempo real tomada originalmente de CPB/NCEAS, para observar también en este caso como se comporta el modelo propuesto.

Sin embargo cabe mencionar que existen limitaciones para todo sistema desarrollado, en este trabajo el sistema multiagente GECOLE presenta la limitación de no ser un sistema genérico que sirva para predecir cualquier base de conocimiento o serie de datos, sino que se realizaron todas las pruebas correspondientes para los ejemplos planteados tomando en cuenta algunas series reales.

Las pruebas realizadas se hicieron con series de tiempo que contenían valores pequeños, que representaban un tamaño a escala de la población, en estas pruebas los resultados son aceptables. En algunas series de tiempo los datos de las observaciones son representadas por valores grandes, el sistema GECOLE en este tipo de series de tiempo proporciona resultados que no son favorables para un buen ajuste y predicción.

Esta consideración pretende plantear que el modelo propuesto tiene limitaciones y que funciona bien para los casos planteados y casos similares, pero para los casos de series de tiempo con valores grandes funciona no tan aceptablemente como se puede apreciar en la gráfica de la Figura 6.11 mostrada en el capitulo 6 Resultados. Esto es bien conocido por mencionar algún ejemplo las redes neuronales artificiales y por eso la diversidad de las diferentes tipos de redes [Diez *et.* al, 1999].

7.4. Perspectivas

Con este trabajo, se inicia una línea de investigación, y una plataforma de desarrollo para poder seguir realizando diferentes pruebas en el área de otras técnicas de modelado de Dinámica de Poblaciones, y una forma de utilizar los sistemas Multiagente.

Como perspectivas se presentan los siguientes apartados:

- La realización de pruebas con otras series de tiempo, para evaluar la capacidad del modelo.
- Integración del modelo predictor, al proyecto de investigación 38274-A de Conacyt, el cual utiliza otras técnicas de predicción. El objetivo es presentar resultados del comportamiento del modelo predictor en el sistema Sigma.
- Evaluar al método con métodos tradicionales, y presentar una comparación de ellos.

¦ Aplicación a un problema real, pensando en sistemas pesqueros con algunas instituciones de ciencias marinas (UABC-Universidad Autónoma de Baja California, CICESE-Centro de Investigación Científica y de Educación Superior de Ensenada).

Bibliografía

[Abramson & Zanette, 1998] Abramson, G., & Zanette D. H., *Statistics of extinction and survival in LotkaVolterra systems.* Phys. Rev. E,57,4572.(1998).

[AgentBuilder, 1999] *AgentBuilder User 's guide.* Reticular Systems, external documentation, http://www.agentbuilder.com/, August 1999.

[Agudelo & Ríos, 1997] Agudelo, J., & Rios A., Sem. *Inteligencia Artificial.* Medellin. Universidad de Antioquia 1997.

[Amsterdam, 1994] Amsterdam, B. V., *Stochastic Models of Age-Structured Populations.* (1994) OPA (Overseus Publishers Association)

[Bäck *et. al, 1996]* Bäck, T., Schwefel, Hans-Paul, *An Overview of Evolutionary Algorithmé for Parameter Optimization, University* of Dortmund.

[Bellifemine, 2001] Bellifemine, Fabio, *Java Agent Development Framework.* What it is and what it is next. Telecom Italia Lab - Torino (Italy). ETAPS 2001, 7th Apr. 2001.

[Bentley, 1999] Bentley, P. J., *Evolutionary Design by Computers.* Morgan Kaufmann Publishers Inc., San Francisco, CA. [BEEB1]. (Contributing Editor) (1999).

[Berryman, 1981] Berryman, A. A., 1981. Population Systems: A General Introduction. Plenum Press.

[Box *&* Jenkins, 1976] Box, G. E. P., Jenkins, G. M., *Time Series Analysis: Forecasting and Control.* Holden Day. (1976).

[Brenner *et.* al, 1998] Brenner, W., Zamekiq, R., and Wittig H., *Intelligent Agents: Foundations and Applications,* Berlin: Springer-Verlag, 1998.

[Bussetta *et.* al, 1999] Busetta, P., Rónnquist, R., Hodgson A., Lucas A., *"JACK Intelligent Agents - Components for Intelligent Agents in Java",* updated from AgentLink Newsletter, http://www.agent-software.com.au/ , October 1999.

[Caswell, 1989] Caswell, H., *Matrix Population Models: Construction, Analysis, and Interpretation.* Sunderland, MA: Snauer Associates, 1989.

[Collis & Ndumu, 1999] Collis, J.; Ndumu, D., *"The ZEUS Technical Manual",* external documentation, http://labs.bt.com/projects/agents/zeux/, September 1999.

[Cullen, 1985] Cullen, M. R., *Linear Models in Biology, Ellis Horwood Series Mathematics and Its Applications.* Ellis Horwood, Chichester, 1985.

[Cristea *et.* a*l*, 2000] Cristea, P., Arsene, A., and Nitulescu B., *Evolutionary Intelligent Agents, Proceedings of the Congress on Evolutionary Cornputation* (CEC-2000), San Diego, USA, p. 1320-1328.(2000).

[DeJong, 1975] De Jong, K. A., *An analysis of the behavior of a class of genetic adaptive systems.* Estados Unidos, Tese (doutorado), - University of Michigan 1975.

[Diez *et.* oZ, 1999] Diez, R. P., Moreno, P. P., Fernández J. P., Fernández N. G., *Aplicación de Redes Neuronales Artificiales al Cálculo de Previsiones a Corto Plazo en el Mercado Eléctrico Español.* Technical Report. ETSII e II de Gijón. Universidad de Oviedo.

[Ferber & Gutknecht, 1998] Ferber, J., y Gutknecht, O., *A meta-model for analysis and design of multi-agent systems,* Proceedings of the 3rd International Conference on Multi-Agent Systems, (ICMAS'98), IEEE, pp. 155-176, August 1998.

[Finklestein & Carson, 1985] Finklestein, L., y Carson, E. R., *Mathematical Modelling of Dynamic Biological Systems.* John Wiley & Sons. 355pp. 1985.

[Fogel *et.* a*l*, 1966] Fogel, L.J., Owens, A.J., & Walsh M.J., *Artificial Intelligence through Simulated Evolution,* New York: Wiley, (1966).

[Goldberg, 1999] David, E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning. (1999).* Addison Wesley.

[Goodman & In Soule, 1987] Goodman, D., In Soule M., *Viable Populations for Conservation.* Cambridge; Cambridge University Press, 1987.

[Grefenstette, 1995] Grefenstette, John J., *Predictive Models Using Fitness Distributions of Genetic Algorithms.* In Foundations of Genetic Algorithms 3, L. Darrell Whitley and Michael D. Vose, editors, pp. 139-162. San Mateo, CA: Morgan Kaufmann, 1995.

[Guessoum & Briot, 1999] Guessoum, Z., Briot, J.P., *From Active Objects to Autonomous Agents,* IEEE Concurrency, July-September, 1999, pp.68-76

[Gutknecht & Ferber, 1997] Gutknecht, O., y Ferber, J., *Madkit: Organizing heterogeneity with groups in a plataform for múltiple multi-agents systems.* December 1997. Technical Report. 97188, LIRMM, 161, rué Ada-Montpellier-France.

[Gutknecht & Ferber, 1998] Gutknecht, O., Ferber, J., *The MADKIT Agent Plataform Architecture.* Agents WorkShop on Infrastructure for Multi-Agent Systems, 1998.

[Hernández, 1998] Hernández, A. A., *Introducción a las Redes Neuronales Artificiales. Soluciones Avanzadas;* pags 25-34. Noviembre 1998.

[Heitkótter & Beasley, 2000] Heitkótter, J., y Beasley, D., *The Hitch-Hiker's Guide to Evolutionary Computation: A list of Frequently Asked Questions (FAQ),* usenet: comp.ai.genetic. Eds. (2000)

[Hitz *k* Werthner, 1996] Hitz, M., & Werthner, H., *Earning Benefits of the Object-Oriented Paradigm in Dynamic System Simulation.* Copyright 1996 IEEE. Published in the Pro-ceedings of the 30th Annual Hawaií, U.S.A.

[Holland, 1992] Holland, J. H., *Adaptation in Natural and Artificial Systems,* 2da edition. Cambridge: MIT Press/Bradford Books, 1992.

[Juchem & Bastos, 2001] Juchem, M., y Bastos, R. M., *Engenieria de Sistemas Multiagentes: Urna Investigacao sobre o Estado da Arte.* Facultad de Informática. PUCRS-Brazil; April-2001.

[Koza, 1992] Koza, J., (1992) : *Genetic Programming.* Cambridge : M.I.T. Press.

[Larousse, 1991] Ramón, G. P., Gross M., *Enciclopedia científica larousse.* Segunda edición actualizada. Ed.Larousse, 1991.

[Leslie, 1945] Leslie, P. H., *On the Use of Matrices in Certain Population Mathematics,* Biometrika 37, 1945, pp. 183-212.

[Lotka, 1925] Lotka, A. J., *Elements of physical biology.* Baltimore: Williams & Wilkins Co. (1925).

-Volterra, *Two Species Model* http://www.stolaf.edu/people-/mckelvey/envision.dir/lotka-volt.html

[Maes, 1997] Maes, P., *Software Agents Tutorial, Conference on Human Factors in Computing Systems,* MIT Media Laboratory, 1997. CHI97.

[Makridakis *et.* al, 1983] Makridakis, S., Wheelwright S.C., and Hyndman R.J., *Forecasting: methods and applications,* New York: John Wiley & Sons. (1998).

[Malthus, 1798] Malthus, T., *An Ensay on the Principie of Population,* London, Printed For J. Johnson, In St. Paula Church-Yard, 1798. http://www.ac.wwu.edu/stephan/malthus/malthus.0.html

[McLulich, 1937] McLulich, D. A., *Fluctuations in the numbers of the varying haré (Lepus americanus).* University of Toronto Studies, Series No. 43. University of Toronto Press, Toronto. 1937.

[Michalewicz, 1992] Michalewicz, Z., *Genetic Algorithms + Data Structures Evolutionary Programs.,* 1992. Springer-Verlag. AI Series, New York. (1992).

[Min-Soo & Seong-Gon, 2001] Min-Soo, K., & Seong-Gon, K., *Parallel-Structure Fuzzy Systems for Time Series Prediction,* International Journal of Fuzzy Systems, Vol.3, No.l, March 2001.

[Morrison, 1991] Morrison, F., *Forecasting for Chaos, Randomness, & Determinism.* Multi-science Press, Inc., 1991.

[Nakaoka, 1993] Nakaoka, M., *Yearly variation in recruitment and its efects on population dynamics in yoldia notabilis (mollusca:bivalvia), analyzed using proyection matriz model.* Res. Popul. Ecol., 35:199-213, 1993.

[Nerc, 1999] NERC Centre for Population Biology, Imperial College (1999) The Global Population Dynamics Datábase, http://www.sw.ic.ac.uk/cpb/cpb/gpdd.html

[Nicholson & Bailey, 1935] Nicholson, A. J. y Bailey, V. A., *The balance of animal populations.* Proccedings of the Zoological Society of London, 3, 551-598. (1935).

[Odum, 1983] Odum, H. T., *Systems Ecology: an Introduction.* John Wiley & Sons. 644 pp. 1983.

[Palacios *et. a*l, 2002] Palacios, C. A., Soto, C. R., Núñez, E. G., *Cooperación en Equipos de Agentes Expertos,* Publicación en extenso de las memorias: XXIV Congreso Internacional de Ingeniería Electrónica, Instituto Tecnológico de Chihuahua, Chih. México. Vol. XXIV, ISSN: 1405-2172, pp. 333-338.

[Pereira & Costa, 2001] Pereira, F., Costa E., *How Learning Improves the Performance of Evolutionary Agents: a Case Study with an Information Retrieval System for a Distributed Environment* (2001).

[Piñero, 1998] Pinero, D., *De Las Bacterias al Hombre: La Evolución.,México* 1998. FCE.

[Priestley, 1981] Priestley, M. B., *Analyisis and Time Series.* Vol.I. Univariate Series. Academic Press Inc. 1981.

[Rendón, 2000] Rendon, G. A., *El Lenguaje Unificado de Modelado (UML),* Universidad del Cauca, Facultad de Ingeniería Electrónica y Telecomunicaciones Departamento de Computación. Popoyán, http://www.inf.udec.cl/intartif/porque.htm, 2000.

[Russell & Norvig, 1995] Russell, S., Norvig, P., *Artificial Intelligence A Modern Approach.* New Jersey, Pretice-Hall, 1995.

[Rumbaugh *et. a*l, 1999] Rumbaugh, J., Jacobson, I., Booch, G., *The unified modeling language reference manual.* Workingham, Inglaterra: Addison Wesley Longman, 1999.

[Sathyan, 2000] Sathyan, N. S. R,. and Chellapilla, K., *Evolving Recurrent Bilinear Perceptrons for Time Series Prediction.* Departament of Electrical and Computer Engineering. Villanova University. Villanova. PA, U.S.A 19085

[Shoham, 1991] Shoham, Y., *Agent-oriented programming extended abstract.* In Proceedings of the IJCAI91 Workshop on Theoretical and Practical Design of Rational Agents, 1991.

[Shoham, 1993] Shoham, Y., *Agent Oriented Programming,* Artificial Intelligence, 6:pp. 51-92, 1993.

[Simón, 2002] Simón, E., *Forecasting foreing Excahnge Rates with Neural Networks.* EPFL (March 2002).

[Sinclair & Shami, 1997] Sinclair, M. C., Shami, S. H., *Evolving Simple Software Agents: Comparing Genetic Algorithm and Genetic Programming Performance.* In: Proc. of the 2nd Intl. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications. IEE Press, London 1997 421426

[Soto & Núñez, 2001] Ramón, S. C, & Núñez, E. G., *Conjuntos Difusos Dinámicos para Predicción de Series de Tiempo.* Centro de Investigación en Tecnologías de Información y Sistemas (CITIS). Universidad Autónoma del Estado de Hidalgo. 2001.

[Soto &: Núñez,2002] Soto, C. R., Núñez, E. G., *Simulación de Sistemas Débilmente Estructurados Basada en Sistemas Genéticos Multiagentes.* Publicación en extenso de las memorias: XXIV Congreso Internacional de Ingeniería Electrónica, Instituto Tecnológico de Chihuahua, Chih. México. Vol. XXIV, ISSN: 1405-2172, pp. 333-338.

[Soto & Núñez, 2003] Soto, C. R., Núñez, E. G., *Soft Modelling of Financial Time Series.* A Publication of the International Association of Science and Technology for Development - IASTED. February 24-26, 2003. Palm Springs, California, USA. ISBN: 0-88 986-337-7. ISSN: 1021-818 1. Anabeim. Calgary. Zurich

[Thompson, 1956] Thompson, *Introduction a The Origin of Species.* Londres, J.M. Dent & Co., 1956.

[Tong, 1990] Tong, H., *Non Linear Time Series. A dynamical system approach.* Oxford Science Publications. (1990).

[Tuljapurkar, 1986] Tuljapurkar Shripad, *Demographic applications of random matriz producís,* pp. 319-326, In Random Matrices and their applications, J.E. Cohén, H. Kesten, C.M. Newman (Eds.), American Mathematical Society, Providence, RI. 1986.

[UML, 1999] *UML Notation Guide.* Disponible por WWW en http://www.rational.com/uml (jun 1999).

[Weigend *et.* a*l*, 1999] Weigend, S. A., Hubberman, B. A., *k* Rumelhart D. E., *Prediciendo manchas solares y cambios con redes conectadas.* (1999).

[Weiss, 1999] Weiss, G., *Multiagent Systems - A moderm Approach to Distributed Artificial Intelligence* (ed.) Gerhard Weiss. The MIT Press Cambridge, Massachussetts London, England, 1999.

[Wooldrige, 1995] Wooldridge, M.J., *Intelligent Agents: Theory and Practice.* Departament of Computing, Manchester Metropolitan University. Jennings, Nicholas. Departament of Electronic Engineering. Queen Mary & Westfield College, 1995.

[Wooldrige & Jennings, 1995] Wooldridge, M., & Jennings N., 1995. *Intelligent Agents: Theory and Practice.* Knowledge Engineering Review, 10(2): 115-152.

[Wooldrige, 1997] Wooldridge, M., *Agent-based software engineering,* IEE Proc. Software Engineering 144 (1) (1997) 26- 37.

[Yule, 1927] Yule, G. U., *An Introduction to the Theory of Statistics,* 8th Edition. C. Griffin and Co., London. 1927.

Apéndice A

Código de GECOLE

En el presente apéndice se muestra el código que se utilizó para construir al sistema multiagente GECOLE. El código estará dividido por las clases que se utilizaron en el sistema.

A.I. Clase geneticCollectiveGame

```
package geneticCollectiveGame;

import madkit.kernel.*;
import madkit.lib.reactive.*;


import madkit.lib.simulation.*;


import madkit.lib.messages.StringMessage;


import java.io.*;
import java.útil.*;
import java.lang.*;


public class geneticCollectiveScheduler extends Scheduler {


int delay=100,nump=3;
Interfaz display;
AgentPredictor[] p;
        int[] RolPresa= new int[l];
```

```
int[] AgentesRol= new int [4] ;
StringD rolIT={"prey","smartpredator","predator","Dumbprey"};
int[] AgentesGrupo= new int[4];
StringG groupIT=-["productorl","predatorl","predator2","productor2"};

int preynumber, predatornumber, envsize=70;
int predatornumber2 = 120;
int preynumber2 = 190;
int [] paramet;

Prey [] preytabi;
Predator[] predatortab;
Predator[] spredatortab;
DumpPreyD preytab;

public EnvironmentViewer ev;
public Environment e;
public EditorAgent ed;
public WebBrowserAgent web;
AgentObsever obsvr;

int laux=0, RolPres=0;double 1=0; int RolPreso=0;
double conteo=2.0,we=1.0; int faa=0, inc=0;

public geneticCollectiveScheduler(){}

public void initGUI(){
    setGUIObject(display = new Interfaz(this));
    System.out.println("Entre al initGUI...");
    }
    public void println(String mess){
    display.println(mess);
    }
    public void setHuntParams(int_paramet[]){
```

```
        preynumber=_paramet[0];
        predatornumber=_parainet [1] ;
  System.out.println("Estoy en el for de CollectiveScheduler...");
  for (int i=0; i<_paxamet.length;i++)
  System.out.println("Resultado del for "+_paramet[i]);
}
  protected void actívate (){
          println('S. D. Poblaciones SMA. ");
          println("Agente geneticCollectiveScheduler");
          println("Agente Interacción Presa-Depredador.");
          println("Agente LVolterra");
          println("Agente MLeslie");
          println("Agente Editor.");
          println("Agente Help.");
}

  public void huntGameCM
  System.out.println("Ejecutando Depredador-Presa...");

  preytabi = new Prey[preynumber];
  predatortab = new Predator[predatornumber2];
  spredatortab = new Predator[predatornumber];
  preytab = new DumpPrey[preynumber2];


  e = new Environment(envsize,envsize,this); launchAgent(e
  ,"environment",false); printlnC'Enviroment e ... ");
  ev = new EnvironmentViewer(e,5);
  launchAgent(ev ,"Depredador-Presa",true);
  printlnC'EnviromentViewer ev .. ");


  foundGroup("ambiente");
  println("Yeah¡ Encontré el grupo ambiente");
```

```
 joinGroup("ambiente");
requestRoleCambiente", "interfaz gráfica");
presas();
DumbDepredador();
SmartDepredador();
Presasi() ;
obsvr = new AgentObsever(AgentesRol);
agentes();
setActive(true);
ev.update();
ev.onScreen.repaint();
}
public void agenteEdit(){
        ed = new EditorAgent();
        launchAgent(ed,"AgentEdit",true);
}
public void agenteWebBrowser(){
        web = new WebBrowserAgent() ;
        launchAgent(web,"AgentWebBrowser",true);
}
public void agentePredictor(){
        println("Lanzando Agentes predictores:");
        p = new AgentPredictor[nump] ;
        for (int i = 0; i < nump; i++){
            AgentPredictor pred = new AgentPredictor();
                launchAgent(p[i] = pred,"AgentPredictor-"+i,true);
                println("AgentePredictor-"+i);
        }
}
  public void agentes (){
        for(int i=0; i<4; i++){
            AgentesRol[i] =(getAgentsWithRole("productorl",rolIT[i])).length;
            System.out.println("Rol["+i+"]:"+AgentesRol[i]);
        }
```

```
        }
        public void live(){
          println('live");
          ciclolifeO;
        }
  public void ciclolife(){
    AnimalActivator al = new AnimalActivator("productorl","prey");
    AnimalActivator a3 = new AnimalActivator("predatorl"."smartpredator");
    AnimalActivator a4 = new AnimalActivator("predator2","predator");
    AnimalActivator a2 = new AnimalActivator("productor2","Dumbprey");
    SingleMethodActivator a5 = new
    SingleMethodActivator("observe","ambiente"."observervador");
    ComunicacionActivator a6 = new
    ComunicacionActivator("ambiente"."SmartPredator");
addActivator(a5);
addActivator(a1);
addActivator(a3);
addActivator(a4);
addActivator(a2);
addActivator(a6);
a6.execute();
update();
println("Inicia");
    int j=0;
  while(true){
        try {
        if (delay==0)
              Thread.yield() ;
        else
          pause(delay);
        if (active){
            update();
            a1.execute();
            a3.execute();
```

```
        a4.execute();
        a2.execute();
        a5.execute();
        }           j++;
                    factorincrement() ;
                    incremento (); behaviorinteraction();
                    if (j==170){
                        efectenvironment() ;
                    }
                        if (j==310){
                  efectHarvesting();
                }
}
        catch(Exception w){}
       } //while
}
 public void factorincrement(){
              we++;
      RolPres=RolPres+((getAgentsWithRole("productorl", "prey")).length);
      RolPres = (-1)*RolPres;
      System.out.println("SegundoRol"+RolPres);
      obsvr.ploter.addPointii(RolPres);
}
  public void increment(){
RolPres=RolPres+((getAgentsWithRole("productor2", "Dumbprey")).length);
RolPres = (int)(Math.round(Math.pow(conteo,we)))*RolPres;
obsvr.ploter.addPointiii(RolPres);
}
public void efectenvironment(){
         for (int i = 0; i <=predatornumber; i++){
         int fa = (int)(Math.round(Math.random()*(i+20)));
         obsvr.ploter.addPointi(fa);
         killAgent(predatortab[i] );
```

```
          remove(predatortab[i]);
          update() ;
                              }
}
public void efectHarvesting(){
           for  (int i = 0;   i <= 90;  i++){
           killAgent(spredatortab[i]);
           remove(spredatortab[i]);
           update( ) ;
           }
}
public void uncertainly(){
           for  (int  i=l;   i<=5;i++){
           faa = (int)(Math.round(Math.random()*Math.exp(i)));}
           System.out.println("Faa"+faa);
           obsvr.ploter.addPointii(faa);
}
public void behaviorinteraction(){
                    for(int jj=0; jj<4; jj++){
                           for(int i=0; i<4; i++){
                           laux = laux+ (getAgentsWithRole(groupIT[jj] ,rolIT[i])).length;
                           AgentesRol [i] = (getAgentsWithRole ("ambiente",rolIT [i])). length;
                           }
                  AgentesRol[jj]= laux;
                  laux =0;
                  }
         obsvr.ploter.addPoint(AgentesRol);
         }
         public void presas(){
            println("Lanzando Agentes presas");
            for (int i = 0; i < preynumber; i++)      {
            println("Creando Prey:"+i);
            Prey p2 = new Prey();
            p2.setEnvironment(e);
```

```
            launchAgent(preytabi[i] =p2,"prey "+i, false);
        }
    }
    public void presasi(){
        println("Lanzando Agentes presas tontas");
         for (int i = 0; i < preynumber2; i++){
                println("Creando Dumbprey:"+i);
                DumpPrey p = new DumpPrey();
                p.setEnvironment(e);
                launchAgent(preytab[i] = p,"prey-"+i, false);
        }
    }
    public void DumbDepredador(){
        println("Lanzando Agentes depredadores tontos");
        for (int i = 0; i < predatornumber2; i++){
        println(Creando predator:"+i);
        DumbPredator p = new DumbPredator();
        p.setrandom();
        p.setEnvironment(e);
       launchAgent(predatortab[i] = p, "predator"+i, false);
       }
    }
    public void SmartDepredador(){
            println("Lanzando Agentes depredadores listos");
            for (int i = 0; i < predatornumber; i++){
            printlnC"Creando Smart predator :"+i);
            SmartPredator pl = new SmartPredator();
            p1.setrandomC);
            p1.setEnvironment(e);
            launchAgent(spredatortab[i] = pl, "smartpredator"+i,false);
            }
    }
        public void remove(Animal e){
        killAgent(e);
```

```
        update();
        ev.update();
}
public void presasmuertas1(){
int len=preytab.length;
if(len>0)
          for (int i = 0; i <= preynumber; i++){
          killAgent(preytab[i] );
          println("Presa muerta:"+preytab[i]);}
}
 public void presasmuertas2(){
    int len=preytabi.length;
    if(len>0)
            for (int i = 0; i <= preynumber2; i++){
                killAgent(preytabi[i]);
                println("Presa muerta:"+preytabi[i]);}
  }
 public void predatormuertos1(){
    int len=predatortab.length;
    if(len>0)
         for (int i = 0; i <= predatornumber; i++){
          killAgent(predatortab[i]);
              println("Por depredador Dump: "+predatortab[i]);}
}
 public void predatormuertos2(){
 int len=spredatortab.length;
 if(len>0)
          for (int i = 0; i <= predatornumber; i++){
          killAgent(spredatortab[i]);
               println("Por depredador Smart:"+predatortab[i]);} }
}
 public void end(){
 println("Muerte de geneticCollectiveScheduler");
 presasmuertas1();
```

```
            presasmuertas2();
            predatormuertos1();
            predatormuertos2();
}
public int getDelay() {return delay;}
public void setDelay(int v) {this.delay = v;}
public void setActive(boolean b){ active=b; }
public boolean getActive(){
                return active;
       }
        boolean active;
}
```

A.2. Clase AgentPredictor

```
package geneticCollectiveGame;

importmadkit.lib.graphics.*;
import madkit.lib.agents.*;

import madkit.kernel.*;
import java.awt.*; import java.io.*;

import j ava.util.*;
import j ava.awt.event.*;

import javax.swing.event.*; import javax.swing.*;

import madkit.lib.messages.*;

public class AgentPredictor extends AbstractEditorAgent {
    String targetGroup;
    String targetRole;
    GuiPredictor pred;
  public AgentPredictor(){}
```

```
public void initGUI(){
pred = new GuiPredictor(this);
setGUIObject(pred);
}
        protected void activate(){
        println("Agente Activado¡ ¡") ;
        requestRole("ambiente"."predictor");
        setTargetGroup("ambiente");
        setTargetRoleC'predictor");
        mensaje("Agente Activado¡¡¡");
}
 public void live(){
   mensaje("Agente Ejecutándose...");
    while(true){
    Message m = waitNextMessage() ;
            handleMessage(m);
   }
}
 public void setTargetGroup(String group){
        targetGroup = group;
         ((GuiPredictor)pred). showCurrentGroup(group);
  }
 public void setTargetRole(String role){
        targetRole = role;
         ((GuiPredictor)pred).showCurrentRole(role);
 }
  public void mensaje(String m){
    ((GuiPredictor)pred).println(m);
    }
    void handleMessage(Message m){
         if (m instanceof StringMessage){
         StringMessage msg = (StringMessage) m;
         String content = msg.getString();
         }
```

```
}
public void end()-C
println("Proceso Terminado¡¡¡ ¡");
        }
}
```

A.3. Clase GuiPredictor

```
package geneticCollectiveGame;

import java.io.*; import java.útil.*; import java.awt.*; import java.awt.event.*;

import javax.swing.event.*;

import j avax.swing.*; import j avax.swing.border.*;

import madkit.lib.graphics.*; import madkit.lib.agents.*;

import madkit.lib.messages.*; import madkit.lib.tools.*;

import madkit.kernel.*;

public class GuiPredictor extends AbstractGuiPredictor implements

 ActionListener {

      JTextArea outputArea;
      JTextAreaWriter jtaOut;
      PrintWriter out, err;
      JLabel currentTargetLabe1;
      AgentPredictor _ag;
      GeneticSearch geneticSearch;
      int pl=0, p2=0, p3=0;
      public JTextField Field_l, Field_2, Field_3;
      public GuiPredictor(){
          super() ;
```

```
    }

    public GuiPredictor(AgentPredictor ag){
     super(ag);
       geneticSearch = new GeneticSearch(this);
       printlnC'Agent Vive¡¡");
       println("Search Application - Genetic Search");
       printlnC'Genetic object class name");
geneticSearch.setGenetic0bjectClassName("GeneticSearch0bj3");
}
 public GuiPredictor(GeneticSearch gs){
       geneticSearch = gs;
  }
  protected void init(){
       super. lnit();
  }
   void showCurrentGroup(String group){
        stdout () . printlnCGroup: "+group);
  }
   void showCurrentRole(String role){
   stdout().println("Role:"+role);
   }
   public void guardar() {
       JFileChooser fileChooser = new JfileChooser();
       fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
       int result= fileChooser.showSaveDialog(this);
       if (result == JFileChooser.CANCEL.OPTION) return;
       File name= fileChooser.getSelectedFile();
        try {
           BufferedWriter bw=new BufferedWriter(new FileWriter(name,true));
           PrintWriter output= new PrintWriter(bw);//new FileWriter(name));
           output.write(getOutPutArea());
           output.close();
                }
                    catch (IOException ioException) {} }
```

```
        JOptionPanel.showMessageDialog(null,”Error en el archivo,
                “Error”, JOptionPanel.ERROR_MESSAGE);
        }
}
public void abrir(){
      JFileChooser fileChooser = new JfileChooser();
      fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
      int resulta fileChooser.showOpenDialog(this); int 1=0; float num=0,r=0;
      String cabeza; int corte=0;
      if (result== JFileChooser.CANCEL_OPTION) return;
      File name= fileChooser.getSelectedFile();
       if(name.exists()) {
         try {
                DataInputStream datos = new
                    DataInputStream(new FileInputStream(name));
                    String text = new String();
                    text=datos .readLine();
                    corte = text.indexOf(" ");
                    cabeza= text.substring(O, corte);
                     num = Float.valueOf(cabeza).floatValué();
                println(“Registros:>”+num;)
                1=(int)num;
                float[] timE =new float[1+1];
                float[] prba = new float[1+1];
                println("L:0  "+1);
                TrainningPorc(l);
                geneticSearch.setMaxNumRecs(l);
                 System.out.println("Envié valores??");
                 for(int i=lj i<=l; i++){
                       try { text = datos .readLine() ;
                       }catch (Exception e)
                       {System.out.println("Error al leer el archivo:"+e); }
                       corte = text.indexOf(" ");
                       cabeza= text.substring(O, corte);
```

```
                    r=timE[l] = Float.valueOf(cabeza).floatValue();
                    printlnO1:- "+i+" : "+timE[l]) ;
                    prba[1]=r;
  }
System.out.println("Envió r :< "+r);
System.out.println("Envió timeE :( "+timE[l]);
System.out.println("Envió prba[] :( "+prba[l]);
System.out.println("Envió prba :( "+prba);
geneticSearch.setNumRecs(prba);
println("No. Registros:< "+1);
}
catch (IOException ioException)
{JOptionPane.showMessageDialog(null,"Error en el archivo",
        "Error",JOptionPane.ERROR.MESSAGE);}
}
else JOptionPane.showMessageDialog(null," El archivo no existe ",
        " Error ".JOptionPane.ERROR.MESSAGE);
}
public void setParametros(){
        Field_l = getFieldO;
P1 = Integer.valueOf (Field.l.getText()) .intValue() ;
geneticSearch.setMaxNumPasses(pl);  //100 //120 //110
println("Numero de generaciones activado");
p2 = Integer.valueOf(Field2.getText()).intValué ();
geneticSearch.setPopulationSize(p2);
println("Tamaño de población activado");
p3 = Integer.valueOf(Field3.getText()).intValue();
geneticSearch.setReplacementSize(p3);
println("Tamaño de remplazo activado");
//Umbral:Primer paso o entrada
geneticSearch.setFitnessThreshold(10.0);
}
publie void TrainningPorc(int numrec){
int porc=0,send=0;
```

```
        porc = numrec;
        System.out.println("Ya entre :o");
        send =(int)(porc * .70);
        System.out.println("Tran:S ="+send);
        println("Tran:S ="+send);
        geneticSearch.setMaxTraining(send);
}
public void actionPerformed(ActionEvent e){
        command(e.getActionCommand());
 }
protected void command(String c){
        if (c.equalsC"Guardar Pred...")){
                 guardar();
                println("Almacenado¡ ¡");
         }
        else if (c.equals("Activación")){
        geneticSearch.start() ;
        try {
geneticSearch.sleep(200);
} catch (InterruptedException e){ }
                println("Activación Lista¡¡");
        }
        else if (c.equalsC"Activar Parámetros")){
                setParametros();
        }
        else if (c.equals("Cargar serie")){
                abrir(); }
        if (c.equals("Cerrar"))
                close() ;
         else if  (c.equalsí"Salir"))
        quit(true);
        }
    }
```

A.4. Clase Interfaz

```
package geneticCollectiveGame;
import java.awt.*; import java.awt.event.*; import javax.swing.*;
 import java.io.*; import java.útil.*; import javax.swing.event.*;

import java.awt.*; import java.awt.image.*; import madkit.kernel.*;

 import java.awt.event.*; import madkit.lib.graphics.*

 import madkit.lib.messages.*; import javax.swing.*;

import madkit.lib.agents.*; import madkit.lib.tools.*;

public class Interfaz extends JPanel{
    JTextArea outputArea;
    JLabel currentTargetLabel;
    protected JPanel commandPanel;
    JTextAreaWriter jtaOut;
    PrintWriter out, err;
    geneticCollectiveScheduler ag;
     Interfaz (geneticCollectiveScheduler _ag) {
     ag = _ag;
       setLayout(new BorderLayout(l0.l0));
       JPanel panel = new JPanel(new GridLayout(3,l)); JMenuBar jMenuBarl =
       new JmenuBar();
       JMenu jMenuFile = new Jmenu();
       JMenuItem jMenuFileExit = new JmenuItem();
       JMenu jMenuHelp = new Jmenu();
       JMenuItem jMenuItemHelp = new JmenuItem();
       JMenuItem jMenuHelpAbout = new JmenuItem();
```

```
JMenu jMenuAgentes = new JMenu() ;
JMenuItem jMenuItemJEdit = new JMenuItem();
 JMenuItem jMenuItemLVolterra = new JMenuItem();
 JMenuItem jMenuItemPredictor = new JMenuItem();
 JMenuItem jMenuItemMLeslie = new JMenuItem();
 JMenuItem jMenuItemSMA = new JMenuItem();

JMenu jMenuAcciones = new JMenu() ;
 JMenuItem jMenuItemCompilar = new JMenuItem();
JToolBar jToolBar = new JtoolBar();
 jToolBar.addSeparator() ;


JButton jButtonJEdit;
ImageIcon eda = new ImageIcon("geneticLifeGame/inicia.gif");
 jToolBar.add(jButtonJEdit = new JButtonC'AgenteEditor",eda));
JButton jButtonPredictor = new JButtonO'Predictor");
JButton jButtonLVolterra = new JButtonC'LotkaVolterra");
JButton jButtonMLeslie = new JButton("Matriz Leslie");
JButton jButtonSMA = new Jbutton("S. MultiAgente");
JButton jButtonHelp = new JButtonC" Agente Help ");
        jToolBar.add(jButtonPredictor);
        //jToolBar.add(jButtonLVolterra);
        //jToolBar.add(jButtonMLeslie);
        //jToolBar.add(jButtonSMA);
        jToolBar.add(jButtonJEdit);
        jToolBar.add(jButtonHelp);
        jMenuFile.add(jMenuFileExit);
        jMenuAcciones.add(JMenuItemCompilar);
        jMenuAgentes.add(JMenuItemJEdit);
        jMenuAgentes.add(jMenuItemPredictor);
        jMenuAgentes.add(jMenuItemLVolterra);
        jMenuAgentes.add(jMenuItemMLeslie);
        jMenuAgentes.add(jMenuItemSMA);
```

```
jMenuHelp.add(jMenuItemHelp);
jMenuHelp.add(jMenuHelpAbout);
jMenuBaxl.add(jMenuFile);
 jMenuBarl.add(jMenuAcciones);
 jMenuBarl.add(jMenuAgentes);
 jMenuBarl.add(jMenuHelp);
/*******************************************/
jMenuFile.setText("File");
jMenuFileExit.setTextC'Exit");
jMenuFileExit.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
     jMenuFileExit_actionPerformed(e) ;
    }
  } );
/*******************************************/
 jMenuAcciones.setText("Acciones");
 jMenuItemCompilar.setText("Compilar");
/*******************************************/
 jMenuAgentes.setText("Agentes");
 jMenuItemJEdit.setText("A. JEdit");
 jMenuItemJEdit.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e){
           jMenuItemJEdit_actionPerformed(e);
        }
 });
 jMenuItemPredictor.setText("M.  Predictor"); jMenuItemPredictor.addActionListener(new
 ActionListener()   {
        public void actionPerformed(ActionEvent e){
           jMenuItemPredictor_actionPerformed(e);
        }
  });
/*******************************************/
 jMenuHelp.setText("Agente Help");
 jMenuItemHelp.setText("Agente Help");
```

```
jMenuItemHelp.addActionListener(new ActionListener() {
public void actionPerformedCActionEvent e) {
      jMenuItemHelp_actionPerformed(e);
      }
});
jMenuHelpAbout.setText("About");
jMenuHelpAbout.addActionListener(new ActionListener()  {
 public void actionPerformed(ActionEvent e) {
   jMenuHelpAbout_actionPerformed(e);
   }
});
/*******************************************/

jButtonJEdit.addActionListener(new java.awt.event.ActionListener(){ public void
   actionPerformed(ActionEvent e)  {
      jButtonJEdit_actionPerformed(e);
      }
});
jButtonJEdit.setToolTipTextC"Editor Agente");
/*******************************************/
jButtonPredictor.addActionListenerCnew java.awt.event.ActionListener() {
    public void actionPerformedCActionEvent e) {
       jButtonPredictor_actionPerformed(e);
    }
    ;
jButtonPredictor.setToolTipText("Agente Predictor");

jButtonLVolterra.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformedCActionEvent e) {
       jButtonLVolterra_actionPerformed(e);
       }
});

jButtonSMA.addActionListener(new java.awt.event.ActionListener()   { public void
    actionPerformedCActionEvent  e)  {
```

```
  jButtonSMA_actionPerformed(e); }
}
});

jButtonSMA.setToolTipText("Sistema Multiagente");

jButtonHelp.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jButtonHelp_actionPerformed(e);
    }
    });
JButtonSMA.setToolTipText("Ayuda en Linea");
  panel.add(jMenuBarl, BorderLayout.NORTH); //this.
  panel.add(jToolBar, BorderLayout.CENTER);  //this.
  setSize(new Dimensión(600, 400));
  add(panel,"North");
  setMinimumSize(new Dimension(530, 350)); setPreferredSize(new
  Dimensión(530, 350)); //setPreferredSize(new Dimensión(259,
  331));
  outputArea = new JTextAreaC" ,10,40);
     JScrollPane outscroller = new JScrollPane() ;
     outscroller.setSize(300,200);
     outscroller.getViewport().add(outputArea);
     add(outscroller,"Center");

     jtaOut = new JTextAreaWriter(outputArea);
     out = new PrintWriter(jtaOut, true);
     err = out;
     JPanel displayLabelPanel = new JPaneKnew BorderLayout());
     add(displayLabelPanel,"South");
     currentTargetLabel=new Jlabel(“Panel principal. ");
     displayLabelPanel.add(currentTargetLabel,"West");
      }
      void setRecipientLabeKString s){
```

```
        currentTargetLabel.setText("Event: "+s);}
      public void println(String sMout .println(s) ;}
      /**File | Exit action performed*/
     public void jMenuFileExit_actionPerformed(ActionEvent e) {
     System.exit(0);
     }
/**Help I About action performed*/
public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
     Interfaz.AboutBox dlg = new Interfaz_AboutBox(this);
     Dimensión dlgSize = dlg.getPreferredSize();
     Dimensión frmSize = getSize();
     Point loe = getLocation() ;
     dlg.setLocation((frmSize.width - dlgSize.width)
         / 2 + loc.x, (frmSize.height - dlgSize.height) / 2 + loe.y);
     dlg.setModal(true);
     dlg.show();
     }
      public void jMenuItemHelp_actionPerformed(ActionEvent e){
            ag.agenteWebBrowser() ;
 }
   public void jMenuItemJEdit_actionPerformed(ActionEvent e){
     ag.agenteEditO ;
}
 public void jMenuItemLVolterra_actionPerformed(ActionEvent e){
     GuiLotkaVolterra panel = new GuiLotkaVolterra(this);
     panel.show();
     }
     public void jMenuItemPredictor_actionPerformed(ActionEvent e){
     ag.agentePredictor() ;
     }
 public void jMenuItemMLeslie_actionPerformed(ActionEvent e){
 GuiMatrizLeslie panel3 = new GuiMatrizLeslie(this);
 panel3.show();

 }
```

```
 public void jMenuItemSMA_actionPerformed(ActionEvent e){
  geneticGamePanel panel2 = new geneticGamePanel(this);
   panel2.show();
  }
  protected void processWindowEvent(WindowEvent e) {
     this.processWindowEvent(e);
  if (e.getIDO — WindowEvent.WINDOW.CLOSING) {
      jMenuFileExit_actionPerformed(null);
       }
    }
    void jButtonSMA_actionPerformed(ActionEvent e) {
      geneticGamePanel panel2 = new geneticGamePanel(this);
            panel2.show();
    }
    void jButtonJEdit_actionPerformed(ActionEvent e) {
           ag.agenteEdit() ;
     }
     void jButtonHelp_actionPerformed(ActionEvent e) {
             ag.agenteWebBrowser();
    }
     void jButtonPredictor_actionPerformed(ActionEvent e) {
          ag.agentePredictor() ;
 }
 }
class geneticGamePanel extends JDialog implements ActionListener {
JTextArea outputArea;
JLabel currentTargetLabel;
protected JPanel commandPanel;
JTextAreaWriter jtaOut;
PrintWriter out, err;

Gráfica grafical=null;
```

```
AgentObsever agentgraf=null;
ParamsLotka param;
Double[] parame;
GetParams paramsWin;

Interfaz in;
geneticCollectiveScheduler ag;
geneticGamePaneKgeneticCollectiveScheduler _ag){
     ag = _ag;
}
geneticGamePanel (Interfaz _in){
    in = _in;
    setTitle("Modelo Sistemas MultiAgentes");
    setSize(350,250);
    getContentPane().setLayout(new BorderLayout(10,10));
    JPanel buttonPanel = new JPanel(new GridLayout(4,4));

    JMenuBar jMenuBarl = new JMenuBar();
    JMenuBar jMenuBar2 = new JMenuBar() ;

    JMenu jMenuAcciones = new JMenu();
    JMenuItem jMenuItemParametros = new JMenuItem() ;
    JMenuItem jMenuItemActivacion = new JMenuItem();
    JMenuItem jMenuItemOpciones = new JMenuItem();
    JMenuItem jMenuItemSalir = new JMenuItem();

   jMenuAcciones.add(jMenuItemParametros);
   jMenuAcciones.add(JMenuItemActivacion);
   jMenuAcciones.add(jMenuItemOpciones);
   jMenuAcciones.add(JMenuItemSalir);

   jMenuAcciones.setTextC'Acciones");
   jMenuItemParametros.setText("Parámetros Gamescheduler");
   jMenuItemActivacion.setText("Activación de Ambiente");
```

```
        jMenuItemOpciones.setText(”Opciones”);
        jMenuItemSalir.setText(“Salir”);

        jMenuBarl.add(jMenuAcciones);
        buttonPanel.add(jMenuBar1);
        buttonPanel. add(jMenuBar2) ;
          JButton buttoni = new JButton("Parámetros Gamescheduler");
                    buttoni.addActionListener(this);
          JButton buttonii = new JButton("Activación de Ambiente");
                    buttonii.addActionListener(this);
          JButton buttonv = new JButton("Opciones");
                    buttonv.addActionListener(this);
          JButton buttonvi = new Jbutton(“Salir");
                    buttonvi.addActionListener(this);
          buttonPanel.add(buttoni);
          buttonPanel.add(buttonii);
          buttonPanel.add(buttonv);
          buttonPanel.add(buttonvi);
getContentPane() . add (buttonPanel, "North");
outputArea = new JTextArea("",10,40);
JScrollPane outscroller = new JscrollPane();
 outscroller.setSize(300,200);
outscroller.getViewport().add(outputArea);
 getContentPane().add(outscroller,"Center");
 jtaOut = new JTextAreaWriter(outputArea);
out = new PrintWriter(jtaOut, true);
          err = out;
 JPanel displayLabelPanel = new JPaneKnew BorderLayout());
 getContentPaneO.add(displayLabelPanel,"South");
 currentTargetLabel=new JLabel("Panel principal. ");
 displayLabelPanel.add(currentTargetLabel,"West");
  printl(“- Agente Activado");
  }
```

```
void setRecipientLabel(String s){
      currentTargetLabel.setTextC'Event: "+s);
}
public void println(String sMout .println(s) ;}
public void actionPerformed(ActionEvent e) {
    String c = e .getActionCommand() ;
    if(c.equals("Parámetros Gamescheduler"))
    {
          int params [] =new int [2] ;
          params[0]=160; params[1]=130;   // 160    130
          final GetParams paramsWin = new GetParams(params,this);
          in.ag.setHuntParams(params);
          WindowListener 1 = new WindowAdapter() {
          public void windowClosing(WindowEvent e) -CparamsWin.setVisible(false);}};
          paramsWin.addWindowListener(l);
          paramsWin.pack();
          paramsWin.setVisible(true);
          println("Activado ->");
          setRecipientLabeK"Enviando parámetros... ");
          String sv="Valores:";
          for(int tec=O; tec<params.length; tec++)
          sv+=params[tec]+" ";
          println(sv);
          }
      else if (c.equals("Activación de Ambiente"))-{
             in.ag.huntGame();
             in.ag.update() ;
              println("Activando parámetros -> ");
              setRecipientLabeK"Ambiente activo") ;
       }
        else if (c.equals("Opciones")){
           println("Opciones cargadas ->"); }
           else if (c.equals(“Salir”)) System.exit(0);
```

```
    }
    Frame getFrameParent(){
     Component c = this;
    while (!(c instanceof Frame))
         if (c == nuil)
                  return(null);
         else
                  c = c.getParent();
                  return((Frame) c);
     }
}
class GetParams extends JFrame {
      int [] paramet;
       JTextField Fieldl,Field2;
      geneticGamePanel _m;

   public GetParams (int[] _params,geneticGamePanel m) {
         super("Depredador-Presa");
        paramet = _params;
        _m = m;
        Fieldl = new JTextField (new Integer(paramet[0]).toString(), 5);
         Field2 = new JTextField (new Integer(paramet[1]).toString(), 5);

        JPanel labelPanell = new Jpanel();
        labelPanell.setLayout(new GridLayout(0, l));
        labelPanell.add(new JLabeK"PARÁMETROS "));
        labelPanell.add(new JLabe("                                                    "));

        JPanel labelPanel = new Jpanel();
         labelPanel.setLayout(new GridLayout(0, l));
        labelPanel.add(new JLabel("Número de presas: "));
         labelPanel.add(new JLabel("Número de depredadores: "));

        JPanel fieldPanel = new JPanelO;
```

```
    fieldPanel.setLayout(new GridLayout(0, 1));
     fieldPanel.add(Fieldl);
     fieldPanel.add(Field2);

    JButton OkButton;
    JPanel buttPanel = new JPanel();
     buttPanel.setLayout(new GridLayout(1, 0));
    buttPanel.setBorder(BorderFactory.createEmptyBorder(20, 80, 20, 80));
    buttPanel.add(OkButton = new Jbutton(“OK"));
    OkButton.setMnemonic('o');
    OkButton.addActionListener(
          new ActionListener(){
               public void actionPerformed(ActionEvent evt) •{
                          okPressedO ;
               }
    });
     JPanel contentPane = new Jpanel();
          contentPane.setBorder(BorderFactory.createEmptyBorder(20, 20, 0, 20));
           contentPane.setLayout(new BorderLayout());
          contentPane.add(“North".labelPanel1);
          contentPane.add(labelPanel);
          contentPane.add(fieldPanel, "East");
           contentPane.add(buttPanel, "South");
          setContentPane(contentPane);
}
 public void okPressed() {
      paramet[0] = Integer.valueOf(Fieldl.getText ()).intValue();
      paramet[1] = Integer.valueOf(Field2.getText ()).intValue ();
       String sv="enviados:";
      for(int tec=0; tec<paramet.length; tec++){
           sv+="Param ->"+paramet[tec]+" ";
           }
           _m.println(sv);
```

```
            for  (int i=0;   i<paramet.length;i++)
            System.out.println(paramet[i]);
            }
} // Fin de la clase
```