Laboratorio computacional en el ámbito de la economía computacional basada en agentes

Computational Laboratory in the field of the computational economics in agent-based

Fabián Andrés Giraldo

Fecha de recepción: 3 de abril 2013
Fecha de aceptación: 5 de mayo de 2013

Resumen

El presente artículo presenta un laboratorio computacional en el ambiente de la economía computacional basada en agentes. Dicho laboratorio permite la especificación de modelos de simulación usando un lenguaje desarrollado denominado UNALCOL. El lenguaje tiene una serie de características entre las cuales se encuentra un entorno integrado de desarrollo que facilita las tareas de programación y una plataforma de simulación para los modelos especificados.

Una característica importante de este lenguaje es que permite la integración con librerías externas para resistir el proceso de soporte a decisiones. Con el fin de validar el funcionamiento del lenguaje se presenta como caso de prueba un modelo de juegos no cooperativos repetitivo, específicamente el dilema del prisionero iterado. Se configuran como estrategias de juegos ALLC (Cooperación), TFT (Tit For Tac) y por último un perceptrón multicapa. Los resultados obtenidos en el proceso de simulación evidencian la cooperación mutua.

Palabras clave: Economía computacional basada en agentes de juegos no cooperativos, laboratorio computacional, dilema del prisionero iterado.

Abstract

This paper presents a computer lab in the ambience of the computational economics in agent-based. This laboratory allows the specification of simulation models using a language called UNALCOL. The language has a number of features among which there is an integrated of development that facilitates the tasks of programming and simulation platform for the specified models.

An important feature of this language is that it allows the integration with external libraries to resist the decision support process. In order to validate the operation of the language is presented as a model test case of repeated noncooperative games, specifically the iterated prisoner's dilemma. This games are configured as game strategies ALLC (Cooperation), TFT (Tit For Tac) and finally a multilayer perceptron. The results obtained in the simulation process evidencing of the mutual cooperation.

Key words: Computational economics in agent-based noncooperative games, computer lab, iterated prisoner's dilemma.

1. Introducción

En años recientes la ciencia de la complejidad ha sido reconocida por diferentes académicos en importantes disciplinas científicas como la física, la economía, las ciencias de la computación y las ciencias sociales, por su capacidad para modelar sistemas no lineales. Dicha ciencia estudia los sistemas como un conjunto de componentes interconectados cuyo comportamiento no es explicable exclusivamente por las propiedades de los mismos, el comportamiento emerge de las estructuras interconectadas.

Con el objeto de modelar los sistemas complejos, se han venido desarrollando una serie de paradigmas de modelamiento y simulación entre los que se encuentran la dinámica de sistemas y la simulación basada en agentes.

La simulación basada en agentes es un paradigma de modelamiento que se caracteriza por comprender cómo varios agentes (autónomos, heterogéneos e independientes), con sus propias metas y objetivos, son capaces de realizar interacciones entre sí y con su entorno. Para el caso particular de este trabajo el objetivo de la simulación basada en agentes es tratar de inferir las propiedades globales de todo el sistema o identificar los patrones de comportamiento emergentes a partir de las reglas que determinan el comportamiento individual de los agentes. El comportamiento global del sistema no es abstraído, sino que emerge durante el proceso de inferencia al ejecutar el modelo [1].

Un escenario de interés particular en donde es utilizada la simulación basada en agentes es la economía computacional, siendo un área del pensamiento económico que rechaza los supuestos tradicionales, encargados

de indicar que la economía es un sistema cerrado que eventualmente logra un estado de equilibrio en el cual supuestos de racionalidad perfecta, inversiones homogéneas, entre otros, deben efectuarse para permitir el análisis matemático. En su lugar esta área del pensamiento ve a la economía como un sistema complejo, adaptativo y dinámico que trata de comprender el comportamiento emergente del sistema macroscópico a partir de la dinámica microscópica de cada agente (humanos, firmas, mercado, etc.) [2].

La economía computacional basada en agentes (ACE) estudia los procesos económicos modelados como sistemas dinámicos de agentes interactuando. Ejemplos de posibles agentes incluyen individuos (consumidores, trabajadores), grupos sociales (familias, firmas, agencias gubernamentales), instituciones (sistemas reguladores del mercado), entidades biológicas (cultivos, ganados, bosques) y entidades físicas (infraestructura, clima y regiones geográficas).

La investigación actual sobre ACE está encaminada en los siguientes aspectos: 1). Comprensión empírica: porque particularidades globales evolucionan y persisten a pesar de la ausencia de control y planificación centralizada; 2). Comprensión normativa: encargada de determinar cómo los modelos basados en agentes pueden utilizarse como laboratorios para el descubrimiento de buenos diseños económicos. El objetivo es evaluar propuestas de diseños de políticas económicas, instituciones, procesos y cómo son deseables socialmente en el tiempo; 3). Visión cualitativa y generación de teorías: analiza cómo pueden los sistemas económicos comprenderse más plenamente a través de un examen sistemático de los comportamientos dinámicos potenciales bajo condiciones iniciales especificadas [3].

Sin embargo existe una serie de líneas de investigación bien definidas en el campo de la economía computacional basada en agentes, entre las que se pueden mencionar [4]:

- Aprendizaje: la idea principal es buscar cómo modelar la mente de los agentes computacionales. Los investigadores en ACE han usado un amplio rango de algoritmos para los procesos de aprendizaje en los agentes. Entre los principales algoritmos se encuentran: aprendizaje por refuerzo, redes neuronales, algoritmos genéticos, programación genética y otros algoritmos de computación evolutiva que permiten capturar aspectos de aprendizaje inductivo.
- Evolución de normas de comportamiento: se trata de estudiar la evolución de la cooperación, a pesar que el engaño produce ganancias inmediatas. La idea es estudiar qué papel juegan la reputación, la confianza, la reciprocidad, la venganza, el rencor y el castigo en el desarrollo de los juegos.
- Formación de redes económicas: estudiar la manera en que las interacciones en las redes económicas están determinadas para escoger de forma deliberada los socios. Una de las preocupaciones claves en esta tarea es la aparición de redes de comercio de compradores y vendedores que determinan sus socios de forma adaptativa, sobre la base de experiencias pasadas.
- Modelamiento y organizaciones: estudia la manera de determinar la forma óptima de una organización para alcanzar unas metas específicas.
- Laboratorio computacional: un laboratorio computacional es un entorno de trabajo preconstruido, con una interfaz gráfica que permite realizar procesos de experimentación en un dominio específico permitiendo estudiar sistemas de múltiples agentes, interactuando por medio

de experimentos computacionales controlados y replicables. Entre los temas de investigación para la construcción de laboratorios computacionales se encuentra la construcción de laboratorios computacionales para cada tipo de aplicación, construcción de plataformas computacionales multifacéticas, mecanismos para realizar validaciones de resultados con datos obtenidos por otras fuentes.

Un aspecto importante que se debe entrar a considerar en los laboratorios computacionales es conocer con qué grado de flexibilidad pueden ser especificados esquemas de aprendizaje para los agentes; a pesar que muchos estudios tienden a realizar sus algoritmos de aprendizaje en forma de simples ecuaciones de actualización con parámetros fijos, la evidencia acumulada indica que esos algoritmos no funcionan bien en todas las situaciones. Un mejor camino para proceder es permitir que los agentes en los laboratorios computaciones aprendan a aprender.

A partir de lo anterior el presente artículo de investigación plantea el desarrollo de un laboratorio computacional en el ámbito de la economía computacional basado en agentes y en el cual se puedan integrar esquemas de aprendizaje sin requerir habilidades avanzadas de programación.

El artículo está estructurado de la siguiente manera. En la sección II se presenta la especificación conceptual de los laboratorios computacionales así como también la descripción del lenguaje de dominio UNALCOL, en la sección III se desarrolla una mención y determinación de los juegos no cooperativos, en la sección IV se despliegan los escenarios de prueba y por último, en la sección V, se establecen algunas conclusiones.

2. Laboratorio computacional

Con el fin de desarrollar un laboratorio computacional para la especificación de simulaciones en el ámbito de la economía computacional basada en agentes, se tendrán en cuenta las ideas introducidas por Dopfer *et al* [5], las cuales plantean el desarrollo de un entorno de trabajo analítico en el ámbito de la economía evolutiva con una arquitectura micro, meso y macro. La idea central es definir un sistema económico en términos de reglas, en el cual el nivel meso es entendido como el conjunto de reglas que gobiernan los sistemas, el nivel micro se refiere a los individuos portadores de reglas (agentes), sus operaciones locales y al sistema que ellos organizan, por último el nivel macro estudia el orden o la coordinación entre las poblaciones de reglas o unidades meso.

A partir de lo anterior se propone una manera de pensar diferente sobre las cuestiones fundamentales de la coordinación, el cambio de la economía y se adapta la perspectiva sobre la evolución económica vista como un escenario de crecimiento de procesos de conocimiento. Como consecuencia directa se indica que un sistema económico puede ser examinado como una estructura compleja de reglas que evolucionan sobre un largo periodo de tiempo.

En las aproximaciones tradicionales las reglas son preconfiguradas en el agente, significa que el conjunto de acciones de los agentes no cambiarán durante el proceso de simulación, sin embargo, se han venido introduciendo mecanismos de aprendizaje utilizando técnicas de aprendizaje reforzado, redes neuronales, programación evolutiva (algoritmos genéticos, programación genética, programación genética gramatical) entre otras, con el fin de cambiar el conjunto

de reglas a medida que avanza el proceso de simulación.

El desarrollo de simulaciones en el ámbito de la economía computacional basada en agentes requiere comprender o construir una plataforma de modelamiento con el fin de configurar los diferentes sistemas que se quieren simular. Actualmente existe una variedad de plataformas (StartLogo, NetLogo, MASON, etc.), sin embargo para la implementación de modelos se deben conocer elementos sintácticos del lenguaje, complicando un poco la tarea para su ejecución. Adicionalmente, proveer a los agentes de mecanismos de adaptabilidad a partir de procesos de aprendizaje no se encuentra en el núcleo central de dichas plataformas y, si lo están, requieren de cierta experticia en programación de computadores y de técnicas de aprendizaje particulares que se deseen aplicar, habilidades de las cuales los modeladores carecen [6].

Con el fin de dar una solución a lo anterior varios trabajos de investigación se han planteado, entre los cuales se encuentra el desarrollado por Okuyama *et al*, donde se desenvuelve el entorno de trabajo MAS-SOC (Multi-Agent Simulations for the SOCial Sciences) con el fin de permitir la creación de simulaciones sociales basadas en agentes, este trabajo va dirigido a personas que no tengan mucha experiencia en programación. Fue introducido también el lenguaje ELMS (Enviroment Description Language for Multiagent Simulation) para la especificación de sistemas multiagentes, dicho lenguaje permite la puntualización de agentes, reglas de comportamiento, percepciones, acciones, ambiente, entre otros [7].

A pesar que es un lenguaje más simple que los provistos por las plataformas mencionadas anteriormente, tiene elementos que pueden dificultar la especificación de modelos puesto que no existe un entorno de desarrollo integrado que facilite las tareas de programación, lo que en efecto no permite la integración de librerías de aprendizaje externas.

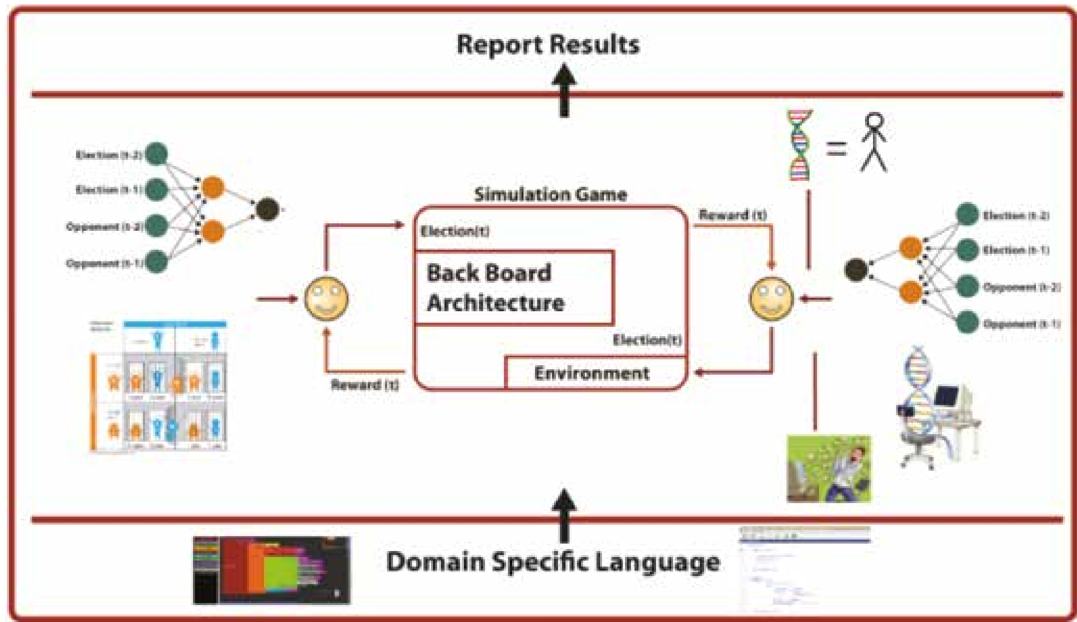
Buscando un esquema unificado para la especificación de simulaciones económicas, se ha planteado el desarrollo de un entorno de simulación bajo la propuesta de agentes inteligentes planteado por Russell *et al* [8], para lo cual es necesario definir la arquitectura y el programa de cada uno de los agentes involucrados en un proceso de simulación. La arquitectura debe permitir definir los sensores, actuadores, entorno y la medida de rendimiento. Por su parte el programa debe implementar la función del agente, es decir, especificar las acciones a ejecutar basado en las percepciones, el estado interno, la función de utilidad esperada en cada juego y los mecanismos de aprendizaje.

En el caso de la economía computacional basada en agentes, cada agente (arquitectura micro) participante debe tener la capacidad de percibir las decisiones del contrincante a través de las reglas de decisión utilizadas en cada una de las iteraciones, la recompensa entregada por el entorno en función de la tabla de recompensas va a alterar de esta forma el comportamiento (arquitectura macro).

Dicha información debe ser almacenada temporalmente en un estado interno con capacidad limitada que posee cada agente, con el fin de que el módulo de aprendizaje pueda evaluar el resultado obtenido y basado en criterios de calidad y experiencia, es posible modificar los elementos de actuación definidos, es decir, alterar las reglas de tal manera que se pueda mejorar la selección de acciones buscando optimizar las medidas de rendimiento.

El esquema conceptual planteado para el laboratorio computacional es presentado en la figura 1.

Figura 1. Esquema conceptual plataforma de simulación



Fuente: elaboración propia

Como se puede observar el laboratorio computacional tiene los siguientes componentes:

1. Lenguaje específico de dominio para la especificación de los modelos de simulación.
2. Plataforma de simulación basada en agentes que soporten los planteamientos especificados por Russell *et al*. Como se puede observar, los agentes envían las decisiones al ambiente en cada una de las iteraciones del juego, las decisiones pueden ser tomadas teniendo en cuenta los resultados producidos por estrategias de aprendizaje que usan como fuente de información el historial de percepciones de cada uno de los agentes.
   El ambiente en la plataforma de simulación recibe las acciones de cada uno de los agentes y a través de las reglas configuradas realiza cada uno de los pagos a los respectivos agentes.
3. Reporte de resultados de las simulaciones especificadas.

A continuación se presenta cada uno de los elementos especificados anteriormente para el laboratorio de simulación que se denominó UNALCOL.

2.1. Lenguaje específico de dominio

Con el fin de poder realizar la especificación de las simulaciones se planteó un lenguaje específico de dominio textual. Para su construcción es necesario comprender los

elementos conceptuales de desarrollo de un compilador.

- Un compilador es un programa que lee un programa escrito en un lenguaje -lenguaje fuente- y lo traduce a un programa equivalente a otro lenguaje -lenguaje objeto-.
- Un compilador se compone internamente de varias etapas, o fases, que realizan distintas operaciones lógicas, entre ellas podemos encontrar el analizador léxico, analizador sintáctico, analizador semántico, generador de código, entre otros. [9]

Entretanto, el analizador léxico tiene la tarea de leer el programa fuente como un archivo de caracteres y dividirlo en tokens; un token es una secuencia de caracteres que representan una unidad de información en el lenguaje fuente, por ejemplo las palabras reservadas, identificadores, símbolos especiales entre otros. Para cumplir con su tarea el analizador léxico debe implementar métodos de especificación y reconocimiento de patrones. Estos métodos son principalmente los de expresiones regulares y los autómatas finitos.

Para el caso del laboratorio computacional se definieron expresiones regulares para las siguientes categorías léxicas:

- Palabras reservadas de alto nivel.
- Palabras reservadas que identifican los tipos de datos
- Símbolos del lenguaje.

Dado lo anterior, se puede proceder a identificar los elementos que van a conformar el analizado léxico. En la figura 2 se presentan los token identificados.

Figura 2. Expresiones regulares lenguaje

```
TOKEN : /* KERWORDS */
{
  < TYPE: "type" > | < AGENT: "agent" >  | < ATTRIBUTE : "attribute" > | < PERCEPT:
"percept" >| < ACTION : "action" >| < PROGRAM: "program" >| < MAIN: "main" > | < END:
"end" > | < AS: "as" > | <ENV:"enviroment" > | < CONFIGURATION: "configuration" > | <
MAXITERATION: "max" > | < SINCRONIZED: "sincronized" > | < TRUE  : "true" > | < FALSE :
"false" > | <NAME: "name" > | < FUNCTION: "function" >| < IMPORT  : "import" >
}

TOKEN : /* DATATYPE */
{
  <STRING: "string"> | <DOUBLE:"double"> | <INTEGER: "integer">| < BOOLEAN:"boolean"> | <
PERCEPTIONS : "perception">
}


TOKEN : /* STRUCTURES */
{
  < IF : "if"> | < THEN : "then"> | < ELSE: "else"> | < TO : "to"> | <DIM : "dim">
 |<AND : "and"> | < OR : "or"> | < NOT : "not"> | < IN : "in"> | < NULL : "null">
}

TOKEN : /* SIMBOL */
{
  < DP: ":"> | < CO: ","> | < PT: "."> | < EQ: "="> | < PA: "("> | < PC: ")">
| < AT: "@">| < MINUS: "-"> | < PLUS: "+"> | < MULTIPLY: "*"> | < DIV: "/">
| < LA: "["> | < LC: "]"> | < AD: "!"> | < MY: ">"> | < MN: "<">
}

TOKEN :{
  < ID: < LETRA > (< LETRA > | <DIGIT>)* > | < #LETRA: [ "a"-"z" ] >
}
```

Fuente: elaboración propia.

Posterior a la definición de las expresiones regulares, se procedió a dilucidar la gramática de libre contexto. En este sentido el analizador sintáctico va a determinar las reglas gramaticales de una gramática de libre contexto (especificación para la estructura sintáctica de un lenguaje de programación). Las reglas gramaticales determinan las cadenas legales de símbolos de tokens.

Para el caso del laboratorio computacional se construyó una especificación gramatical LL(1) soportada por el metacompilador JAVACC. Los componentes de dicha gramática son:

- Símbolo inicial de la gramática: UNALCOL.
- Símbolos no terminales: *Unalcol*, *Agents*, *Estructure*, *AsignationAttributes*, *Condition*, etc.
- Reglas de producción: cada símbolo no terminal está asociado con una regla de producción, a continuación se exponen algunas reglas de producción particulares. Es importante anotar que en Javacc, los elementos entre <> corresponden a símbolos terminales, y los restantes corresponden a símbolos no terminales.

En primera instancia un modelo de simulación en UNALCOL debe tener configurados los agentes, el ambiente y los parámetros de configuración para ser utilizados en la ejecución de la simulación.

Para la definición del cuerpo del agente se tienen en cuenta la regla de producción agents, la cual está conformada por los siguientes componentes.

- Integración con librerías externas para el proceso de toma de decisiones, en este caso se debe especificar la definición de una variable en la cual se indique la clase ejecutable. En la gramática el componente léxico inicial es IMPORT.
- Definiciones de atributos del agente: representa las propiedades internas que definen el agente. Dichos atributos pueden ser observados por el ambiente y otros agentes. En la gramática el componente léxico que indica el inicio de la definición de atributos es ATTRIBUTE.
- Definición de percepciones del agente: son las características observables por parte del agente. En la gramática el componente léxico que indica el inicio de la definición de percepciones es PERCEPT.
- Definición de acciones: son las posibles decisiones que puede tomar el agente en el ambiente. En la gramática el componente léxico que indica el inicio de la definición de percepciones es ACTION.
- Definición del programa del agente: se definen las reglas de decisión basados en los atributos y las percepciones obtenidas a través del ambiente. Es importante indicar que en las reglas de decisión se puede hacer uso de las variables definidas en los IMPORT para enlazarse con librerías externas.

Como se observa en la figura 3 las reglas de decisión deben ser especificadas en términos de reglas IF-THEN-ELSE.

Figura 3. Regla de producción para la definición del agente

```
Unalcol →
     ( Agents ) *  EnviromentStructure   Main
 Agents →
 <AGENT><ID>
   (
    <IMPORT><DP>   (   <ID > <AS > <ID> (<PT><ID>)*   )+  )?
    <ATTRIBUTE><DP>
    (
      <ID> ( <PA>   <CONSTANT>   <PC>   )?
     <AS> ( <STRING> |  <DOUBLE> | <INTEGER > | <BOOLEAN > | < PERCEPTIONS > |   <AGENT>  |  <ID>  )
    )+
   <PERCEPT><DP>     (      <ID>   <IN>   (      <ENV >     | < AGENT >  ) )+
    <ACTION><DP>   <ID>    (    <CO>         <ID>     )*
    <PROGRAM> <ID >   <DP>   ( Estructure  |  AsignationsAttributes  )+
<END><AGENT>

Estructure →
 ( <AT >    (    <PROGRAM >   | < ACTION >    | < PERCEPT >    ) )?
 <IF> Condition  <THEN>    (  ( AsignationsAttributes  )+   |    Estructure   )
 (
   <ELSE> ((  AsignationsAttributes) +    |    Estructure    )   )?
  <END> <IF>

Condition →
   Asignations   ( (<AND >|<OR >|<NOT>)   Asignations   )*
```

Fuente: elaboración propia.

Dentro de la definición de reglas se debe aclarar el tipo de regla, entre las cuales se encuentran: @PROGRAM, @ACTION, @PERCEPT.

- Si se especifica el metadato @PROGRAM en la definición de la regla, se indica que dicha regla corresponde al programa que define la acción que debe llevar a cabo el agente.
- En caso que se especifique el metadato @ ACTION, se indica al agente cuáles son los cambios internos que se deben ejecutar producto de la decisión tomada. Los cambios se ven reflejados en los diferentes atributos que definen al agente.
- Por último en caso que se especifique @ PERCEPT, se indican al agente las reglas a aplicar luego de obtener, a través de los sensores, las características observables entregadas por el ambiente.

Otro de los elementos que deben ser definidos para la realización de una simulación en el lenguaje UNALCOL, es el ambiente, el cual queda completamente definido a través de los siguientes componentes (figura 4).

- Atributos del ambiente.
- Programa del agente: tal como se indicó anteriormente el ambiente recibe las decisiones tomadas por cada uno de los agentes en cada periodo de simulación. A partir de esto se da a cada uno de ellos una recompensa.

Figura 4. Regla de producción para la definición del ambiente

```
EnviromentStructure →
  <ENV>
  <ATTRIBUTE> <DP>
    (   <ID>  (  <PA>   <PC>  )?  <AS>
        ( <STRING> |   <DOUBLE> | <INTEGER > | <BOOLEAN > | < PERCEPTIONS > | <AGENT> | <ID>  )
    )+
  <PROGRAM> <ID >  <DP>    ( Estructure    |     Asignations    )+
 <END> <ENV>
```

Fuente: elaboración propia.

Por último se deben especificar las configuraciones de la simulación que serán tenidas en cuenta en el proceso de ejecución del modelo. Entre los elementos a especificar está:

- El nombre del modelo.
- El número de iteraciones a ejecutar por el modelo de simulación.
- Especificación si se requiere que el modelo se ejecute en modo síncrono o asíncrono. Si se denota que el modelo sea ejecutado de modo síncrono se indica que el ambiente debe esperar a que cada uno de los agentes envíen las decisiones. Luego de recibirlas, se debe proceder a realizar los pagos a cada uno de ellos con base en las reglas de decisión especificadas en el programa.

Por último se deben declarar cada uno de los agentes y realizar la inicialización de los respectivos atributos (figura 5).

Figura 5. Regla de producción para la inicialización de los parámetros de la simulación

```
Main →
  <MAIN>  ( Definitions )*  ( Iniciations  )*
  (
   <CONFIGURATION><PT ><NAME ><EQ > <ID >
   <CONFIGURATION><PT ><MAXITERATION ><EQ > <CONSTANT >
   <CONFIGURATION><PT ><SINCRONIZED ><EQ > (<TRUE>          | <FALSE >)
 )?
 <END> <MAIN >

Definitions →  <DIM> <ID>   <AS> <ID >

Iniciations →
  ( <ID>  |  <ENV>  ) <PT>  <ID>
   <EQ>
  (   (  <ID >  |    <CONSTANT >  )
     |  <LA>   <ID >    (   <CO>   <ID > )*  <LC>
  )
```

Fuente: elaboración propia.

Como parte del lenguaje específico de dominio se procedió a construir un entorno de desarrollo Integrado (IDE) con el fin de facilitar las tareas de programación involucradas. El editor construido tiene soporte para los siguientes elementos.

- Coloreado de sintaxis: los token definidos en el analizador léxico cuando aparecen en el programa fuente se presentan en diferentes colores.
- Numeración de líneas: el programador tiene a disposición el número de línea a medida que el programa se especifica, dicha información es importante cuando se produce un error léxico o sintáctico.
- Autocompletación - Ayuda: si un programador no conoce exactamente la palabra reservada que debe utilizar, el editor le presenta una lista de opciones cuando es presionada la combinación de teclas CRT + SPACE.
- Plantillas: Con el fin de agilizar el proceso de programación se definieron plantillas de código comúnmente utilizadas: Entre estas se pueden encontrar las plantilla para la definición del agente (***ag*** + CTR + SHIF - SPACE), reglas (***ifr*** + CTR + SHIF - SPACE), ambiente (env + CTR + SHIF - SPACE) y main (***mai*** + CTR + SHIT - SPACE).

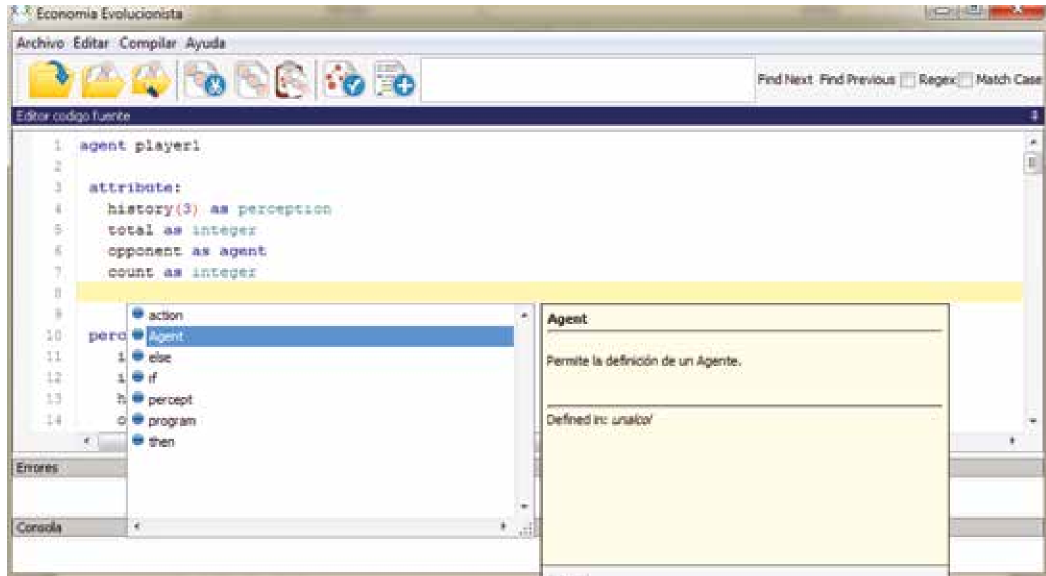
Con el fin de cumplir con los requerimientos se utilizaron las siguientes herramientas.

- RsintaxTextArea [10]: adiciona la funcionalidad de resaltado de sintaxis, numeración de línea y autocompletación a un componente jTextArea del lenguaje JAVA.
- AutoComplete: permite definir las palabras y las plantillas que pueden ser completadas por el editor, usando la combinación de teclas definidas.
- TokenMakerMaker: permite personalizar el analizador léxico para el RsintaxTextArea.

En la figura 6 se presenta una imagen del editor construido para el lenguaje UNALCOL. Entre las características adicionales a mencionar se encuentra.

- Tiene las características de un editor de texto tradicional: abrir archivos con extensión ***.alife***, guardar, guardar como, compilar, generar código, seleccionar, copiar, cortar.
- Tiene soporte para realizar búsquedas sobre el código fuente a través de palabras clave, expresiones regulares.

Figura 5. Editor de código fuente UNALCOL



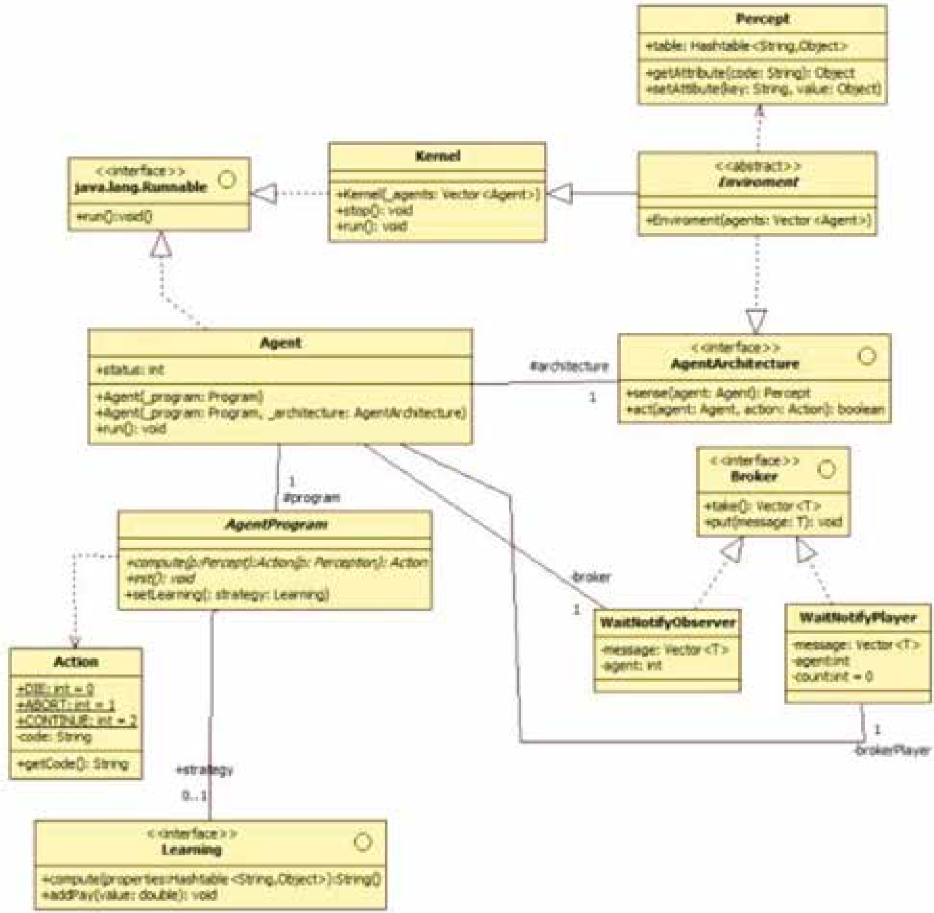
Fuente: elaboración propia

2.2. Plataforma de simulación

Con el fin de permitir procesos de simulación se tuvo en consideración los planteamientos especificados por Russell *et al*. En la figura 6 se presentan las interfaces y clases abstractas principales que permiten la especificación de un modelo de simulación.

- *Agent*: clase que implementa Runnable que permite la especificación de agentes y controla el ciclo de vida del agente. Tiene la función de observar el ambiente, obtener percepciones e invocar al programa para la obtención de las respectivas acciones a ejecutar.
- *AgentProgram*: interfaz que define el esquema de especificación de programas de agente. Dicho programa queda completamente especificado con la definición del método *compute*, el cual recibe las percepciones y retorna la acción a ejecutar.
- *AgentArchitecture*: interfaz que permite especificar los mecanismos para los sensores y actuadores del agente.
- *Broker*: interfaz que define los métodos necesarios para gestionar la concurrencia. Específicamente toma en cuenta los métodos *take* y *put*; con el método *take* los agentes pueden tomar los pagos del realizador por el ambiente cuando el modelo de simulación es síncrono, por su parte con el método *put*, los agentes ponen las decisiones a disposición del ambiente.
- *Learning*: clase abstracta que define los elementos para la integración de estrategias de aprendizaje basadas en librerías externas. Dicha interfaz aduce que se deben sobreescribir los métodos *compute* y *addPay*. El método compute recibe las percepciones y retorna a la acción, por su parte el método *addPay* es utilizado cuando se debe dar una calificación a la acción ejecutada.

Figura 6. Clases principales simulador UNALCOL



Fuente: elaboración propia.

Con el fin de poner un escenario de prueba que permita plantear un modelo de simulación en UNALCOL, se plantea el desarrollo de un modelo para la especificación de juegos repetitivos no cooperativos, modelos que entran dentro de la línea de investigación de normas de comportamiento, especificado anteriormente.

3. Juegos no cooperativos

En un juego estratégico normal los jugadores escogen estrategias puras de forma simultánea e independiente y reciben un pago que depende del perfil de y maniobras utilizadas.

En el presente artículo se estudiará un juego clásico de la teoría de juegos denominado juego de suma no nula: el dilema del prisionero, formulado por el matemático Tucker en base a las ideas de Flood y Dresher en 1950 [11].

En el dilema del prisionero existen cuatro posibles resultados: ambos jugadores deciden cooperar (reward), ambos jugadores de-

ciden no cooperar (punishment), el jugador A decide cooperar mientras que el jugador B no coopera (sucker), y el jugador A decide no cooperar mientras que el jugador B decide cooperar (temptation).

Para el caso del dilema del prisionero el juego constituye un dilema si se cumplen las siguientes desigualdades. Ecuación (1):

$$T > R > P > S \text{ y } 2R > S + T \quad (1)$$

Los valores típicos de la matriz de pagos son presentados en la tabla 1

Tabla 1. Matriz de pagos dilema del prisionero

	Jugador B		
Jugador A		Cooperar	No cooperar
	Cooperar	R=3,R=3	S=0,T=5
	No cooperar	T=5,S=0	P=1,P=1

Fuente: elaboración propia.

Si existe una única oportunidad de jugar y basados en supuestos de racionalidad, la mejor decisión es siempre no-cooperar, por lo tanto esta opción es denominada como una estrategia dominante en el juego (equilibrio de Nash).

Muchas variaciones de los juegos cooperativos han sido propuestas; una de ellas es el dilema del prisionero iterado (IPD) presentada por Axelrod en 1984, en el cual dos contrincantes juegan repetidamente al dilema del prisionero.

La clave de los juegos repetitivos es que ambos jugadores pueden jugar nuevamente, lo cual posibilita desarrollar estrategias de juego basadas en interacciones de juegos previos con el fin de maximizar los pagos recibidos. Teóricamente el movimiento de un jugador puede influir en el comportamiento de su oponente en el futuro.

A continuación se procede a especificar dos escenarios:

- La competencia entre dos agentes cuyas estrategias de decisión son ALLC (siempre cooperar) y TFT (Tit For Tat).
- La competencia entre un jugador del dilema de prisionero que usa TFT y un jugador que utiliza como esquema de decisión una red neuronal.

4. Escenarios de prueba

4.1. Competencia entre agentes con estrategias estándar

En la figura 7 se presenta la especificación de los agentes Player1, Player2, en el cual se definen los componentes de los agentes: atributos, percepciones, acciones y programa.

En la Fig. 7 se presenta el jugador 1 (Player1), el cual tiene configurado como estrategia de decisión la cooperación. El jugador 2 por su parte, tiene como programa la estrategia TFT, es decir que el agente toma como decisión la acción que fue ejecutada por el contrincante en la iteración anterior (history(iteration-1).opponent.action).

Dado que *history* es una variable de tipo perception, almacena la información de las decisiones y recompensas de cada uno de los agentes en el modelo de simulación de acuerdo al tamaño especificado. Para el caso particular del ejemplo se están realizando los filtros de selección de la percepción de la ite-

ración anterior (history(iteration-1.), específicamente la acción ejecutada por el oponente en la iteración anterior (history(iteration-1).opponent.action).

Figura 7. Programa para los agentes

```
agent player1
 attribute:
   history (1) as perception
   total as integer
   opponent as agent

 percept:
   id in agent
   iteration in enviroment
   history in agent
   opponent in agent
   reward in agent

 action:
   CO, DEF, EXIT

   program COOPERATE:
   @Program
    if iteration = 0 then
       action=CO
    else
       action=CO
    end if

   @Action
    if action=CO then
      color=yellow
    else
      color=blue
    end if

   @Percept
    if reward !=null then
       total =  total + reward
    end if
end agent
```

```
agent player2
 attribute:
   history(3) as perception
   total as integer
   opponent as agent

 percept:
   id in agent
   iteration in enviroment
   history in agent
   opponent in agent
   reward in agent

 action:
   CO, DEF, EXIT

   program TFT:
   @Program
    if iteration =0 then
       action=CO
    else
      if history(iteration-1).opponent.action=CO  then
            action=CO
      else
            action=DEF
      end if
    end if

   @Action
    if action=CO then
        cx=50
        cy=10
        color=yellow
    else
      cx=10
      cy=10
      color=blue
    end if

   @Percept
    if reward !=null then
       total =  total + reward
    end if
end agent
```

Fuente: elaboración propia.

Para el caso del ambiente se especifica la matriz de pagos usando reglas de decisión. Por ejemplo, si el jugador 0 (Player1) seleccionó en la iteración anterior la acción cooperar (history(iteration).player(0).action=CO) y el jugador 1 (player2) seleccionó cooperación, se debe dar como pago de acuerdo a la matriz del dilema del prisionero tres unidades a cada uno de los jugadores (player(0).reward =3, player(1).reward=3) y así sucesivamente. Lo anterior es presentado en la figura 8.

Figura 8. Programa para la especificación del ambiente

```
enviroment
 attribute:
   player() as agent


 program Enviroments:
  @Program
  if history(iteration).player(0).action=CO AND  history(iteration).player(1).action=CO  then
     player(0).reward =3
     player(1).reward=3

  else
     if history(iteration).player(0).action=CO AND  history(iteration).player(1).action=DEF  then
          player(0).reward=0
          player(1).reward=5
     else
          if history(iteration).player(0).action=DEF AND  history(iteration).player(1).action=CO then
             player(0).reward=5
             player(1).reward=0
          else
            if history(iteration).player(0).action=DEF AND  history(iteration).player(1).action=DEF then
                player(0).reward=1
                player(1).reward=1
            end if
          end if
     end if
  end if

end enviroment
```

Fuente: elaboración propia.

Por último se especifican los parámetros de configuración como se presentan en la figura 9. Se declaran dos agentes, uno por cada tipo de agentes declarados, se realiza la inicialización de los atributos para cada uno de ellos, se define el nombre del modelo como dilema con un total de 20 iteraciones y se especifica que se debe realizar un modelo sincronizado.

Como resultado de la ejecución se obtiene como resultado la cooperación mutua, representada en un puntaje de 60 para cada uno de los jugadores, como se presenta en la figura 10.

Figura 9. Parámetros configuración de la simulación

```
main
 /*
  Declaracion de los agentes
 */
 dim fabian as player1
 dim isabel as player2

 fabian.total=0
 fabian.opponent=isabel
 fabian.id=fabian
 fabian.cx=10
 fabian.cy=20
 fabian.color=blue

 isabel.total=0
 isabel.opponent=fabian
 isabel.id=isabel
 isabel.cx=30
 isabel.cy=20
 isabel.color=blue

 enviroment.player=[fabian,isabel]

 configuration.name=dilema
 configuration.max=20
 configuration.sincronized=true

end main
```

Fuente: elaboración propia

4.2. Competencia entre agentes con estrategias estándar y red neuronal

Con el fin de validar la integración con librerías externas en la figura 10 se presenta un ejemplo.

Figura 10. Parámetros configuración de la simulación

Fuente: elaboración propia.

En este caso el Player2 usará la clase unalcol.agents.nn.DilemaNn, para que sea descubierta como se indicó anteriormente se debe implementar la interfaz Learning y estar presente en el CLASSPATH en el momento de la ejecución del modelo.

En la figura 11 se presenta el programa de la clase utilizada para el proceso de toma de decisiones. Como se puede observar la clase DilemmaNn extiende la clase abstracta *Learning*, por lo tanto se implementa el método abstracto *compute*.

El método compute recibe la información del historial de percepciones del ambiente y como resultado obtiene la acción a ejecutar. En este caso particular se tiene una perceptrón multicapa que usa como proceso de entrenamiento ResilenPropagation. Al tener la red neuronal entrenada se puede proceder a realizar el proceso de toma de decisiones usando la red neuronal. En este caso se toman las acciones ejecutadas por los agentes en la iteración anterior, con dicha información se puede proceder a calcular la decisión enviada por la red neuronal y transmitida al programa del agente.

Los resultados obtenidos indican cooperación mutual dado que ambos jugadores deciden cooperar cuando se coopera en la iteración inmediatamente anterior.

Figura 11. Programa enlaza con librería externa

Fuente: elaboración propia.

Figura 12. Red neuronal (1)

```
package unalcol.agents.nn;

import java.util.Hashtable;
public class DilemmaNn extends Learning{

    private  BasicNetwork network;
    private  double INPUT[][] = { { 0.0, 0.0 }, { 1.0, 0.0 },
                    { 0.0, 1.0 }, { 1.0, 1.0} };

    private double IDEAL[][] = { { 0.0 }, { 0.0 }, { 1.0 }, { 1.0 } };

    public DilemmaNn(){
         // create a neural network, without using a factory
        network = new BasicNetwork();
        network.addLayer(new BasicLayer(null,true,2));
        network.addLayer(new BasicLayer(new ActivationSigmoid(),true,3));
        network.addLayer(new BasicLayer(new ActivationSigmoid(),false,1));
        network.getStructure().finalizeStructure();
        network.reset();

        MLDataSet trainingSet = new BasicMLDataSet(INPUT, IDEAL);
        final ResilientPropagation train = new ResilientPropagation(network, trainingSet);
        int epoch = 1;
        do {
                train.iteration();
                System.out
                                .println("Epoch #" + epoch + " Error:" + train.getError());
                epoch++;
        } while(train.getError() > 0.01);

        double weight[]=this.network.getFlat().getWeights();
        for(int i=0;i<weight.length;i++)
            System.out.print(weight[i] + ",");

        System.out.println();
    }
```

Fuente: elaboración propia.

Figura 13. Red neuronal (2)

```
@Override
public Object compute(Hashtable<String, Object> properties) {

    String  id = (String) properties.get("ID");
    Integer iteration =(Integer) properties.get("ITERATION");
    java.util.Vector history= (java.util.Vector) properties.get("HISTORY");
    String  opponent = (String) properties.get("OPPONENT");

    if(iteration>=1){
        int move=-1;
        int moveOpponent=-1;

        if(((String)searchPerception(history,id,iteration- 1,"ACTION")).equals("CO"))
            move=1;
        else
            move=0;

        if(((String)searchPerception(history,opponent,iteration- 1,"ACTION")).equals("CO"))
            moveOpponent=1;
        else
            moveOpponent=0;

        double XOR_TEST[][]=new double [1][2];
        XOR_TEST[0][0]=move;
        XOR_TEST[0][1]=moveOpponent;

        BasicMLDataSet dataSet = new BasicMLDataSet(XOR_TEST, null);
        final MLData output =network.compute(dataSet.getData().get(0).getInput());

        if(output.getData(0)==1.0){
            return "CO";
        }else{
            return "CO";
        }

    }else{
        return "DEF";
    }
```

Fuente: elaboración propia.

5. Conclusiones

Luego del diseño e implementación de un laboratorio computacional para la especificación de modelos de simulación en el ámbito de la economía computacional basada en agentes, se pueden indicar las siguientes conclusiones.

- Los laboratorios computacionales, tal como se evidenció en el proceso de desarrollo de los juegos repetitivos, facilitan la implementación de este dispositivo para personas con poca experticia en programación.
- La integración de librerías externas para el soporte a la toma de decisiones permite a los agentes adaptar las estrategias de decisión con el fin de maximizar los beneficios. Lo anterior puede ser evidenciado en la utilización de las redes neuronales para el caso de los juegos no cooperativos.

6. Referencias

[1] H. Scholl, "Agent based and system dynamics modeling: A call for cross study and joint research", Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-

Volume 3, IEEE Computer Society, pp. 3003, 2001.

[2] M. Fontana, "Can neoclassical economics handle complexity? the fallacy of the oil spot dynamic", Journal of Economic Behavior Organization 76 , no. 3, 584-596, 2010.

[3] L. Tesfatsion, "Agent-based computational economics: A constructive approach to economic theory", Handbook of Computational Economics, vol. 2, Elsevier, pp. 831-880, 2006.

[4] L. Tesfatsion, "Agent-based computational economics: Modeling economies as complex adaptive systems", Staf general research papers, Iowa State University, Department of Economics, 2008

[5] K. Dopfer, J. Foster, J. Potts, "Micro-meso-macro", Journal of Evolutionary Economics 14 (2004), no. 3, 263-279.

[6] F Giraldo, J Gomez. "Aprendizaje de estrategias de decisión en juegos repetitivos no cooperativos", Revista tecnura Universidad Distrital, Bogotá Colombia, 2013

[7] F Okayuma, R Bordini, A Rocha, "ELMS: An enviroment Description Languaje for MultiAgent Simulation", First International Workshop, E4MAS 2004, New York, NY, July 19, 2004.

[8] Russell, S. J., & Norvig, P. (2003). Artificial intelligence: A modern approach. Upper Saddle River, N.J: Prentice Hall/ Pearson Education.

[9] Aho, A. V.; Sethi, R. y Ullman, J. D. "Compiladores Principios, Técnicas y Herramientas. Addison-Wesley Iberoamericana S.A.1990

[10] Fifesoft 2013, RsintaxTestArea, http://fifesoft.com/rsyntaxtextarea/

[11] R. Chiong, "Applying genetic algorithms to economy market using iterated prisoner's dilemma." In Proceedings of the 22nd Annual ACM Symposium on Applied Computing (pp. 733-737). Seoul, Korea: ACM Press, 2007.