AVANCES
EN CIENCIAS E INGENIERÍAS
ARTÍCULO/ARTICLE SECCIÓN/SECTION C

# Una nueva infraestructura para el desarrollo de agentes autónomos en RoboCup

**Enrique V. Carrera**[1*]

[1]*Colegio de Ciencias e Ingeniería - El Politécnico, Universidad San Francisco de Quito*
*Diego de Robles y Vía Interoceánica, Quito, Ecuador*
[*]*Autor principal/Corresponding author, e-mail: vcarrera@usfq.edu.ec*

Editado por/Edited by: F. J. Torres, Dr.
Recibido/Received: 01/29/2010. Aceptado/Accepted: 02/25/2010.
Publicado en línea/Published on Web: 05/21/2010. Impreso/Printed: 06/01/2010.

**Abstract**

ROBOCUP is an international competition for autonomous robots intended to promote research and education in Artificial Intelligence. The competition contains several separate leagues, one of which is called *Simulator League* and deals with purely simulated players. A team of players for this league can be written in any programming language. However, a major problem when constructing ROBOCUP players is to implement the basic functionality of each player, since there are several problems related to information retrieval, processing, and management, as well as the development of basic skills for the soccer players. Thus, this paper proposes a new infrastructure oriented to simplify the programming of ROBOCUP teams allowing us to concentrate on the job of controlling players without worrying about the low-level tasks previously mentioned. Besides describing the proposed infrastructure, this paper evaluates our Java implementation of the infrastructure using isolated players and complete ROBOCUP teams.

**Keywords.** Autonomous agents, robot soccer, software development.

**Resumen**

ROBOCUP es una competencia internacional para robots autónomos orientada a promover la investigación y educación en el campo de la Inteligencia Artificial. La competencia incluye varias ligas separadas, una de las cuales se denomina *Liga de Simulación* y trata únicamente con jugadores de fútbol virtuales. Un equipo para esta liga puede ser escrito en cualquier lenguaje de programación. Sin embargo, uno de los mayores problemas al momento de programar los agentes autónomos es implementar la funcionalidad básica de cada jugador, pues existen varios inconvenientes asociados a la recuperación, procesamiento y administración de la información, así como también al desarrollo de las habilidades típicas de un jugador de fútbol. Siendo así, este artículo propone una nueva infraestructura orientada a facilitar la programación de equipos para ROBOCUP, permitiendo que los programadores se concentren en las estrategias de alto nivel aplicadas por los agentes antes que en las tareas de bajo nivel previamente mencionadas. Además de describir la infraestructura propuesta, este artículo evalúa una implementación en *Java* de la mencionada infraestructura usando tanto jugadores aislados como equipos completos.

**Palabras Clave.** Agentes autónomos, fútbol de robots, desarrollo de software.

## Introducción

Varios problemas estandarizados han sido la fuerza motriz básica para el desarrollo de las investigaciones en Inteligencia Artificial (IA). El más típico ejemplo de un problema estándar es el juego de ajedrez por computador [1]. Las investigaciones en este juego llevaron al descubrimiento de varios algoritmos de búsqueda poderosos. Siguiendo esa línea, ROBOCUP (*Robot World Cup Soccer Initiative*) propuso un nuevo problema estándar para la investigación en IA y Robótica. La propuesta busca utilizar el juego de fútbol como una plataforma para un amplio rango de investigaciones en IA, como por ejemplo: principios de diseño de agentes autónomos, colaboración entre múltiples agentes, adquisición de estrategias, razonamiento en tiempo real y fusión de sensores [2].

El objetivo a largo plazo de ROBOCUP es eventualmente producir equipos de robots que puedan competir con humanos en 2050. Para alcanzar esta meta, ROBOCUP mantiene una competencia anual para robots jugadores de fútbol y agentes inteligentes. La competencia incluye varias ligas, una de las cuales trata con robots enteramente simulados. Esta liga es denominada *Liga de*

ISSN 1390-5384

*Simulación* y libera a los investigadores de lidiar con problemas propios de los robots físicos como el reconocimiento de objetos, la comunicación, problemas de hardware, y otras limitaciones actuales de la robótica.

La Liga de Simulación usa un simulador de fútbol denominado *Soccer Server* [3]. Usando este simulador, un equipo de agentes escritos en cualquier lenguaje de programación que ofrezca facilidades de comunicación por *sockets* UDP/IP y manejo de cadenas de caracteres puede jugar un partido contra otro equipo. Sin embargo, uno de los mayores problemas al programar agentes para la Liga de Simulación es la implementación de la funcionalidad básica de cada jugador de fútbol. Existen varios inconvenientes relacionados a la recuperación, procesamiento y administración de la información, así como al desarrollo de las habilidades mínimas presentes en todo jugador de fútbol.

En términos generales, los programadores deben preocuparse por detalles de bajo nivel como la comunicación por *sockets* UDP, el análisis de cadenas de caracteres, la administración de la información, la ejecución de comandos, etc. Adicionalmente, habilidades básicas como encontrar el balón, mover el balón, patear en la dirección deseada, etc. deben también ser desarrolladas con la finalidad de implementar estrategias de juego de más alto nivel. De esta manera, los equipos ROBOCUP existentes son sistemas de software extremadamente sofisticados. Por ejemplo, el código C++ base del equipo *Trilearn 2003* [4], ganador de la competencia ROBOCUP 2003, tiene más de 32 mil líneas de código, mismo excluyendo las estrategias de control de alto nivel usadas por este equipo para ganar el campeonato.

Basado en lo anterior y considerando que no existen plataformas de software ampliamente utilizadas para el desarrollo de sistemas multi-agente, este artículo propone una nueva infraestructura de software para la implementación de agentes autónomos orientados a participar en ROBOCUP. La propuesta se enfoca en simplificar la programación de un equipo completo para la Liga de Simulación ofreciendo un API (*Application Programming Interface*) simple, flexible y genérico. Si bien, la primera implementación de nuestra infraestructura se basa en la programación Java, no existe nada en la propuesta que dependa del lenguaje de programación utilizado.

Para evaluar la infraestructura propuesta, se han construido varios jugadores individuales, así como equipos completos basados en estrategias de juego simples. Todos los jugadores utilizan la funcionalidad y habilidades ofrecidas por nuestra infraestructura a través del API correspondiente. De esta forma se demuestra que es posible implementar un equipo ROBOCUP completo y funcional, utilizando pocas líneas de código y sin mucha complejidad.

## Fundamentos Teóricos

Esta sección discute la motivación por detrás de las competencias de ROBOCUP, así como el simulador oficial utilizado en estas competencias. Un análisis del uso de ROBOCUP en la enseñanza de IA es también realizado.

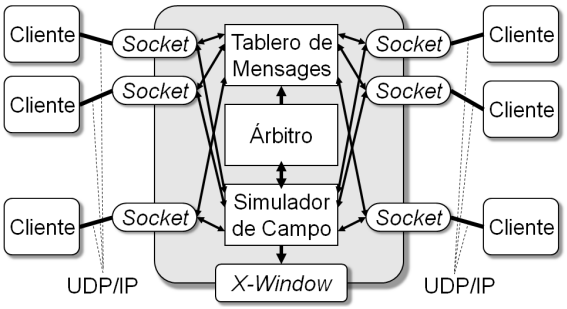


**Figura 1: Estructura general del** *Soccer Server***.**

### *RoboCup y Sistemas Multi-Agente*

ROBOCUP es un intento por fomentar la investigación en IA al proveer un problema estándar en el que una amplia gama de tecnologías pueden ser integradas y examinadas. Desde el punto de vista de los sistemas multiagente, el fútbol es un buen ejemplo de problemas con muchas propiedades del mundo real, pero todavía moderadamente abstracto. El juego del fútbol se puede caracterizar principalmente porque:

- El ambiente cambia de forma dinámica y en tiempo real.
- La robustez es más importante que la elaboración.
- La cooperación dentro de un equipo es obviamente ventajosa.
- La información sobre el estado del universo es únicamente parcial.
- La capacidad de adaptación es una propiedad deseada.
- La comunicación entre jugadores no es necesariamente precisa.

En el pasado, varios dominios han sido utilizados como ejemplo para evaluar el desempeño de los sistemas multi-agente, pero normalmente se han tratado de ambientes simples y pequeños. Un partido de fútbol, por otro lado, es un ambiente complejo y relativamente grande, y las características detalladas arriba demuestran que este juego ofrece varias formas para evaluar el comportamiento de este tipo de sistemas [3]. Adicionalmente, el fútbol es un deporte conocido a nivel mundial con una gran trayectoria a lo largo de la historia.

### *El Simulador de RoboCup*

En la Liga de Simulación, el juego entre 2 equipos virtuales de fútbol es simulado y controlado usando una arquitectura cliente-servidor. El servidor, denominado *Soccer Server*, provee la cancha de fútbol virtual y simula todos los movimientos de los jugadores y el balón. Cada cliente puede proveer la 'inteligencia' de un jugador conectándose al *Soccer Server* a través de una red

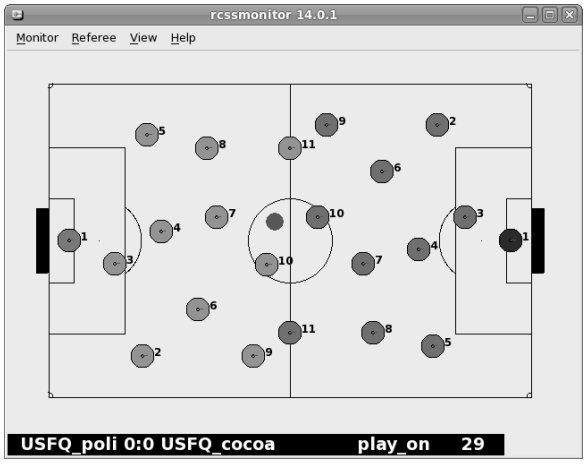


**Figura 2: Visualización de un partido de fútbol a través del** *Soccer Server Monitor***.**

de computadores (*i.e.*, usando *sockets* UDP/IP) y especificando las acciones que el jugador debe llevar a cabo. Como respuesta, el cliente recibe información tanto visual como auditiva capturada por los sensores del jugador correspondiente. Con la finalidad de que los clientes pueden ser programados en cualquier lenguaje, toda la comunicación entre el servidor y los clientes utiliza cadenas de caracteres en formato ASCII.

La figura 1 muestra una visión general de como el *Soccer Server* se comunica con sus clientes. Como se observa, el *Soccer Server* consta de los siguiente 3 módulos:

1. El simulador de campo crea el mundo virtual básico (*i.e.*, la cancha de fútbol), calcula los movimientos de los objetos, y verifica las posibles colisiones. La cancha de fútbol y todos los objetos en ella son bidimensionales, al punto que los jugadores y el balón son tratados como círculos. También existen varias marcaciones especiales en el campo que son utilizadas por los clientes para determinar la posición del jugador, su orientación, o la velocidad y dirección de su movimiento.
2. El tablero de mensajes maneja la comunicación auditiva entre el *Soccer Server* y sus clientes. Versiones últimas del *Soccer Server* permiten que cada equipo también tenga un entrenador que se comunica con sus jugadores a través de este tablero de mensajes.
3. El árbitro garantiza el cumplimiento de las reglas del juego. Este módulo advierte los goles, determina cuando el balón no está en juego, controla la posición de los jugadores en los tiros libres, tiros de esquina y saques, etc. Los juzgamientos del árbitro son anunciados a todos los clientes mediante mensajes audibles.

Como se aprecia en la figura 1, el *Soccer Server* también ofrece una interface para conectar un monitor *X-Window*. Una captura de pantalla del *Soccer Server Monitor* oficial es mostrada en la figura 2.

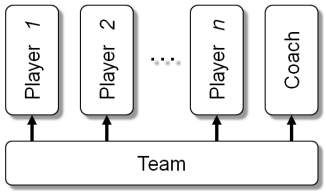


**Figura 3: Estructura lógica de un equipo de fútbol.**

Una característica importante del *Soccer Server* es la inclusión automática de ruido [5]. De forma a reflejar la naturaleza del mundo real, el servidor introduce varios tipos de incertidumbres en la simulación. De esta manera, valores aleatorios son adicionados al movimiento de los objetos, a los parámetros de los comandos, a la información de los sensores, etc. produciendo que los agentes interactúen con un mundo no-determinístico.

### *RoboCup en la Enseñanza de IA*

Un problema en la enseñanza de ciencias abstractas es encontrar ejemplos ilustrativos que presenten los conceptos respectivos. Una alternativa es PBL (*Problem Based Learning*), el planteamiento de problemas a resolver mientras el profesor cambia su papel de experto al de guía. De esta forma, los estudiantes adquieren conocimientos durante la búsqueda de soluciones a los problemas planteados.

ROBOCUP encaja perfectamente en el paradigma PBL, ya que incluye problemas fácilmente definidos con una amplia variedad de soluciones potenciales, requiriendo mayor investigación y un profundo entendimiento por parte de los estudiantes. Como ROBOCUP plantea problemas que no tienen soluciones 'correctas', los estudiantes son desafiados a mejorar las ideas presentadas en libros texto o aquellas implementadas por sus compañeros.

## Infraestructura Propuesta

La infraestructura propuesta sustenta su accionar en 3 clases base que pueden ser extendidas por los programadores conforme los requerimientos del equipo a crear. Estás 3 clases base son: `Team`, `Player` y `Coach`.

### *Team*

El componente de más alto nivel en nuestra infraestructura ROBOCUP es el equipo de fútbol representado por la clase `Team`. Esta clase inicia configurando los parámetros principales de la simulación como son el nombre del equipo, el número de jugadores, la dirección IP del servidor, etc. y posteriormente invoca a los controladores de cada uno de los jugadores. El código que controla cada jugador es una clase derivada de `Player`, explicada más adelante. Adicionalmente, la clase `Team` puede también crear una instancia de la clase `Coach` que hace las veces de entrenador.

La relación entre las diversas clases base asociadas a un equipo se describe gráficamente en la figura 3. Es importante notar, que con la finalidad de cumplir las reglas

vigentes en las competencias de ROBOCUP, cada agente ejecuta de forma completamente independiente con relación a sus compañeros de equipo (*i.e.*, no existe comunicación local entre los jugadores surgidos a partir de una misma clase `Team`). De esta manera, el comportamiento de cada agente depende exclusivamente de la información sensorial colectada desde el *Soccer Server* y de su propia estrategia de control.

***Player***

Este componente contiene el código base que controla cada uno de los jugadores que participan de un partido de fútbol. El código de `Player` puede ser extendido de forma diferente para cada jugador o tener extensiones similares para un grupo de jugadores. La estructura de esta clase no es trivial, una visión general de como está conformado cada jugador se muestra en la figura 4. Como puede observarse, `Player` fundamenta su operación en los siguientes 7 elementos:

- *Communicator*. Este elemento está a cargo de realizar la comunicación UDP entre el agente y el *Soccer Server*. Adicionalmente, debido a que el *Soccer Server* interactúa con los clientes usando una escala de tiempo discreta (*i.e.*, los agentes pueden emitir comandos cada 100 ms y reciben la información visual correspondiente cada 150 ms), este elemento marca el inicio de cada ciclo al recibir una nueva actualización de la información visual.
- *Parser*. Este elemento analiza e interpreta toda la información sensorial que viene desde el servidor y traduce los comandos del agente al lenguaje del *Soccer Server*. Un factor clave en este módulo es la noción de independencia de lado: no importa el lado de la cancha en el que juegue el agente, toda la información ofertada a los elementos superiores es relativa a su propio lado (*i.e.*, no existe arco izquierdo o derecho, sino arco propio y arco oponente, no existen banderas superiores o inferiores, sino banderas derechas e izquierdas dependiendo del lado en que juega el agente).
- *WorldState*. Los datos mantenidos por este elemento reflejan la visión que tiene el agente del mundo real. Esta visión es generalmente incompleta y con ruido, pues se sustenta en la interpretación de la información sensorial enviada por el *Soccer Server*. Con la finalidad de tener un sistema de referencia espacial simple, un sistema de coordenadas rectangulares es utilizado para posicionar todos los objetos en la cancha. El centro de la cancha corresponde al punto `(0, 0)`, y los ejes de coordenadas son relativos a la dirección hacia donde ataca el jugador. De esta forma, se tiene acceso a una visión mucho más genérica del campo de juego.
- *PlayerState*. Los datos en este elemento resumen el estado del mundo real y la situación del agente desde su propio punto de vista. Se basa no sólo en información sensorial, sino en mediciones del estado interno del robot. Este módulo permite responder cuestionamientos como el modo de visión utilizado por el jugador, la cantidad de energía disponible, la velocidad experimentada, etc.

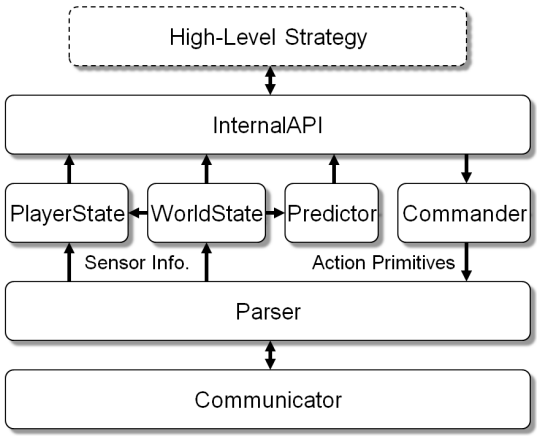


**Figura 4: Estructura interna de la clase base `Player`.**

- *Predictor*. Este elemento incluye algoritmos de regresión que ayudan a predecir la posición del balón y de los agentes vecinos. Para ello, se mantiene un histórico de las posiciones absolutas donde han sido observados esos elementos. Note que la predicción no es necesariamente exacta debido al ruido incluido en la información sensorial y a la escala de tiempo discreta usada por el servidor.
- *Commander*. Este elemento decide la mejor forma de ejecutar los comandos emitidos por la estrategia de alto nivel conforme el estado interno del agente y el estado de la cancha en ese instante. Se han implementado comandos de bajo y alto nivel. Los comandos de bajo nivel corresponden a ordenes implementadas directamente en el *Soccer Server* como `kick` y `turn`. Los comandos de alto nivel incluyen acciones como 'interceptar el balón', 'pasar el balón al compañero más cercano', y otras que dependen de cálculos más sofisticados (*e.g.*, la aceleración y dirección del agente, la potencia y dirección de la patada).
- *InternalAPI*. Este elemento resume toda la funcionalidad ofrecida por los módulos inferiores en un conjunto de llamadas que están al alcance de cualquier estrategia de alto nivel.

Dentro de la clase base, la estrategia de alto nivel no está definida y es justamente donde los programadores deben centrar su mayor esfuerzo al momento de extender esta clase base. La estrategia de alto nivel puede ser extremadamente simple como en los ejemplos mostrados en las próximas secciones, o incluir tecnologías propias de IA como redes neuronales (RNA), inteligencia de enjambres [6], etc.

***Coach***

Este componente al igual que `Player` contiene varios elementos que ayudan a implementar las estrategias de

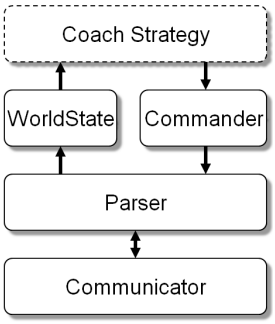


**Figura 5: Estructura interna de la clase base `Coach`.**

control y colaboración a nivel de equipo. Como puede observarse en la figura 5, esta clase es una adaptación simplificada de `Player`, ya que no se requiere un predictor de movimientos ni se tiene acceso al estado interno de los jugadores. Adicionalmente, se ha eliminado el elemento concentrador de llamadas `InternalAPI` debido a que normalmente la estrategia del entrenador accede a un subconjunto bastante reducido de los métodos y campos existentes en `WorldState` y `Commander` (*e.g.*, el entrenador sólo puede emitir comandos `say` para comunicarse con sus jugadores [5]).

## Implementación en Java

La infraestructura propuesta en la sección anterior no depende directamente de ninguna característica del lenguaje de programación utilizado. Sin embargo, debido a que Java es un lenguaje orientado a objetos, multiplataforma, completamente portable y con excelente soporte al uso de interfaces de usuario gráficas, Java fue la primera opción para la implementación de nuestra infraestructura.

Todo el desarrollo de la nueva infraestructura se basó en la utilización de *OpenJDK* 1.6.0 a través de *NetBeans* 6.7.1. Además, para la implementación del módulo `Parser` se usó *JavaCC* (versión 4.1), un generador de análisis sumamente popular en aplicaciones Java.

El mapeo entre la estructura original de la propuesta y las clases Java implementadas es prácticamente directo. Sin embargo, entre los aspectos más sobresalientes que tuvieron que definirse durante la fase de implementación podemos mencionar:

- La estrategia de control de alto nivel debe ser implementada a través del método `strategyCycle` que es invocado al final de cada ciclo de comunicación.
- La clase `InternalAPI` incluye 2 campos principales (`field` y `player`) a través de los cuales se ofrece acceso a todos los métodos públicos implementados por `WorldState`, `Predictor`, `PlayerState` y `Commander`.
- La implementación de `Predictor` únicamente reserva espacio para las últimas 5 ubicaciones en las que han sido observados los jugadores y el balón.

```
import jplayer.*;

public class myTeam extends Team {
  public myTeam(String name, String host, int port) {
    super(name, host, port, 4); //team with 4 players
  }

  public Player getPlayer(int number) {
    switch (number) {
      case 1: return new myPlayer();
      case 2:
      case 3:
      case 4: return new otherPlayer();
    }
  }

  public static void main(String[] args) {
    Team t = new myTeam(args[0], "localhost", 6000);
    t.connectAll();
  }
}
```

**Figura 6: Código Java extendiendo la clase `Team`.**

```
import jplayer.*;

public class myPlayer extends Player {
  public myPlayer() {
    super();
  }

  public void strategyCycle() {
    //this should be the high-level strategy
    FieldItem ball = field.getBall();
    if (ball.getDistance() < 0.5)
      player.kickToGoal();
    else
      player.interceptBall();
  }
}
```

**Figura 7: Código Java extendiendo la clase `Player`.**

- La implementación de `Parser` soporta todos los protocolos de *Soccer Server* desde la versión 4 hasta la versión 13.
- En el caso particular de la clase `Coach`, ésta accede a los métodos públicos implementados por `WorldState` y `Commander` mediante los campos `field` y `coach`, respectivamente.

De acuerdo a lo expresado, la figura 6 muestra un ejemplo de como crear un equipo de fútbol mediante la extensión de la clase `Team` en Java. Puede observarse, que después de configurar cada uno de los parámetros principales, el código asigna los agentes de control de cada jugador conforme su número de identificación. Si el equipo tiene un entrenador, su número de identificación sería el 0.

De forma similar, la figura 7 presenta un ejemplo de como extender la clase `Player` para crear un agente de control para uno o varios jugadores. Como se observa, la estrategia de control debe incluirse dentro del método `strategyCycle` haciendo uso de los campos heredados `field` y `player`. En el ejemplo presentado, el jugador intenta siempre acercarse al balón, y en caso de conseguir su objetivo (*i.e.*, distancia $< 0.5$), patea en la dirección del gol contrario.

Finalmente, la figura 8 muestra un ejemplo de como extender la clase `Coach` de manera a crear un entrenador

```
import jplayer.*;

public class myCoach extends Coach {
  public myCoach() {
    super();
  }

  public void strategyCycle() {
    //this should be the high-level strategy
    if (field.getStatus() != field.PLAY_ON)
      coach.say("Go to initial position")
  }
}
```

**Figura 8: Código Java extendiendo la clase `Coach`.**

para el equipo. En el ejemplo presentado, el entrenador es sumamente simple y únicamente genera un mensaje cada vez que el juego se interrumpe por cualquier motivo. Note también la posibilidad de acceso a los campos heredados `field` y `coach`.

## Evaluación

De forma a evaluar las características de la infraestructura propuesta, se han creado jugadores individuales y equipos completos con estrategias de juego simples. Estos elementos serán colocados en situaciones controladas para analizar la facilidad de uso y poder de acción de nuestra infraestructura.

Todos los experimentos fueron realizados en un computador *Pentium Dual Core* a 3.4 GHz ejecutando Linux 2.6.31 (Fedora 12) con *OpenJDK* 1.6.0. También fueron instalados los paquetes *Soccer Server* 14.0.2 y *Soccer Server Monitor* 14.0.1.

### *Arquero vs. Goleador*

Este experimento confronta un agente atacante (con características goleadoras) contra un arquero. El atacante intenta 100 veces marcar un gol pateando desde posiciones aleatorias ubicadas entre 25 y 30 unidades del arco rival. Del otro lado, el arquero, hace su mejor esfuerzo por agarrar el balón. Este experimento muestra la robustez y eficiencia de los métodos de alto y bajo nivel ofrecidos por la infraestructura. En el caso del arquero se usa principalmente 'interceptar el balón', mientras que el atacante simplemente invoca 'patear hacia el arco rival'.

Los resultados del experimento son presentados en la tabla . Como se puede ver, el arquero intercepta el balón el 61 % de las veces. Si bien el atacante escoge el mejor ángulo de disparo para hacer el gol, los goles no ocurren tan frecuentemente. De hecho, 5 % de las veces el balón sale desviado. Considerando el indeterminismo introducido por el *Soccer Server*, creemos que estos números son bastante representativos de un comportamiento 'normal' en partidos de fútbol reales.

### *Pases del Balón*

Para evaluar la capacidad de coordinación entre varios agentes usando los métodos de alto y bajo nivel ofrecidos por nuestra infraestructura se ha construido un equipo de 4 jugadores que intenta pasar el balón entre ellos. Para dificultar su tarea, un agente contrario trata de interceptar el balón, lo cual limita las posibilidades de pase dentro del equipo de agentes cooperantes.

Los resultados de este experimento son mostrados en la tabla . Después de 100 intentos de pase, casi 9 de cada 10 tuvieron suceso. Note que la estrategia de alto nivel no busca solamente realizar un pase, sino también mantener el balón lejos del agente contrario, y desde el punto de vista del agente receptor, inclusive la captura del balón fue realizada con éxito la mayoría de las veces.

### *Partido de fútbol*

Si bien las evaluaciones anteriores ya presentan resultados interesantes, no hay mejor indicación de que tan buena es una infraestructura de desarrollo de Agentes RoboCup que la construcción de un equipo de fútbol completo y enfrentarlo en un partido 11 contra 11. Obviamente, como las estrategias de alto nivel no son el enfoque principal de este trabajo, se implementaron mecanismos de control simples donde cada jugador intenta mantenerse en una zona asignada previamente, capturar cuando sea posible el balón, y pasarlo a un compañero mejor posicionado. Adicionalmente, el arquero tiene una zona de cobertura más reducida y su prioridad es interceptar el balón. Por otro lado, los atacantes priorizan el patear hacia el arco contrario cuando es factible.

Al enfrentarse 2 equipos basados en las mismas estrategias de control simple, se percibió un equilibrio en el juego que no deja entrever mayores conclusiones. Quizás, esta no es la mejor forma de evaluar un equipo de agentes, pero algunos trabajos futuros mencionados en la última sección de este artículo nos permitirán realizar mejores evaluaciones del desempeño de un equipo basado en la infraestructura aquí propuesta. No obstante, es importante mencionar que durante el partido entre estos 2 equipos, se consume en media 20 % del tiempo de CPU.

## Trabajos Relacionados

Los primeros intentos por ofrecer bibliotecas para usarlas en la construcción de agentes RoboCup fueron normalmente la liberación de partes del código fuente de equipos ya existentes. Uno de los equipos que más suceso tuvo en este sentido fue *CMUnited* ganador de RoboCup 1998 y 1999 [7]. El problema en este caso es que el código incluye poca o ninguna documentación y su funcionalidad está estrechamente ligada a la estructura del equipo original.

| **Comportamiento final** | **Número** |
|---|---|
| El arquero atrapa el balón | 61 |
| El atacante marca un gol | 34 |
| El balón abandona la cancha | 5 |
| **Total** | **100** |

**Tabla 1: Arquero vs. Atacante.**

| Comportamiento final | Número |
|---|---|
| El pase es realizado con suceso | 88 |
| El atacante intercepta el balón | 5 |
| El balón abandona la cancha | 7 |
| **Total** | **100** |

**Tabla 2: Pases del balón.**

Surge entonces la necesidad de bibliotecas bien estructuradas y documentadas para que cualquier programador pueda utilizarlas [8]. Ejemplos de este tipo de bibliotecas son *Libsclient* y *RoboLog*. *Libsclient* es una biblioteca de funciones en C que provee funcionalidades básicas a los clientes ROBOCUP. Por otro lado, *RoboLog* es una biblioteca Prolog construida sobre C++ [9], lo que facilita la utilización de inferencia lógica en la programación de los agentes ROBOCUP.

Conforme aumenta la funcionalidad de estas bibliotecas, también aumenta su tamaño, y pronto aparece la necesidad de utilizar el paradigma de orientación a objetos. Este cambio facilita el reuso de código y extiende la modularidad de las bibliotecas. Ejemplos de esta tendencia son *RoboSoc* y *Atan*. *RoboSoc* es implementado en C++ y esta orientado al desarrollo de agentes con propósitos educativos [10]. En cambio, *Atan* es parte del proyecto *VSOC* que pretende construir un equipo ROBOCUP donde cada jugador basa su control en el uso de RNA [11]. *Atan* está basado en el lenguaje Java.

Al igual que varios trabajos previos, esta infraestructura nace de la necesidad de crear un equipo ROBOCUP orientado al estudio de la inteligencia de enjambres [6]. Al percibir las falencias de las plataformas de desarrollo existentes, nuestra meta fue ofrecer funcionalidades de alto nivel a más de las básicas ya existentes en esos otros ambientes. Además, se hizo énfasis en la modularidad del código para que las estrategias de control de alto nivel puedan inclusive reemplazar parte del código interno con funcionalidades distintas o implementaciones optimizadas de las ya existentes.

Finalmente, nuestra infraestructura busca ofrecer soporte a diversos tipos de estrategias de control de una manera bastante general. Si bien existen propuestas para usar IA a través de RNA, algoritmos genéticos o inteligencia de enjambres [12, 13, 14], existen otras que combinan principalmente algoritmos de programación convencional con ciertas rutinas de IA [15, 16, 17].

## Conclusiones

Este trabajo demuestra la factibilidad de implementar un equipo ROBOCUP flexible y colaborativo con pocas líneas de código. Es así que la infraestructura propuesta ayudará a que la *Liga de Simulación* pueda ser utilizada fácilmente en la enseñanza de IA mediante estrategias PBL, mejorando el entendimiento de los diversos conceptos y aplicaciones asociados a IA. La flexibilidad y modularidad incluida en nuestra infraestructura permite abordar los problemas inherentes a ROBOCUP con diversos grados de profundidad, permitiendo tanto el uso de reglas `if-else` simples como quizás esquemas de control adaptable. Por otro lado, la facilidad de usar comunicación auditiva entre los agentes y la posibilidad de incluir un entrenador, abren nuevas áreas de experimentación para el estudio de la colaboración en sistemas multi-agente.

En un futuro cercano, pretendemos hacer algunas mejoras y extensiones a nuestra infraestructura. Entre las principales podemos mencionar:

1. El desarrollo de estrategias de control de alto nivel basadas en técnicas típicamente asociadas a IA.
2. El desarrollo de un sistema de entrenamiento automático basado en el comportamiento de equipos ganadores, muy útil al trabajar con RNA.
3. La adaptación de una serie de experimentos que puedan ser usados como laboratorio para la enseñanza de IA en la USFQ.

## Referencias

[1] Berliner, H. J. 1975. “Chess as problem solving: the development of a tactics analyzer.” PhD thesis Carnegie Mellon University Pittsburgh, PA, USA.

[2] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. 1997. “RoboCup: The robot world cup initiative.” In W. Lewis Johnson and Barbara Hayes-Roth, (ed.), Proceedings of the First International Conference on Autonomous Agents, New York, NY: ACM Press. 340–347.

[3] Noda, I., Matsubara, H., Hiraki, K., and Frank, I. 1998. “Soccer Server: A tool for research on multi-agent systems.” *Applied Artificial Intelligence*. 12, 233–250.

[4] Kok, J. R., Vlassis, N., and Groen, F. 2003. “UvA Trilearn 2003 – team description.” In D. Polani, B. Browning, A. Bonarini, and K. Yoshida, (ed.), Proceedings of RoboCup 2003 International Symposium, Padua, Italy: Springer-Verlag.

[5] Chen, M., Dorer, K., Foroughi, E., Heintz, F., Huang, Z., Kapetanakis, S., Kostiadis, K., Kummeneje, J., Murray, J., Noda, I., Obst, O., Riley, P., Steffens, T., Wang, Y., and Yin, X. 2003. “RoboCup Soccer Server – User Manual.” Technical Report 7.07. The RoboCup Federation.

[6] Carrera, E. V. 2006. “Applying Collaborative Intelligence to RoboCup.” In Proceedings of the 32nd Latin-American Conference on Informatics. Santiago, Chile.

[7] Stone, P., Riley, P., and Veloso, M. 2000. “CMUnited-99: RoboCup-99 simulator world champion team.” *AI Magazine*. 21, 33–40.

[8] Tambe, M. 1997. “Towards Flexible TeamWork.” *Journal of Artificial Intelligence Research*. 7, 83–124.

[9] Murray, J., Obst, O., and Stolzenburg, F. 2000. “Robolog koblenz.” In RoboCup-99: Robot Soccer World Cup III London, UK: Springer-Verlag. 628–631.

[10] Heintz, F. 2000. “Robosoc a system for developing robocup agents for educational use.” Master’s thesis Department of Computer and Information Science, Linkoping University Linkoping, Sweden.

[11] Qasem, A., Wagner, W., and Woll, R. April 2010. Atan Project. http://atan1.sourceforge.net/.

[12] Aronsson, J. 2003. "Genetic Programming of Multi-agent Systems in the RoboCup Domain." Master's thesis Department of Computer Science, Lund University-Lund, Sweden.

[13] Kutsenok, A. 2004. Swarm AI: "A solution to soccer" Master's thesis Department of Computer Science, Rose-Hulman Institute of Technology. Terre Haute, IN.

[14] Browne, K., McCune, J., Trost, A., Evans, D., and Brogan, D. 2002. volume **2377** of Lecture Notes in Computer Science. Springer. Berlin, Germany. 499–502.

[15] Kok, J. R., Vlassis, N., and Groen, F. 2004. "UvA Trilearn 2004 – team description." In D. Nardi, M. Riedmiller, and C. Sammut, (ed.), Proceedings of RoboCup 2004 International Symposium, Lisbon, Portugal: Springer-Verlag.

[16] Gutmann, J.-S., Hatzack, W., Herrmann, I., Nebel, B., Rittinger, F., Topor, A., and Weigel, T. 2000. "The CS Freiburg Team – playing robotic soccer based on an explicit world model." *Artificial Intelligence Magazine*. 21, 37–46.

[17] Noda, I. and Stone, P. 2003. "The RoboCup Soccer Server and CMUnited Clients: Implemented Infrastructure for MAS Research." *Autonomous Agents and Multi-Agent Systems*. 7, 101–120.