Evolving object oriented agent programs in Robocup Domain

David E. Suárez
Student
Universidad Distrital
Member of Laboratory of Automatic, Microelectronic and Computational Intelligence (LAMIC)
Bogotá, Colombia
desuarezp@estudiantes.udistrital.edu.co

Julián Y. Olarte
Student
Universidad Distrital
Member of Laboratory of Automatic, Microelectronic and Computational Intelligence (LAMIC)
Bogotá, Colombia
jyolarter@estudiantes.udistrital.edu.co

Sergio A. Rojas
Faculty mentor
University College of London
Universidad Distrital
Member of Laboratory of Automatic, Microelectronic and Computational Intelligence (LAMIC)
Bogotá, Colombia
srojas@udistrital.edu.co

ABSTRACT
This paper describes the application of object oriented genetic programming to the automatic generation of agents under the Object Oriented Paradigm. To generate the agent programs code, we evolve concurrently the methods that represent the agent-environment interaction. We use like terminals and operations the objects that correspond to the context elements. This study uses the simulation league of the Robot World Cup (Robocup) like a testing environment. The fitness function used evaluates the behavior of agent player in several levels that indicates the learning progress. The experimental results indicate that is possible the agent programs evolution under the Object Oriented Paradigm.

Categories and Subject Descriptors
I.2.2–Automatic Programming, Program Synthesis; I.2.11–Distributed Artificial Intelligence, Intelligent agents; D.3.2–Language Classifications, Object-oriented languages.

General Terms
Algorithms, Design, Languages

Keywords
Genetic Algorithms, Genetic Programming, Agent programs.

1. INTRODUCTION
An agent is a computational process that implements the autonomous, communicating functionality of an application [6]. Also, the agents "live" in a dynamic virtual or real environment and perceive and act to achieve goals in a specific context. These entities require the creation of algorithms that generate intelligent answers to environment messages for the solution of problems and conflicts. Some recent research [5][13] suggest that an agent (in other words their algorithms) can be generated automatically using Computational Intelligence techniques like Genetic Algorithms or Genetic Programming.

Genetic Programming Technology (GP)[3] allows the automatic generation of computer programs by natural selection and Darwin's Evolution Theory. The process uses a large initial population of programs that are random combinations of the problem's specific variables and functions. Through the process, all generated random programs are evaluated using fitness functions and then they are altered with genetic operations like mutation or crossover. These genetic operations depend on the schema representation used in the programs and many alternatives to standard GP are developed by the GP community. All these schemas look for generating typically functional or list based algorithms, except some research based on object oriented programming [4][9][10].

Last and best implementations of Intelligent Agents and Multi-agent Systems have been developed on the object oriented programming (OOP) paradigm [7][11]. It creates a great challenge for the recent investigation about automatic generation of agents programs because the implementations under the OOP have high quality and they have characteristics as re-use, modularity and high abstraction [3].

All recent research about automatic generation of agent programs use the standard GP as the generation method, but to the authors' best knowledge, no work the generation of object oriented agent programs. This project proposes to use the research of several authors (e.g., Bruce [4], Langdon [9] and Lucas [10]) who have been working with a GP method for the generation of object oriented programs to generate object oriented agents, in a similar way that [5][13]. The proposal for the generation of object oriented programs include simultaneous induction of several methods of the same object [4][9] and the use of complex objects as iterators. Other additional functionality might be the use of reflexion to explore the API object library system [10].

Genetic and Evolutionary Computation Conference (GECCO)'05, June 25-29, 2005, Washington, DC, USA.

Many experimentation environments exist for intelligent artificial agents. One of these is the Robot World Cup initiative (Robocup) [2]. We decided to use this platform because it provides a testing environment for artificial real agents and simulated agents which have to play a game similar to soccer. It is a very interesting problem to researchers in computational intelligence because it deals with navigation, planning, communication, cooperation and taking of decision making for issues that must be solved in real time in a highly dynamically changing environment. There are many different methods of producing players to compete in the Robocup competition and the most popular of which appears to be agent based research. And the best way to generate agents would be to generate them using the object oriented paradigm.

2. EVOLVING OBJECT ORIENTED SOFTWARE AGENTS

Most of implementations of GP search for the generation of lists or functional based programs, but some investigations have dealt with the evolution of programs guided by objects. The most similar prior work to this paper in evolving object oriented programs is Abbott's initial exploration of Object Oriented Genetic Programming (OOGP) [1] and Lucas's exploiting reflection in OOGP [10]. These studies, and Bruce's and Landong's research, demonstrate that it is possible to simultaneously evolve object member functions and their cooperation in the self-organization of the internal memory of the same one.

In an agent, the interaction model with the environment defines its internal state. A specific example of information in a physical agent is the visual and body perception information. The facts found in the environment change the internal state of an agent and produce a specific response for the situation.

In object oriented programming, the responses of an agent might be modelled with an object method call and the environment information is passed as an argument. The actions of the agent are executed by calls to objects that reference the environment. For example, in Aglets platform [8] the object model defines a set of methods that represents the responsibilities of the agent in relation to the context. These methods' set are standard for all agents and define the calls that the environment execute over each agent to inform their own changes [7]. All these methods which represent the agent interaction can be evolved in a similar way to the stacks and queues examples evolution in the Bruce's and Landong's research.

We wish to evolve the code that implements the response agent's methods given a set of method signatures. The methods' implementation is a sequence of instructions where each instruction consists of a method reference (the Method to be invoked) and an object reference (the Object to invoke it on). Other elements in the methods implementations to be evolved might be: control instructions (like *if* or *while* sentences), and scope brackets.

In the agent context the principal object to be used is a reference to the environment and the principal methods of this reference are messages about the change of the agent state and queries about environment's items (e. g., shared elements, databases, and general info).

3. ROBOCUP AS EXAMPLE APPLICATION

The Robocup context has appropriate characteristics that allow software agents to work in. Here, it is possible to exploit the autonomy, social ability, reactivity, and pro-activity properties. The Robocup is divided in several categories; one of these categories is the simulation league, where the players are software agents programs. In the simulation league, games are carried out between two teams of eleven players each, under a client/server scheme with UDP/IP communication protocols. The Robocup organizers provide the server and the monitor for the simulation. The players are developed by each research group, and they are connected as clients to the server.

The simulation server (SoccerServer) takes charge of controlling all information of the game like ball and players' position or the time of game. The monitor (SoccerMonitor) makes the two-dimensional visualization of the game field in the screen. The clients request information from the server, and based on this, they execute control commands such as to rotate, to kick, to move, to speed up, and others.

Each player determines his position in a relative way to certain objects like goals, limit lines, ball and other players. The visual information is received every pre-determined period of time (see Figure 1.). Additionally, the players can receive auditive and body sense information. The auditive info is composite by the decisions and referee's orders, and possibly messages sent by their team partners and coach. The body sense info represents the action of the environment to the player, for example, the lost of corporal energy.

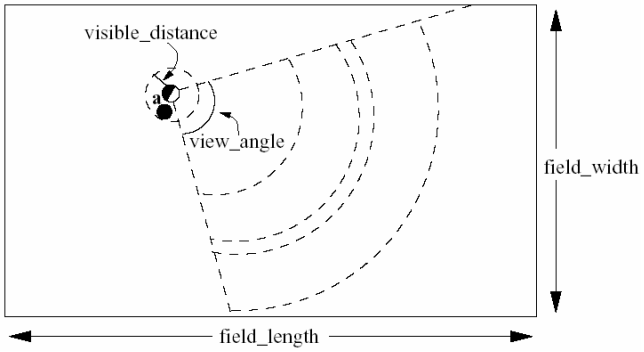

Figure 1. The visible range of an individual agent in the soccer server.

Our goal is to automatically develop players which are designed according to the object oriented paradigm and have interaction with the soccer server. For this purpose we use the Java language programming to generate our Agents. In fact, we have to evolve three principal methods: one to manage the visual information, other for perceiving sound information and the last one for the body perception information. The code of our player (see, Figure 2) has three principal methods that receive as parameters, objects that represent the complex audio, visual and body perception information of the players. The reason to evolve the three methods (and not alone one of them) is which in the object oriented programming all the methods of a class

collaborate to each other at the same time to complete the class objective.

Table 1. Terminals and functions used in these experiments.

Terminals	Type	Description
ball rightGoal leftGoal	Objects	We use this three principal objects. All these have this next properties:
m_isVisible	Bool	Visibility
m_distance	Float	Distance to the object
m_direction	Integer	Direction to the object
m_distChange	Integer	Delta of the distance
m_dirChange	Float	Delta of the direction
Functions	**Type**	**Description**
turn(float)	void	Function that spins the player
move(float, float)	void	Function that moves the player
kick(float,float)	void	Function to kick the ball
dash(float)	void	Function to dash *n* steps
catchBall(float)	void	Function to take the ball (goalie)
turnNeck(float)	void	Function to turn the head
add(float,float)	Float	Arithmetic sum
sub(float,float)	Float	Arithmetic subtraction
mult(float,float)	Float	Arithmetic multiplication
divi(float,float)	Float	Arithmetic division
mod(float,float)	Float	Arithmetic modulus
equals(bool,bool)	Bool	Comparative equals (= =)
less(bool,bool)	Bool	Comparative less than (<)
greater(bool,bool)	Bool	Comparative greater than (>)

Evolution process experiments begin with the random generation of our players using the terminals and functions described in the table 1.

```
public class PlayerExample extends PlayerAgent{
   // Some declarations
   // Principals function to evolve
   public void onSee(VisualInfo inf) {
       // some evolve code...
   }
   public void onSenseBody(PerceptiveInfo inf){

        // here used the info object.
        // here used the server object
        // to send messages...
   }
   public void onHear(AudibelInfo inf){
        // some evolve code...
   }
   // other functions ...
}
```

Figure 2. Example code of the standard functions to evolve in an example player agent.

Then a tree representation of agent programs, which include all response functions, is constructed (see, Figure 3). We use Strongly Typed Genetic Programming with type inheritance [12]; it allows using several data types and their inheritance hierarchy. Also, Bruce's research confirms that, in this context, STGP is as much as or more efficient than the standard GP. To perform the type inheritance checkup, we used the Java reflection API to discover the inheritance hierarchy. This indicates that in our experiments, we do not use data primitive types.

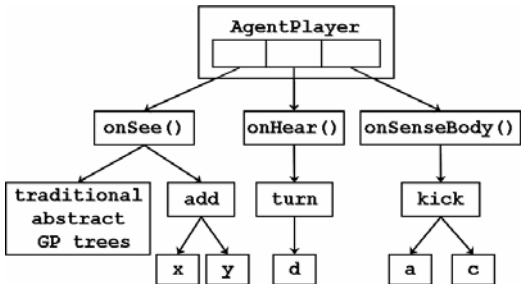

Figure 3. Tree representation of object oriented code agent.

The next step is to take each agent in the population to measure their performance with the fitness function. This function evaluates the players' behavior in the playing field considering parameters as goals, shots, passes, and other elements of the game. For each type of player, there is one fitness function.

The fitness function is designed to induce the player to improve the behavior to score, in the established time for each game. The agent's actions have a learning order; first, the ball search, second, the proximity to it, third, the kick to the ball and, lastly to score. The fitness function is calculated in the following way:

$$Fitness = \frac{(V_b * P_1) + (N_b * P_2) + (K_b * P_3) + (G_p * P_4)}{Total_{cycles}} \quad (1)$$

Where V_b is the number of times that the ball is in the range of player's vision; N_b is the average of the distance to the ball per the number of times that the ball is visible for the player; K_b is the number of kicks in direction to enemy goal; and G_p is the number scores. All variables depend of the number of cycles per game ($Total_{cycles}$). Each action has a learning weight associated and represents the level of the each player, for example, the visibility of the ball has weight P_1. From this it follows that:

$$P_1 < P_2 < P_3 < P_4 \quad (2)$$

When the fittest agents are selected, they are submitted to several genetic operations. These operations can be both, mutation and crossover functions, and they work in a similar way than traditional GP. The whole process is repeated over many generations until the finalization criterion is reached.

Our developed evolution system called PROLE2 is shown in the Figure 4. It is based on the model described by [13]. It is divided in two parts; the first one is the Robocup system implementation and the other one is the agent code generator. The generated agents are tested in the soccer server and its fitness value is returned to the agent code generator to continue the evolution.

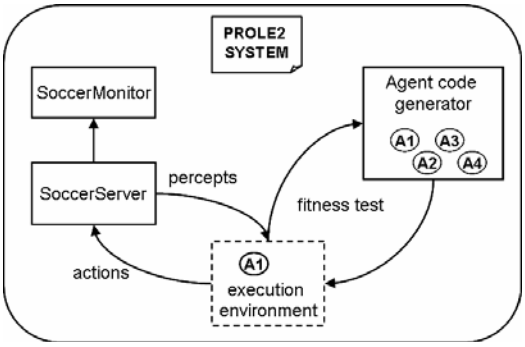

Figure 4. Framework for the evolution of artificial object oriented players.

For the experiments a population of 30 players was used during 60 generations. To each player it was granted a time of 600 cycles to play. The roulette method was used as the selection method and the genetic operators were crossover and mutation. The experiments objective was generate autonomous behaviors in the agents, exploiting the object oriented programming benefits.

4. DISCUSSION

Given the variability and complexity of the environment and the very long run time required to get practical results, this project requires a second experimentation phase, where it would be possible to analyze other items in the agents generation like the collective behavior [13], intelligence emergence and other topics.

The use of Strongly Typed GP used in these experiments is the same that propose [12] with the difference that we don't have to make a type tree hierarchy, if not that, we used the java API reflection.

The generated agents in the first experiments exhibited autonomous behaviors like running behind the ball or searching and kicking the ball. In response to the sound information, the agents did not evolve any coherent behavior. Possibly it occurred because of the deficiency of meaning of set sound stimulus; we had not planned a sound stimulation strategy. Other element that can be used to the evaluation of agent programs is the object oriented metrics.

In particular we demonstrated the possibility of the generation of object oriented agents programs using the previous work on OOGP and Automatic generation of agents. We also demonstrated the emergence of simple behaviors in a system that is considerably constrained and with high complexity. We believe that this work presents an approach to the implementation of evolutionary techniques in commercial systems like all agents platforms which are developed over object oriented paradigm.

In the future, this research will look for to finish a complete team for the Robocup simulation league, and later, it will look for commercial applications for the Object Oriented Agents automatically generated.

5. ACKNOWLEDGMENTS

Our thanks to the Universidad Distrital, and to all members of the Laboratory of Automatic, Microelectronic and Computational Intelligence (LAMIC).

6. REFERENCES

[1] Abbott, R., Guo, J., and Parviz, B., *Object-Oriented Genetic Programming*. The 2003 International Conference on Machine Learning; Models, Technologies and Applications (MLMTA). 2003.

[2] Birk A., Coradeschi. S., Tadokoro S. (Eds.). *RoboCup 2001: Robot Soccer World Cup V*. Berlin: Springer-Verlag. 2002.

[3] Booch, G., *Object oriented analysis and design with applications*. Second edition. Redwood City, CA: Benjamin/Cummings Publishing Company, Inc. 1994.

[4] Bruce, W., *The Application of Genetic Programming to the Automatic Generation of Object-Oriented Programs*. Ph.D. Tesis. On line: ftp://cs.ucl.ac.uk/bruce.thesis.ps.gz. 1995.

[5] Cliff, D., P. Husbands, J-A. Meyer, and S.W. Wilson, Eds. 1994. *From Animals to Animats 3*. MIT Press.

[6] FIPA Rationale. *Foundation for Intelligent Physical Agents*. On line: http://www.cselt.stet.it/fipa/fipa_rationale.htm. 1996.

[7] Garcia, A., Lucena, C. y Cowan, D. *Agents in Object-Oriented Software Engineering*. Elsevier: Software: Practice & Experience. 2003.

[8] IBM, Research. *Aglets*. On line: http://www.trl.ibm.com/aglets/, Last change: 14 March, 2002.

[9] Langdon, W. *Evolving data structures with genetic programming*. In Proceedings of the Sixth Int. Conf. on Genetic Algorithms. San Francisco, CA: Larry J. Eshelman, Ed., pp. 295-302, Morgan Kaufmann.1995.

[10] Lucas, S. *Exploiting Reflection in Object Oriented Genetic Programming*. EuroGP 2004: 369-378. 2004.

[11] Omicini, A. *From Object to Agent Societies: Abstractions and Methodologies for the Engineering of Open Distributed Systems*. AI*IA/TABOO Joint Workshop. 2000.

[12] Shoenefeld, D., Haynes, T., Wainwright, R. *Type Inherietance in Strongly Typed Genetic Programming.* Chapter in book: Advances in Genetic Programming 2, Editors: Kennet E. Kinnear, Jr., And Peter J. Angeline. MIT Press, 1996.

[13] Spector, L. *Automatic Generation of Intelligent Agent Programs*. In IEEE Expert. Jan-Feb 1997, pp. 3-4