AGLIPS: An educational environment to construct behaviour based robots

Luis Moniz[1], Paulo Urbano[1], and Helder Coelho[1]

Faculdade de Ciências de Lisboa
Campo Grande, 1749-016 Lisboa, Portugal
{hal,pub,hcoelho}@di.fc.ul.pt

Abstract. We present a tool to aid the construction and management of behaviour-based robot in a simulated environment that shows collective behaviour. This framework is based on two different tools: the extended Aglets multiagent platform plus the Player/Stage environment for robotics simulation. The framework AGLIPS links these two heterogeneous environments into a single platform, providing us with a tool for constructing agents and an environment for experimenting them. The resulting testbed merges the Aglets platform features and the dynamic and unpredictable characteristics of the Player/Stage environment producing a joint tool capable of combining social and physical aspects of agents useful for more advanced experiments.

1 Introduction

Verifiable scientific experiments require that they can be repeated independently of time and space. The designer needs tools to construct, manipulate, and measure the experiment. Most of physical experiments with behaviour-based agents are very time and cost consuming, and most of the time is spent in hardware calibration and trouble shooting. Only a small amount of the resources is used in the design of real robot mind. Another problem arises from the nature of a real experiment, and it resides on the proper analysis of the outcomes. Due to the distributed nature of the robots and to the fact that no central control exists, it is very difficult to get data and identify what were the processes that originated certain behaviour. Motivated by these problems, we developed a tool to build behaviour-based agents that evolve in a simulated environment. This framework incorporates characteristics of intelligent agents environments (Aglets)[3] and multiagent simulation tools (Player/Stage)[2, 5]. The possibility of repeating an experiment, by changing some initial parameters, allows the user to conduct simulations and observe how different conditions influence different behaviours. The remainder of the paper is organised as follows. Section 2 will address the individual features of both platforms and how they interact. Section 3 presents the Virtual Laboratory architecture and what is available to build robots. Section 4 shows an experimental example of chasing behaviour.

2 Aglips Framework

Our platform framework is constructed based on two different tools: the Aglets environment and the Player/Stage platform. The Aglets environment is an cognitive and mobile agent oriented environment with a set of tools to construct agents and a platform where the agents can be executed and controlled. The Player/Stage platform is a simulated environment to build and simulate behaviour-based robots whose code can be downloaded into real machines.

2.1 The Aglets Environment

The Aglets environment is a platform inside which all the agents, mandatory written in JAVA, are executed. This platform provides the agents with a set of services (message transport, mobility, cloning, etc) and an execution model that must be followed. The Aglet life cycle is event driven, and it based in the Aglet response to these events. An Aglet must be constructed regarding the model that regulates its life cycle, we must define what to do when it is created, when it moves or when it is removed from the platform. In the platform, each agent is identified by an aglet proxy (a kind of internal address), associated with it at creation time. The platform supports a message transport service based on aglets proxies. The knowledge of these proxies is crucial to all the agents in order to support the communication among them. The environment does not provide a central mechanism to track the agents and know who is active when. For instance, when an agent moves from one platform to another its proxy changes, making that all the other agents loose its contact. The system provides also a control interface and aglet viewer (Tahiti) with which the user can launch aglets, control their state, and remove them from the platform. The Aglet architecture structure is based on its life cycle. For instance, when an Aglet is created inside the platform, its creation method is invoked, and then when it is activated the run method is invoked. The same happens to all main events on the Aglet cycle, to each one there is an Aglet behaviour pre-determined to be invoked. The definition of an aglet behaviour is made by redefining some of these methods, identifying which actions should be performed in each situation. This architecture and the use of the JAVA language to define agents have some advantage when redefining Aglets based on previous ones; it is only needed to define what is different. For instance, if we have an Aglet that checks whenever email arrives, we can use this agent to define other that remove all spam mail, based on some criteria.

2.2 The Player/Stage Environment

The Player/Stage platform simulates a team of mobile robots moving and sensing in a two-dimensional environment. The robots behaviours are controlled through the Player component of the system. The Stage component provides a set of virtual devices to the Player, various sensors models, like a camera, a sonar and a laser, and actuators models, like motors and a griper. It also controls

the physical laws of robot interaction with each other and the obstacles. In our current environment only the sonar, laser and motors are usable. This tool provides a controllable framework to build tests and experiments in a simple robotic environment. The physical environment is defined in set of files containing the number and position of each robot, the host/port from which each robot can be controlled and the obstacles disposition (in a graphics file).

2.3 Interface Environment

The interface environment provides the user with a set of tools to access and control the Aglets and Player/Stage environments. This environment has three different tools: the agents' manager, the agents' consoles, and stage monitor. The stage monitor provides us with a graphical representation of the environment and the robots during the simulations. It gives little control over the robot simulation. The agents manager is an interface to allow the low level control of the Aglets platform, for instance, it gives tools to launch/kill/move an agent, start a simulation, monitor what agents are active, etc. In the figure 1 we present an example of this interface.

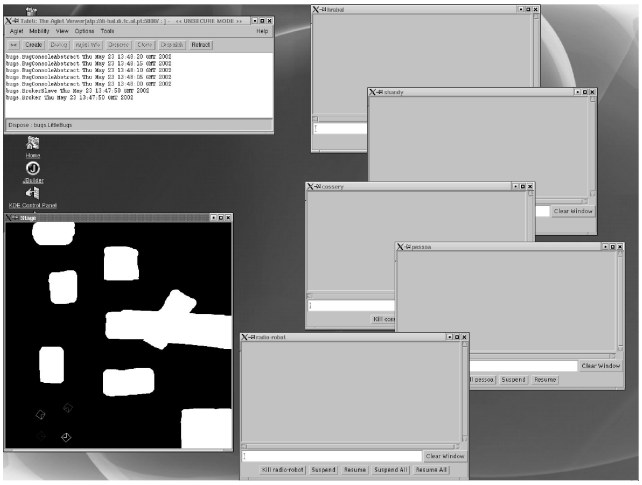

Fig. 1. The Aglips Interface. This picture presents a view of the Aglips environment: agent manager, stage viewer, agent/robot consoles and agent controller

In order to connect all these different environments we had to support heterogeneous communication tools. The communication Aglets and robots are supported by sockets and a Java API interface. This channel supports all sensors and actuators. The basic mechanism of communication between agents is through message passing. This service is supported by special agents inside the Aglets platform that provide the environment with white pages service and message routing. The direct communication among robots is not supported.

3 The Virtual Laboratory

Along with the complexity of the environment, the design of the agent is decisive for the collection of its behaviours. The agent's behaviour is determined by its control program, as well as the available actuators and sensors and their positioning on the agent's body. The virtual laboratory allows users to build experiments in a modular way. Agents are composed of a control mind agent in the Aglets platform, a body equipped with sensors and actuators in the Player/Stage environment, and the user interface device (see Fig.2). The user can build up a library of different agents, control programs and environments, which can be combined for different experiments. The environment can be tailored to the experiment: the user defines the number, size, and disposition of obstacles, number and position of robots, and the behaviour of each robot.

In the next sections, we will overview the framework and present what tools are available to construct the desired experiments.

3.1 Agents, Robots and Clips

The integration of the Player/Stage environment with the Aglet platform allowed us to add some new features to the environment and to associate an aglet to manage each robot. This aglet controls the robot behaviour using the Player interface (sensing and actuating), and it is capable of communicate with the other aglets (robots) through the platform. This extension provides the robots with a communicating channel (peer to peer and broadcast) that provides them with complex message exchange capabilities. Additionally we add a GPS to the system, providing the robot with the knowledge of its absolute position in the environment. We also associate a simple console command line and display to each aglet. Through this console it is possible to track the aglet execution and communicate directly with it. We also add the possibility of add a special aglet without a robot attached. This aglet revealed itself useful to track the aglet simulation and to communicate with the other agents, for instance, to broadcast a message to all of them.

The Fig. 3 shows how the robot architecture is organised in layers. The bottom layer corresponds to the features provided by the Aglet platform. This corresponds to the sequence of actions executed during the Aglet life cycle and some system tools, like send messages, mobility commands, etc. The next layer (Aglet Extended architecture) adds new functionalities to the base architecture. These overlap some original functionalities and encapsulate other in order to hide some repetitive tasks. For instance, when the agent is activated the first action it must perform is to registrate itself with the broker, indicating who it is and what are its main characteristics (kind, group, ...).

The Robot Base Architecture layer extends the Aglets with a CLIPS virtual machine to help the definition of behaviours instead of the original Java virtual machine. To simplify the design of the robot behaviour we choose to describe it using CLIPS. The CLIPS language is a rule-based language with a forward chaining based reasoning engine. The user can define the robot behaviour in

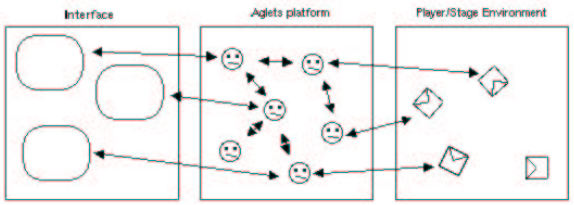

Fig. 2. The Aglips architecture. This figure shows an example of an Aglips supported experiment. Note that not all aglets had an associated robot or console, they can support other tasks, such as, experiment controllers, aglets managers or performance meters

terms of first order logic rules, in the form of pairs conditions/actions. To support the feature we had to incorporate Jess CLIPS interpreter engine in our Aglets the as the CLIPS virtual machine. When an agent is created the file, where is behaviour is defined, is passed as a parameter and loaded into the agent Jess interpreter. Then the agent moves on standby mode. The agent is activated only when it receives the run command. We also defined a set of robot modules to support some Java functionalities in the CLIPS interpreter and to link the robot with the stage environment. This level gives supports the construction of the robot controllers. These are special purpose agents to issue simulation control commands to the other robots, these commands are also given in CLIPS. For instance, the user can (through a robot controller) command a simulation robot to go to a specific location, run 10 rules of its behaviour or add a new rule to the agent.

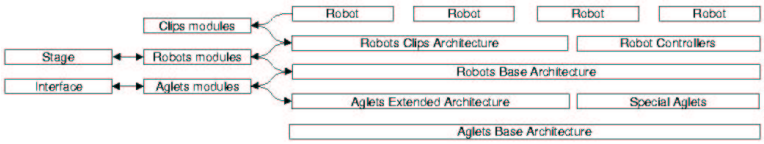

Fig. 3. The Aglips internal architecture is organized in layers, each built based on the precedent and extended using modules

The Robot Clips Architecture layer allows us to tune our robots to specific experiments without need to access to the system or modify the core architecture. This layer is completely defined in CLIPS and it is specific to each robot. This layer will be detailed in section 4.

Beside these layers, we also define a set of three modules to be used as interface between layers. These modules are sets of functions and routines that translate some of the layer internal features and gives external access to them. For instance, we defined two classes of routines in the robots modules: functions

to interface with Stage/Player environment, and functions to accede to the Aglets environment capabilities. These functions are mainly written in Java and then they are exported as foreign code into the Jess environment.

In the next sections, we will detail some of the features added and how they can be used to track our experiments.

3.2 Controller Agents

This class of agents is used to control the other agents during the simulation. They can issue or broadcast commands to the other agents, provoking a predefined reply from them. The reply is defined in various levels relatively to the place where the answer is embedded in the architecture. This corresponds to three levels in the architecture: Aglets extended architecture; Robots base architecture; Robots CLIPS architecture (see Fig. 3).

In the first two levels the agent is compelled to fullfil the request due to its own architectural constrains. These commands are embedded in the robot architecture and they are not subject to be ignored. For instance, the "suspend" command forces the robot to a stop, no more rules are executed and its speed is put to zero until the resume command is issued. The answer to the last group of commands depends on the robot architecture. These commands are issued to the robot, and it then chooses what to do.

Individual Controller Agents Each robot is associated with a controller agent, which has its own private console. Each of the consoles has full control over the respective robot. Through the console we can issue commands to the robot, like suspend and resume its activity, and even kill the robot agent. We can also obtain information concerning the robot perception and state and change its parameters and behaviour. This console provide the user with information concerning the associated robot internal states, allowing the correct tunning of its behaviour.

Global Controller Agent This special built-in agent represents the observer and is used to control the other agents in the simulation. The observer can suspend and resume all robot agents. The controller can also interrupt on-line every other robot agent order to send it commands: we may want to inspect its state or to force it to begin a particular behaviour.We consider two agent classes which must be controlled: (1) the robot agents and (2) the special agents. So, the observer can broadcast commands to all agents, to all robot agents, to a particular agent, and to a list of agents.

These commands are not restricted to this agent, other agents can also use them. But, for the global controller, which is the agent that set-up and run an experiment, they are essential commands. We must remark that when an agent broadcasts a message or a command it will never receive the message neither the command.

This agent can be used mainly for control purposes, it had no access to the internal state of any other agents, unless they explicitly provide that information.

Table 1. Commands issued by the observer to globally control each of the individual robots

height Directive	Agent	Command	Parameters
(**all-do**		*command-name*	*arg1 ... argn)*
(**robots-do**		*command-name*	*arg1 ... argn)*
(**you-do**	agent-name	*command-name*	*arg1 ... argn)*
(**list-do**	agent-list	*command-name*	*arg1 ... argn)*

3.3 Monitor Agents

We can use the Aglets facilities to produce intrusive monitor agents, capable of generating real-time information about the environment and the robots. Instead of exhibiting a transparent behaviour in the simulation, these agents request the data directly from the robots and produce some analysis. In order to allow this behaviour, the robots must understand the Monitor requests and answer accordingly to them. Next we present a fragment of a Monitor behaviour.

Example of a Monitor agent behaviour

```
...
  (robots-do activate)            ;activate all robots
  (robots-do safe-wader)          ;start wandering behaviour
  (wait ?time)                    ;wait some time
  (robots-do send-data ?myself)   ;ask for behaviour data
  (wait ?timeout)                 ;wait a timeout
  (calc-statistics)               ;calculate with received data
...
```

We want to extend this class of agents incorporating more complex, detached and transparent agents, capable of producing real time statistics, like how many times each robot got stalled, which is the more efficient, the explored world, etc, without interfere in the simulation. We should also note that these agents are fully customized by the user, and it is possible to build Monitor agents with specific characteristics, specially tunned to observe some given experiment.

3.4 Statistics

Built into the robot architecture is the log generation feature. These mechanisms automatically generate an entry in a log file each time a rule is activated. Each log is composed by a set of measures like time, robot position, robot speed and robot state. This information can later be subject of a statistical analysis in order to study the robot behaviour and performance.

We can see an example of statistical treatment of information produced by the simulation in the Fig.4. This example shows the tracks of each robot during the execution of the wandering behaviour during 5 minutes.

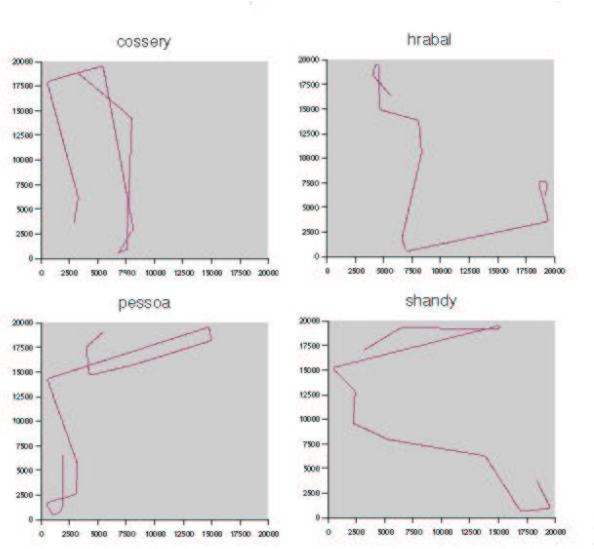

Fig. 4. Tracking robots. This figure shows an example of construction of a graphic representation of the robots path when performing the wander behavior in the cave environment

We can also use this log to make performance tests of the robots behaviour and the workbench performance. For instance, it is possible to get how many times a robot gets stucked in the environment or what is the difference between two consecutive log entries. The Fig.5 shows an example of this workbench performance measure. We can observe regular peaks and a long-term degradation of performance. The long-term degradation corresponds do the reducing of the time between peaks, meaning that the robot is taking more time between successive activation of rules.

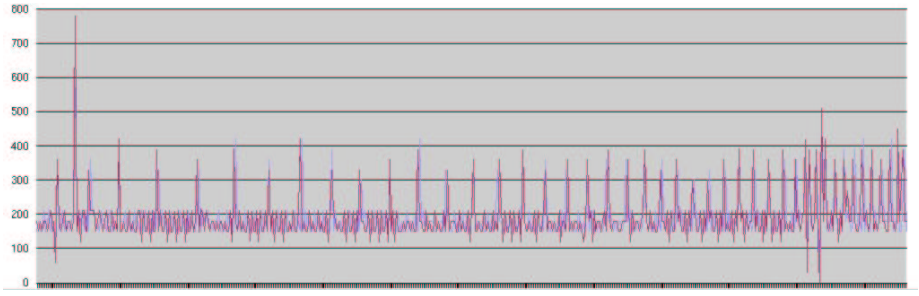

Fig. 5. Performance measure. Time difference between successive log entries

Obtaining this performance measure is fundamental to our robot/workbench tuning; the reason of this degradation could be a robot fault (someone starts eating computer cycles on executing a rule) or a workbench problem that is jamming the system. Identifying the correct motive can direct us to correct design flaws of the entire system

4 Designing and Running Experiments

Let us imagine we want to build behaviour based robots that exhibit a flocking behaviour in an world with obstacles. To achive this, they have to keep all moving together and, at the same time, avoid obstacules.

We have several alternatives to develop this behavior: we can use a low level approach and decribe all the robot behaviour in pure CLIPS; or, we can built small pieces of CLIPS code that can be assembled together to built a higher level description. We choose the second option to solve the problem. In the table 2 we present some of these CLIPS blocks that we be used to produce the behaviour.

Table 2. Some macro commands available in the CLIPS module to buid robotic behaviours

Some commands in CLIPS module		
(robots-inside-cone ?x ?y ?r ?angle)	(closest-robot ?x ?y)	(velocity)
(robot-inside-cone ?x ?y ?r ?angle)	(distance ?x ?y ?hx ?hy)	(robot-list)
(towards ?x ?y ?hx ?hy)	(one-robot)	(Xcor)
(robots-inside-circle ?x ?y ?r)	(robot-inside-circle ?x ?y ?r)	(Ycor)
(Compass)	(xcor-of robot)	(ycor-of robot)
(compass-of robot)	(velocity-of robot)	

Our robot behaviour can then be describeded in the following form: If there is an obstacle the robot must turn away from it (velocity given by obstacules avoidance function) and if there is some other robot in the perception field then turn towards the direction that robot is facing, otherwise maintain velocity (flocking vector function). Both these functions produce a pair (translation velocity, rotation velocity) that should be applied to the robots motors to produce the desired behaviour. It is the obstacle avoidance behaviour that prevents the robots to bump in each other, due the sonar sensor do not differenciate standing obstacles from robots. Next, we present a fragment of the CLIPS code that produce this behaviour.

Example of a flocking behaviour with obstacle avoidance

```
...
if (< (min (sonar)) 750) then
  Vvector = obstacule-avoidance-vector(sonar) else
```

```
  Vvector = flocking-vector((robot-inside-cone) (xcor) (ycor)
                                                (perception-range)
                                                (angle-range))
(setSpeed Vvector)
...
```

We should note that this behaviour description is similar to a subsumption architecture with two levels, where the obstacle avoidance level can inhibit the flocking behaviour. Some of these experiments can be observed at *http://ai.di.fc.ul.pt/projects.*

5 Conclusions

In this paper we present a tool to aid the construction and management of behaviour-based robot in a simulated environment. We have focused mainly on the overall architecture of our system and we have presented, in a shallow form, a number of aspects that deserved to be treated in more depth. These include a complete description of the workbench and its internal details of functioning, the methodology to build behavior based robots and all the available modules built in our system.

We introduced a number of features in our architecture which we believe to be particularly important in future developement of simulations. This include the combination of the CLIPS rule based language to describe behaviours and the capability of examine the agent mind during the simulation. This feature allows us to have a complete and accurate image of the agent mind and understand the shown behaviour. Another inportant festure is the inclusion in the system of special agents to control and observe the simulation. Instead of the tradicional approach of pre-determined behaviours, these agents can be completly contructed by the user and tunned for the experiment specific needs.

In the current stage of development the educational characteristics of the system are not fully developed. We do believe that this framework can become an acceptable tool to introduce agent programing basic concepts in a controlable environment. The capability of statistics data generation by the agents can be an important tool to analyse the overall performance of the agents and the society evolution.

References

1. Ronald C. Arkin. *Behavior-based Robotics.* The MIT Press, 1998.
2. R. T. Vaugham B. Gerkey, K. Stoy. *Player Robot Server*, version 0.8c user manual edition, 2001.
3. Mitsuru Oshima Danny B. Lange. *Programming Deploying Mobile Agents with Java Aglets.* Peachpit Press, 1998.
4. Yoav Shoam. Agent oriented programming. *Artificial Intelligence*, 60:139–159, 1993.
5. Richard T. Vaugham. *Stage: A Multiple Robot Simulator*, version 0.8c user manual edition, 2000.