An Environment Description Language for Multirobot Simulations

Antonio Chella, Massimo Cossentino, Giuseppe Tomasino
Dip. di Ingegneria Automatica ed Informatica
University of Palermo
Viale delle Scienze, 90128 Palermo, Italy
E-mail: maxco@unipa.it

Abstract

A relevant problem of simulation (but also of real systems) is the knowledge that robots have of the environment. Generally this is very poor, or quite lacking, and the robots have to explore the world around them with their sensors.

In a simulation process, sensor bearings are particularly lacking and the knowledge of the environment should be built using informations obtained in other way.

We have designed an environment description language for multi-robot agent-based simulators. The result of our approach is a complete test system for robots that operate in indoor environments whose behaviors are based upon agents. The resulting language was called EDL.

1. Introduction

Robotics applications and behaviors are becoming more and more complex and, as a consequence, their testing more and more onerous. Examples are in transportation of things on demand or on schedule in hospitals and communities, night surveillance tasks in large buldings like museums, banks, and so on. We could also enumerate all tasks where the human presence may be dangerous for the human health, such as chemical or nuclear industrial process, space experiments, and so on.

In addition, to test cooperative behaviors of multirobot systems, it is necessary to involve several robots. This kind of test could be a problem, because each robot is an expensive resource, and the same robots are shared among several different projects at the same time.

In such a scenario, simulation has proved very important to perform initial tests of multi-robot systems.

A relevant role in an intelligent system, and specifically in a multi-robot system, is played by its *knowledge*. The knowledge about the environment is particularly important because from the environment the robot receives a lot of perceptions, that it uses to plan its own actions in order to meet its design objectives. Then a relevant problem is that the necessary knowledge is initially very poor, or quite lacking, and the robots have to acquire it by exploring the world around them. In a real system they can use their sensors, but in a simulation process sensor bearings are particularly lacking, and the knowledge should be built using informations obtained in other way.

In the following sections we present EDL (Environment Description Language), an environment description language designed to be used in conjunction with a multi-robot agent-based simulator. EDL has been thought to generate indoor environments description in a very simple way, but it is enough flexible to deal with complex problems.

Using EDL is possible to simulate the robots learning about environment in very simple manner, because the user defined map of these environments, is always available for the vision agent of the robots, and so it can get which parts of the map its sensors are "seeing" during the navigation.

2. The language

EDL can describe and represent in a simple manner, but also with great precision, the features of an indoor environment. The result is a map that contains the description of all the elements and can be profitable used in robot simulations.

The simulation area is contained in a rectangular motion field, and its dimensions (length and width) are user-defined. These dimensions are real numbers that refer to a Cartesian co-ordinates system with its origin point in the bottom left-hand corner of the map. The user has to specify the components of the environment, and their specific attributes; these parameters are real numbers representing the specific geometric features of each component, for example its length, its width, and its position referring to a Cartesian co-ordinates system. All the values are measured in centimetres.

In EDL the components of the environment are called *members*. Members currently supported are:

- **Point**, representing a point of the plane;
- **Corridor**, representing a corridor inside the environment;
- **Wall**, representing a partition wall;
- **Door**, representing a passage through a partition wall ;
- **Junction**, representing corridors crossing;
- **Notice**, representing a graphical element of the environment (for example a painting).

To define the structure of the EDL members we referred to Saphira artifacts geometry [3].

2.1 Point

The point member is used to represent an oriented point of the Cartesian plane. Its parameters are the values of abscissa and ordinate in the Cartesian co-ordinates system, but also its *orientation*, referring to the global co-ordinates system. It is the angle (α in fig. 1) formed between the local co-ordinates system fixed on the point, and the global co-ordinates system.

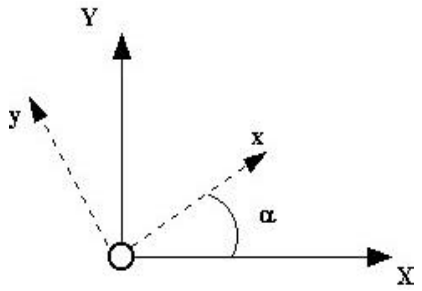

Figure 1: orientation of the point

We adopted the oriented point instead of a simple not oriented point because it can be useful in the description of the movement of a robot. In fact, in this situation, the target of the motion is often represented not only by the co-ordinates but also by the orientation that the robot should have.

2.2 Corridor

The corridor member is used to represent a corridor of the environment. Its parameters are the length, the width and the position of its barycentre. The location of this element inside the environment is obtained by the specification of the point P in the middle of the corridor; obviously, it is oriented in the same direction of its length.

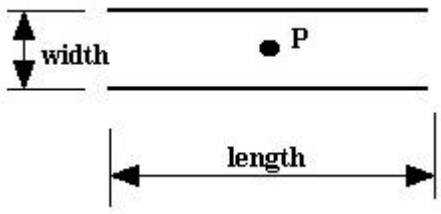

Figure 2: example a corridor

2.3 Wall

The wall member is used to represent a partition wall or any other not crossable obstacle inside the environment. Its parameters are the length, and the co-ordinates of the point P in the middle of the wall. The orientation of P is in the direction of the length of the wall.

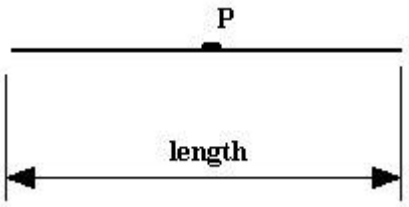

Figure 3: example of a wall

2.4 Door

The door member is used to represent a door, or any other passage in a partition wall inside the environment. Its parameters are the width of the passage and the co-ordinates of the point P in the middle of the door. The element is oriented along the perpendicular of the width.

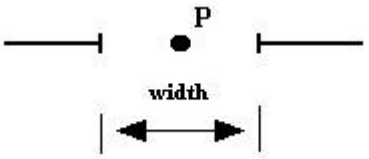

Figure 4: example of a door

2.5 Junction

The junction member is used to represent the crossing between two corridors. When a corridor meets another one, it is necessary to define some junctions, if is possible for the robot to pass through corridors crossing. In fact, if we do not introduce the junction, the robot is obstacled by the boundary of the corridors because the two corridors are in superposition between them.

The parameters of this member are its width and the co-ordinates of the point P in the middle of the junction. This point is oriented in the crossing direction (orthogonal to the width of the junction).

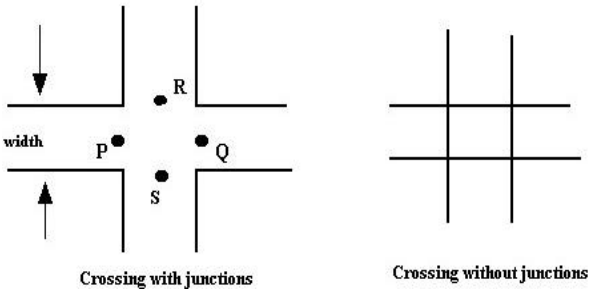

Figure 5: example of junction

We can note that in a T junction, it is necessary to define three members, as showed in figures 5; on the contrary, if the junction is cross-shaped, it is necessary to define four members, as showed in fig. 6.

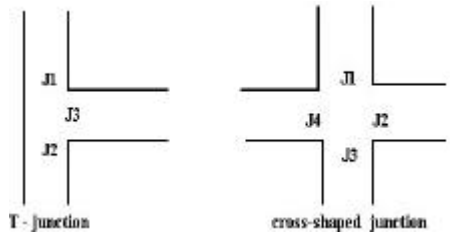

Figure 6: T junction and cross-shaped junction

2.6 Notice

The notice member is used to represent a graphical information about the neighbouring environment. A robot can use it to obtain an additional information, that can be of various kind. For example a notice can be an image that the vision agents of the robot can use to distinguish the various rooms of the environment.

The sign can also be thought as a traffic signal, positioned at a fixed height from the floor.

The information is contained into an image file, that the robot's vision agents can obtain and process. The parameters of the notice member in the EDL language are the name of the image file, the co-ordinates of the barycentre, and the height from the floor.

This last choice permits us to extend the two dimensional representation of the world obtaining a two and half dimensions model.

2.7 Implementation

To design the EDL language we used an object oriented approach. The coding phase has been performed in the C++ language.

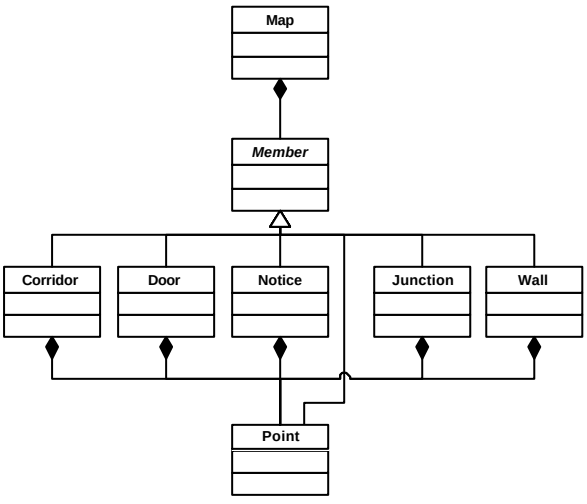

Figure 7: The EDL class diagram

Each element of the language is derived from the class *member* (see fig.7 for a class diagram of the language architecture)

The class *member* is an abstract class that represent a generic element of the map. In fact, a specific class has been defined for each element of the language inheriting it from the *member* class.

Discussing the structure of the various member in the previous sections, we have already seen that all of them use a 'point' element to specify their own position. For this reason, in fig. 7 we can see that the same members have a one to one 'aggregate' relationship with the 'point' class.

The map of the environment is represented through the 'map' class that can aggregate more instances of the EDL members. This aggregation gives place to the collection of elements that we can find in the environment described by the map.

The map of an environment is represented in a plain text file that contains the dimensions of the motion field and the list of all the members of the map. For each member we have to specify its characteristic values too.

The member's list must be write in a prefixed format, according to the syntax rules that constitute the grammar of the language.

In fig. 8 we can see the EDL description of the environment graphically depicted in fig. 9.

We can notice that the grammar of the description is very simple and, consequently, it can be modified by hand also in the case of very complex environments

```
LENGTH 900.0
WIDTH  500.0

1 CORRIDOR:
      x= 450.0
      y= 50.0
      theta= 0.0
      long= 900.0
      wide= 100.0

2 CORRIDOR:
      x= 450.0
      y= 250.0
      theta= 90.0
      long= 500.0
      wide= 100.0

3 JUNCTION:
      x= 400.0
      y= 50.0
      theta= 0.0
      wide= 100.0

4 JUNCTION:
      x= 500.0
      y= 50.0
      theta= 0.0
      wide= 100.0

5 JUNCTION:
      x= 450.0
      y= 100.0
      theta= 90.0
      wide= 100.0

6 DOOR:
      x= 250.0
      y= 100.0
      theta= 90.0
      wide= 100.0

7 DOOR:
      x= 500.0
      y= 350.0
      theta= 0.0
      wide= 100.0
```

Figure 8: an example of map description text file

This description file is not the file that is really processed by the simulator. In fact it should be processed by a language translator module, that produce a new file whose format is more suitable for the needs of the simulation program. This new format is characterised by the *cfg* extension.

The syntax of this *cfg* file is designed so that it is very simple for the map class constructor method to read it and to build the list of the members. The *cfg* file is composed by one row for each member of the map plus one (the first) for

the environment dimensions. Each row contains all the values of the corresponding member separated by a token .

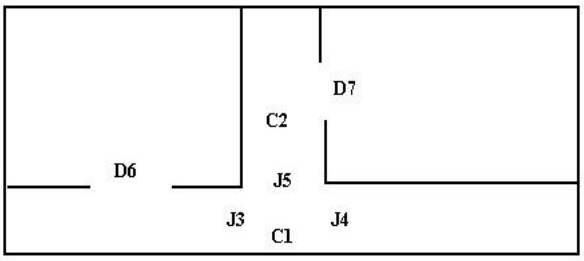

Figure 9: the map described in the text file of fig.8

Because during its navigation in the real world a robot can discover the position of the obstacles using its sensors and it cannot discriminate among all kind of members,

The constructor of the class *map*, adds an additional wall to the member's list. This wall is the border and it is used to prevent the robot from going out of the map.

3. An example

We applied the EDL language to a multi-robot simulation software built upon the Ethnos multi-agent operating system that had been developed for the RoboCup research project. It implemented a rectangular soccer field with two goals. We have modified this structure to support all the environments that can be described with EDL.

Ethnos (Expert Tribe in a Hybrid Network Operative System, [1, 2]) is a real-time programming environment. It can be used to support agent-oriented multi-robot systems and can provide advanced communication capabilities among different agents (called experts) of the same robot and among different robots. An expert in Ethnos is a concurrent agent responsible for a specific deliberative or reactive behavior. All experts are members of a tribe which are distributed in separeted villages (network computers) depending on their computational task.

From the communication perspective, Ethnos supports and optimises transparent inter-robot information exchange across different media (cable or wireless); in fact it permits the communication among different agents, through a public dashboard and a publish/subscribe method, either in the same robot or in different robots. From the runtime perspective it provides support for the real-time execution of periodic and aperiodic tasks, schedulability analysis, event handling, and resource allocation and synchronization. From the software engineering perspective, it provides support for rapid development, platform independence and software integration and re-use.

To demonstrate the use of the EDL language together with the Ethnos simulator we have chose a simple example in which a team of two robots pursues a single robot playing the role of the prey [12]. The environment is like a labyrinth, and it has been built using EDL.

In the following description we will omit the description of the prey robot that is not very interesting. We will focus our attention upon the two pursuing robots.

To develop each robot we have used a multi-agent architecture where each agent was dedicated to a specific task [11].

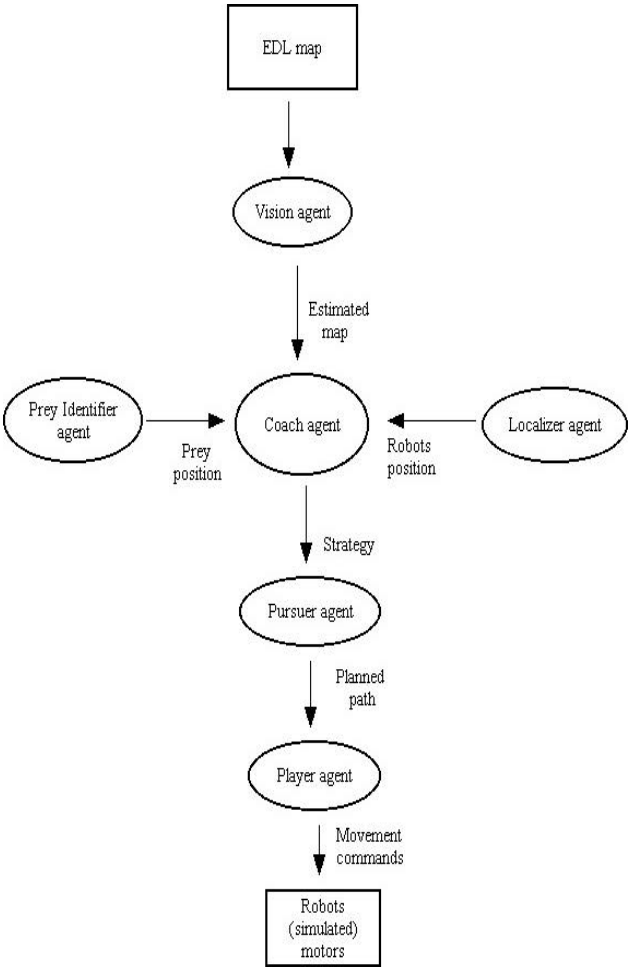

Figure 10: the architecture of our example

We have used 6 agents. Five of them are present in both the robot and the last (the Coach) is supposed to run in another computer :

- the Vision agent acquires information from the environment. It access the map and retrieve data about the elements that are near to the robot.
- the Prey Identifier agent is responsible for detecting the prey.
- the Localizer agent retrieve all informations about the position of the robots of the team.
- the Coach agent is responsible for the definition of the movement strategy of the team. At this end, it receives all necessary data from the Vision agent, the Prey Identifier agent and the Localizer agent.

- The Pursuer agent defines the path from the robot to the prey according to the planned strategy.
- The Player agent generates all movement commands for the robots simulated motors.

In fig. 10 we can see the relations among the agents that we have used.

During the simulation process, each robot moves in the environment. To 'see' the world it uses a vision agent.

This agent reads the map and returns what a real robot would see if it would be in that position (and orientation) in the real world. This is the only agent in the robot who is able to acces the map [8].

Obviously the vision agent has to be changed with an image processing agent that uses a real video acquisition device, before testing the software in a 'real' robot.

In order to better simulate the reality, the robot can only 'view' the objects that are closer to itself, and that are not covered by other ones. In fact, generally the sensors of a real robot have natural limits in their range of sight.

To co-ordinate and synchronise the actions of the pursuing robots, we have defined the Coach agent. It has not sensors to see the world or the prey and so it receives all the information from the other agents. From the Vision agent, it receives information about the portion of the environment map that each robot can see; from the Localizer agent, it receives informations about the position of each robot; from the Prey Identifier agent, it receives informations about the estimated position of the prey, if one robot can 'see' it [6].

By processing these informations, the Coach chooses the best strategy for the team. One of the options is to surround the prey with the two robots of the team. Obviously, if the prey has not been localised yet, a further exploration of the environment has to be planned.

Because the Coach agent is an high level agent, it doesn't communicate directly with the robot team. On the contrary, it sends the strategy to the Pursuer agent that implements the directives.

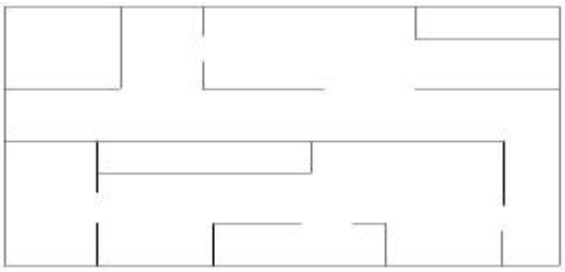

Figure 11: the labyrinth we have used for our example

It plans the paths for the robots of the team [13, 14]. The result of its elaboration is the path that each robot has to follow[7]. This information is sent to the Player agent of each robot that directly controls the movement.

The labyrinth we have used in the example, is represented in fig. 11.

The following pictures present a sequence of images captured from the simulator. Note that only a part of the labyrinth is represented because of the chosen magnification factor. Two robots pursue a prey. The prey has been localised and according to its position and the position of the two robots, the surrounding strategy has been chosen.

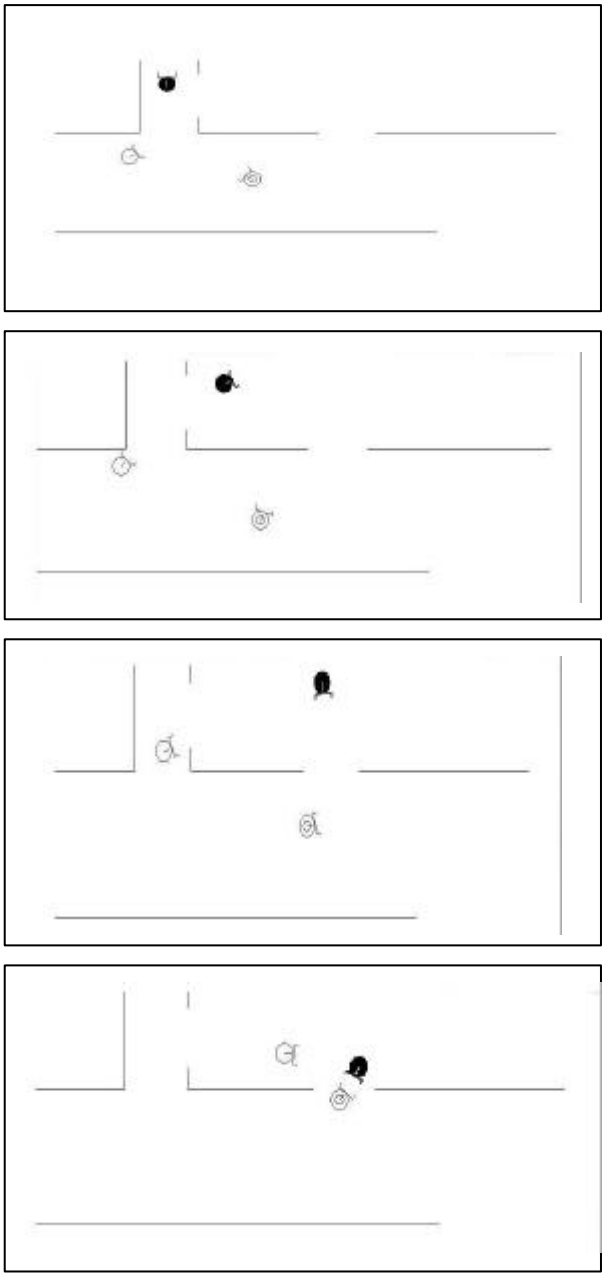

Figure 12: A sequence of simulation

In figure 12 it is possible to see how EDL is used to represent the map of the environment.

Only some elements of the environment are shown.

In fact, the robots know only the part of the map that they have previously explored or that they have in sight.

The black lines represent the walls of the map. The prey is the dark robot.

4. Conclusion

Using the EDL language we may build all kinds of structured environments including corridors, rooms, partition walls, doors, and so on.

In these hypothesis, it is possible to represent domestic environments, offices, warehouses, and so on.

This language has been thought to be used in conjunction with many different multi-robot simulators.

It has proved very useful to support the design of many different examples during the test of some agent-oriented design process with UML [4, 5].

We are now developing a fipa-os [9, 10] simulator in Java that will use EDL to define the map of the simulation environment.

A possible extension of the EDL towards the representation of three dimensional environments has been planned for the very next future.

5. Acknowledgments

The authors would like to thank the AIR lab of the Polytechnic of Milan, and particularly A. Bonarini, and M. Matteucci that have provided us with the original multi-robot software simulator.

6. References

[1] M. Piaggio and R. Zaccaria, "An Efficient Cognitive Architecture for Service Robots", The Journal of Intelligent Systems, Freund & Pettman, Endholmes Hall, England, Vol 9:2, March-April 1999.

[2] M. Piaggio and R. Zaccaria, "Distributing a Robotic System on a Network - the ETHNOS Approach", Advanced Robotics, The International Journal of the Robotics Society of Japan, Vol. 12, N.8, VSP Publisher, Aprile1998.

[3] K. G. Konolige: "Saphira - Robot Control System", SRI International, Artificial Intelligence Center. On line at: http://www.ai.sri.com/~konolige/saphira

[4] Chella, M. Cossentino, U. Lo Faso, "Designing agent-based systems with UML", Proc. of ISRA'2000, Monterrey, Mexico. Nov. 2000.

[5] Chella, M. Cossentino, U. Lo Faso, "Applying UML use case diagrams to agents representation", Proc. of AI*IA 2000 Conf., Milano, Sept. 2000.

[6] J. Rossman, "Virtual reality as a control and supervision tool for autonomous systems", in Proc. Int. Conf. Intell. Auton. Syst., Karlsruhe, Germany, Mar. 1995.

[7] Y. Wakita, S. Hirai and K. Machida, "Intelligent monitoring system for limited communication path: Telerobotic task execution over internet", in Proc. IEEE/RSJ IROS'95, Pittsburgh, PA, Aug. 1995.

[8] G. Tascini, L. Regini, A. Montesanto, P. Puliti, "Interazione Sistema Autonomo – Mondo in Ambiente Virtuale", Proc. of AI*IA 2000 Conf., Milano, Sept.

[9] M. Makelainen, "Open Source FIPA Agent Platform", Merito Forum, March 2000.

[10] S. Posland, P. Buckle, R. Hadingham, "The FIPA-OS Agent Platform: Open Source for Open Standards", Manchester, UK, April 2000.

[11] R. C. Arkin, T. Bulch, "Cooperative Multiagent Robotic Systems", 1997.

[12] T. Bulch, R. C. Arkin, "Behavior-Based Formation Control for Multirobot Teams", 1999.

[13] A. Tsoularis, C. Kambhampati, "On-line Planning for Collision Avoidance on the Nominal Path", Journal of Intelligent and Robotic Systems, Kluwer Academic Publishers, Netherlands, October 1997.

[14] E. Freund, H. Hoyer, "Collision Avoidance in multi-robot systems", in H. Hanafusa and H. Inoue (eds), Robotics Research, The Second Int. Symp., MIT Press, Cambridge, MA, 1985, pp. 135-146.