# APPLICATION OF A NATURAL LANGUAGE INTERFACE TO THE TELEOPERATION OF A MOBILE ROBOT

**J.M. González Romano*, J. Gómez Ortega** and E.F. Camacho****

**Dpto. Lenguajes y Sistemas Informáticos. Facultad de Informática. Universidad de Sevilla. Avda. Reina Mercedes s/n. 41012 Sevilla.*
***Dpto. Ing. de Sistemas y Automática. Escuela Superior de Ingenieros. Universidad de Sevilla. Camino de los Descubrimientos s/n. 41092 Sevilla.*

Abstract: This paper describes the application of a natural language interface to the teleoperation of a mobile robot. Natural language communication with robots is a major goal, since it allows for non expert people to communicate with robots in his or her own language. This communication has to be flexible enough to allow the user to control the robot with a minimum knowledge about its details. In order to do this, the user must be able to perform simple operations as well as high level tasks which involve multiple elements of the system. For this ones, an adequate representation of the knowledge about the robot and its environment will allow the creation of a plan of simple actions whose execution will result in the accomplishment of the requested task. *Copyright © 1998 IFAC*

Keywords: Telerobotics, Man-machine interfaces, Natural language.

## 1. INTRODUCTION

One important goal in the robotic field is the developing of friendly human-robot interfaces that allow inexperienced operators to deal with robots. For instance, a robotized assembly system could be better controlled by an expert in assembly systems instead of an expert in robotics. The design of such an interface is a complex task because a number of different technologies in robot control, computer vision, man-machine communication and learning fields have to be put together.

With respect to the man-machine interface, one interesting approach is the use of natural language interfaces (NLI), which allow the interaction with the robot through commands given in the own robot operator's language. NLI have been successfully used in a wide variety of fields like communication with expert systems, database access systems or operating systems. Their main advantages are that a very short training period is required and that they seem *natural* to the operator.

In the robotics field, several works can be found in the literature where NLI are used for robot control and operation. Two of the first systems, developed in the seventies, at the beginnings of the Artificial Intelligence, were ROBOT (Harris, 1997) and SHRDLU (Winograd, 1972). Both systems worked with a simulated robot. The improvement in computational facilities has lead to the development of more powerful systems which can interact in real time with real robots. Selfridge (Selfridge and Vannoy, 1986) developed a NLI for a robotized assembly system which allows the operator to hold a *conversation* with the system in order to carry out several manipulation and vision tasks like object recognition and assembly to built more complex objects. SAM (Brown, 1992) combines written and spoken language; the robot has a video camera in order to recognise the objects. Torrance (Torrance, 1994) developed a NLI to a mobile robot through which the user can command the robot to move through an environment and to memorise it.

This paper presents the application of a NLI to the teleoperation of a NOMAD 200 mobile robot. The work is focused on the communication between the robot and the human operator. The operator will be able to issue high level commands to the robot, and the NLI will decompose them in a set of lower level commands which will be executed directly by the robot navigation and control system. The operator does not need any knowledge about this robot-level commands. The information needed for this task is stored in a previously created knowledge database. Section 2 presents the main characteristics of the NLI. Section 3 describes the tasks that can be carried out by the system. Finally, section 4 shows the conclusions.

## 2. DESCRIPTION OF THE NLI

The NLI is a part of a more complex system aimed at

communicating in natural language with complex interactive systems (González, 1997b). This system uses a NLI to acquire the knowledge of a generic complex system and another NLI to operate it. The system is composed of two different parts: the acquisition module and the operation module. The first one works off-line, while the second one works on-line. The acquisition module allows the user to describe a system by introducing phrases in his own natural language, and stores the acquired knowledge in a knowledge database. By using this knowledge, the operation module allows to operate the system through simple or complex natural language commands. The accomplishment of the latter ones requires a previous planning of the actions to be carried out.

Fig. 1 shows the block diagram of the developed system, from which there exists a running prototype (González, 1991). This prototype connects to the real system through its control system, which receives the commands from the prototype and executes them over the system. It is also possible to connect the prototype to a program which simulates the behaviour of the system when the connection is not possible or for operator training purposes. The prototype has been applied to different complex systems such as an industrial process (González, 1993) or an electric network (González, 1997a, b). This paper describes its application to a mobile robot wich navigates in a structured environment.

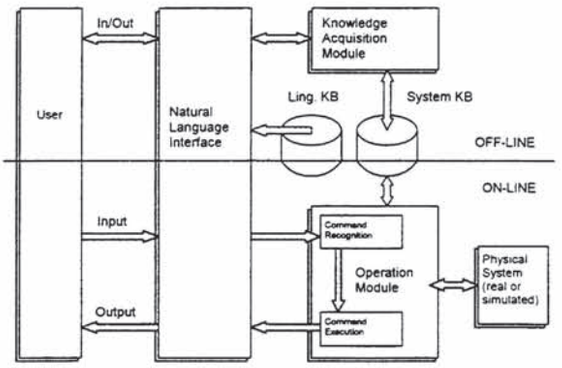


Fig. 1 Block diagram of the developed prototype

### *2.1 The Knowledge Database*

The knowledge database is an important part of the prototype. It consists of two parts: the linguistic knowledge database and the system knowledge database. The first one contains the linguistic knowledge that is necessary to analyze and extract the meaning of the user's phrases, and is composed of a dictionnary and a grammar. The second one contains all the necessary knowledge to describe the system, and can be divided into declarative and procedural knowledge. The first one concerns the different kinds of entities that can be found in the system: **classes**: (different kinds of objects present in the system), **objects** (particular instances of the classes), **connections** (topological relations between objects), **groupings** (groups of objects related to each other by their topology or their function) and **measurements**: (sensors which allow for some relevant magnitudes of the system to be known).

Procedural knowledge includes a set of functions that represent the means through which the goals of the system can be fulfilled. These functions are related to the different entities found in the system (classes, objects, groupings) or to the whole system. A function has the following components (Fig. 2):

- **Goal** ($g$): is the goal fulfilled by the function.
- **Prerequisites** ($r_i$): are conditions that the system must necessary accomplish before applying the means for the fulfillment of the goal of the function. A prerequisite can be a condition on a state or the value of a property.
- **Means** ($m_{ij}$): are the operations whose execution results in the fulfillment of the goal of the function. They can be simple actions over objects or classes, or other functions. In general there will exist some sequences of means in parallel: some means will execute at the same time, as they are independent, while others will have to do it in sequence, as every means depends on the result of another.
- **Criteria** ($c$): is the criteria whose accomplishment implies that the goal of the function has been fulfilled. It is a condition over the value of some property, and will always be true provided that the means have been correctly executed.
- **Posterior actions** ($p_{ij}$): are operations that must be executed once the goal of the function has been fulfilled, in order to leave the system in a specific state. They can be simple actions or functions.

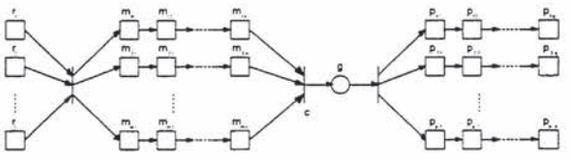

Fig. 2 Representation of a function

The execution of a function may need the previous execution of other functions that act as means of it, and may launch the execution of other functions that act as posterior actions of it. When executing a function, it will be decomposed into its constituents until there are any functions left, thus obtaining a network made of simple actions and conditions that willl be called the **actions network**.

### *2.2 Knowledge Representation*

The elected formalism for the representation of knowledge has been that of frames (Minsky, 1975). This is due to the hierarchical organization of the knowledge, at the declarative (hierarchical structure of clases) and the procedural level (hierachical structure of functions). Thus the knowledge database will be composed of class, object, grouping, measurement and function frames.

## 3. TELEOPERATION OF THE MOBILE ROBOT

The system to which the prototype has been applied is a Nomad 200 mobile robot (Nomadic, 1997) which navigates in a partially structured environment. It is composed of a base with a turret mounted on it. The base has two driving and one steering wheel, allowing forward and backwards translation movements and left and right turning. The turret is capable of rotating 360° over itself independently of the base. Fig. 3 shows a photograph of the robot. The goal of the application is to teleoperate the robot from a terminal, through which it will be given commands in order to perform certain tasks, such as walk to a named place or walk forward avoiding all the obstacles it can find along its way.

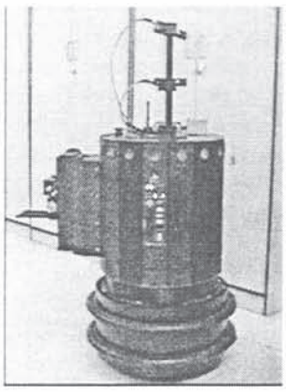

Fig. 3 Photograph of the NOMAD robot

### *3.1 System description*

The robot navigates in a partially structured environment composed of walls, furniture, doors and obstacles. Fig. 4 shows the different classes of objects. The hierarchical structure of the defined functions is shown in Fig. 5. Level 0 corresponds to simple actions over the robot. Functions begin in level 1 and increase its complexity in upper levels. As an example, let it be the level 2 function *walk_watching*. Its goal is to make the robot walk forward avoiding the obstacles it can find along its way. To do this the robot is asked to walk forward. The bumpers are checked during this movement; if a collision is detected, then the robot stops, walks backwards a little bit (*unwalk 100*), walks around the obstacle (*surround_obstacle*), and follows its way (*walk watching*). The frame for this function is shown in Table 1.

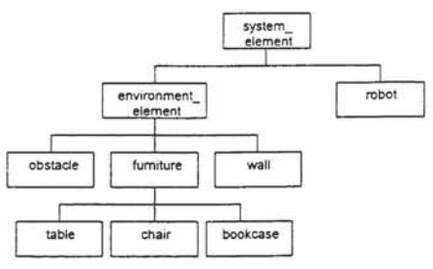


Fig. 4 Class structure of the robotic system

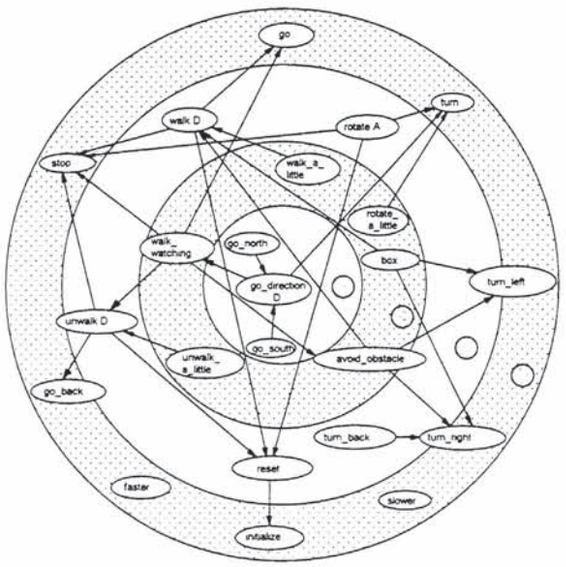


Fig. 5 Structure of the functions of the robotic system

Table 1 Frame for the function walk_watching

| | |
|---|---|
| NAME | walk_watching |
| DESCRIPTION | walk forward avoiding obstacles |
| ASOC_TYPE | class |
| ASOC_NAME | robot |
| PRE | () |
| MEANS | (go) |
| CRITERIA | bumper = 1 |
| POST | (stop, unwalk 100,<br>surround_obstacle, walk watching) |

### *3.2 Robot operation*

The Nomad 200 robot can be operated in two ways. The first one consists in executing the programs which control the robot in the robot itself, as it has its own CPU. Programs are transferred to the robot through the network, and are executed once they are inside it. The second one consists of using the control system located in another machine which is connected to the robot through an ethernet radio link. This second way is more desirable, as it allows to have a friendly development system and a graphical simulation environment which allows to test the programs before executing them on the real robot.

Since the control software of the robot is located in a different machine than the prototype is, it has to be settled a method to communicate both machines in order to send the commands to the robot and to receive its state. For this purpose the sockets have been used. A socket is a way of transferring information between processes which execute under UNIX operating system. Processes can execute in the same machine or in different machines connected through a network, as in this case, which is illustrated in Fig. 6. Communication with sockets is based on the client-server model. There are two server processes which run in the machine where the robot control software is located, and two client processes which are launched

by the prototype to send and receive information from the robot. Fig. 7 shows the communications schema.

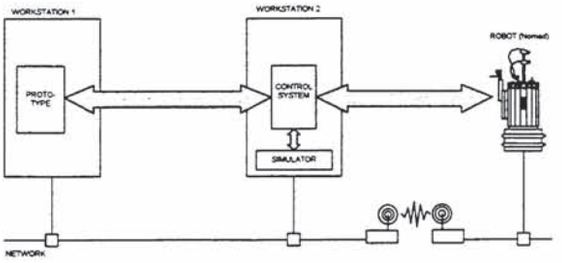


Fig. 6 Connection of the prototype to the Nomad robot

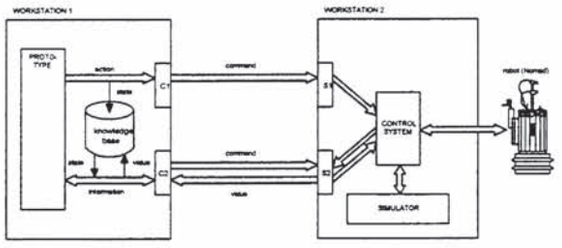


Fig. 7 Communication between prototype and robot

The two possible communication types are:

a) Execution of a command over the robot: the prototype launches a client C1 which connects to the server S1 and sends the command, which is translated and in turn sent to the robot for its execution. The client updates the system knowledge database and ends its execution.
b) Request for information about the robot: the prototype launches a client C2 which connects to the server S2 and receives the actual state of the robot, with which it updates the system knowledge database. Then it shows the user the requested information and ends it execution.

*Simple commands*. Simple commands are those which apply directly to an element or group of elements, and can be divided into two cathegories: action execution commands and information request commands. The first one includes commands with which the user requests the execution of a specific action over an object or set of objects. The following are examples of this kind of commands:

```
> go the nomad robot.
> accelerate.
> stop nomad.
> turn right.
```

Action execution commands imply the performing of a specific action over a specific object; the recognition process of these commands consists of identifying the action which has to be done and the object to which it has to be applied. Once this has been done, it should be checked whether the action can be applied to the object, and if the actual state of the object allows for the application of that action. If everything is correct, the action will be executed, and the object will be set to its new state.

Information request commands are those commands with which the user aks for a specific information about an object or a set of objects. For instance,

```
> show the position of the nomad robot.
> show its velocity.
```

The requested information corresponds to the state or property value of an object or grouping. The recognition process of these commands consists of identifying the object or grouping whose state or property is to be known and, in this latter case, the corresponding property, which must be a valid property and must have an associated measurement.

*Complex commands: actions network.* Complex commands are those that imply the execution of a plan of actions in order to fulfill a specific goal. The plan of actions is a sequence of simple actions over some objects in a predetermined order, and is generated from the knowledge stored in the system knowledge database. Given a goal, there will exist in the knowledge database at least a function which will have this goal as its goal. The frame for this function will be the starting point to generate the plan of actions which allows to fulfill the command. Thus, the different means, prerequisites and posterior actions of this frame will be analysed one by one. Each means $m_{ij}$ and posterior action $p_{ij}$ from this function can be a simple or complex command. Every prerequisite $r_i$ can be a complex command or a condition over a state or the value of a property. Simple and complex commands, as well as conditions, have a structure that is represented by a Petri Net (Silva, 1985). Each structure has at least an input place and an output place. The input place will be marked when the net is activated, whereas the marking of the output place will mean the ending of its traversal.

A simple command is represented by two places and one transition (Fig. 8). The input place represents the action to be executed over the object to which the command is applied, and will be marked when the command starts its execution. The transition represents the state to which the object is to be taken, and will be fired when the action over it has been done. At this moment the output place representing the fulfillment of the simple command will be marked.

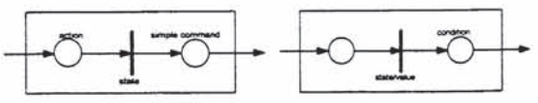


Fig. 8 Structure of a simple command and a condition

For complex commands there will exist in the knowledge database another function whose goal corresponds to that command, and that will have its own means, prerequisites, criteria and posterior actions. The network representing this function will be a subnetwork of the one corresponding to the main

function, and should in turn be expanded, should there have another complex command between its means, prerequisites and posterior actions. Its structure is shown in Fig. 9. There is an input place which will be marked when the execution of the complex command begins, and this mark will propagate automatically to all the prerequisites. The output place of the net is the goal place. When the marks reaches this place the goal will be fulfilled, in spite of the end of the posterior actions $p_{ij}$.

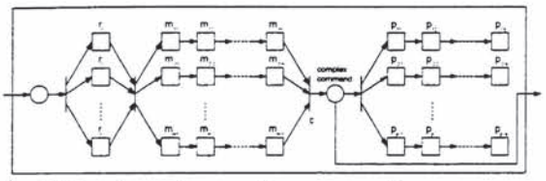

Fig. 9 Structure of a complex command

Finally, a condition type prerequisite is represented by two places and a transition, as it is shown in Fig. 8. The input place will be marked when the evaluation of the prerequisite starts. The transition represents the desired state or property value, and will be fired when it has the adequate value, resulting in the marking of the output place.

If every means, posterior action and prerequisite is successively divided until there only are simple actions and conditions, a network, the **actions network**, will be obtained. Places in this network represent direct actions over objects and transitions represent conditions over the state of the objects or the value of their properties. The initial marking corresponds to the first simple actions to be executed and the first conditions to be checked. The marks will propagate as the transitions are fired, as a result of the accomplishment of the conditions, until the mark reaches the goal place. Fig. 10 shows the generic structure of an actions network.

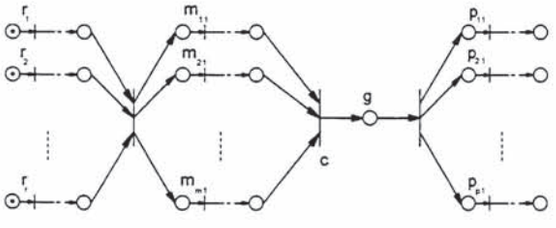


Fig. 10 Structure of an actions network

The goal will be fulfilled when the place $g$ is marked. In order for this to happen, condition $c$ should be true when all of its input places are marked. This will happen once the corresponding means $m_{ij}$ have been accomplished. The initial marking of these means will in turn depend on the marking of the ending places of the prerequisites $r_i$. On the other hand, once the goal $g$ has been fulfilled its output transition will be fired marking the input places of the posterior actions $p_{ij}$. The initial marking of the network will correspond to the initial places of the prerequisites $r_i$, and will propagate towards the goal as transitions are fired.

To summarise, in order for a goal to be fulfilled there have to be accomplished, in the first place, all the requisites. These are, thus, necessary but not sufficient conditions for the fulfillment of the goal. Once the prerequisites are accomplished, the accomplishment of every means results in the fulfillment of the goal, provided that the criteria is true. The accomplishment of the means plus the criteria is, therefore, the necessary and sufficient condition for the main goal to be fulfilled. Finally, once the goal place has been reached, the posterior actions will be executed.

### *3.3 Operation examples*

As an example of simple commands a sequence of actions over the robot is shown.

```
> where is nomad?
     NOMAD IS CURRENTLY AT LABORATORY 1
> go nomad.
     ROBOT NOMAD GOING
> turnright.
     ROBOT NOMAD TURNING
> stop.
     ROBOT NOMAD STOPPED
> turnleft.
     ROBOT NOMAD TURNING
> go.
     ROBOT NOMAD GOING
> turnright.
     ROBOT NOMAD TURNING
> stop.
     ROBOT NOMAD STOPPED
> where is nomad?
     NOMAD IS CURRENTLY AT LABORATORY 2
> what is its velocity?
     THE VALUE OF NOMAD'S VELOCITY IS 0
```

As an example of the execution of complex commands, it is shown the execution of the goal *walk_watching*, which consists in making the robot walk forward avoiding possible obstacles. Fig. 11 shows the graphical representation of the frame representing this function, which appeared in Table 1. The function frames that are necessary to execute this command are those of the function *walk_watching* itself, the functions *unwalk* and *surround_obstacle*, which appear as its posterior actions, the function *walk*, which is a means of *surround_obstacle*, and the function *reset*, which is a means of both *walk* and *unwalk* functions. Fig. 12 shows the relation between all this frames. Fig. 13 shows the simulation environment window during the execution of the complex command, and Fig. 14 represents the generated actions network.

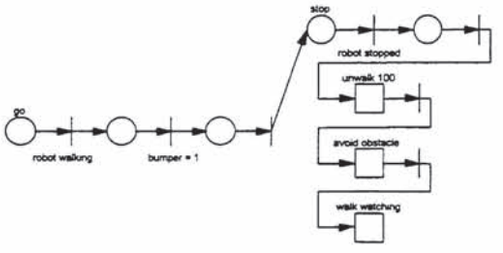


Fig. 11 Structure of the function *walk_watching*

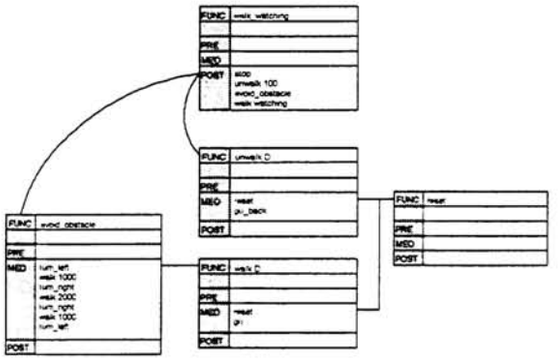

Fig. 12 Frame structure for function *walk_watching*

## 4. CONCLUSIONS

This paper has shown the application of a natural language interface to the teleoperation of a mobile robot. The adequate description of the system knowledge allows the user to perform high level operations, which are decomposed in other lower-level actions until there is a sequence of simple actions which are executed over the robot, thus having a telescopic vision of the system. The developed prototype could be applied to simulate the execution of goals. In order to do this, it suffices to simulate the behaviour of the system through software and build the plan of actions corresponding to the goal. If the plan is successfully built, the goal can be achieved and could be executed over the real system. In other case, a different plan should be created. This way it is avoided to start executing actions and reach a point in which no more actions can be executed due to a prerequisite not accomplished or an action that cannot be performed, resulting in a half-executed plan and a goal not achieved. This incomplete execution is a problem as it could prevent the goal from being fulfilled with another plan. To summarise, it has been developed a tool which can simplify the operation of complex interactive systems, as the mobile robot to which it has been applied in this paper.

## 5. REFERENCES


Brown, M.K, B.M. Buntschuh and J.G. Wilpon (1992). SAM: A Perceptive Spoken Language Understanding Robot, *IEEE Transactions on Systems, Man and Cybernetics*, vol. **22**, no. 6, pp. 1390-1402.

Harris, L.R (1977) A High Performance Natural Language Processor for Data Base Query, *ACM SIGART Newsletter*, vol. **61**.

Minsky, M (1975). A Framework for Representing Knowledge, The Psychology of Computer Vision, P.H. Winston, Ed, McGraw-Hill, pp. 211-277.

Nomadic Technologies Inc (1997). NOMAD Language Reference Manual.

González Romano, J.M, J.A. Ternero and E.F. Camacho (1991). Natural Language Interface for Process Control Centers. *Preprints 3rd IFAC International Workshop on Artificial Intelligence in Real Time Control*, Napa (California).


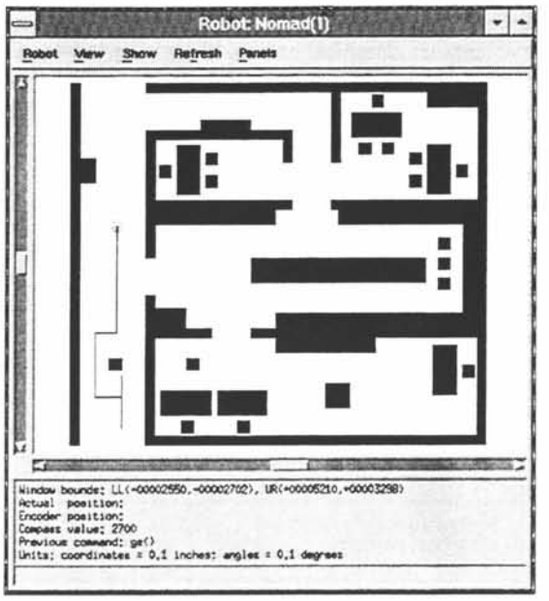


Fig. 13 Execution of the goal *walk_watching*

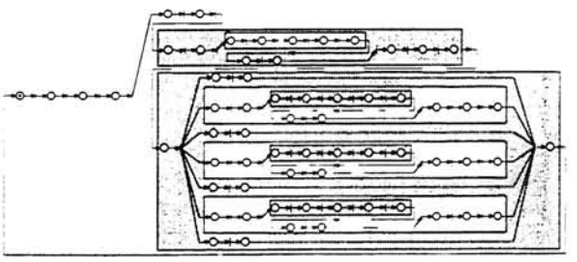

Fig. 14 Actions network for the goal *walk_watching*


González Romano, J.M. and E.F. Camacho (1993). Goal-Oriented Man Machine Interface in Control. Application to a Pilot Plant. *Preprints of the 12th IFAC World Congress*, pp. 455-458, Sydney (Australia).

González Romano, J.M. and E.F. Camacho (1997a). Utilización de un interfaz en lenguaje natural para la realización de operaciones en centros de control de redes eléctricas, *VII Conferencia de la AEPIA (CAEPIA '97)*, Málaga (Spain).

González Romano, J.M (1997b). Aplicación del lenguaje natural a la adquisición de conocimientos y operación de sistemas complejos, Doctoral dissertation, Univ. Sevilla.

Selfridge, M. and W. Vannoy (1986). A Natural Language Interface to a Robot Assembly System, *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 3, pp. 167-171.

Silva, M. (1985). *Las Redes de Petri en la Automática y la Informática*, AC.

Torrance, M.C. (1994). Natural Communication with Robots (doctoral dissertation), Massachusetts Institute of Technology.

Winograd, T. (1972). *Understanding Natural Language*, Academic Press.