# A generic natural language interface for task planning — application to a mobile robot

José Mariano González Romano[a,*], Eduardo Fernández Camacho[b], Juan Gómez Ortega[b], Miguel Toro Bonilla[a]

[a]*Departamento Lenguajes y Sistemas Informáticos, Facultad de Informática y Estadística, Universidad de Sevilla, Avda Reina Mercedes s/n, 41012 Sevilla, Spain*

[b]*Departamento Ingeniería de Sistemas y Automática, Escuela Superior de Ingenieros Industriales, Universidad de Sevilla, Camino de los Descubrimientos s/n, 41092 Sevilla, Spain*

**Abstract**

This paper presents a generic natural language interface that can be applied to the teleoperation of different kinds of complex interactive systems. Through this interface the operators can ask for simple actions or more complex tasks to be executed by the system. Complex tasks will be decomposed into simpler actions generating a network of actions whose execution will result in the accomplishment of the required task. As a practical application, the system has been applied to the teleoperation of a real mobile robot, allowing the operator to move the robot in a partially structured environment through natural language sentences.

*Keywords:* Natural language; Man–machine interfaces; Telerobotics; Mobile robots; Task planning

## 1. Introduction

Complex interactive systems are present in a great variety of fields. In these systems, sophisticated control techniques are used in combination with human operators in order to satisfy some specific goals. The system's operators act from a control room in two ways: on the one hand, they are informed of the state of the system at every moment and, on the other, they perform actions by the system in order to satisfy some goals. The difficulty of the operation depends on the complexity of the system, which comes from three main sources: the complexity of the domain, the complexity of control and the complexity of interaction. Although the first of these can hardly be reduced, it is possible to reduce the complexity of control and interaction. In the first case, this is achieved by automating the system, and in the second case designing an adequate interface for communication between the operator and the system. Nevertheless, completely automated systems are extremely problematic, because in situations where surprises can lead to critical situations and matters of life and death, responsible decisions cannot be left to such systems alone. Instead, final decisions must be left to humans, because people have the ability to master unprecedented situations. Under these circumstances, the design of friendly interfaces which decrease the complexity of interaction and allow the operators to act in a simple and quick way in every kind of situation becomes even more important.

Although it can be said that, in general, man–machine interfaces of control centers are quite user friendly, they have some drawbacks such as the need for an extensive training period for operators and the great quantity of information, some of it irrelevant, they tend to present to the operator. Natural language interfaces can be used in control centers to help the operator to manipulate the system by entering commands in his own language. A natural language interface (NLI) requires a short training period for the operator (just the time to learn the grammatic rule s and the vocabulary) and gives him the flexibility to focus attention on specific system elements or consider wider parts of the system, modifying the

* Corresponding author. Tel.: + 34-95-455-2768; fax: + 34-95-455-7139.

*E-mail addresses:* mariano@lsi.us.es (J. M. G. Romano), eduardo@cartuja.us.es (E. F. Camacho), juango@cartuja.us.es (J. G. Ortega), mtoro@lsi.us.es (M. T. Bonilla).

degree of abstraction in which the system is viewed as a function of its current state. Natural language can also be used to generate explanations about the causes of a given situation: the operator can be given an explanation of what is happening in his own language. Natural language can be used alone or in combination with other means of interaction such as hyper-text or graphics to create a more flexible, multimodal interface. In conclusion, natural language can be, in general, of great interest as a tool to simplify the work of control center operators.

This paper presents a natural language-based interface for man–machine communication in control centers. This interface allows the operator to perform the usual control center tasks during normal system operation, which consist of issuing commands to the system elements or asking for information on its state. Commands can be classified into two groups: simple and complex commands. The first ones correspond to actions that apply directly to system elements, and are executed provided that the state of the elements allows for their execution. Complex commands pursue the accomplishment of a certain goal and involve, in general, several system elements. The accomplishment of the goal implies the execution of a set of actions in a predetermined order by these elements. This will be called a *plan of action*.

The plan of action associated with a certain goal can be known a priori or can be unknown. In the first case, it is a part of the system's knowledge database and can be applied immediately. In the second case it has to be automatically generated. In order to do this the system has to be adequately described, so that the functional relations between its elements are known. Thus, given a goal, it will be possible to find a plan of action that allows for its accomplishment. The generated plan will be appended to the system's knowledge database so it can be reused later.

The plan of action is executed whilst it is being constructed, so it is not necessary to construct it before its execution with complex planning techniques. This is especially important in real-time systems, such as those considered in this paper.

The paper is organized as follows: Section 2 analyses the state of the art in man–machine interaction in complex systems. Section 3 identifies all the relevant information involved in a generic complex system and selects a formalism for its representation. In Section 4, the main properties of a natural language interface designed to operate a complex system are studied, focusing on the process of generating plans of action to accomplish complex goals. Section 5 presents the system that has been developed and that has been applied to a mobile robot, which navigates in a partially structured environment. Section 6 shows the experimental results obtained. Finally, Section 7 shows the conclusions and some future work.

## 2. Related work

Natural language interfaces have been applied to a great variety of fields such as database access, interfacing with expert systems, knowledge acquisition, communication with robots, human–computer dialog or tutorial systems.

In the field of database access, NLI have proved themselves to be a very useful tool for simplifying access to large amounts of information. As an example a geographical information system (Kasturi, Fernández, Amlani and Feng, 1989) or a multimedia database (Stock, 1994) can be cited.

In the field of interaction with complex systems, there are some works on the application of a NLI to communication with robots. Several researchers have developed robotic systems that are controlled by means of natural language sentences. The use of natural language allows operations to be performed on robots and their environment and specify goals to be satisfied without the need for previous knowledge about robots or computers.

Amongst the first systems two classical cases must be cited, ROBOT (Harris, 1977) and SHRDLU (Winograd, 1972). Both of them allow for robot interaction, although they deal with a simulated robot rather than a real one. SHRDLU allows the user to command a robot that moves in a world made of blocks. With the improvement of computational power it becomes possible to build more powerful systems, which allow real-time interaction with real robots. Selfridge and Vannoy (1986) present a NLI to a robot assembly system which allows the user to talk with the system in order to do manipulation and visual tasks, such as recognizing the components that are present in an image and putting them together to make a more complex component. The knowledge base consists of a set of if–then rules. When these rules are insufficient to make a component, the system asks the user for an explanation of the plan to make it. It uses a learning-by-being-told technique instead of deducing new knowledge from the knowledge base.

Other NLI systems such as SAM (Brown, Buntschuh & Wilpon, 1992) combine the use of written and spoken natural language. The user can use a telephone or a keyboard to command the robot. The robot is a manipulator arm with six axes and a grip, and has a mounted camera with which it can recognize the objects placed on a surface. The user can introduce a description of the objects. This description will later be used to command the robot to perform actions over them.

Torrance (1994) uses a NLI to command a robot to move and memorize its environment. The user can control the robot with commands. A plan is a sequence of actions to get the robot between two named places. Thus, it is a very specific type of plan, valid for robot navigation but not for other kinds of complex systems. This is a common feature in the majority of systems designed for

natural language communication with robots. The one proposed in this paper is designed to work with generic complex systems instead, although the examples shown correspond to a particular application.

Hwang, Cheng and Watterberg (1996) describe a learning user interface (LUI) for robot task-planning and programming. The robot operator interacts with the LUI with commands in a natural-language-like form. LUI has a basic set of commands and can learn new commands given a demonstration of how to execute the command in terms of a sequence of known commands. In contrast, the approach proposed here is to acquire all the knowledge in a previous phase through the same natural language interface.

PRS (Ingrand, Chatila, Alami & Robert, 1996) is a set of tools and methods used to represent and execute plans and procedures. PRS has been adapted to autonomous robots and has been used in different implementations as a high level supervision and control language for mobile robots. PRS has a library of plans, each describing a particular sequence of actions and tests that may be performed to achieve a given goal. The main difference between PRS and the system presented in this paper is that PRS uses a procedural representation, while the proposed system uses frames to represent goals. In both cases the representation preserves the control information (i.e. the sequence of actions and tests) embedded in the plans, while keeping some declarative aspects.

Knoll, Hildebrandt and Zhang (1997) present a system that integrates natural language with vision to control an assembly cell consisting of two cooperating robots and a variety of sensors. The authors conclude that in a limited domain like assembly the use of natural language in robotic systems may be of great help even to inexperienced users.

In a review of the state of the art in human–robot communication Klingspor, Demiris and Kaiser (1997) conclude that it is necessary to develop techniques that allow untrained users to make efficient and safe use of robots. The users do not want to operate the robot. Instead, they want to use it to accomplish some task. For this reason they need a friendly interface that allows them to intuitively instruct the robot and that provides them feedback in a way that they can understand.

Previous work on this subject carried out by the authors led to the development of a system that was applied to the operation of a pilot plant (González, Ternero & Camacho, 1992; González & Camacho, 1993). Through the NLI, the plant operator was able to perform simple actions such as open or close valves, as well as complex goals such as filling a tank up to a given level or maintaining the temperature of a tank. This system was later applied to the operation of an electrical network (González & Camacho, 1997), allowing the control center operators to remotely perform actions over the network elements: open/close switches, feed loads, regulate voltages, etc. The behavior of the network was simulated by software. The work presented in this paper shows the application of the system to a new, real complex system, which has allowed the system to be tested in a real environment. Moreover, the mobile robot is very different from the other two systems, so several modifications had to be made to the system, which has been enriched with new features and is now able to deal with a larger variety of systems.

The main difference between the system presented here and the rest of the systems discussed in this section is its generic nature. The developed system has not been specifically designed for human–robot communication but for human–computer communication. For this reason it can be applied to many different complex systems and not just to a specific kind of system.

## 3. The knowledge database

The knowledge database contains all the system's knowledge, which can be divided into two parts: declarative knowledge and procedural knowledge. The first one contains the description of the different entities that can be distinguished in the domain being studied. The second one deals with the behavior of the system, and includes procedures or functions that allow the goals of the system to be achieved.

### *3.1. Declarative knowledge*

Declarative knowledge describes the static structure of the system, and is structured as follows:

- *Classes:* different categories of objects that can appear in the domain.
- *Objects:* specific instances of a class.
- *Connections:* topological relations between objects.
- *Groupings:* set of objects related to each other for functional or topological reasons.
- *Measures:* sensors that give access to the values of some relevant magnitudes of the domain.

*Classes* are characterized by having some *properties* and some associated *actions*, and have a hierarchical structure. Higher level classes are more generic and lower level ones are more specific. There is an inheritance mechanism: the properties of a class are inherited by all of its *subclasses*. Classes can be *simple* or *compound*. The latter are those whose objects are a set of objects with a certain structure. The introduction of compound classes allows certain repetitive structures of the system to be defined as classes, thus simplifying its description. Another characteristic of a class is its *connectors*: objects connect to each other through connection elements that are attached to its connectors.

An *object* is a particular instance of a class. Objects from a class have a *value* for each one of the class

properties. In addition, an object can have *exclusive properties*, defined by the object itself rather than by the class it belongs to, that are not shared by the rest of the objects of the class. Objects can be in different *states* throughout their lives, each one corresponding to a different behavior of the object. The state of an object is variable and can be changed either by acting directly upon it or as a result of the behavior of other related objects.

Objects connect to each other through special kinds of objects called *connections*. A connection starts in a connector of an object and finishes in a connector of another object. If an object is connected to several objects there will be several connections which could start from the same or different connectors. A connection can either be orientated or not. In the first case, a first (or input) end and a second (or output) end of the connection exist. It is then possible to refer to the input and output objects of the connection.

Objects can be grouped into *groupings* according to functional or topological criteria. In the first case, all the objects share a common function. In the second case, the objects are just placed next to one another. The objects composing the grouping maintain their own personality, but in certain cases the grouping can be taken as a whole, being a special object with its own properties and a behavior that will be a function of those of the component objects.

In general, there are a lot of measurable magnitudes in a system, corresponding to the different properties of objects and groupings. However, not all of them will usually be accessible to the operator, only those with an associated sensor will. A *measure* is defined by the object or grouping to which it is associated, the property of the object or grouping it measures and the value of the property.

### 3.2. Procedural knowledge

Declarative knowledge allows the state of the system at a given moment and the state changes of every object in the system to be described. Another kind of knowledge to be represented is that of procedures, i.e., conditional sequences of actions that can be run to achieve given goals. Procedural knowledge includes a set of functions that represent the means by 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:

- *Goal:* is the goal fulfilled by the function.
- *Prerequisites:* are conditions that the system must necessarily accomplish before applying the means for the fulfillment of the goal of the function. They can be conditions as to the state of a class or object or the value of a property of a class, object or grouping, or can be another function.
- *Means:* are the operations whose execution results in the fulfillment of the goal of the function. They can be simple actions on objects or classes, or other functions. In general, several sequences of means in parallel will exist: some means will be executed at the same time, as they are independent, while others will have to be executed in sequence, as every means depends on the result of another.
- *Criteria:* is the criteria whose accomplishment implies that the goal of the function has been fulfilled. It is a condition on the value of some property of a class, object or grouping, and will always be true provided that the means have been correctly executed.
- *Posterior actions:* 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 on objects or classes, or other functions. As in the case of the means, there can also be several sequences of parallel posterior actions.

The execution of a function may need the previous execution of other functions that act as its means, 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 no functions left, thus obtaining a network made of simple actions and conditions that will be called the *action network*.

Functions can have parameters that modify their behavior. These parameters can appear either in the means or in the prerequisites, the criteria or the posterior actions. These will not be completely defined until the function is executed and the parameter has a definite value. It will then fill in the corresponding slot in the function structure and the function will be executed.

### 3.3. Knowledge representation

All the knowledge has to be identified, acquired and represented in some way in order to have a complete description of the system. After carefully analyzing the kind of knowledge to be represented, the elected formalism for its representation has been that of *frames* (Minsky, 1975). This is due to the hierarchical organization of their knowledge at the declarative level (structure of classes) and at the procedural level (the means, prerequisites and posterior actions of a function can be other functions that can, in turn, be other functions, and so on). Thus, the knowledge database will be composed of class, object, connection, grouping, measurement and function frames. With all these types of frames it is possible to represent all the declarative and procedural knowledge of a system in a suitable and efficient way for its manipulation. Part of this knowledge is fixed, whereas another part is variable. The state of objects, the

values of the properties of objects and groupings, and the values of the measures can change during the system operation, while the rest of the knowledge remains the same.

## 4. Operation of complex systems

In this section the communication process between the operator and the system to be controlled is studied. The operator is usually placed in a control center, from which he acts upon the system and is constantly kept informed of its state. There are two kinds of communication: one between the operator and the control system, and the other between the control system and the physical system. The communication studied here is of the first type, which is done through an interface. This interface reads the operator's commands, interprets them and generates control actions that the control system will send to the physical system. The results produced by these actions on the system will be sent back to the operator through the same interface.

The proposal of this paper is to use a NLI between the operator and the control system that allows for interaction in both senses: the operator will introduce sentences representing commands to the physical system, which will be analyzed and interpreted by the interface, resulting in the generation of commands in a control language that will be sent to the physical system. These commands will be executed, resulting in an updating of the system's state. Finally, the control system will communicate to the operator the new state and this will close the cycle. If needed, the interface will request the operator to introduce additional information in order to complete missing data, setting up a dialog with him. Fig. 1 shows this process.

The following section shows the characteristics of the NLI used for the recognition of the operator's commands, which will be called the recognition interface.

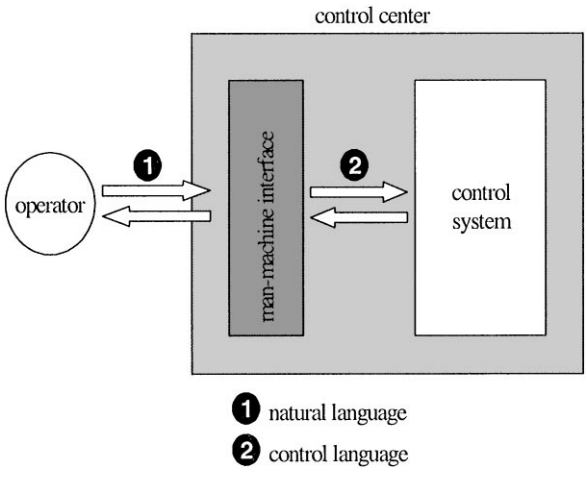


Fig. 1. Interaction between the operator and the control system.

### 4.1. Natural language interface

The two essential components of a NLI are the vocabulary and the grammar. The vocabulary of the recognition interface includes all the words the operator can use to give his commands. It is composed of generic words, valid for every kind of system, and words that are specific to the system, such as the names of its particular entities. The grammar is adapted to every possible phrase the operator can use to give his commands. Fig. 2 shows the representation in terms of a recursive transition network (RTN), of a part of this grammar. In contrast to the vocabulary, the grammar is fixed and cannot be modified by the operator.

### 4.2. Man–machine dialog

The dialog between the operator and the system through the recognition interface are as follows (Fig. 3):

1. The operator introduces a phrase representing a command to the system from the keyboard.
2. The phrase is analyzed for correctness, using the linguistic knowledge database (composed of the vocabulary and the grammar).
3. The phrase is interpreted, identifying the kind of operation requested by the operator and the element or elements of the system affected by the operation.
4. The requested operation and the specified elements are checked for validity.
5. The control language command is generated and sent to the physical system through the control system.
6. The knowledge database is updated and the operator is informed of the result of the operation.

The interpreter transforms the natural language sentence in an internal representation of its meaning after having identified the requested operation and the involved

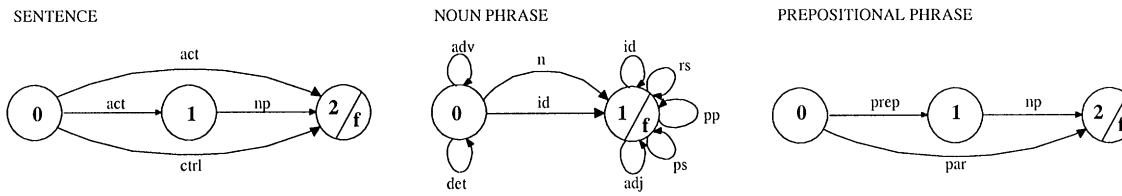


Fig. 2. A fragment of the recognition interface grammar.

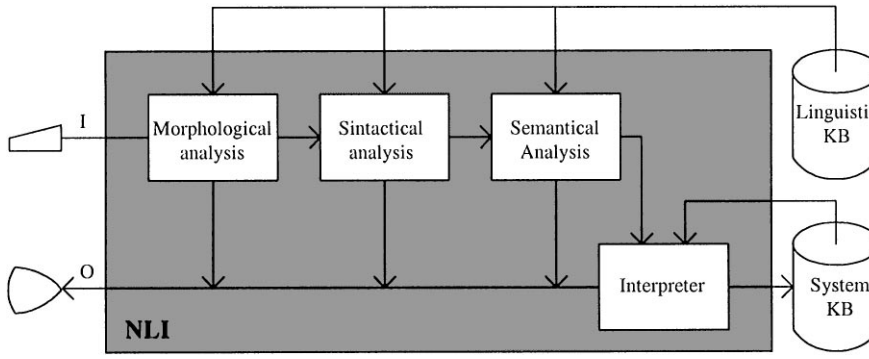


Fig. 3. Man–machine dialog.

elements. As a result of this interpretation, two possibilities can take place:

- Both the requested operation and the specified elements are valid: the operation is executed, the operator is presented its result, and the knowledge database is updated.
- The operation is not valid or the specified elements do not exist: the operator is informed of the impossibility of executing the operation.

In the second case, the operator can be asked either to give additional information or to solve possible conflicts, through a dialogue that will finish when the information has been precisely specified.

### *4.3. Types of commands*

In this section the two types of commands, simple and complex, the operator can give are studied in more detail.

#### *4.3.1. Simple commands*

Simple commands are those which apply directly to an element or group of elements, and can be divided into two categories: 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 recognition process of these commands consists of identifying the action that has to be made and the object to which it has to be applied. Once this has been done, whether the action can be applied to the object, and if the actual state of the object allows for the application of that action should be checked. If everything is correct, the action will be executed and the object will be set to its new state. The following sentences, taken from the operation of an electrical network, are examples of these kind of commands:

> open the cell C1S1.
> open the bus tie cell of generator G5.
> open the cell that joins G1 with the busbar BC1S1.
> open the cells associated to line L1.
> open all the cells connected to the main busbar of substation sub1.
> open all the cells from substation sub1.

Information request commands are those commands with which the user asks for specific information about an object or a set of objects. 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 must be a valid property and must have an associated measure. The following are examples of information request commands:

> show the position of the nomad robot.
> show its velocity.
> show the voltage of the main busbar of substation sub1.
> show the state of all the cells of sub1.
> show the state of all the cells connected to the main busbar BC1S1.
> show the active power of all the loads of the net.

In both types of commands the object to which the command is applied can be directly specified or can be specified in an implicit way, by means of its topological or functional relation to other objects, its pertinence to a grouping or its location. In every case, the corresponding frames will have to be retrieved from the knowledge database in order to identify the object. On the other hand, the object can be unspecified, in which case it will be obtained from the previous sentences of the dialog. Finally, commands can be applied to multiple objects: all the objects of a named class, all the objects belonging to a grouping, all the objects related to a same object or all the objects placed at a given location. In this case it will be necessary to identify all of the objects, and it may happen that the command can only be applied to some of them, since not all the objects will, in general, have the same state.

#### *4.3.2. Complex commands*

Complex commands are those that imply the execution of a sequence of simple actions to some objects in a predetermined order, what will be called a plan of action, in order to fulfill a specific goal. To achieve this,

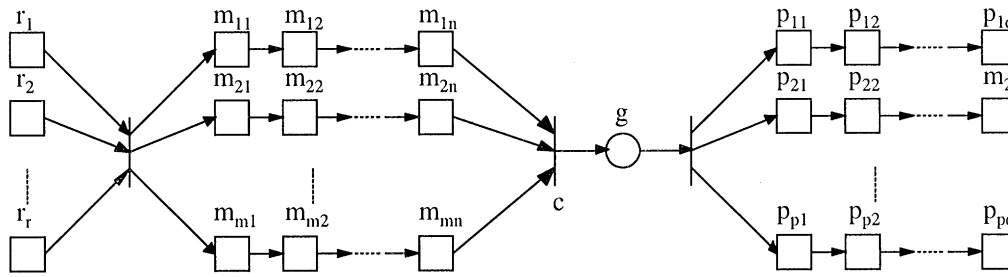


Fig. 4. Representation of a function frame.

the goal has to be decomposed into simpler subgoals until there are only simple actions that can be applied to specific objects. The execution of these actions will result in the accomplishment of the goal. The necessary knowledge to generate a plan of actions is stored in the function frames. These frames contain all the information about the operations that can be performed on the system, each one described in term of the others. Thus, a frame representing a complex function can be expressed in terms of other less complex functions that will also have their corresponding frames. The process of generating a plan of actions consists then of exploring frames that will call up other frames and so on, until frames that only have direct actions over objects are found. The execution of the simple action network built this way will result in the accomplishment of the goal.

### *4.4. Generation of a plan of action*

In order to understand the process of goal decomposition it is necessary to recall the concept of the function frame. This frame describes a function that can be associated to a class, an object, a grouping or the whole system. Fig. 4 shows the graphical representation of a function frame.

Given a goal, at least one function will exist in the knowledge database which will have this goal as its goal $g$. The frame for this function will be the starting point to generate the plan of action that allows the command to be fulfilled. Thus, the different means, prerequisites and posterior actions of this frame will be analyzed 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 represented by a Petri net. 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. 5). The input place represents the action to be executed on the object to which the command is applied, and will be marked when the command starts up its execution. The transition represents the state to which the object is to be taken, and will be fired when the action on it has been made. At this moment the output place representing the fulfillment of the simple command will be marked.

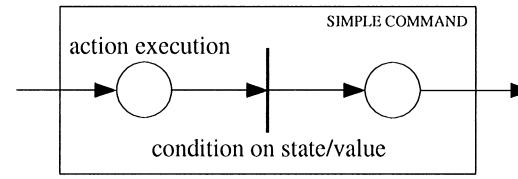


Fig. 5. Structure of a simple command.

For complex commands another function will exist in the knowledge database 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 if it were to have another complex command between its means, prerequisites and posterior actions. Its structure is shown in Fig. 6. There is an input place that will be marked when the execution of the complex command begins, and this mark will automatically propagate to all of its prerequisites. The output place of the net is the goal place. When the marks reach this place the goal will be fulfilled.

Finally, a condition-type prerequisite is represented by two places and a transition, as is shown in Fig. 7. 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 once 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 are only simple actions and conditions left, a network, the *action network*, will be obtained. Places in this network represent actions being executed on objects while transitions represent conditions on 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

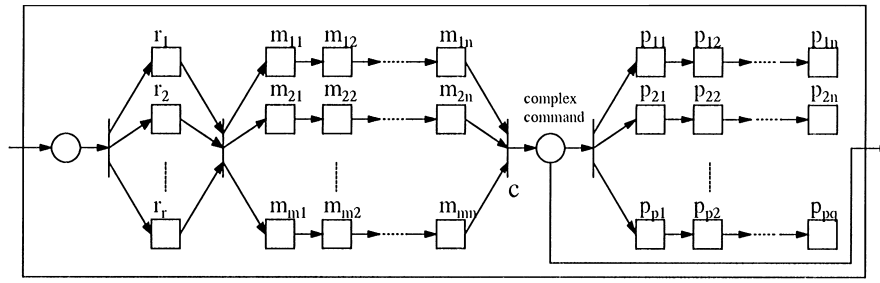


Fig. 6. Structure of a complex command.

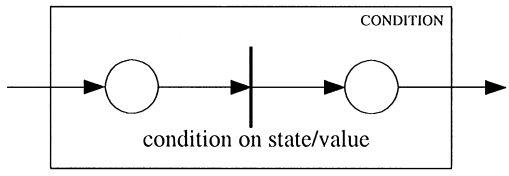


Fig. 7. Structure of a condition.

be checked. The marks will propagate as the transitions are fired, as a result of the conditions being fulfilled, until the mark reaches the goal place. Fig. 8 shows the generic structure of an action network.

The action network can formally be defined as a 6-tuple $\mathbf{R} = < P, T, c, o, \alpha, \beta >$ where:

- $P$ is a finite and non-null set of places that represents the states of the objects. These states are associated to actions that are applied to the objects or conditions that are evaluated on them.
- $T$ is a finite and non-null set of transitions that represents certain conditions on the states or property values of the objects and groupings of the system.
- $c$ is a transition that represents the accomplishment criteria of the function goal.
- $o$ is a place that represents the goal to be fulfilled. When this place is marked, the goal will have been fulfilled.
- $P \cap T = \emptyset$, i.e. places and transitions are disjoint sets.
- $\alpha : P \times T \rightarrow \mathbf{N}$ is the *previous incidence function*, which specifies the input places of the transitions. A place $p$ is an *input place* of a transition $t$ if an orientated arc from $p$ to $t$ exists, i.e. $\alpha(p, t) \neq 0$.
- $\beta : P \times T \rightarrow \mathbf{N}$ is the *posterior incidence function*, which specifies the output places of the transitions. A place $p$ is an *output place* of a transition $t$ if an orientated arc from $t$ to $p$ exists, i.e. $\beta(t, p) \neq 0$.

A marking $M$ of the network $\mathbf{R}$ is an application from $P$ in $\mathbf{N}$ that gives the number of marks that exist at every place of $P$.

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

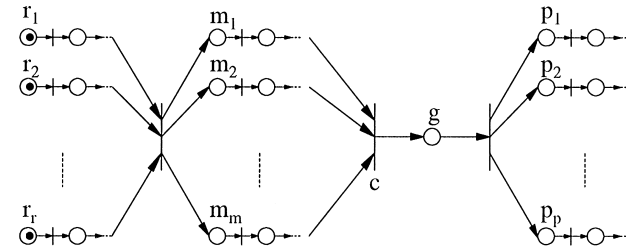


Fig. 8. Structure of an action network.

the marking of the ending places of the prerequisites $r_i$. On the other hand, once 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 the transitions are fired.

Every generated network is added to the knowledge database so that it can be used later if needed. Thus, if when generating a new action network one of the subgoals is a goal whose action network has already been generated, it will be taken from the knowledge database where it was stored and will be included in the action network that is being generated. This way avoids generating the same network again.

## 5. Application to a mobile robot

In order to practically apply all the ideas presented in this paper, a system has been developed. This system is composed of two different parts: the acquisition module and the operation module. The first one, described in González (1997), works off-line, while the second works on-line. Both of them share a common knowledge database. The acquisition module allows the knowledge of different types of systems to be introduced and stored in different knowledge bases. By using this knowledge, the operation module allows the system to be operated. The block diagram of the developed system is shown in Fig. 9.

The developed system is connected to the real system through its corresponding control system, which will

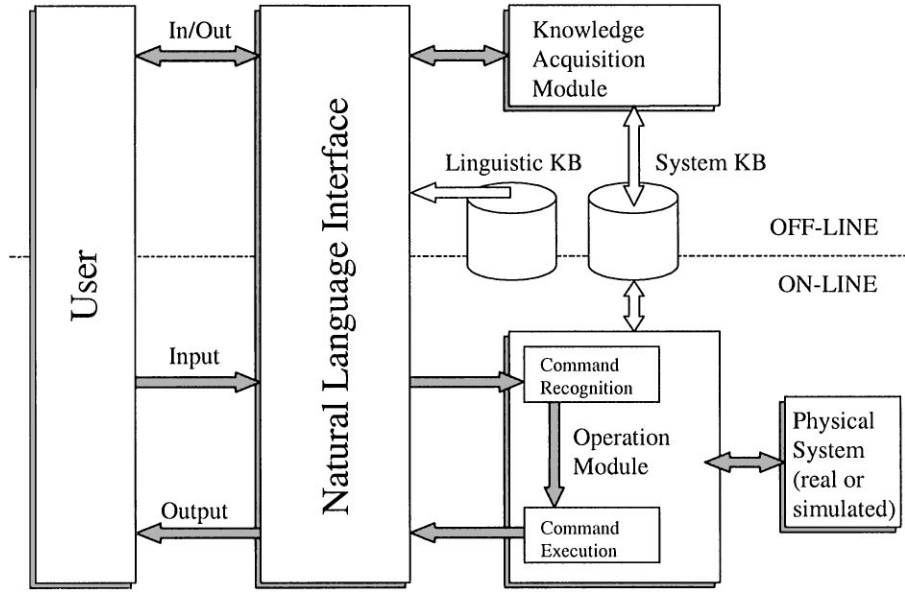


Fig. 9. Block diagram of the developed system.

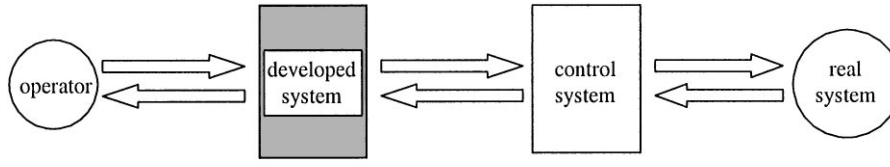


Fig. 10. Connection of the developed system to the real system.

receive the commands and will execute them over it (Fig. 10). It is also possible to connect the developed system to a program that simulates the behavior of the real system if this one is not available or for operation tests or operator training purposes.

This system has been applied to several complex systems with different characteristics: a pilot plant (González et al., 1992; González & Camacho, 1993) an electrical network (González & Camacho, 1997) and a mobile robot that navigates in a partially structured environment (González, Gómez & Camacho, 1998). In the first two applications the system behavior has been simulated by software. In the mobile robot application, which is presented in this paper, the system has been connected to the real robot. A hierarchy of functions has been developed that allows the operation of the robot from a lower level represented by simple movements (go, turn, etc.) to a higher level represented by more complex operations (follow wall, avoid obstacle, etc.). In this way, the robot can easily be teleoperated from a terminal when moving in a partially structured environment.

### *5.1. System description*

The robot, a Nomad 200 mobile robot (Nomadic Technologies Inc, 1997), is composed of a base with

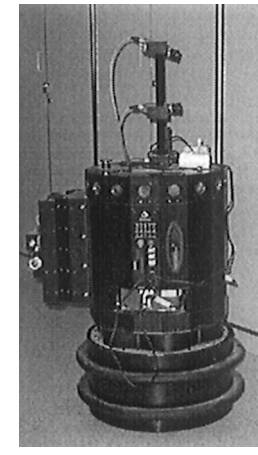

Fig. 11. Photograph of the NOMAD 200 robot.

a turret mounted on it. The base has two driving and one steering wheels, allowing forward and backwards displacement movements and left and right turning. The turret is capable of rotating 360° over itself independently of the base. The robot is equipped with a structured light-based laser range sensor, ultrasounds and a CCD camera. Fig. 11 shows a photograph of the robot, while Fig. 12 shows a schematic representation of its environment.

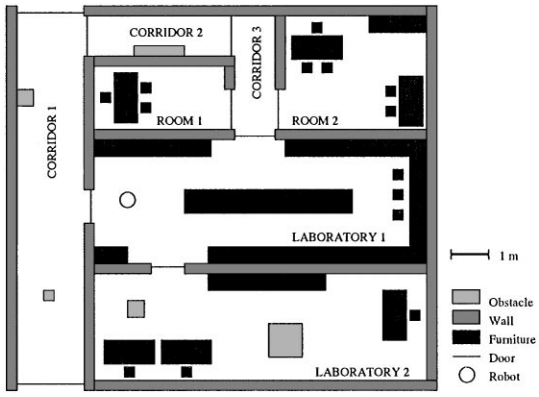


Fig. 12. The robot environment.

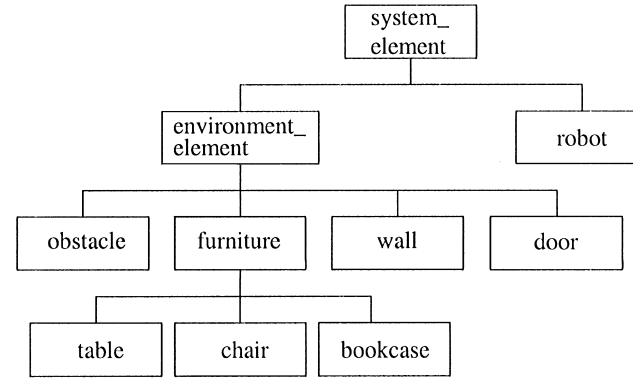


Fig. 13. Class structure of the robotic system.

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 may find along its way. The robot navigates in a partially structured environment composed of walls, furniture, doors and obstacles and the operator uses the information provided by the camera for the teleoperation of the robot. Fig. 13 shows the different classes of objects. As for connection classes, there exists a class *door* with a superclass *connection_element*.

Every room of the environment in which the robot moves is a grouping, and is made up of objects of the types *table*, *chair* and *bookcase*. Rooms connect to each other through objects of class *door*.

The hierarchical structure of some of the defined functions is shown in Fig. 14. Level 0 corresponds to simple actions over the robot. Functions begin at level 1 and increase their complexity in upper levels. For example, level 2 function *avoid_obstacle*, which consist of avoiding an obstacle placed in front of the robot by going round it, depends on level 1 function *walk_D* (move the robot forward a fixed distance *D*) and level 0 functions *turnright* and *turnleft*.

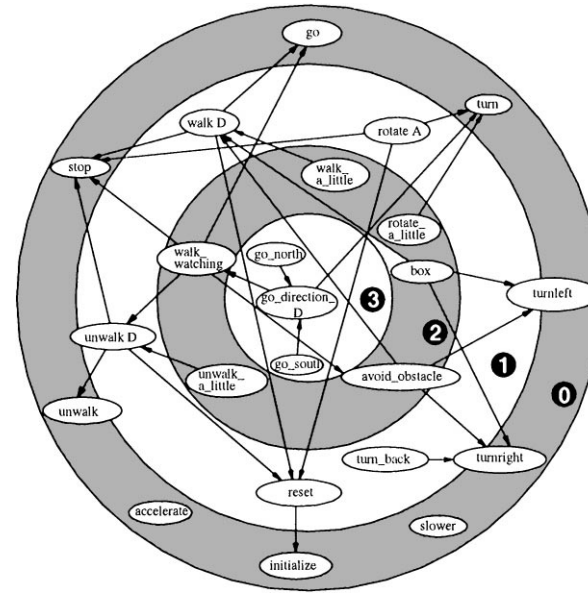


Fig. 14. Structure of the robotic system functions.

### 5.2. Robot operation

The Nomad 200 robot can be operated in two ways. The first one consists of 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 for a user friendly development system and a graphical simulation environment that allows the programs to be tested before executing them on the real robot. Fig. 15 shows the network connections between the developed system (located in workstation 1), the control system (located in workstation 2) and the real robot.

Since the control software of the robot is located in a different machine than the system, a method has to be chosen that allows communication between both machines in order that commands can be sent to the robot and its state can be received. For this purpose the UNIX sockets have been used. There are two server processes which run in the machine where the robot control software is located, and two client processes that are launched by the system to send and receive information from the robot. Fig. 16 shows a schema of the communication between these processes.

The two possible communication types are:

(a) Execution of a command on the robot: the system 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

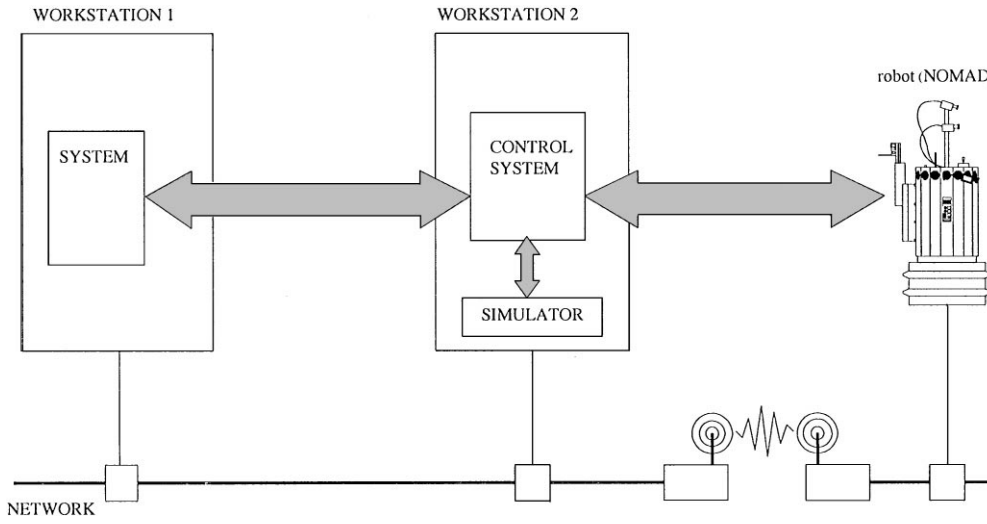


Fig. 15. Connection between the system and the real robot.

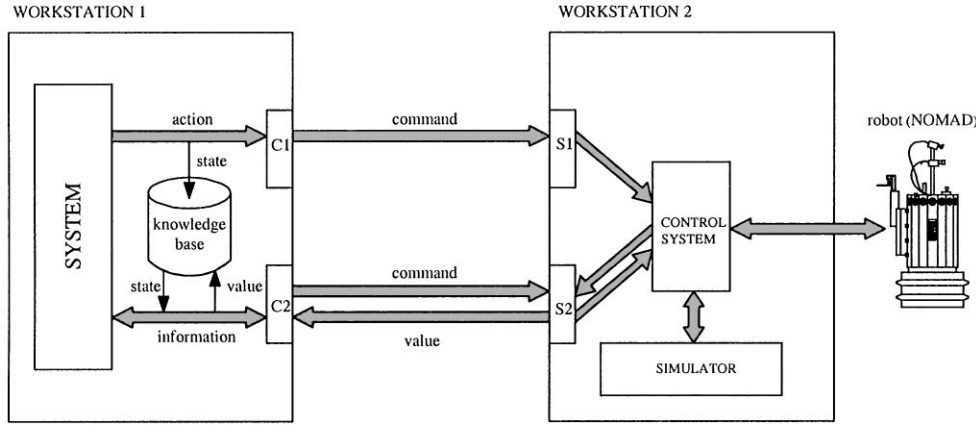


Fig. 16. Communication between the developed system and the robot.

updates the system knowledge database and ends its execution.

(b) Request for information about the robot: the system 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. It shows the user the requested information and ends its execution.

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

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

### 5.3. *Example of a complex command*

As an example of the execution of complex commands the execution of the goal *walk_watching*, which consists of making the robot walk forward avoiding possible obstacles, is shown (Fig. 17). The function frames that are necessary to execute this command are those of the function *walk_watching* itself, the function *avoid_obstacle*, which appears as its posterior action, the function *walk*, which is a means of *avoid_obstacle*, and the function *reset*, which is a means of the *walk* function. Fig. 18 shows the relations between all these frames. Note that the function *walk_watching* has a recursive definition. It just asks the

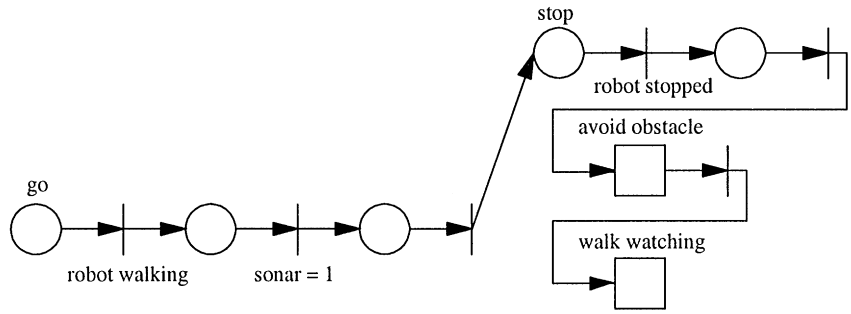


Fig. 17. Structure of the function *walk_watching*.

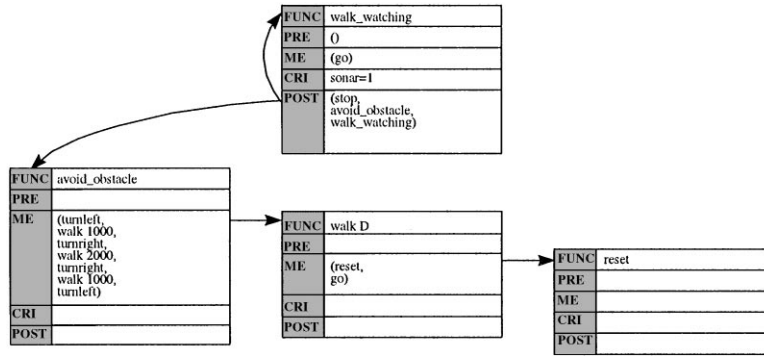


Fig. 18. Frame structure for the function *walk_watching*.

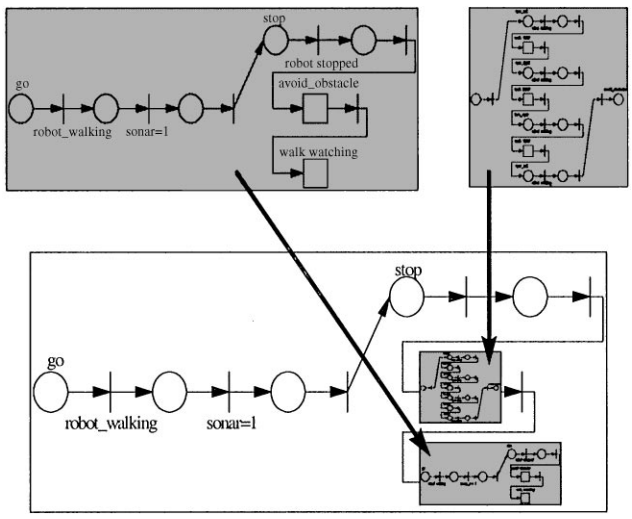


Fig. 19. Building process of the action network for the function *walk_watching*.

robot to wander around until it gets blocked. This definition could be changed without problems; for instance, once the robot has rounded the obstacle it could be asked to walk a fixed distance and then stop. In any case, if a new command is sent to the robot the movement will end, as every time a command is given the previous one finishes (except for turns that can coexist with translational movements).

Fig. 19 shows the building process of the action network. First, the knowledge base is checked in search of a function that fulfills the goal *walk_watching*. Once the function has been found and its frame has been obtained, every prerequisite, means and posterior action of the function is analyzed in search of new goals. In the example, two subgoals are found: *avoid_obstacle* and *walk_watching* itself. For every found subgoal, the knowledge base is searched again to find a function that fulfills this goal. The prerequisites, means and posterior actions of these functions are in turn analyzed for new subgoals. In the example, *avoid_obstacle* has as a means the subgoal *walk*. The process continues until there are no subgoals left. The final result is shown in Fig. 20, where the network corresponding to the function *reset* has been placed into the network corresponding to the function *walk*, this in turn has been placed into the network corresponding to the function *avoid_obstacle*, and this in turn has been placed into the original network.

When the action network is executed the corresponding actions will be sent to the robot one after another until the goal is fulfilled. If the path is blocked the goal will fail, the robot will stop and the system will give an explanation of the problem. In general, if some prerequisite or means cannot be accomplished the goal will fail, causing the system to present the user with an explanation of the reason for this failure. If

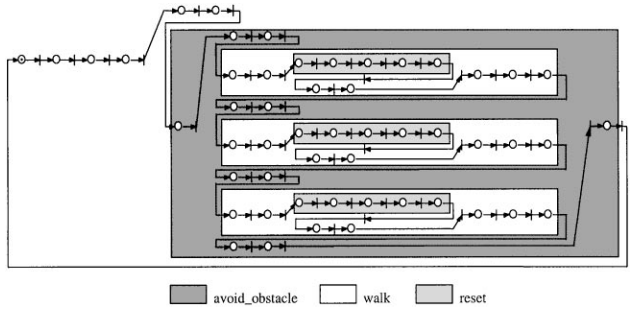


Fig. 20. Action network for the goal *walk_watching*.

there are other ways to fulfill the goal they will be tried in turn until one of them succeeds or until they all fail.

## 6. Experimental results

The system has been practically tested with the synchro-drive-type NOMAD 200 robot. Two experiments have been carried out in a typical partially structured indoor environment (see Fig. 12). Several series of simple and complex commands have been executed over the robot in this environment.

As is well known, one of the problems that arise in mobile robot navigation is the estimation of the robot position and orientation (posture estimation). This problem is especially important in a teleoperated system such as the one presented in this work, as the decisions and commands given to the robot by the operator are based on its estimated posture. In this work a sensor fusion-based posture estimation system has been used. This system makes an estimation based on the odometry and a match between the map obtained by distance measurements given by the laser range sensor and a map of the environment previously stored in memory. Such estimation is made every 7 m travelled by the robot or each time a turning command is executed (commands *turn* and *rotate*).

### 6.1. Experiment 1

The goal of the first experiment is to guide the robot from a starting point (1) to a goal point (12) travelling a total distance of about 27 m and avoiding all possible obstacles. Fig. 21 shows the environment window used by the operator as an interface for the execution of the series of commands, which is listed in Fig. 22. The figure shows the points at which the commands are given to the robot. There are simple (*go*, *turnright*) and complex (*rotate*, *estimate posture*, *walk_watching*) commands. Every time a new command is given the previous one finishes, except for turns that can coexist with translational movements. In this way, when the robot is moving and a turning command is given, a circular movement is obtained (see points 6, 9 and 10 in the figure). The series ends with the complex command *walk_watching* whose action network was discussed in the previous section, followed by a *stop* command. The posture is automatically estimated every time the robot turns. In the figure, the prime numbers correspond to the corrected robot positions. An

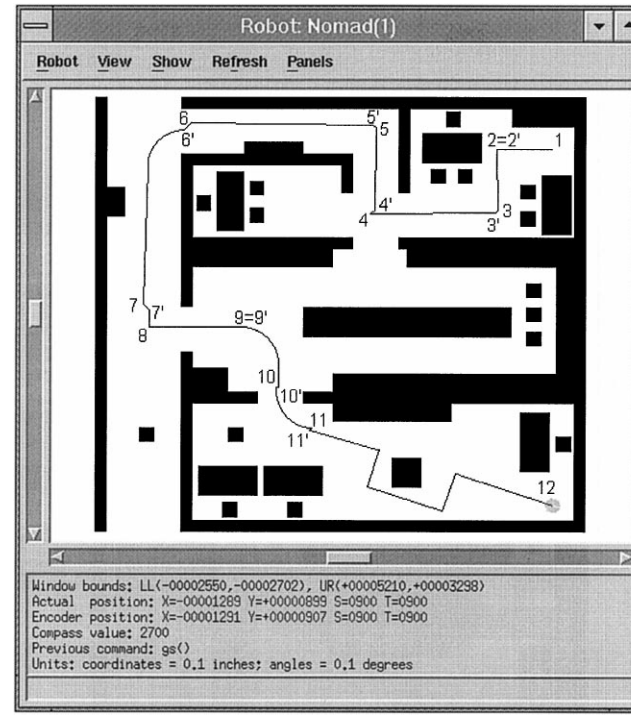


Fig. 21. Execution of experiment 1.

```
1.  go nomad.
2.  stop.
    turnleft.
    go.
3.  stop.
    turnright.
    go.
4.  stop.
    turnright.
    go.
5.  stop.
    turnleft.
    go.
6.  turnleft.
7.  stop.
    estimate posture.
7' walk a little.
8.  turnleft.
    go.
9.  turnright.
10. rotate 60 to the left.
11. walk watching.
12. stop.
```

Fig. 22. Series of commands for experiment 1.

| FUNC | goto <object> |
|---|---|
| PRE | () |
| ME | (face <object>,<br>walk_watching <object>) |
| CRI | pos=pos(object) |
| POST | (stop,<br>estimate posture,<br>correct posture) |

| FUNC | walk_watching <object> |
|---|---|
| PRE | () |
| ME | (go) |
| CRI | sonar(front)=1 |
| POST | (avoid_obstacle_left,<br>goto <object>) |

| FUNC | avoid_obstacle_left |
|---|---|
| PRE | (sonar(left)=0) |
| ME | (turnleft,<br>go) |
| CRI | sonar(right)=0 |
| POST | () |

Fig. 23. Function frames for experiment 2.

additional posture estimation command is given by the operator at point 7′, before entering laboratory 1. The operator then decides to correct the position with the command *walk_a_little* to avoid collision with the door.

### 6.2. Experiment 2

The second experiment shows how the system can be extended in order to cope with new goals. In particular, a new goal consisting of guiding the robot to a named place has been considered. Some new functions have been defined for this purpose, the most important being the following:

- *goto* $<$ *object* $>$, a function that makes the robot face an object and then makes it walk towards it avoiding all possible obstacles along the way. The function ends when the robot reaches the position where the object is located. Then the robot stops and estimates its posture, correcting it if necessary.
- *avoid_obstacle_left*, a function which avoids an obstacle located in the front of the robot by turning the robot to the left and then moving it in that direction until the obstacle has been avoided. A similar function *avoid_obstacle_right* can also be defined so that the operator can use the most appropriate one at each moment.
- *walk_watching* $<$ *object* $>$, a new version of function *walk_watching* with an object as a parameter. The robot walks until an obstacle is found. Then the obstacle is avoided by the left and the function *goto* is called with the object as its parameter.

Fig. 23 shows the frames for these new functions. Fig. 24 shows the new experiment results obtained when executing the commands listed in Fig. 25, whose goal is to take the robot from laboratory 2 to room 2.

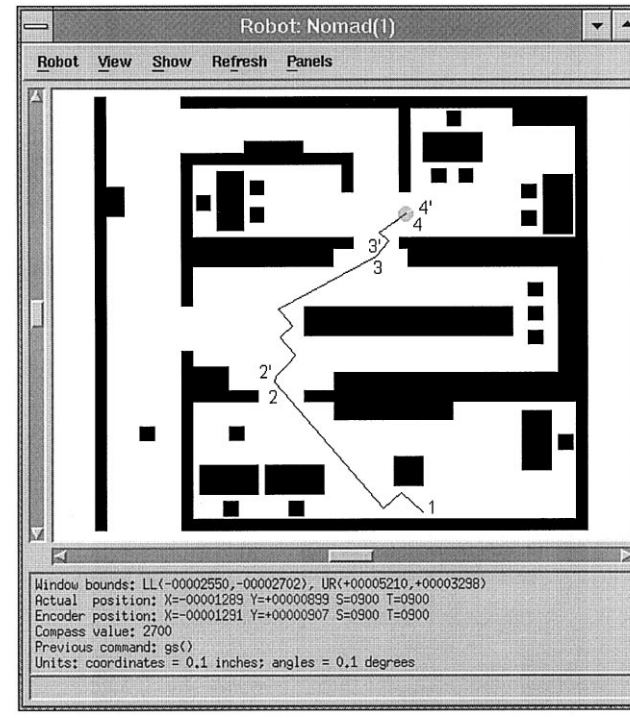


Fig. 24. Execution of experiment 2.

```
1. goto the door of laboratory2.
2. goto the main door of laboratory1.
3. goto the door of room2.
```

Fig. 25. Series of commands for experiment 2.

## 7. Conclusions and future work

In this paper a structure for the operation of complex interactive systems based on the use of natural language processing techniques has been proposed. These techniques can simplify work with such systems and allow for its use in a comfortable and natural way by non-experts.

A natural language interface has been designed that has been used for operating a system from its control center. This interface has a grammar and a vocabulary that covers almost every kind of dialog that can take place in the system operation. The interface allows for natural language communication between the operator and the system, setting up a series of dialogs with which the control center operator can perform every kind of action on the system. These actions can be simple actions over the system components, as well as more complex goals that imply the generation and execution of a sequence of simple actions on some objects of the system in a predetermined order.

In order to execute a goal a procedure has been established that generates a plan of simple actions that accomplishes the goal from the information stored in the

knowledge database. This plan is added to the knowledge database so it can be reused if the same goal, or another goal from which this one is a subgoal, is executed. Special attention has been paid to allow for system manipulation through high level user defined tasks. These tasks are defined one after the other in a telescopic way so that the operator has the possibility of using the level of abstraction he considers most convenient at every moment.

A system has been developed that has been applied to several complex systems of great interest. In particular, the application to a mobile robot that navigates in a partially structured environment has been presented. A description of the robotic system is given, its knowledge database has been generated and some typical operations were performed. The behavior of the robotic system was obtained by connecting the system to the real robot. The developed system could be applied to simulate the execution of goals. In order to do this, it is sufficient to simulate the behavior of the system through software and construct the plan of action corresponding to the goal. If the plan is successfully constructed the goal can be achieved and could be executed on the real system, otherwise a different plan should be created. This method avoids actions being initiated and then reaching a point at which no more actions can be executed due to an unaccomplished prerequisite or an action that cannot be performed, resulting in a half-executed plan and an unachieved goal. This incomplete execution is a problem as it could prevent the goal from being fulfilled with another plan.

The contribution of this paper is the development of a generic NLI that can be applied to different kinds of complex systems. The NLI allows the user to first describe the system and then to perform operations on it, including the definition and execution of high-level tasks. The system can be used to simulate the execution of tasks, which is of great interest for operator training purposes. To show the benefits of the system it has been applied to the teleoperation of a real mobile robot, allowing the user to move the robot in a partially structured environment through natural language sentences.

As future research related to this work, the following lines can be pointed out:

- Integration of natural language with other complementary interaction types, such as hyper-text, graphics or menus, in a multimodal interface that gives the user the possibility of using the type of interaction he considers the most adequate to the kind of operation he wants to perform at all times. For instance, the system can present the user with shorter sentences, hyper-linked between themselves and possibly with term definitions, instead of longer and harder to read sentences.
- Increase the goal types. In addition to the goals considered here, those of reaching a certain system state, two other possible kinds of goals exist: to prevent or to maintain a state.
- Choice amongst several action networks for fulfilling a goal. It will be necessary to specify an optimization criteria in order to select the most adequate network.

## References


Brown, M. K., Buntschuh, B. M., & Wilpon, J. G. (1992). SAM: A perceptive spoken language understanding robot. *IEEE Transactions on Systems, Man and Cybernetics*, *22*(6), 1390–1402.

González Romano, J. M., Ternero, J. A., & Camacho, E. F. (1992). Natural language interface in control. Application to a pilot plant. In A. Ollero, & E. F. Camacho, *Proceedings of the IFAC symposium on intelligent components and instruments for control applications (SICICA'92)* (pp. 199–204). Oxford: Pergamon Press.

González Romano, J. M., & Camacho, E. F. (1993). Goal-oriented man machine interface in control. Application to a pilot plant. *Preprints of the 12th IFAC world congress*, Sydney, Australia (pp. 455–458).

González Romano, J. M. (1997). *Application of natural language to knowledge acquisition and operation of complex systems*. Doctoral Dissertation, University of Seville.

González Romano, J. M., & Camacho, E. F. (1997). Use of a natural language interface to make operations in electrical networks control centers. *VII congress of the AEPIA (CAEPIA'97)*, Málaga, Spain.

González Romano, J. M., Gómez Ortega, J., & Camacho, E. F. (1998). Application of a natural language interface to the teleoperation of a mobile robot. *IFAC workshop on intelligent components for vehicles (ICV'98)*, Seville, Spain.

Harris, L. R. (1977). ROBOT: a high performance natural language processor for data base query. *ACM SIGART Newsletter*, *61*, 39–40.

Hwang, Y. K., Cheng, P. C., & Watterberg, P. A. (1996). Interactive task planning through natural language. *Proceedings of the IEEE international conference on robotics and automation*, Minneapolis, MN, USA (pp. 24–29).

Ingrand, F. F., Chatila, R., & Alami, R., & Robert, F. (1996). PRS: A high level supervision and control language for autonomous mobile robots. *Proceedings of the international conference on robotics and automation*, Minneapolis, MN (pp. 43–49).

Kasturi, R., Fernández, R., Amlani, M. L., & Feng, W. (1989). Map data processing in geographic information systems. *Computer*, *22*(12), 10–21.

Klingspor, V., Demiris, J., & Kaiser, M. (1997). Human–robot communication and machine learning. *Applied Artificial Intelligence*, *11*(7–8), 719–746.

Knoll, A., Hildebrandt, B., & Zhang, J. (1997). Instructing cooperating assembly robots through situated dialogues in natural language. *Proceedings of the IEEE international conference on robotics and automation*, Alburquerque, NM (pp. 888–894).

Minsky, M. (1975). A framework for representing knowledge. In P. H. Winston, *The psychology of computer vision* (pp. 211–277). New York: McGraw-Hill.

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

Selfridge, M., & Vannoy, W. (1986). A natural language interface to a robot assembly system. *IEEE Journal of Robotics and Automation*, *RA-2*(3), 167–171.

Stock, O. (1994). Natural language in multimodal human–computer interfaces. *IEEE Expert*, *9*(2), 40–44.

Torrance, M. C. (1994). *Natural communication with robots*. Doctoral Dissertation, Massachusetts Institute of Technology.

Winograd, T. (1972). *Understanding natural language*. New York: Academic Press.