11th IEEE International Conference on Methods and Models in Automation and Robotics MMAR'2005, 29 August – 1 September 2005, Międzyzdroje, Poland. (on CD) ISBN-83-60140-85-5

SYSTEMATIC APPROACH TO THE DESIGN OF ROBOT PROGRAMMING FRAMEWORKS

CEZARY ZIELIŃSKI[†]

[†]Institute of Control and Computation Engineering, Warsaw University of Technology, ul. Nowowiejska 15/19, 00–665 Warsaw, Poland, C.Zielinski@ia.pw.edu.pl

Abstract. The paper presents a transition function based formalism for specifying robot programming frameworks. It deals with systems consisting of multiple embodied agents (e.g., robots), influencing the environment through their effectors and gathering information from the environment through their sensors. The presented examples pertain to behavioral and hybrid behavioral-deliberative agents, but are not limited only to that.

Key Words. robot programming frameworks, multi-agent systems

1. INTRODUCTION

This research[*] stemmed from the necessity of quickly producing controllers for diversity of robots executing significantly differing tasks. The controllers for the following systems have been designed using the methodology and tools described in this paper:

- High-stiffness serial-parallel robot having a large work-space (fig. 1(a)) [16, 19, 14, 31], thus well suited to milling and polishing tasks [15],
- Direct-drive robot without joint limits (fig. 1(b)) [18, 20], hence applicable to fast transfer of objects,
- Two IRp-6 robot system (fig. 1(c)) acting as a two-handed manipulation system [32, 25].

How fast a new controller can be produced depends on the quantity of readily available software that can be reused from former projects and the extent to which it can be modified. This paper deals with the problem of designing universal control software for systems:

- composed of multiple embodied (having a material body) agents with initially unknown:
  - number and type of effectors,
  - number and type of receptors,
  - communication means between the agents,
  - information processing capabilities,

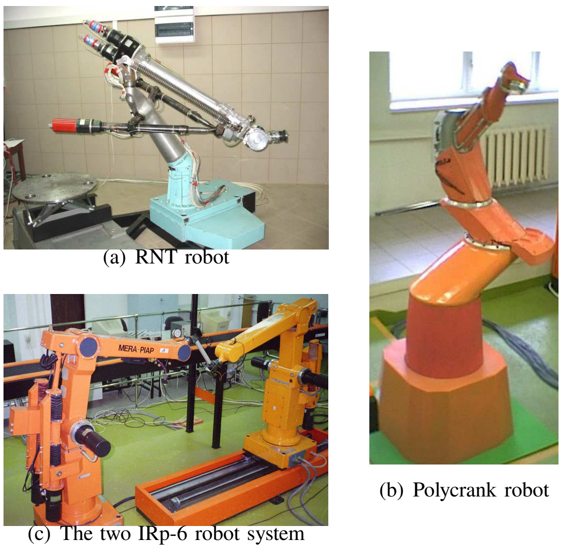
(a) RNT robot
(b) Polycrank robot
(c) The two IRp-6 robot system

Fig. 1. The considered robot systems

[*] This work was supported by Polish Ministry of Science and Information Technology grant: 4 T11A 003 25.

- with initially unspecified task to be executed by both the system and separate agents.

As the variability of systems fulfilling such a general specification is enormous the tool for the construction of such systems must be very versatile. The answer to this problem, that is provided by computer science, is a tool named a programming framework [13]. Programming frameworks are application generators with the following components:

- library of software modules (building blocks out of which the system is constructed),
- a method for designing new modules that can be appended to the above mentioned library,
- a pattern according to which these modules can be assembled into a complete system jointly exerting control over it and realizing the task at hand.

Robot programming frameworks have been developed for quite a while (e.g., `RCCL` [11], `KALI` [10, 3], `PASRO` [4, 5], `RORC` [26], `MRROC` [26], `MRROC++` [27, 28], G^{en}oM [9, 1], `DCA` [21], `TCA` [24], `TDL` [23], `Generis` [17], `SmartSoft` [22]). Currently efforts are being made to produce public domain generic robot control software (e.g., the OROCOS project [8]). Taking into account all of activities associated with producing generic robot software, focusing on formal specification of programming frameworks for systems composed of heterogeneous embodied agents seems to be an important research area. Rational design of a programming framework requires some specification tool. Here we shall concentrate on specific programming frameworks, namely the ones dealing with multi-agent systems, where the agents have material bodies capable of influencing the state of physical environment. The proposed meta-tool has been verified by presenting several specifications of agents utilizing behavioral, fuzzy and deliberative approaches to their control [30]. This formalism is a generalization of the approach used to define and implement the `MRROC++` [28, 29] robot programming framework. `MRROC++` based systems have either a hierarchical structure or a structure composed of independent entities. In both cases no direct interaction between the agents was possible, besides the indirect interactions through sensing or upper layers of hierarchy. This formalism includes direct interactions between agents. The paper concentrates on behavioral aspects [2] of control of agents.

Expressing ideas in natural languages tends to be inexact and somewhat superficial. Introduction of a formalism, that uses mathematical symbols, imposes rigor and precision on the discussion. Expressing our thoughts formally renders a deeper understanding of the topic and often discloses, otherwise hidden, properties of the proposed methods of solving the problem at hand. In our case the problem is formulated as: how to describe in a general and exact fashion the diverse behaviors that are necessary for the robots to adequately operate in complex environments. Moreover, we want the proposed description to be easily transformable into an implementation of the proposed ideas in the form of the control software coded in one of the programming languages.

2. STRUCTURE AND STATE OF AN EMBODIED AGENT

Let us consider a multi-agent system consisting of n_a embodied agents. The state of an agent a_j is:

$$s_j = \langle c_j, e_j, V_j, T_j \rangle, \quad j = 1, \ldots, n_a \tag{1}$$

c_j	–	state of the control subsystem of the agent (variables, program etc.),
e_j	–	state of the effector of the agent,
V_j	–	bundle of virtual sensor readings utilized by the agent,
T_j	–	information transmitted/received to/from the other agents.

For brevity, and because of contextual obviousness, the denotations assigned to the subcomponents of the considered system and their state will not be distinguished.

A bundle of virtual sensor readings contains individual virtual sensor readings:

$$V_j = \langle v_{j_1}, \ldots, v_{j_{n_{v_j}}} \rangle \tag{2}$$

Each virtual sensor v_{j_k}, $k = 1, \ldots, n_{v_j}$, produces an aggregate reading from one or more exteroceptors (further on called receptors). The data obtained from the receptors usually cannot be used directly in agent motion control, e.g., to control the arm motion, one would need the grasping location of the object that is to be picked and not the whole bit-map delivered by a camera. In some other cases a simple sensor in its own right would not suffice to control the motion (e.g., a single touch sensor), but several such sensors deliver meaningful data. The process of extracting meaningful information for the purpose of motion control is performed by virtual sensors. Thus the kth virtual sensor reading obtained by the agent a_j is formed as:

$$v_{j_k} = f_{v_{j_k}}(c_j, R_{j_k}) \tag{3}$$

where R_{j_k} is a bundle of receptor readings used for the creation of the kth virtual sensor reading.

$$R_{j_k} = \langle r_{j_{k_1}}, \ldots, r_{j_{k_{n_r}}} \rangle \tag{4}$$

where n_r is the number of receptor readings $r_{j_{k_l}}$, $l = 1 \ldots, n_r$ taken into account in the process of forming the reading of the kth virtual sensor of the agent a_j.

The responsibility of the agent's control subsystem c_j is to: gather information about the state of the

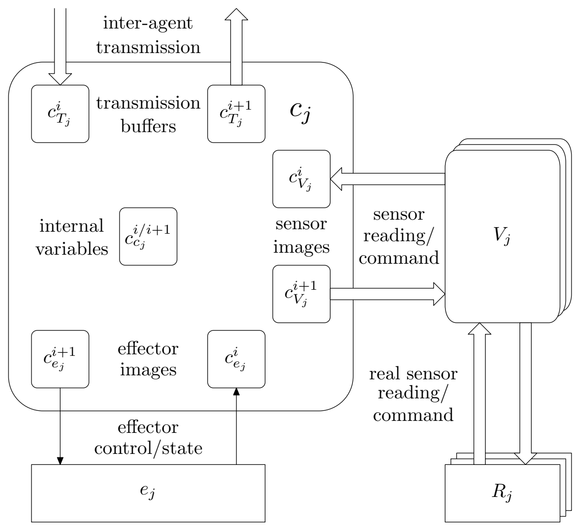

Fig. 2. A single embodied agent a_j, $j = 1, \dots, n_a$

environment through the associated virtual sensor bundle V_j, obtain the information from the other agents $a_{j'}$ ($j' \neq j$), monitor the state of its own effector e_j, and to process all of this information to produce: a new state of the effector e_j, influence the future functioning of the virtual sensors V_j, and send out information to the other agents $a_{j'}$. As a side effect the internal state of the control subsystem c_j changes. To do this effectively several components of the control subsystem can be distinguished:

c_{e_j}	–	image of the effector (this is a perception of the effector by the control subsystem, e.g., motor shaft positions, joint angles, end-effector location),
c_{V_j}	–	images of the virtual sensors (i.e., current virtual sensor reading and configuration),
c_{T_j}	–	inter-agent transmission (i.e., information mutually transmitted between the agents),
c_{c_j}	–	all the other relevant variables.

From the point of view of the system designer the state of the control subsystem changes at a servo sampling rate or a low multiple of that (usually referred to as a control step, e.g., 1 ms). If i denotes the current instant, the next considered instant is denoted by $i+1$. The control subsystem uses $c^i_j = \langle c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j} \rangle$ to produce $c^{i+1}_j = \langle c^{i+1}_{c_j}, c^{i+1}_{e_j}, c^{i+1}_{V_j}, c^{i+1}_{T_j} \rangle$, i.e.:

$$\left\{ \begin{array}{lcl} c^{i+1}_{c_j} & = & f_{c_{c_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j}) \\ c^{i+1}_{e_j} & = & f_{c_{e_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j}) \\ c^{i+1}_{V_j} & = & f_{c_{V_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j}) \\ c^{i+1}_{T_j} & = & f_{c_{T_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j}) \end{array} \right. \tag{5}$$

or more compactly:

$$c^{i+1}_j = f_{c_j}(c^i_j) \tag{6}$$

This is the program of the agent's actions for the whole of its lifespan. Obviously specifying just one such function describing the activities of the agent for for its whole existence would be a formidable task. Thus this function must be decomposed into more elementary functions. A robot programming framework must supply templates of such functions and patterns according to which they are assembled.

The control subsystem obtains the input values $c^i_{e_j}$, $c^i_{V_j}$, $c^i_{T_j}$ through transmission from the other components of the agent itself (e.g. effector, virtual sensors) or the other agents. It also must transmit the computed values $c^{i+1}_{e_j}$, $c^{i+1}_{V_j}$, $c^{i+1}_{T_j}$ to the other elements of the agent or the other partners (fig. 2). If we do not want to make any assumptions about the order of those transmissions and do not assume that the input and output images are of the same type (this occurs very rarely) duplicates of those entities must be stored. Only the internal variables c_{c_j} should have a single representation. Hence the control subsystem total state can be represented as (fig. 2):

$$c_j = c^{i/i+1}_j = \langle c_{c_j}, c^i_{e_j}, c^{i+1}_{e_j}, c^i_{V_j}, c^{i+1}_{V_j}, c^i_{T_j}, c^{i+1}_{T_j} \rangle \tag{7}$$

In general, the state of a system is represented by the values of state variables. Those variables hold the cumulated information about the past evolution of the system in such a form that, having the additional knowledge about future external influences on the system, we are able to predict the future behavior of this system. From what has been said it seems that just the elements of c^i_j would suffice to form the state vector of an agent, and that the elements of c^{i+1}_j are just the produced control. However, each component of c^i_j and c^{i+1}_j has to be stored within the agent's control subsystem (fig. 2), so each of those entities has a state of its own. There is a certain duration of time that both sets of those values have to coexist in the system simultaneously. Hence, it is reasonable to include the states of all the components in the state vector of the agent's control subsystem. The denotations c^i_j and c^{i+1}_j not only refer to different control subsystem components, but also different time instants at which their contents are utilized.

3. CONTROL OF MOTION OF AN EMBODIED AGENT

The art of programming boils down to mastering complexity. In the sixties it was noticed that only the correct structuring of programs can overcome complexity. Programs must be decomposed into small and well defined modules (procedures and functions were introduced) and a clear relationship between those modules must be established (e.g., `goto` had to be purged, intermodule communication means had to be specified). The formalism proposed here facilitates the creation of a clear structure of programs controlling the activities of an embodied agent.

Internal functioning of an agent is defined by the transition functions (5). The flexibility of a programming framework is attributed to the ability of expressing diverse approaches to programming the actions of each agent, and so the proposed formal description should enable easy formulation of diverse control strategies. Here we shall concentrate on schemes for behavioral control of agents. To decompose the control subsystem instead of providing a single set of functions (5), describing the motion of an agent throughout its life, many sets of such functions are specified. They define small motion segments, and the final result is obtained by their concatenation. The shortest duration of such a segment is the servo sampling rate or its low multiple. Such a short period is not practical, if one wants to describe the actions of an agent performing a certain task (e.g., foraging, assembling, reaching a destination in a cluttered environment). Thus, motion steps have to be grouped into sequences. The formation of such groupings is of interest to us here.

Thus instead of a single function f_{c_j}, n_f partial functions are defined:

$$c_j^{i+1} = {}^m f_{c_j}(c_j^i), \qquad m = 1, \ldots, n_f \tag{8}$$

Variability of agents is due to the diversity of those functions. The more functions of this type are provided by a programming framework the more types of agents we can construct.

In the case of a purely reactive system, sometimes also called a reflex system, the choice of the function ${}^m f_{c_j}$ is based on testing predicates ${}^q p_{c_j}$, $q = 1, \ldots, n_p$ which take as arguments only the components of $c^i_{V_j}$. In pseudo-code it can be expressed as:

$$\texttt{if } {}^q p_{c_j}(c^i_{V_j}) \texttt{ then } c_j^{i+1} := {}^m f_{c_j}(c_j^i) \texttt{ endif} \tag{9}$$

However, here we shall consider systems that decide, which function to choose, on the basis of all of the available information, i.e. all components of c_j^i. Moreover, in actual systems an endless loop containing the conditional instruction (9) must be constructed. Thus, for systems, where only one predicate can be true at a time, the pseudo-code will assume the following form:

```
loop
  // Determine the current state of the agent
  $e_j \rightarrowtail c^i_{e_j}$;   $V_j \rightarrowtail c^i_{V_j}$;   $c_{T_{j'}} \rightarrowtail c^i_{T_j}$;
  // Compute the next state of the agent
  if $^1 p_{c_j}(c_j^i)$ then $c_j^{i+1} :=^1 f_{c_j}(c_j^i)$ endif
  if $^2 p_{c_j}(c_j^i)$ then $c_j^{i+1} :=^2 f_{c_j}(c_j^i)$ endif
  ....................................
  if $^{n_p} p_{c_j}(c_j^i)$ then $c_j^{i+1} :=^{n_p} f_{c_j}(c_j^i)$ endif
  // Transmit the results to the other subsystems
  $c^{i+1}_{e_j} \rightarrowtail e_j$;  $c^{i+1}_{V_j} \rightarrowtail V_j$;  $c^{i+1}_{T_j} \rightarrowtail c_{T_{j'}}$;  $i := i + 1$;
endloop
```
(10)

where the comments are preceded by a double slash and the symbol "$\rightarrowtail$" denotes transmission of data. Those transmissions result in: data input, execution of motion, configuration of virtual sensors and transmission of messages to other agents. In each step i one iteration of the loop (10) will be executed. Thus in each control step i, one and only one out of the n_p predicates ${}^q p_{c_j}$ must be true, hence a single function ${}^m f_{c_j}$ is selected as the one designating the next state of the agent.

From the point of view of clarity of the description of the task it is useful to group the steps of the commanded evolution of control subsystem state into sequences. Those sequences will be called primitive behaviors.

$${}^q b_j = \{c_j^{i+1}, c_j^{i+2}, \ldots, c_j^{i+n_s}\} \tag{11}$$

where n_s is the number of steps in a behavior and q denotes a numeric identifier of this reaction. Each sequence of states $c_j^{i+1}, c_j^{i+2}, \ldots, c_j^{i+n_s}$ is generated by one of the functions ${}^m f_{c_j}$, thus this function is defining the primitive behavior. The pseudo-code (9) represents a single-step behavior, i.e., $n_s = 1$. In the case of a multi-step behavior the pseudo-code assumes the following form:

$$\texttt{if } {}^q p_{c_j}(c_j^i) \texttt{ then } {}^q b_j(c_j^i) \texttt{ endif} \tag{12}$$

In the case (12) the decision as to which behavior should be executed is taken once every n_s steps.

```
loop
  // Determine the current state of the agent
  $e_j \rightarrowtail c^i_{e_j}$;   $V_j \rightarrowtail c^i_{V_j}$;   $c_{T_{j'}} \rightarrowtail c^i_{T_j}$;
  // Select and execute the next behavior
  if  $^1 p_{c_j}(c_j^i)$  then  $^1 b_j(c_j^i)$  endif
  if  $^2 p_{c_j}(c_j^i)$  then  $^2 b_j(c_j^i)$  endif
  ..................................
  if  $^{n_p} p_{c_j}(c_j^i)$  then  $^{n_p} b_j(c_j^i)$  endif
endloop
```
(13)

Here the required computations (i.e., computation of $c_j^{i+\epsilon}$, $\epsilon = 1, \ldots, n_s$) and the execution of behaviors (i.e., transmission: $c^{i+1}_{e_j} \rightarrowtail e_j^{i+1}$) are bundled together within ${}^q b_j(c_j^i)$, $q = 1, \ldots, n_p$. The loop can be constructed in such a way that if none of the predicates ${}^q p_{c_j}(c^i_{V_j})$ is true a default behavior, called the main reaction or a goal pursuing reaction, is executed. The other reactions deal with some abnormal situations – hindering attaining of the goal [26].

If we use (11) and (13) as a combined definition of a behavior a recursive definition results, where (11) defines a primitive behavior, and (13) defines a complex behavior consisting of subbehaviors. In that case within the behavior a local set of predicates can be used, thus producing a hierarchy of reactions with variable granularity. One way to deal with assigning predicates to levels of behavior is to look at the time needed to process the information from the sensors, i.e. $c^i_{V_j}$. The more time required to perform the processing the higher the level of behavior that the associated predicate triggers.

If in the definition (13) one assumes that the last transition within a behavior forces the return to the initiation of the loop (this is equivalent to the use of the `break` instruction of the `C` language) only the behavior associated with the first true predicate will be executed. If in such a case the `if` instructions are placed in the loop in the order of importance of the associated behaviors a system with prioritized behaviors results. This is equivalent to the suppression mechanism of subsumption [6, 7].

In the case (10), where the computation of the next effector state and its execution are separate, several predicates ${}^q p_{c_j}$ can be true simultaneously. In that case the values of several partial functions ${}^m f_{c_j}$ have to be composed together, so the pseudo-code is:

```
loop
   // Determine the current state of the agent
   $e_j \rightarrowtail c^i_{e_j}$;   $V_j \rightarrowtail c^i_{V_j}$;   $c_{T_{j'}} \rightarrowtail c^i_{T_j}$;
   clear$({}^q c^{i+1}_j)$ for: $q = 1, \ldots, n_p$;
   // Compute the next control subsystem state
   if ${}^1 p_{c_j}(c^i_j)$ then ${}^1 c^{i+1}_j := {}^1 f_{c_j}(c^i_j)$ endif
   if ${}^2 p_{c_j}(c^i_j)$ then ${}^2 c^{i+1}_j := {}^2 f_{c_j}(c^i_j)$ endif
   ......................................
   if ${}^{n_p} p_{c_j}(c^i_j)$ then ${}^{n_p} c^{i+1}_j := {}^{n_p} f_{c_j}(c^i_j)$ endif
   // Compute the aggregate control
   $c^{i+1}_j :=$ composition$({}^q c^{i+1}_j)$ for: $q = 1, \ldots, n_p$;
   // Transmit the results to the other subsystems
   $c^{i+1}_{e_j} \rightarrowtail e_j$;   $c^{i+1}_{V_j} \rightarrowtail V_j$;   $c^{i+1}_{T_j} \rightarrowtail c_{T_{j'}}$;
endloop
```
(14)

Many composition operators can be conceived. Competitive methods are based on some form of selecting one value out of the computed values, e.g.:

$$c^{i+1}_j = \max_m \{{}^m f_{c_j}(c^i_j)\} \tag{15}$$

The mechanisms of inhibition (elimination of some components by others) and suppression (substitution of some components by others), introduced in [6, 7], can be produced by supplying an adequate selection function.

Cooperative methods are based on some form of superposition of the computed values, e.g.:

$$c^{i+1}_j = \sum_{m=1}^{n_f} \frac{w_m}{w} \, {}^m f_{c_j}(c^i_j), \qquad w = \sum_{m=1}^{n_f} w_m \tag{16}$$

where w_m are the weights of particular components.

The proposed program structures either rely on the evaluation of the predicates in each step (e.g., (10) and (14)) or on a fixed length of the sequence within

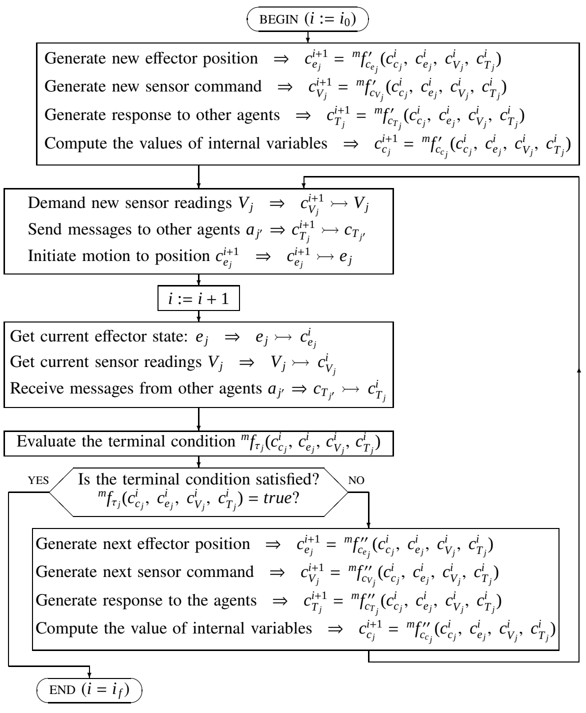
BEGIN $(i := i_0)$
Generate new effector position $\Rightarrow$ $c^{i+1}_{e_j} = {}^m f'_{c_{e_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j})$
Generate new sensor command $\Rightarrow$ $c^{i+1}_{V_j} = {}^m f'_{c_{V_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j})$
Generate response to other agents $\Rightarrow$ $c^{i+1}_{T_j} = {}^m f'_{c_{T_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j})$
Compute the values of internal variables $\Rightarrow$ $c^{i+1}_{c_j} = {}^m f'_{c_{c_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j})$
Demand new sensor readings V_j $\Rightarrow$ $c^{i+1}_{V_j} \rightarrowtail V_j$
Send messages to other agents $a_{j'}$ $\Rightarrow$ $c^{i+1}_{T_j} \rightarrowtail c_{T_{j'}}$
Initiate motion to position $c^{i+1}_{e_j}$ $\Rightarrow$ $c^{i+1}_{e_j} \rightarrowtail e_j$
$i := i + 1$
Get current effector state: e_j $\Rightarrow$ $e_j \rightarrowtail c^i_{e_j}$
Get current sensor readings V_j $\Rightarrow$ $V_j \rightarrowtail c^i_{V_j}$
Receive messages from other agents $a_{j'}$ $\Rightarrow$ $c_{T_{j'}} \rightarrowtail c^i_{T_j}$
Evaluate the terminal condition ${}^m f_{\tau_j}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j})$
YES
Is the terminal condition satisfied? ${}^m f_{\tau_j}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j}) = true$?
NO
Generate next effector position $\Rightarrow$ $c^{i+1}_{e_j} = {}^m f''_{c_{e_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j})$
Generate next sensor command $\Rightarrow$ $c^{i+1}_{V_j} = {}^m f''_{c_{V_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j})$
Generate response to the agents $\Rightarrow$ $c^{i+1}_{T_j} = {}^m f''_{c_{T_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j})$
Compute the value of internal variables $\Rightarrow$ $c^{i+1}_{c_j} = {}^m f''_{c_{c_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j})$
END $(i = i_f)$

Fig. 3. `Move` instruction of an agent a_j

a primitive behavior (as in (13)). A very general concept of motion instructions for multi-robot systems was introduced in `MRROC++` [27, 28, 29]. It can be extended even further to include direct inter-agent communication. The `Move` instruction has similar properties as (13), thus it can be treated as a generalization of the above proposals. The semantics of a general `Move` instruction is presented in fig. 3. Here the function f_{c_j} (6), due to its inherent complexity, has been decomposed into a sequence of separate pairs of functions. Each pair influences the agent during i_f steps, where a third function ${}^m f_{\tau_j}$ determines the number of steps. The function ${}^m f'_{c_j}$ defines the action of the agent in the first step – usually, from the computational point of view, it differs from all the other motion steps within the instruction execution. The function ${}^m f''_{c_j}$ specifies the behavior of the agent in all motion steps, but the first one. In this way, one function (f_{c_j}), defining the system evolution for whole of its lifetime, has been divided into a sequence of triplets of functions ${}^m f'_{c_j}$, ${}^m f''_{c_j}$ and ${}^m f_{\tau_j}$, which specify the actions of an agent for a period of time when a single `Move` instruction is executed. Each function ${}^m f_{\tau_j}$ determines when the control system should switch from one `Move` instruction to another. Each such instruction is governed by a different set of functions: ${}^m f'_{c_j}$, ${}^m f''_{c_j}$ and ${}^m f_{\tau_j}$. Thus those functions should constitute the parameters of the `Move` instruction. This solution was adopted in `MRROC++`.

Let us limit the arguments of the `Move` instruction to just the three relevant to this discussion, i.e., ${}^m f'_{c_j}$, ${}^m f''_{c_j}$ and ${}^m f_{\tau_j}$. Now a behavior can be composed of a sequence of `Move`$({}^m f'_{c_j}, {}^m f''_{c_j}, {}^m f_{\tau_j})$ instructions. As previously, such a sequence will be selected by a predicate within an `if` instruction.

```
if ${}^{q}p_{c_j}(c^i_j)$ then Move( ${}^{m}f'_{c_j}$, ${}^{m}f''_{c_j}$, ${}^{m}f_{\tau_j}$); ... endif
```
(17)

Here the duration of the execution of each component (i.e., `Move`) in a sequence is determined by its termination function ${}^{m}f_{\tau_j}$. As the moment at which the terminal condition is satisfied is usually caused by an external event, the synchronous character of the pseudo-code (9) or (12) is lost here – thus, from the point of view of the programmer, the pseudo-code (17) becomes asynchronous and event driven.

Each of the `Move` instructions has two separate functions generating the steps: one for the first step (${}^{m}f'_{c_j}$) and one for every other step (${}^{m}f''_{c_j}$). The sequence is chosen according to the value of the predicate ${}^{q}p_{c_j}$, so the decision process is not invoked too often. Hence the resulting pseudo-code is:

```
loop
    // Determine the current state of the agent
    $e_j \rightarrowtail c^i_{e_j}$;   $V_j \rightarrowtail c^i_{V_j}$;   $c_{T_{j'}} \rightarrowtail c^i_{T_j}$;
    // Select and execute the next behavior
    if ${}^{1}p_{c_j}(c^i_j)$ then Move( ${}^{1}f'_{c_j}$, ${}^{1}f''_{c_j}$, ${}^{1}f_{\tau_j}$);...
       endif
    if ${}^{2}p_{c_j}(c^i_j)$ then Move( ${}^{2}f'_{c_j}$, ${}^{2}f''_{c_j}$, ${}^{2}f_{\tau_j}$);...
       endif
    .......................................
    if ${}^{n_p}p_{c_j}(c^i_j)$ then Move( ${}^{m}f'_{c_j}$, ${}^{m}f''_{c_j}$, ${}^{m}f_{\tau_j}$);...
       endif
endloop
```
(18)

In the case of the pseudo-code (18) only one predicate ${}^{q}p_{c_j}$ can be true at the moment the decision is being made. To enable the situation where several predicates can be true simultaneously some modifications are necessary. The `Move` instruction not only computes the value of c^{i+1}_j, but also causes its transmission to the other subsystems of the agent, thus no composition of partial results is possible within the program (18). To make this possible a slight modification of the `Move` definition is necessary and an introduction of a separate entity responsible for the composition of partial results obtained from each `Move` instruction being executed in parallel. The modification of the flowchart defining the `Move` instruction (fig. 3) consists in exchanging the contents of the operational block initiating the motion to position $c^{i+1}_{e_j}$ (i.e., executing: $c^{i+1}_{e_j} \rightarrowtail e_j$) for the transmission of partial result ${}^{q}c^{i+1}_j$ to this new entity (e.g., a thread). Once this entity has collected the partial results from all currently active `Move` instructions, it can compute the final value according to one of the formulas (15) or (16) and then transmit the result to the other subsystems of the agent for execution.

Besides the implementation of the `Move` instruction a `Wait` [29] instruction is usually implemented. It provides means for waiting for an initial condition to be fulfilled. This condition defines the instant that a following motion can commence. A Boolean function ${}^{m}f_{\iota_j}$ defines this condition. Its value is checked in each step of the execution of the `Wait` instruction.

The actions of both the `Move` and `Wait` instructions are described by referring to discrete time *i*. The introduction of Boolean functions ${}^{m}f_{\tau_j}$ and ${}^{m}f_{\iota_j}$, which determine the instants when the execution of each instruction should be terminated changes the perception of the behavior of the agent. As those instants are not known *a priori*, they have to be associated with events that cause the conditions represented by those functions to be fulfilled. The `Move` and `Wait` instructions change the low level synchronous (time driven) view of the system into a high level asynchronous (event driven) view.

4. ADDING DELIBERATION TO BEHAVIOR

Deliberation assumes the use of artificial intelligence techniques [12] to find a plan (i.e., sequence) of actions leading the execution of a task (goal) set forth before the agent. This is implemented by search techniques. Search requires the following entities:

- search space (i.e., problem domain) composed of search space states – not to be mistaken with the state of the environment or the agent itself,
- initial state, belonging to the search space – from this state the search commences,
- operators, which transform the current search space state into the next states (those operators may result either from production rules or be the side-effect of application of predicate logic [12]),
- data structure accumulating the generated states (i.e., the search tree or graph),
- goal test deciding whether the generated state is the goal state
- path cost function, which evaluates the quality of the obtained search space state – usually it takes into consideration the cost of both the path traversed so far and the remaining path to the goal state (e.g., A^* algorithm or its derivatives).

In the case of deliberative systems c_{c_j} must contain the data structures accumulating the generated states (i.e., problem solution). Deliberation is a search process starting in the initial state of the problem solution. This state includes a partial description of the current state of the agent, but also other search related information. As the operators are applied new problem solution states are generated. The operators are equivalent to transition functions transforming one problem solution state into another. Heuristics are included in the path cost function and help in discarding the produced states that either do not lead to a solution or are along a far from optimal search path, thus avoiding a combinatorial explosion in the search process. The path leading from the initial state and ending in a goal state describes a plan of actions that the agent should try to execute. Assuming that the plan generation starts with $c^i_{c_j}$ the plan (result of the search process) would be included in $c^{i+1}_{c_j}$ and would influence the generation of the state of the agent in the next steps, i.e., $c^{i+1}_j, c^{i+2}_j, \ldots$. If the planning process takes a lot of time, the plan might not be ready in the instant $i+1$. In that case the function (6) would have to generate c^{i+1}_j without the plan, so that c^{i+1}_j would have to result in halting the effector – to be on the safe side.

5. CONCLUSION

The agents of the described system can act:

- purely independently,
- they can interact directly through an exchange of transmitted data (c_{T_j}),
- they can also interact indirectly by sensing the other agents or the results of their actions (by using their receptors),
- they can be coordinated by a hierarchically higher entity, i.e., a coordinator.

The coordinator can be treated as an abstract agent, i.e., an agent that does not posses an effector (a body). Nevertheless, there is no reason to assume that the abstract agent should not gather directly the information from the environment, thus it can have a virtual sensor bundle of its own. In this case it would deliver global information about the environment, e.g., a camera gathering the information about the global state of the football pitch in a RoboCup match. The structure of the coordinator can be subdivided into a hierarchy of abstract subagents. In a simple case each sub-agent can control a group of embodied agents, treating them as a set of effectors and a set of sensors. In turn those groups can be controlled by a higher level virtual agent.

Transition function based formalism facilitates the design and implementation of multi agent systems by decomposing a large system into components that can be designed and implemented by providing the code for the specified functions. Each of the control subsystem components (7) can be treated as an object in an object-oriented programming sense. Thus the communication with the other agents, effectors or virtual sensors can be handled internally by the methods of these objects (`MRROC++` uses this method). The objects provide, through their public interfaces, only the data that is necessary for the computation of the control of the agent. Those objects provide data that is utilized by transition functions (5) resident in the control subsystem to compute the next state of this subsystem.

The formalism by enumerating the arguments of the transition functions (5) ensures that all the necessary interconnections between the components of the system are present. The examples showed the numerous possibilities of designing such systems, e.g., behavioral agents with diverse methods of composition of the final control signals, deliberative systems and hybrid deliberative-behavioral systems.

The presented formalism has been used to specify the MRROC++ robot programming framework, and currently it is being used to extend that framework onto multi-agent systems described in this paper. Those systems will be composed of several mobile robots. Moreover, a two-handed system able to manipulate objects using vision and force information and having the sense of hearing and an ability of speaking is under construction. This will be a laboratory model for testing control algorithms used in service robots.

6. REFERENCES

1. R. Alami, R. Chatila, S. Fleury, M. Ghallab M., and Ingrand F. An architecture for autonomy. *Int. J. of Robotics Research*, 17(4):315–337, 1998.
2. R. C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, 1998.
3. P. Backes, S. Hayati, V. Hayward, and K. Tso. The KALI multi-arm robot programming and control environment. In *Proc. NASA Conf. on Space Telerobotics*. 1989.
4. C. Blume and W. Jakob. *PASRO: Pascal for Robots*. Springer-Verlag, Berlin, 1985.
5. C. Blume and W. Jakob. *Programming Languages for Industrial Robots*. Springer-Verlag, Berlin, 1986.
6. R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.
7. R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, (47):139–159, 1991.
8. H. Bruyninckx. Orocos – open robot control software. http://www.orocos.org/, 2002.
9. S. Fleury and M. Herrb. Genom user's guide. Report, LAAS, Toulouse, December 2001.
10. V. Hayward, L. Daneshmend, and S. Hayati. An overview of KALI: A system to program and control cooperative manipulators. In K. Waldron, editor, *Advanced Robotics*, pages 547–558. Springer-Verlag, 1989.
11. V. Hayward and R. P. Paul. Robot manipulator control under unix RCCL: A robot control C library. *Int. J. Robotics Research*, 5(4):94–111, Winter 1986.
12. G. F. Luger and W. A. Stubblefield. *Artificial Intelligence and the Design of Expert Systems*. Benjamin-Cummings, Redwood, 1989.
13. M. E. Markiewicz and C. J. P. Lu-

cena. Object oriented framework development. *ACM Crossroads*, 7(4), 2001. Also: http://www.acm.org/crossroads/xrds7-4/frameworks.html.

14. K. Mianowski. Parallel and serial-parallel robots for the use of technological applications. In *Parallel Kinematic Machines PKM'99, November, Milano, Italy*, pages 39–46. 1999.
15. K. Mianowski, K., M. Wojtyra, and S. Ziętarski. Application of the unigraphics system for milling and polishing with the use of rnt robot. In *Workshop for the users of UNIGRAPHICS system, Frankfurt, November*, pages 98–104. 1999.
16. K. Mianowski and K. Nazarczuk. Parallel drive of manipulator arm. In *CISM-IFToMM Symp. on Theory and Practice of Robots and Manipulators Ro.Man.Sy 8, Cracow, Poland*, pages 143–150. July 1990.
17. E. R. Morales. Generis: The ec-jrc generalised software control system for industrial robots. *Industrial Robot*, 26(1):26–32, 1999. Also: http://www.erxa.it/Eng/GENERIS/description.html.
18. K. Nazarczuk and K. Mianowski. Polycrank – fast robot without joint limits. In *CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators Ro.Man.Sy'12, Wienna, 6–9 June*, pages 317–324. Springer-Verlag, 1995.
19. K. Nazarczuk, K. Mianowski, A. Olędzki, and C. Rzymkowski. Experimental investigation of the robot arm with serial-parallel structure. In *9-th World Congress on the Theory of Machines and Mechanisms, Milan, Italy*, pages 2112–2116. 1995.
20. K. Nazarczuk, K. Mianowski, and S. Łuszczak. Development of the design of polycrank manipulator without joint limits. In *CISM-IFToMM Symp. on Theory and Practice of Robots and Manipulators Ro.Man.Sy 13, Zakopane, Poland, 3–6 July*, pages 285–292. 2000.
21. L. Petersson, D. Austin, and H. Christensen. Dca: A distributed control architecture for robotics. In *Int. Conference on Intelligent Robots and Systems IROS'01*. 2001.
22. C. Schlegel and R. Wörz. Interfacing different layers of a multilayer architecture for sensorimotor systems using the object-oriented framework smartsoft. In *3rd European Workshop on Advanced Mobile Robots, Eurobot'99, Zürich, Switzerland*. September 1999.
23. R. Simmons and D. Apfelbaum. A task description languagefor robot control. In *International Conference on Itelligent Robots and Systems IROS'98. Victoria, Canada*. October 1998. Also: http://www-2.cs.cmu.edu/ tdl/.
24. R. Simmons, R. Goodwin, C. Fedor J., and Basista. Task control architecture: Programmer's guide to version 8.0. Carnegie Mellon University, School of Computer Science, Robotics Institute, May 1997. Also: http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/TCA/www/tca.orig.html.
25. Wojciech Szynkiewicz. Motion planning for multi-robot systems with closed kinematic chains. In *9th IEEE Int. Conf. on Methods and Models in Automation and Robotics MMAR'2003, Międzyzdroje*, pages 779–786. August 25-28, 2003.
26. C. Zieliński. *Robot Programming Methods*. Publishing House of Warsaw University of Technology, Warsaw, 1995. Also: http://www.ia.pw.edu.pl/~zielinsk/.
27. C. Zieliński. Object–oriented programming of multi–robot systems. In *4th Int. Symp. Methods and Models in Automation and Robotics, Międzyzdroje, Poland*, pages 1121–1126. August 26–29 1997.
28. C. Zieliński. The MRROC++ system. In *1st Workshop on Robot Motion and Control, RoMoCo'99, Kiekrz, Poland*, pages 147–152. June 1999.
29. C. Zieliński. By how much should a general purpose programming language be extended to become a multi-robot system programming language? *Advanced Robotics*, 15(1):71–95, 2001.
30. C. Zieliński. A unified formal description of behavioural and deliberative robotic multi-agent systems. In *7th IFAC International Symposium on Robot Control SYROCO 2003, Wrocław, Poland*, volume 2, pages 479–486. September 1–3 2003.
31. C. Zieliński, K. Mianowski, K. Nazarczuk, and W. Szynkiewicz. A prototype robot for polishing and milling large objects. *Industrial Robot*, 30(1):67–76, January 2003.
32. C. Zieliński and W. Szynkiewicz. Control of two 5 d.o.f. robots manipulating a rigid object. In *IEEE Int. Symp. on Industrial Electronics ISIE'96, Warsaw, Poland*, volume 2, pages 979–984. June 17–20 1996.