4th Int. Workshop on Robot Motion and Control, RoMoCo'04, 17–20 June 2004, Puszczykowo, Poland. pp. 79–84

Specification of behavioural embodied agents

Cezary Zieliński
Warsaw University of Technology (WUT)
Institute of Control and Computation Engineering
ul. Nowowiejska 15/19, 00-665 Warsaw, POLAND, e-mail: C.Zielinski@ia.pw.edu.pl

Abstract

The paper[1] *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 ambient through their sensors. The presented examples of its usage pertain to behavioural and hybrid behavioural-deliberative agents, but are not limited only to that.*

1 Introduction

This paper deals with the problem of designing universal control software for systems:

- composed of multiple embodied (i.e. having a material body) agents with initially unknown:
  - number and type of effectors (devices within the agents influencing the state of the environment),
  - number and type of sensors (devices within the agents gathering the information about the state of the environment)
  - communication means between the agents,
  - information processing capabilities of the agents,
- 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. Because of the above mentioned variability of both the structure and the components of those systems the control software must be open to the highest degree. The answer to the so posed problem, that is provided by computer science, is a tool named a programming framework [5]. In general, programming frameworks are application generators with the following components:

- library of software modules (i.e. 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.

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. Such a programming framework is a tool for constructing multi-agent control systems, but we still need a meta-tool for defining such a framework. This paper presents a formal meta-tool for specifying the modules of a programming framework that is well suited to building control systems consisting of one or more embodied agents. The proposed meta-tool has been verified by presenting several specifications of agents utilizing behavioral, fuzzy and deliberative approaches to their control [10].

Summarizing, the paper proposes a formal method of describing multi-agent systems lending itself to structuring and implementation of a general programming framework for producing such systems. The presented formalism is a generalization of the approach used to define and implement the `MRROC++` [8, 9, 11] 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 is possible. Obviously indirect interactions through sensing or upper layers of hierarchy are possible. The formalism presented here includes direct interactions between agents. Moreover, diverse approaches to designing the internal principles of operation of each agent are at the focus of this work.

Many behavioral [1] and deliberative [6] robot control methods have been developed. Besides investigating task execution by a single robot, multi-agent systems are being developed. Unfortunately only some of those control methods have a formal description. To aggravate the situation, those control methods that have formal specifications use different formal means of description. This paper presents an outline of a unified formal description of systems composed of many embodied agents using behavioral or deliberative control methods. Such a description is needed to specify in an implementation independent way the functional-

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

ity of programming frameworks. Transition function based approach to the formal description of the above mentioned systems has been assumed to facilitate the future implementation of thus specified programming frameworks. The presented description is a generalization of the formal description used for the specification of `MRROC++` [8, 9].

2 Embodied agents

For the purpose of brevity, and because of contextual obviousness, the denotations assigned to the components of the considered system and their state will not be distinguished. Let us consider a multi-agent system consisting of n_a agents. The state of a single embodied agent a_j, $j = 1, \dots, n_a$, is:

$$s_j =< c_j, e_j, V_j > \tag{1}$$

c_j	–	state of the **control subsystem** of the agent (i.e. memory: variables, program),
e_j	–	state of the **effector** of the agent,
V_j	–	bundle of **virtual sensor** readings utilized by the agent.

A bundle of virtual sensor readings contains n_{v_j} individual virtual sensor readings:

$$V_j =< v_{j_1}, \dots, v_{j_{n_{v_j}}} > \tag{2}$$

Each virtual sensor v_{j_k}, $k = 1, \dots, n_{v_j}$, produces an aggregate reading from one or more exteroceptors. Exteroceptors, for brevity further on called simply **receptors** or real sensors, include all the measuring devices gathering information from the environment of the system. Interoceptors and proprioceptors, i.e. devices for measuring the internal state of the system (e.g. position encoders, resolvers), are not treated here as proper receptors, because they supply data about the state of the effectors (internal state of the agent) and not the environment of the agent, and thus are associated with the effector. 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 named data aggregation and 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.

$$R_{j_k} =< r_{j_{k_1}}, \dots, r_{j_{k_{n_r}}} > \tag{4}$$

where n_r is the number of receptor readings $r_{j_{k_l}}$, $l = 1 \dots, n_r$, taken into account in the process of forming the reading of the kth virtual sensor of the agent a_j. It should be noted that (3) implies that the reading of the virtual sensor depends also on c_j. In this way the agent has the capability of configuring the sensor as well as delivering to the virtual sensor the relevant information about the current state of the agent (including its effector). This might be necessary in the case of computing the reading of a virtual sensor having its associated receptors mounted on the effector (e.g. artificial skin).

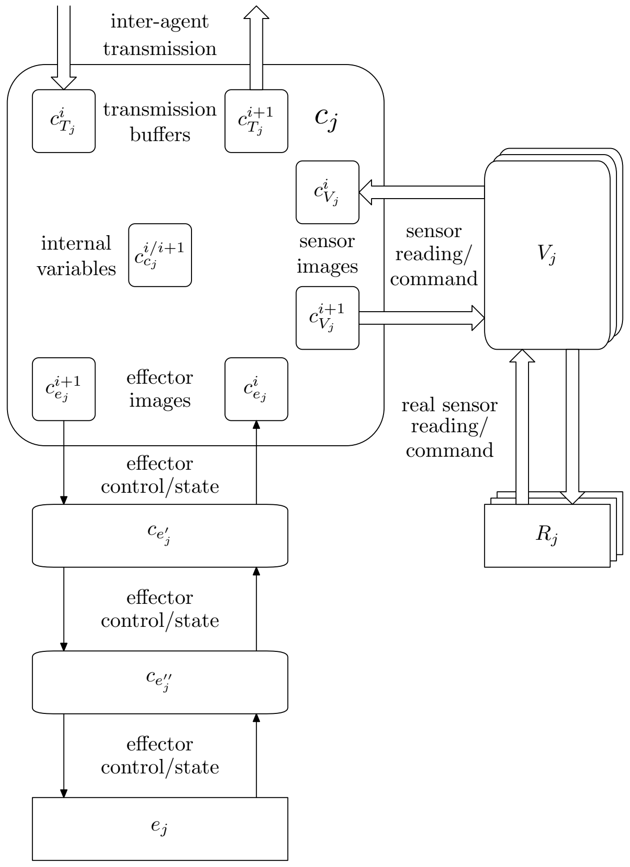

Figure 1: A single embodied agent a_j, $j = 1, \dots, n_a$

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

Thus the control subsystem is partitioned:

$$c_j =< c_{c_j}, c_{e_j}, c_{V_j}, c_{T_j} > \tag{5}$$

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. If i denotes the current instant, the next considered instant is denoted by $i+1$. The control subsystem uses $c^i_j =< c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j} >$ to produce $c^{i+1}_j =< c^{i+1}_{c_j}, c^{i+1}_{e_j}, c^{i+1}_{V_j}, c^{i+1}_{T_j} >$, i.e.:

$$\left\{\begin{array}{lcl} c^{i+1}_{c_j} & = & f_{c_{c_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j}) \\ c^{i+1}_{e_j} & = & f_{c_{e_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j}) \\ c^{i+1}_{V_j} & = & f_{c_{V_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j}) \\ c^{i+1}_{T_j} & = & f_{c_{T_j}}(c^i_{c_j}, c^i_{e_j}, c^i_{V_j}, c^i_{T_j}) \end{array}\right. \tag{6}$$

or more compactly:

$$c^{i+1}_j = f_{c_j}(c^i_j) \tag{7}$$

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 to the other partners (fig. 1). 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 state can be represented as (fig. 1):

$$c_j =< c_{c_j}, c^i_{e_j}, c^{i+1}_{e_j}, c^i_{V_j}, c^{i+1}_{V_j}, c^i_{T_j}, c^{i+1}_{T_j} > \tag{8}$$

Usually the control subsystem of an agent is not implemented as a single entity (e.g. process). In that case there is no direct link between e_j and c_{e_j}. Both the input and the output effector images are being transformed through several intermediate layers (fig. 1). For example, in `MRROC++` the Effector Control Process (`ECP`), which is equivalent to the agent's control subsystem, sends motion commands formed in the output effector image and receives the updates of the input effector image through Effector Driver Process (EDP) and Servo Group Process (`SG`) [8, 11]. This only introduces multi-stage transmissions: $c^{i+1}_{e_j} \rightarrowtail e^{i+1}_j$ and $e^i_j \rightarrowtail c^i_{e_j}$, i.e. :

$$\begin{array}{ccccccc} c^{i+1}_{e_j} & \rightarrowtail & c^{i+1}_{e'_j} & \rightarrowtail & c^{i+1}_{e''_j} & \rightarrowtail & e^{i+1}_j \\ e^i_j & \rightarrowtail & c^i_{e''_j} & \rightarrowtail & c^i_{e'_j} & \rightarrowtail & c^i_{e_j} \end{array} \tag{9}$$

where the symbol "$\rightarrowtail$" represents a transmission within a small fraction of a single time step. Sometimes the creation of a virtual sensor reading requires a multi-stage process, e.g. Arkin [1] defines perceptual schemas that have that property. This also can be dealt with by a multi-stage aggregation and transmission similar to the one presented by (9).

3 A multi-agent system

The needs of communication between the agents are defined by the arguments of the transition functions (6). If the computation of any of the output values requires some input data, this input data has to be delivered to the agent's control subsystem, and this defines a connection between the source of this data and the agent's control subsystem.
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 (through the receptors) the other agents or the results of their actions, or they can be coordinated by a hierarchically higher entity, i.e. a coordinator. The **coordinator**, for the sake of consistency, 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. Let us distinguish the coordinator by the subscript 0. In the face of the above its state will be:

$$s_0 =< c_0, V_0 > \tag{10}$$

Hence the system state is described by:

$$s =< s_0, s_1, \ldots, s_{n_a} > \tag{11}$$

The structure of the coordinator will not be discussed further in this paper. Let it suffice to say that it can be subdivided into other abstract subagents.

4 Internal functioning of an agent

Internal functioning of an agent is defined by the transition functions (6), (7). The flexibility of a programming framework is located in the ability of expressing diverse approaches to programming the actions of each agent, and so the formal description should enable easy formulation of diverse control strategies. Usually instead of computing a single set of functions (6) many partial functions are evaluated and the final result is obtained by a certain composition of the partial results. Thus n_f partial functions are defined:

$$c_j^{i+1} = {}^m f_{c_j}(c_j^i), \qquad m = 1, \ldots, n_f \tag{12}$$

Variability of agents is due to the diversity of those functions. The more functions of this type is provided by a programming framework the more types of agents we can construct. Moreover, the partial functions (12) do not have to take as arguments all of the components of c_j^i. Here some interesting possibilities are investigated. The diversity of those possibilities is simultaneously the verification of the power of the proposed specification meta-tool. The following examples pertain to reactive, behavioural and hybrid behavioural-deliberative control methods, but neither the presented formalism nor the resulting frameworks are limited to only those kinds of robot control.
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 $c_{V_j}^i$. In pseudo-code it can be expressed as:

$$\textit{if } {}^q p_{c_j}(c_{V_j}^i) \textit{ then } c_j^{i+1} := {}^m f_{c_j}(c_j^i) \textit{ endif} \tag{13}$$

In actual systems an endless loop containing the conditional instruction (13) must be constructed. Thus, the pseudo-code will assume the following form:

```
loop
    // Compute the next effector state
    if  ${}^1 p_{c_j}(c_{V_j}^i)$   then  $c_j^{i+1}$  :=  ${}^1 f_{c_j}(c_j^i)$   endif
    if  ${}^2 p_{c_j}(c_{V_j}^i)$   then  $c_j^{i+1}$  :=  ${}^2 f_{c_j}(c_j^i)$   endif
    ........................................
    if  ${}^{n_p} p_{c_j}(c_{V_j}^i)$  then  $c_j^{i+1}$  :=  ${}^{n_p} f_{c_j}(c_j^i)$  endif
    // Transmit the results of computations
    // to the effector
    $c_{e_j}^{i+1} \rightarrowtail e_j^{i+1}$
endloop
                                                  (14)
```

where the comments are preceded by a double slash and the symbol "$\rightarrowtail$" denotes transmission of data and as a result the execution of motion as required.
In each step i one iteration of the loop (14) will be executed. If in each iteration, and thus in each control step i, only one out of the n_p predicates ${}^q p_{c_j}$ is true, then a single function ${}^m f_{c_j}$ is selected as the one designating the next state of the agent. Usually in such a case $n_p = n_f$ and therefore the endless loop contains n_p instructions of the type (13). Obviously different predicates can cause the same reaction, but usually this is not the case, hence the functions in (14) have been numbered 1 through n_p.
The system that is described in [7] performs the selection of the function ${}^m f_{c_j}$ for several consecutive steps, i.e. the selection is less frequent than the evaluation of the function, so the reaction is composed of several steps. In that case a behavior is defined as a sequence of states:

$$b_j^i = \{c_j^{i+1}, c_j^{i+2}, \ldots, c_j^{i+n_s}\} \tag{15}$$

where n_s is the number of steps in a behavior (reaction). As this behavior originates in c_j^i it is labelled with the superscript i, in other words it depends on c_j^i. The pseudo-code (13) 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:

$$\textit{if } {}^q p_{c_j}(c_{V_j}^i) \textit{ then } b_j^i(c_j^i) \textit{ endif} \tag{16}$$

In the case (16) the decision as to which behavior should be executed is taken once every n_s steps. Nevertheless, the transition between the state $c_j^{i+\epsilon}$ and $c_j^{i+\epsilon+1}$, where $\epsilon = 0, \ldots, n_s - 1$ is still computed on the basis of the functions ${}^m f_{c_j}$. The control program is composed of an endless loop containing a sequence of instructions of the form (16). In that case each iteration of the loop contains several control steps i.

```
loop
    // Select the next behavior
    if  ${}^1 p_{c_j}(c_{V_j}^i)$   then  ${}^1 b_j^i(c_j^i)$   endif
    if  ${}^2 p_{c_j}(c_{V_j}^i)$   then  ${}^2 b_j^i(c_j^i)$   endif      (17)
    ................................
    if  ${}^{n_p} p_{c_j}(c_{V_j}^i)$  then  ${}^{n_p} b_j^i(c_j^i)$  endif
endloop
```

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: $c_{e_j}^{i+1} \rightarrowtail e_j^{i+1}$) are bundled together within ${}^q b_j^i(c_j^i)$, $q = 1, \ldots, n_p$. The loop can be constructed in such a way that if none of the predicates ${}^q p_{c_j}(c_{V_j}^i)$ 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.
If we use (15) and (17) as a combined definition of a behaviour a recursive definition results, where (15) defines an atomic behaviour, and (17) defines a complex behaviour consisting of subbehaviours. In that case within the behaviour a local set of predicates can be used, thus producing a hierarchy of reactions with

a variable granularity. One way to deal with assigning predicates to levels of behaviour 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 behaviour that the associated predicate triggers.
Rodney Brooks in [2, 3] uses inhibition and suppression (exchange) mechanism for the evaluation of the final value of c^{i+1}_j – in this case the state of augmented finite state automatons. Each of the automatons generates signals that are the result of a process equivalent to the computation of functions ${}^m f_{c_j}$. The suppression and inhibition mechanism is equivalent to a selection based not only on the values of input signals but also on the priorities of automatons, i.e. priorities of the partial functions. In this way higher level behaviors can subsume the lower level ones.
In the case (14), 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. Many composition operators can be conceived. Competitive methods are based on some form of selecting one value out of the computed values, e.g.:

$$c^{i+1}_j = \max_m \{{}^m f_{c_j}(c^i_j)\} \tag{18}$$

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

$$c^{i+1}_j = \sum_{m=1}^{n_f} {}^m f_{c_j}(c^i_j) \tag{19}$$

(19) can be generalized by introducing weighted sums. The purely reactive approach can be generalized if the predicates ${}^q p_{c_j}$ take as arguments not only $c^i_{V_j}$, but also other components of c^i_j, especially $c^i_{c_j}$. A purely deliberative system results, if for the generation of c^{i+1}_j, functions having as arguments only $c^i_{c_j}$ are used. This is rarely feasible in the case of embodied agents, as the state of the environment cannot be fully predicted, hence sensor readings $c^i_{V_j}$ are necessary. In this way a hybrid deliberative-reactive systems result. Deliberation assumes the use of artificial intelligence techniques [4] 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 state 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 [4]),
- 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 or hybrid 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. $c^{i+1}_j, c^{i+2}_j, \ldots$. In the case of hybrid systems this plan can be treated as a goal pursuing reaction (main reaction). 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 (7) 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.
In the case of hybrid systems two approaches to plan generation can be followed. In the first case, the plan can be treated as a template and fixed corrective reactions handle minor problems with its execution. The plan constitutes a goal pursuing reaction and always the same corrective reactions are executed regardless of the step (action) in the plan. In the second case, the plan is generated with a changing set of corrective reaction for each action (plan step). This is equivalent to a system that changes both the goal pursuing reaction and the set of corrective reactions in each plan step. Nevertheless, in both cases such problems can arise that further execution of the plan will be impossible. In that case replanning has to be done – obviously with the current state of the agent included in the initial problem solution state.
Besides using $c^i_{c_j}$ as an argument of functions generating the value of c^{i+1}_j, also $c^i_{T_j}$ can be used. This implies direct cooperation of agents. Thus obtained information can be used both in the planning process as well as a parameter of the reactions.

5 Conclusions

The presented mathematical generic specification of multi-agent systems is based on transition functions and selection predicates. Both of those concepts are the basis of high-level general purpose computer programming languages. Mathematical functions can easily be transformed into a programming language (e.g. `C`, `C++`, `Pascal`) functions, procedures or OOP methods, while selection predicates are well grounded in control flow instructions of programming languages (e.g. in `C`: *if then else*, *switch* etc.). Within each agent each of the control subsystem components (8) 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 (6) resident in the control subsystem to compute the next state of this subsystem. Thus the transformation of the specification into software is straight forward. Specific cases of such transformations have been presented in [7, 9].

The formalism by enumerating the arguments of the transition functions (6) ensures that all the necessary interconnections between the components of the system will be present. The presented examples showed the numerous possibilities of designing such systems, e.g. purely reactive agents with diverse methods of composition of the final control signals, deliberative systems and hybrid deliberative-reactive systems. Purely deliberative agents have not been considered here, because the discrepancy between a computationally realistic world model (a model that includes the search space) and the real environment is so big that it is highly improbable that an agent without sensors updating the state of the world model frequently could act efficiently. Usually it would fail to attain its goal even because of minor discrepancies between the model and reality.

The presented formalism also poses many open research problems, especially regarding detailed methods of controlling an agent. For instance: for a given task the selection of behaviors b_j^i of an agent a_j is not obvious. The definition of functions ${}^{m}f_{c_j}$ defining a behavior and the selection of predicates ${}^{q}p_{c_j}$ triggering a behavior is also a difficult job. The proposed hierarchical composition of the reactions within a single agent is even more complex to implement for a real task. Even less obvious is the assignment of subtasks to agents in a multi-robot behavioral system. All of the above proposals need future investigations. The treatment of abnormal situations (hardware or software errors) in both behavioral and deliberative systems also requires research.

However, it should be noted that the enumerated problems pertain to the implementation of a specific activity of an agent and not to the structure of the system, hence the presented formalism serves its purpose of providing a formal representation for describing the structure of multi-agent programming frameworks. It points out what has to be contained in the framework both in terms of building blocks and in terms of its structure, i.e. what are the legal ways of assembling those building blocks. The formalism can also be used for describing the functioning of a single agent, but the control algorithm is external to the introduced notation. In other words, first we must have an idea of the control algorithm and only later we can express it in the presented formalism. The formalism only is a specification tool that facilitates the later implementation of the control system.

References

[1] R. C. Arkin. *Behavior-Based Robotics.* MIT Press, Cambridge, Mass., 1998.

[2] 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.

[3] R. A. Brooks. Inteligence without representation. *Artificial Intelligence*, (47):139–159, 1991.

[4] G. F. Luger, W. A. Stubblefield. *Artificial Intelligence and the Design of Expert Systems.* Benjamin-Cummings, Redwood, 1989.

[5] M. E. Markiewicz, C. J. P. Lucena. Object oriented framework development. *ACM Crossroads*, 7(4), 2001. Also: http://www.acm.org/crossroads/xrds7-4/frameworks.html.

[6] S. Russell, P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, N.J., 1995.

[7] C. Zieliński. Reaction based robot control. *Mechatronics*, 4(8):843–86, 1994.

[8] C. Zieliński. The MRROC++ system. *1st Workshop on Robot Motion and Control, RoMoCo'99, Kiekrz, Poland*, strony 147–152. June 28–29 1999.

[9] 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.

[10] C. Zieliński. A unified formal description of behavioural and deliberative robotic multi-agent systems. *Proc. 7th IFAC International Symposium on Robot Control SYROCO 2003, Wrocław, Poland*, wolumen 2, strony 479–486. September 1–3 2003.

[11] C. Zieliński, W. Szynkiewicz, K. Mianowski, K. Nazarczuk. Mechatronic design of open-structure multi-robot controllers. *Mechatronics*, 11(8):987–1000, November 2001.