A Behaviour-Based Kernel Architecture for Robot Control*

João Sequeira**
jseq@isr.ist.utl.pt

M. Isabel Ribeiro
mir@isr.ist.utl.pt

Instituto Superior Técnico/Instituto de Sistemas e Robótica (IST/ISR)
Av Rovisco Pais 1, 1096 Lisboa CODEX, Portugal
Tel.: +351-1-8418057, Fax: +351-1-8418291

Abstract: This paper identifies the basic building blocks in a *behaviour*-based architecture for robot control. The major contribution of the paper is the definition of a state space able to embed semantic relationships between *behaviours* and the definition of a special operator to generate motion in the state space. The controllability of the resulting architecture is verified in terms of the adopted *behaviour* definition. The resulting architecture can be considered as a kernel, expandable through the inclusion of external functionalities aiming at increasing its performance.
Keywords: *Behaviour*, Semantics, Action, Algebraic group, Architecture.

1 INTRODUCTION

In the robotics field *behaviours* have often been considered simply as the map $\mathcal{B} : Perception \mapsto Actions$ and been associated with the use of *natural language* concepts, embedding a capability for semantic representation. A behavioural framework, not accounting for the inclusion of semantics, was proposed in [Willems, 1991] as the foundation of a theory of dynamical systems. Other behavioural applications include compiler design for digital hardware design languages (see [Knapp, 1996]).
A framework supporting semantic representation (from the viewpoint of *behaviour*-based paradigms) must consider non bijective maps $\mathcal{B}$ and also the possibility of having multiple $\mathcal{B}$s qualitatively identical. This leads to an extended definition of *behaviour* as a set of actions sharing some equivalence relation property.
In some sense, this paper provides an extension to the framework defined in [Willems, 1991] where a *behaviour* is defined as a trajectory in a state space with the natural equivalence relation between *behaviours* being given by different parametrizations for the same *behaviour* model, thus not considering the possibility of having distinct trajectories representing the same *behaviour*. Different *behaviour*-based approaches include a recent proposal for a *behaviour* design methodology, not supported in any specific theoretical framework, that appears in [M. Colombetti et al., 1996].
This paper proposes an approach to *behaviours* for robotic devices control that results in a kernel architecture with the ability to support a semantic representation, i.e., a client of a number of servers, each providing particular functionalities to performance increasing.
Throughout the paper it is assumed the weak controllability (see [R. Hermann and A. Krener, 1977]) of the robotic devices involved, as it seems unreasonable to specify a behaviour for a device not able to follow it.
The paper is outlined as follows. Section 2 defines the support state space. The architecture is specialized in Sections 3 and 4. Section 5 explains the interface between the abstract levels defined in the two previous sections. Problem specific concepts are treated in Section 6. A brief explanation on the role of semantics in the paper's work is given in Section 7. Section 8 concludes the paper.

2 *BEHAVIOUR* PRINCIPLES

In this work, an agent is defined by the information pipeline *Environment* $\mapsto$ *Sensors* $\mapsto$ *Behaviours* $\mapsto$ *Actuators* $\mapsto$ *Robotic device*. This concept of agent is able to encapsulate the main aspects of a *behaviour* approach and allows problems with multiple heterodox robotic mechanisms to be treated under the same framework. The refered information pipeline defines the sets containing the necessary and sufficient information to represent the agent and the tasks it has to execute:

S, internal variables, e.g., battery level, sensing data,

Q, the robot C-space,

*Partially sponsored by the programme PRAXIS XXI, project COOPERA - 2/2.1/TPAR/2087/95.
**Work partially sponsored by NATO grant 9/A/95/PO.

$A = \{a_i : Q \times \mathbb{T} \mapsto Q,\ i = 1, \ldots\}$, the space of available actions, with Q representing the initial configurations[3] for the actions and $\mathbb{T}$ the time,

The pair (q, a) (the notation $a(q)$ will be extensively used) completely determines the trajectory of the robot in the C-space between any two consecutive events, thus representing an appropriate state variable for the agent. This state variable contains the information about the current motion strategy of the agent and also some memory of the close past, embedded on the initial configuration for the action.

3 STRUCTURE OF THE ACTION SET

The elements of A satisfy the following definition:

Definition 3.1 (The space A) *Let $a(q)\big|_t$ represent an action, parameterized by the initial configuration q, at the time $t \geq 0$[4]. Each trajectory verifies $\forall_{q \in Q},\ a(q)\big|_t \subset B_a(q),\ t \geq 0$, with $B_a(q) \subset Q$ a connected set containing q and with a boundary enclosing a fixed shape volume.*

From the definition results that it is possible to associate multiple actions with the same set $B(q)$. The representation of semantics is achieved through the following definition for the natural equivalence relation in A,

Definition 3.2 (Equality relation in A) *Let $V(q) \subset B_i(q_i) \cap B_j(q_j)$ be a fixed size neighbourhood of an arbitrary $q \in Q$. Two actions $a_i(q_i)$ and $a_j(q_j)$ are said to be* **equal** *(the relation being represented by $a_i(q_i) = a_j(q_j)$) if and only if the following conditions are met:*

$$\begin{array}{l} a_i(q_i), a_j(q_j) \subset\ B_i(q_i) \cap B_j(q_j), \\ \forall_{t_j \geq 0}, \exists_{t_i \geq 0} : a_i(q_i)\big|_{t_i} \in V(a_j(q_j)\big|_{t_j}). \end{array}$$

Equal actions generate similar trajectories. The first condition accounts for spatial similarity of the trajectories, while the second accounts for the time similarity.

Let the set $W_a^l(q)$ contain the configurations locally weakly accessible from q. Due to the arbitrary nature of the $B_i(q)$ spaces[5], it may happen that q lies on the boundary of $R_i(q) = B_i(q) \cap W_a^l(q)$. In those situations the execution of $a_i(q)$ is interleaved with the necessary maneuvers to keep the robot inside $R_i(q)$.

A basic structure, allowing the generation of motion in the space A, can be obtained through the definition of a closed operation in A, e.g., giving A the structure of an algebraic group.

Proposition 3.1 (The group operation) *Consider that an agent is executing the action $a_j(q_j)$. At time t_0, some decision mechanism requires the agent to switch to the action $a_i(q_i)$, which will start to be executed at an unspecified time $t_1 > t_0$ at a neighbourhood of q_i. The operation $\circ : A \times A \mapsto A$, $a_{i \circ j}(q_j) = a_i(q_i) \circ a_j(q_j)$, verifying $a_{i \circ j}(q_j) \subset B_i(q_i) \cup B_j(q_j)$, if $B_i(q_i) \cap B_j(q_j) \neq \emptyset$[6] and $a_{i \circ j}(a_j(q_j)\big|_{t_0})\big|_{t_1} \in B_i(q_i)$ gives to A the structure of a group.*

[3] Recall the weak controllability assumption.

[4] The notations $a(q)$, a and $a(q)\big|_t$ will be used to represent, respectively, a trajectory starting at configuration q, a trajectory generated by the same flow but with an unspecified starting configuration, and a configuration of trajectory $a(q)$ at time t.

[5] The *behaviour* designer must be free to define these sets.

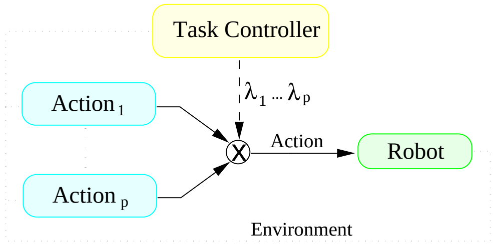


Figure 1: Conceptual *behaviour* architecture.

See appendix A for a demonstration of this proposition.

The operation $\circ$ generates an action with the same *behaviour* as the sequence $a_j(q_j)\big|_{0 \leq t < t_0}$, $a_i(q_i)\big|_{t \geq 0}$ and, if necessary, a path linking $a_j(q_j)\big|_{t_0}$ and $a_i(q_i)\big|_0$ and accounting for the maneuvering the agent may have to make to switch between the actions. In some sense, it encapsulates and/or hides the agent kinematic constraints from the programmer (considered as the *behaviour* designer).

Figure 1 shows the conceptual architecture supported in this algebraic structure. The block $\otimes$ implements the group operation $\circ$ taking into account the interleaving between the execution of any action and any maneuvering that may be necessary. It is defined by[7]

$$\otimes \equiv \begin{cases} a_i(q_i) \circ a_i(q_i) & \text{if } a_i(q_i)\big|_t \in \mathrm{Bd}(R_i(q_i)) \\ a_i(q_i) \circ a_j(q_j) & \text{otherwise.} \end{cases}$$

The actions to be combined are defined by the *Task Controller* block through the control variables λ_i.

A differential structure can be induced in A, under the assumption that the trajectories generated by any action are linear combinations of more elementary actions,

$$a(q) = \sum_{i=1}^{p} \lambda_i b_i(q),$$

with $\lambda_1, \ldots, \lambda_p \in \mathbb{R}$, $b_1(q), \ldots, b_p(q)$ the elementar actions starting at q. The existence of the $b_i(q)$ is guaranteed by the weak controllability assumption (they represent trajectories generated by the basis elements, at configuration q, of the Lie algebra of vector fields on Q). An advantage of having a differential structure is that a continuum of actions can be applied, enabling the use of continuum optimization methods. This is specially important in the case of the *behaviour* paradigm where the main interest is not the computation of a Lie algebra basis of vector fields at each configuration[8], but rather a set of vector fields

[6] The operation has no meaning if $B_i(q_i) \cap B_j(q_j) = \emptyset$.

[7] $\mathrm{Bd}(X)$ denotes the boundary of the set X.

[8] In general, this is a difficult problem [Laumond, 1993].

able to generate a subset of A. Using Definition 3.2, a *behaviour* can be defined as a subset $B \subset Q$, spanned by all the actions that are considered to generate similar trajectories. An alternative definition is to consider a *behaviour* as a set of vector fields, $\{\mathbf{b_i},\ i = 1, \dots, p\}$ on Q, which, in general, are uncompletely specified (e.g., due to environmental uncertainties) and such that the corresponding actions generate similar trajectories.

3.1 CONTROLLABILITY

Let $\mathbb{I} = \{1, \dots, p\}$ with p the number of *behaviours*, $\mathbb{Q} \subset Q$ be a finite set of configurations, and assume that

$$\bigcup_{\mathbb{I} \times \mathbb{Q}} B_{\mathbb{I}}(\mathbb{Q}) = Q. \tag{1}$$

Then the agent keeps the controllability property inherited from the robotic device. Expression (1) reflects the fact that if a covering of Q using a finite number of *behaviour* transitions (expressed by the finite cardinal of $\mathbb{Q}$) exists, then the agent is controllable.

The cardinal of $\mathbb{Q}$ depends on the ability of the *Task Controller* to select, given a set of *behaviours*, a number of configurations such that, by appropriate sequencing of *behaviours* using the elements in $\mathbb{Q}$ as *via* configurations, the agent is able to move between any two arbitrary configurations.

3.2 A PRACTICAL EXAMPLE

In this section, *behaviour* control is used to make a mobile platform to follow, in the xy plane, a reference trajectory, specified by two *behaviours*.

Consider an agent with $Q = \{(x, y, \theta)\ \ x, y \in \mathbb{R},\ \theta \in [0, 2\pi[\}$, and equipped with the following a set of *behaviours*, each defined by a single vector field,

$$\mathbf{b_1} = \begin{bmatrix} 1 \\ 0.5\cos(0.5x) \\ \theta_{\text{free\_1}} \end{bmatrix} \quad \mathbf{b_2} = \begin{bmatrix} 1 \\ 2x \\ \theta_{\text{free\_2}} \end{bmatrix}$$

where $\theta_{\text{free\_1}}, \theta_{\text{free\_2}}$ represent the unspecified part of the vector fields.

If the kinematics of the agent does not impose any special restrictions, then given two arbitrary actions it is possible to define a path between them by considering a translation in the space defined by (λ_1, λ_2). This is the case if the kinematics is described by a general equation of the type $\dot{q} = \mathbf{L_n} \cdot \mathbf{u_n}$ where $q \in \mathbb{R}^n$, $\mathbf{u_n}$ is a $n \times 1$ control vector, and $\mathbf{L_n}$ is a rank n matrix. For a mobile robot of the type

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} = \mathbf{L_2} \cdot \mathbf{u_2} \tag{2}$$

translations between arbitrary points (x, y) may not be possible. The conditions under which the direct switching between actions (without linking path) is possible are given by the configurations (x, y, θ) and

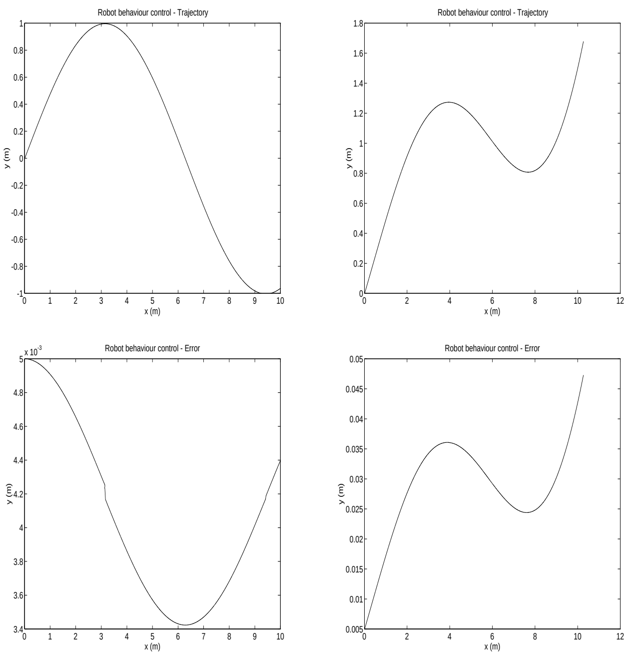


Figure 2: Unicycle control using *behaviours*.

the parameters λ_1, λ_2, such that the control $\mathbf{u_2}$ lies in a surface, in the control space, defined by

$$\lambda_1 \mathbf{b_1} + \lambda_2 \mathbf{b_2} = \mathbf{L_2} \cdot \mathbf{u_2} \tag{3}$$

for any λ_1, λ_2.

In the general case, the computation of the link path can be made by the generation of Lie bracket combinations of the vector fields in $\mathbf{L_2}$ that span the tangent space at configuration q, $T_q Q$. For this particular example, the link path is composed only of pure rotations, i.e., the composition of the actions generated by $\mathbf{b_1}, \mathbf{b_2}$ can be made by a pure rotation of $\theta_f - \theta$ rads, with θ_f given by

$$\tan(\theta_f) = (\lambda_1 0.5\cos(0.5x) + \lambda_2 2x)/(\lambda_1 + \lambda_2),$$

and the group operation is $\circ \equiv +$ acting only in the orientation coordinate.

Figure 2 shows two simulated examples, with sampling interval 0.001s, for $\lambda_1 = 1, \lambda_2 = 0$ (the two leftmost graphics) and $\lambda_1 = 1, \lambda_2 = 0.025$. The upper most figures refer to the actual path of the mobile robot, while the lower most correspond to the errors relative to the references defined by the *behaviours*.

4 AGENT LOWER LEVELS

The two rightmost levels of an agent information pipeline are represented by a continuous dynamical system, of the form

$$\dot{q} = f(q, u) \tag{4}$$

where the state variable, q, lies in an m-dimensional, C^∞ manifold, Q, $u \in \mathbb{R}^p$ with $p \leq m$ is the control variable, and $f(.)$ is a C^∞ vector field on Q. By the

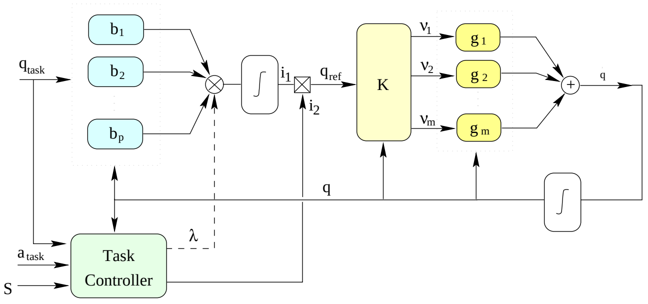


Figure 3: *Behaviour*-based architecture.

initial weak controllability assumption, there exists a set of vector fields $\mathbf{g_1}, \cdots, \mathbf{g_m}$ that span T_qQ and such that equation (4) can be rewritten as

$$\dot{q} = [\mathbf{g_1}, \cdots, \mathbf{g_m}] \cdot \nu = \mathbf{L}(q) \cdot \nu \qquad (5)$$

where $\nu \in \mathbb{R}^m$ is a new control variable[9]. From the point of view of developing a *behaviour* paradigm, it can be assumed that a controller $K : Q \mapsto \{\nu\}$, able to drive the robotic device between any two configurations, is available. This kind of motion does not take advantage of two important characteristics of *behavioural* control: the extended planning horizon (i.e., some algorithm continuously generating references to K is needed), and the ability to represent semantic concepts.

The agent is thus composed of a cascade of two similar structures, $(\mathbf{g_1}, \ldots, \mathbf{g_m}, K, \int, \oplus)$, called the *robot structure*, and $(\mathbf{b_1}, \ldots, \mathbf{b_p}, \mathit{TaskController}, \int, \otimes)$, called the *behaviour structure*, interfaced by an interface block, represented by the symbol ⊠. The action of ⊠ is to switch between the motion strategies based on (i) *behaviours* and (ii) point-to-point control. Figure 3 shows the resulting architecture.

5 *INTERFACE* BLOCK

The point-to-point control functionality, provided by the *robot structure*, is used in situations where no adequate *behaviour* is available. The enable of this functionality is made by the *Interface* block, represented by the symbol ⊠, controlled by the *Task Controller* block (see Table 1).

Active inputs	Action
i_1	$q_{\text{ref}} = i_1$
i_1, i_2	$q_{\text{ref}} = i_2$
i_2	$q_{\text{ref}} = i_2$

Table 1: Working algorithm for the ⊠ building block.

6 *TASK CONTROLLER* BLOCK

The *Task Controller* block uses specific information about the class of tasks that the agent is able to perform, (e.g., the criteria that define which action to apply under the range of environmental conditions considered) to define: (i) the controls λ_i that are sent to the $\otimes$ block, and (ii) whenever necessary, point-to-point actions, that are sent directly to the ⊠ block.

The following examples assess the performance of the *behaviour* approach developed in a navigation task[10].

A mobile robot, of the type described by equation (2), is controlled using two *behaviours*: *obstacle avoidance* and *go direct to goal*. For the purpose of this example, a task is defined as a point $(x_{\text{task}}, y_{\text{task}}) \in \mathbb{R}^2$. An obstacle is placed in the direct path between the starting point and the task.

In a first experiment (Figure 4-a), the *behaviours*, including no data sensed from the environment, are defined by

$$\mathbf{b_{oa}} = \mathrm{Rot}(z, \alpha) \begin{bmatrix} 1 \\ 2x \\ \theta_{oa} \end{bmatrix} \quad \mathbf{b_g} = \begin{bmatrix} x_{\text{task}} - x \\ y_{\text{task}} - y \\ \theta_g \end{bmatrix},$$

where $\mathrm{Rot}(z, \alpha)$ is a matrix describing a rotation of $\alpha = \arctan(y_{\text{task}} - y, x_{\text{task}} - x)$ around the z axis and θ_g and θ_{oa} are the free variables.

The operation $\circ$ is defined similarly to the example of section 3, and the *Task Controller* is defined by the function

$$\begin{bmatrix} \lambda_{oa} \\ \lambda_g \end{bmatrix} = \begin{bmatrix} 1 - \exp(-1/d_o) \\ \exp(-1/d_o) \end{bmatrix} \qquad (6)$$

where d_o represents the Euclidean distance to the obstacle, measured by a sensor along the orientation defined by the vector $w = (x_{\text{task}} - x, y_{\text{task}} - y)$. If there is no obstacle along w the distance returned is ∞.

In the second experiment (Figure 4-b), the previous set of *behaviours* is augmented with a *wall following* defined by

$$\mathbf{b_{wf}} = \begin{bmatrix} \cos(\gamma) \\ \sin(\gamma) \\ \theta_{wf} \end{bmatrix}$$

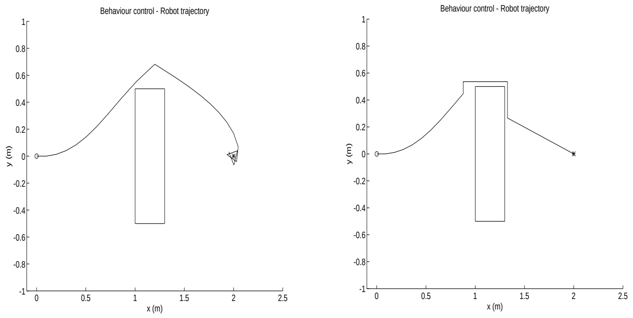
with γ representing the orientation of the wall relative to the vehicle. The principle used to design the *Task Controller* is to avoid obstacles, using the parabolic *behaviour*, $\mathbf{b_{oa}}$, when far from them, and to follow the walls, when near, until the direction to the goal has a clearance of 0.2m. In addition to the first experiment, the robot is equipped with two sensors that give distances along the orientations $\theta \pm \pi/2$, i.e., laterally.

In the first experiment, the trajectory is piecewise smooth. The first smooth part is due to the influence of the obstacle avoid *behaviour*, while the second is due to the go direct to goal *behaviour*.

In the second experiment the advantage of the *Interface* block, ⊠ can be seen. When the first wall following part of the trajectory finishes, the lateral sensors do not detect any obstacle. If the robot turns into the direction of the goal, it will face again the obstacle, leading to more maneuvering. Since the right lateral

[9] This is a consequence of the controllability rank condition, [R. Hermann and A. Krener, 1977].

[10] The initial and final task positions are marked, respectively, with o and $*$.



(a) Experiment 1 - no sensing data on *behaviours*.

(b) Experiment 2 - using sensing data on *behaviours*.

Figure 4: Vehicle trajectories.

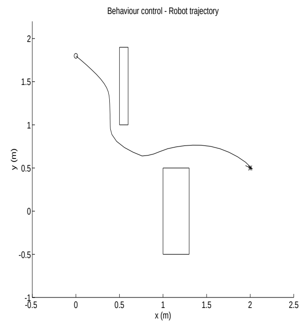


Figure 5: Vehicle trajectories.

sensor had already given the indication of free space, the *Task Controller* forces the starting of the next action in a point that avoids this problem. This effect also causes the small contour around the obstacle before going directly to the goal.
In the third experiment (Figure 5) the *behaviour* $\mathbf{b_{oa}}$ includes the sensory information, d_o, from the environment. The *behaviour* set used is defined by

$$\mathbf{b_{oa}} = \begin{bmatrix} -w_x/d_o \\ -w_y/d_o \\ \theta_{oa} \end{bmatrix} \quad \mathbf{b_g} = \begin{bmatrix} x_{\text{task}} - x \\ y_{\text{task}} - y \\ \theta_g \end{bmatrix},$$

where (w_x, w_y) represents the direction of minimum distance, d_o, to any obstacle. The *Task Controller* uses a constant combination of *behaviours* defined by $\lambda_{oa} = 0.1, \lambda_g = 1$.
The three experiments presented in this section are representative of the potential of *behaviour*-based control. The performance of the *behaviour* approach depends on constraints such as the inclusion/absence of sensory data in the *behaviours*, the quality of such data and the design of the *Task Controller*. The experiments illustrate the differences in performance between the three situations considered under the combinations of those constraints. The inclusion of sensory data in the *behaviours* may lead to smooth trajectories (experiment 3) or to a trajectory that spans a small area around the obstacle (experiments 1 and 2). Globally, the experiments show the relevance of the study of agents equipped with unsophisticated sensors and working in hostil environments, where the acquisition of good sensor data may be difficult.

7 USING SEMANTICS

In the presented examples, semantics is represented by leaving unspecified some coordinates of the vector fields generating the *behaviours* (no *robot structure* was used, and the kinematic constraints were taken care by the *behaviour structure*). Another application of the semantic representation property of the presented approach to *behaviours* is in the design of the *Task Controller*. The symbolic nature of *behaviours* (they express a concept/symbol) leads naturally to Finite State Machine (FSM) based implementations for the *Task Controller*. To minimize the number of states of the FSM, it is desirable to have actions that adapt themselves to some variety of environmental operating conditions but generating identical (in the sense of Definition 3.2) *behaviours*. The use of "adaptive" actions allows a reduction in the number of FSM states (see [J. Sequeira et al., 1997] for some preliminary results).

8 CONCLUSIONS

The paper describes basic concepts necessary to establish a *behaviour*-based paradigm able to cope with the representation of semantic concepts.
The flexibility of the *behaviour* approach was shown in the examples of Section 6. The ability to cope with semantic representation (a direct consequence of the *behaviour* definition used) suggests that functionalities aiming at performance increasings can be seen as add-ins to the kernel architecture of Figure 3. From an implementation viewpoint, the architecture of Figure 3 acts as a client of a number of servers, each representing an add-in. Examples of possible servers are: learning algorithms and sensor data pre-processors, e.g., histograms to the equalization of ultrasonic data, or environment modeling using the variety of methods available (sonar, laser, video cameras, etc).
Preliminary experimental results were first presented in [J. Sequeira et al., 1996], using only the kernel, and in [J. Sequeira et al., 1997], using one server to "adapt" actions to the environment. The evaluation of the architecture of Figure 3, under a variety of common tasks, requiring both isolated and cooperating macroscopic behaviours, is ongoing work.

ACKNOWLEDGEMENTS

The authors whish to thank to an anonymous reviewer for pointing out reference [Willems, 1991].

REFERENCES

[J. Sequeira et al., 1997] J. Sequeira, J. Millan, M. I. Ribeiro, and J. Gonçalves (1997). Embedding Learning in Behaviour-Based Architectures: A Conceptual Approach. In Monostori, L., editor, *Proceedings of The*

Second World Congress on Intelligent Manufacturing Processes & Systems. Springer. Budapest, Hungary.

[J. Sequeira et al., 1996] J. Sequeira, M. I. Ribeiro, and J. M. G. Gonçalves (1996). Cooperation Among Multiple Robotic Mechanisms. In *Proceedings of the SIRS'96*. Lisbon, Portugal.

[Knapp, 1996] Knapp, D. (1996). *Behavior Synthesis: Digital System Design Using the Synopsis Behavioural Compiler*. Prentice Hall.

[Laumond, 1993] Laumond, J. P. (1993). Singularities and Topological Aspects in Nonholonomic Motion Planning. In Z. Li and J. Canny, editors, *Non-Holonomic Motion Planning*, pages 149–199. Kluwer Academic.

[M. Colombetti et al., 1996] M. Colombetti, M. Dorigo, and G. Borghi (1996). Behaviour Analysis and Training: A Methodology For Behavior Engineering. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(6).

[R. Hermann and A. Krener, 1977] R. Hermann and A. Krener (1977). Nonlinear Controllability and Observability. *IEEE Transactions on Automatic Control*, AC-22(5).

[Willems, 1991] Willems, J. C. (1991). Paradigms and Puzzles in the Theory of Dynamical Systems. *IEEE Transactions on Automatic Control*, 36(3).

A THE CLASS OF GROUP OPERATIONS $\circ$

To demonstrate Proposition 3.1 it is necessary to check the conditions under which the properties of an algebraic group are verified. The demonstration relies on the assumption that some basic functionalities in the control of the agent are available and in its compatibility with the structure defined in the text for the elements of A.

Null element

The existence of a null action, a_e, corresponding to the null element of the group, is quite reasonable. For each event it must be always possible *to do nothing*, or, *to do something that does not compromise future decisions*.
Let q_i, q_e be two arbitrary configurations. From the action composition definition,

$$a_e(q_e) \circ a_i(q_i) \subset B_e(q_e) \cup B_i(q_i),$$

and, in order to verify

$$a_e(q_e) \circ a_i(q_i) = a_i(q_i),$$

it is required that $\forall_{q_i, q_e \in Q}\ B_e(q_e) \subseteq B_i(q_i)$. Defining

$$B_e(q) = \min_{q,i} \max\{\bigcap_i B_i(q)\}$$

the time similarity condition is automatically verified, since a_e will always generate trajectories not moving "too much" away from the current configuration, so that any subsequent action has the same effect (in the sense of Definition 3.2) as it would have had if a_e had not been applied.

Similarly, it seems reasonable to be able to take any action after having taken a null action. The previous reasoning in the construction of $B_e(q)$ is applied and,

$$a_i(q_i) \circ a_e(q_e) \subset B_i(q_i) \cup B_e(q_e) = B_i(q_i),$$

with $\forall_{q_e, q_i}, B_e(q_e) \subseteq B_i(q_i)$, to verify

$$a_i(q_i) \circ a_e(q_e) = a_i(q_i)$$

Therefore, the existence of a null element, a_e (representing one of the agent functionalities), is compatible with the structure defined for A.

Inverse element

Consider an agent executing an action a_i. At time t_1 a_i^{-1} is chosen to start execution, aiming at driving the agent back to a configuration such that a subsequent application of a_i, at time t_2, would have had the same effect as if no a_i^{-1} would have been applied. Using Definition 3.2, this statement can be represented by

$$a_i(a_i^{-1}(a_i(q)\big|_{t_1})\big|_{t_2}) \circ (a_i^{-1}(a_i(q)\big|_{t_1}) \circ a_i(q)) = a_i(q),$$

which, to be verified, requires that

$$a_i^{-1}(a_i(q)\big|_{t_1}) \circ a_i(q) = a_e(q)$$

holds.
Using the definition of $\circ$, the action composition

$$a_i^{-1}(a_i(q)\big|_{t_1}) \circ a_i(q) \ \subset\ B_i^{-1}(a_i(q)\big|_{t_1}) \cup B_i(q)$$

must be equal to

$$a_e(q) \subset B_e(q).$$

To keep the uniqueness of the null element, the equality above (expressed by $a_i^{-1}(a_i(q)\big|_{t_1}) \circ a_i(q) = a_e(q)$) must hold for arbitrary q. Defining $B_i^{-1}(a_i(q)\big|_{t_1}) = B_i(q)$ the inverse element yields

$$a_i^{-1}(q) \subset B_i(q),$$

for arbitrary q, i.e., a_i^{-1} lies in the region spanned by a_i. By interchanging a_i with a_i^{-1} in the initial reasoning the expression $a_i(a_i^{-1}(q)\big|_{t_1}) \circ a_i^{-1}(q) = a_e(q)$ is also verified.

Associative property:

Consider the compositions given by

$$(a_3(q_3) \circ a_2(q_2)) \circ a_1(q_1) = a_{3\circ2}(q_2) \circ a_1(q_1) = a_{(3\circ2)\circ1}(q_1).$$

Using the definition of $\circ$, the composition $a_{2\circ1}(q_1) = a_2(q_2) \circ a_1(q_1)$ generates a trajectory lying in the region $B_1(q_1) \cup B_2(q_2)$ and $a_{(3\circ2)\circ1}(q_1)$ lies in $(B_1(q_1) \cup B_2(q_2)) \cup B_3(q_3) = B_1(q_1) \cup B_2(q_2) \cup B_3(q_3)$.
Similarly, for the compositions given by

$$a_3(q_3) \circ (a_2(q_2) \circ a_1(q_1)) = a_3(q_3) \circ a_{2\circ1}(q_1) = a_{3\circ(2\circ1)}(q_1),$$

$a_{3\circ(2\circ1)}(q_1)$ lies in the region defined by $B_3(q_3) \cup (B_2(q_2) \cup B_1(q_1)) = B_3(q_3) \cup B_2(q_2) \cup B_1(q_1)$.
The relation $a_{(3\circ2)\circ1}(q_1) = a_{3\circ(2\circ1)}(q_1)$ is verified and that amounts to say that the operation $\circ$ is associative.

This article was processed using LaTeX