A port–arbitrated mechanism for behavior selection in humanoid robotics

Ali Paikan
iCub Facility
Istituto Italiano di Tecnologia
Via Morego, 30, 16163, Genoa, Italy
Email: ali.paikan@iit.it

Giorgio Metta
iCub Facility
Istituto Italiano di Tecnologia
Via Morego, 30, 16163, Genoa, Italy
Email: giorgio.metta@iit.it

Lorenzo Natale
iCub Facility
Istituto Italiano di Tecnologia
Via Morego, 30, 16163, Genoa, Italy
Email: lorenzo.natale@iit.it

***Abstract*—Software engineering and best practices promote modularity and composability to reduce debugging and development time of software applications in robotics. This approach, however, increases the complexity of the system and the effort needed to properly coordinate interactions between modules. On the other hand programming robots to cope with an unstructured environment requires the implementation of highly reactive systems. Behavior-based architectures have been proposed as a programming paradigm to build complex, yet, reactive systems by integrating simpler modules. They require however that modules establish special connections dedicated to carry coordination signals. In a distributed architecture these signals must be properly synchronized with the ones that carry data.**

This article proposes a novel method for developing reactive systems by coordinating concurrent, distributed behaviors. In our approach arbitration exploits the connections that deliver data messages between modules and, for this reason, i) it intrinsically reduces the number of links required for coordination and ii) it can be built without changing existing modules. The proposed architecture is discussed in detail and tested on a real scenario on the iCub humanoid robot.

I. Introduction

Recent approaches to robot programming in the literature [1], [2], [3] push the idea that software should be organized in modules each performing a well-defined, possibly simple, job and that complex tasks should be then solved by proper integration of a subset of these modules running concurrently. Integration and coordination in large systems is challenging and daunting task. The typical approach is to delegate the coordination to special objects that manage the activities of the individual behaviors. Central coordinators are difficult to reuse and implement robustly. This solution easily leads to brittle systems and scalability problems.

Programming robust applications considering, reactiveness, scalability and re-usability has always been at the center of attention of researchers. Different control architectures such as deliberative, reactive or hybrid [4] have been studied in a wide verities of robotic domain. Among them, the behavior-based approach inspired from Brooks' subsumption architecture [5] has particular interest due to its fast response to external events. Traditionally it has been used in robotic applications in which reactiveness is crucial. It requires however the addition of special connections that carry coordination signals; in distributed architectures these signals must also be synchronized with the ones that carry data. Nevertheless such approaches have been successfully used in mobile robotics [6], [7] (See section II).

We introduce a mechanism for coordination of behaviors in a distributed architecture. Our approach is based on port arbitration, in that it uses the same links which are already exploited to transfer data among behaviors. It therefore intrinsically reduces the number of extra connections needed for coordination. Differently from traditional behavior based architectures, modules are coordinated by arbitrating their connections. In other words inhibition of a module is achieved by suppressing the data it receives. Since modules can receive data from multiple sources, this allows a finer degree of granularity (i.e. a module can inhibit only a subset of the connections to another module thus allowing the latter to process data from other connections).

We implemented our approach using the YARP middleware [1] and tested it by developing a complex behavior on the iCub humanoid robot [8]. We show that using our approach we could implement a behavior that involves a sequence of actions by integrating pre-existing blocks and without the need to develop a special purpose module responsible for coordination. More importantly we developed our behavior incrementally.

Section II highlights some of the features of our proposed mechanism and compare it with the related approaches. Section III presents the problem addressed in the paper and describe the arbitration mechanism and its properties. Section IV suggests some policies for tuning the connection parameters. In Section V we describe how we have used our approach for implementing a specific behavior on the iCub humanoid robot and in Section VI we conclude the paper.

II. Related work

The original concept of behavior-based system is Brooks' subsumption architecture [5] in which reactive behaviors are used in a multilayer system and behaviors from higher (priority) levels can inhibit and suppress others. However, coordinating behavior solely based on inhibition tends to limit the flexibility and reusability of the system [4]. To overcome this limitation Maes [9] proposes a bottom-up selection mechanism in non-hierarchical network of behaviors. Coordination in [9] is done by using three kind of links between the behaviors (predecessor, successor and conflictor) and adjusting preconditions in which behaviors can operate. According to Tyrrell [10] and Hayashi et al [11] this mechanism is not well suited for human-like action selection problems since the binary values used for precondition result in a loss of information. [11] also proposes an action selection method based on motivation levels that resembles the dopamine system in animals. A continuous waveform of motivation signals are divided in different consciousness levels with some preassigned behaviors in each level. Based on the signal level, corresponding behaviors are chosen for execution. Since behaviors are statically prioritized, this approach imposes the same limitation of the subsumption architecture (i.e. a behavior may need to be in different consciousness level depending on the order in which it appears in a sequences of action). In contrast, our approach does not limit behaviors to be statically prioritized.

Different behavior selection mechanisms are compared in [12], [13] and [14]. An alternative approach to competitive action selection is a cooperative mechanism in which recommendations from multiple behaviors are combined to form a control action that represents their consensus. An example of this type of mechanism is DAMN [15]. It uses a centralized arbitrator to fuse the collected commands from different behaviors and select the action which best satisfies the prioritized goals of the system. Nowadays, due to heterogeneity of data type and the complexity of the control systems, the proposed methodology is practically limited to low-level control. However, in our approach, behaviors can help each other to become active by enabling the relevant connections. The centralized coordination mechanism has been successfully used in different applications, however it can encounter scalability problems due to the overhead associated to transferring a relevant amount of information over several links to the coordinator [16]. In contrast our approach does not use any central coordinator. Inspired by voluntary action selection in human [17], arbitration is done using regulated stimulation levels of the outputs of behaviors. Ayllu [18] is an architecture for distributed multi–robot behavioral control which allows standard port–arbitrated–behaviors interaction (message passing, inhibition, and suppression) to take place over IP networks. This architecture shares some concepts with our approach but it does not support important features such as stimulation and excitation.

Integrated Behavior–Based Control (iB2C) [19] is an architecture of behavior–based systems which supports a wide variety of action selection and coordination mechanisms such as priority–based and state–based arbitration, winner–take–all, superposition and voting. Coordination in iB2C is done by using separate signals for coordination (i.e. activation, stimulation, inhibition, target rating). This in fact introduces extra links between behaviors and causes extra overhead that could be non-negligible in a distributed system. In our approach coordination is performed using properties of connections and it exploits links already used to transfer data thus intrinsically minimizing the overhead. Another advantage of our approach is that it can be implemented at the level of the middleware and it does induce dependencies in the implementation of the individual modules.

Kertesz [20] introduces dynamic behavior network (DBN) which has some similarities to our approach such as stimulation, inhibition and excitation of the behaviors in a network. Behaviors in DBN use a set of preconditions and the stimuli from other connected behaviors to determine their actual states at any given time (normal, failed, activated, finished). In contrast, our approach does not change any conditions or internal states of behaviors to activate or deactivate them. Instead, a behavior can decide whether to accept an incoming data or

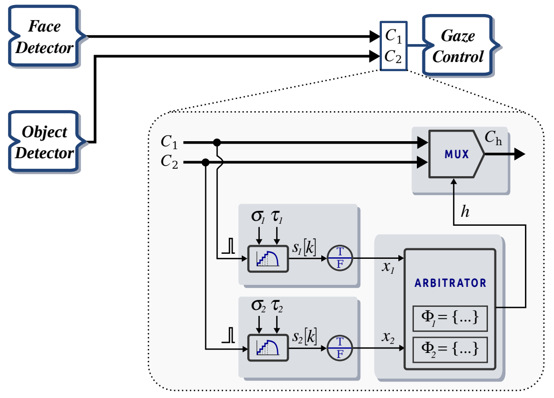


Fig. 1. Two behaviors `Face Detector` and `Object Detector` are connected to the same input interface (dotted box) of `Gaze Control` using connections C_1 and C_2. It shows how properties of the connections are employed by the port arbitrator of the `Gaze Control` to coordinate other behaviors. Please refer to the text for the definition of each symbol.

not based on the configuration of connections. Furthermore, our coordination mechanism allows behaviors to be really parallelized and distributed whereas in DBN parallelism is simulated and the computation of the stimuli is done in iterations. Similar to [19] and [20], in our approach the complexity of the problem can be decomposed in different subsystems. In section V we show that using our approach how a complex system is divided in subsystems that are configured, tested individually and finally combined together to implement the desired behavior.

III. Arbitrating behaviors

There is no concise definition of behavior in the literature. We refer to behavior as a computational unit with a set of preconditions and goals. It has a set of input ports to receive information from other behaviors and a set of output ports to stream out the result of its activity. A behavior checks for the condition in which to become active (e.g. upon receiving data), processes and sends the results through its output ports. We make the following assumptions for each behavior:

- The preconditions in which a behavior gets activated are local to the behavior itself and they are not visible to other behaviors. In other words behaviors cannot directly activate or deactivate others.
- Data is streamed out if and only if the behavior is active. For example, an object detector sends object position information through its output port only if the object has been detected.

We focus on the typical scenario of a publish-subscribe architecture in which modules (behaviors) can communicate asynchronously using connection points (ports). The key features we require are i) the output of a behavior can be connected to one or more input ports of other behaviors and ii) multiple outputs from different behaviors can be connected to the same input port of another behavior. For practical reasons we developed our architecture on top of the YARP middleware [1].

In the example from Figure 1, `Face Detector` and `Object Detector` can both send 3D position information to `Gaze Control` which controls the robot's head to gaze accordingly. Behaviors run in parallel and can be distributed over a cluster of computers that communicate through network interfaces. Since there is no synchronization among behaviors, data can be delivered to an input port at any time, potentially causing conflicts. For example, in a simple scenario where a person keeps an object in front of the robot, Face Detector and Object Detector compete to get data to Gaze Control. An appropriate coordination mechanism avoids conflicts and, at the same time, it allows obtaining different behaviors (looking at a face or looking at the object).

We propose an arbitration mechanism between multiple, competitive connections to the input port of a behavior. As an example Figure 1 illustrates the arbitration in the case of a port with two connection (here `Face Detector` and `Object Detector` are competitive connections to `Gaze Control`)[1]. Messages arrive to a port from different channels (connections) and generate events. Arbitration happens in three stages: computation of the activation values, evaluation of the rules and selection. First events are accumulated with leaky integrators to produce stimulation values for all connections. A connection becomes active when its stimulation level reaches a certain threshold. In the second stage, port arbitrator selects a single connection among the ones that are active at each time by evaluating a set of rules (i.e. written in first order logic) associated to each connection. This "winner" connection delivers data to the behavior whereas data from the other connections gets discarded. As it is shown in Figure 1, the port arbitrator is implemented as a multiplexer that let, at most, one active connection deliver its data to the component at each time.

We describe here the parameters of each connection. In the following sections we demonstrate how these parameters can be used to properly arbitrate multiple connections. Each connection C_i has the following parameters:

$$\Psi_i = < \sigma_i,\ \tau_i,\ \Phi_i > , \quad i \in \{1\ ..\ m\}$$

Ψ_i is a list of the properties of i^{th} connection (identified by C_i) to an input port with m connections. σ_i is the stimulation gain and τ_i is the damping time. Φ_i is the selection rule associated to the connection C_i. In the following sections we explain how the selection rules are represented in first order logic based on the activation state of the connections.

A. Computation of the activation values

Figure 2 illustrates how the activation value of each connection is computed. Arbitration cycles and data delivery happen at discrete events in time and are triggered whenever new messages arrive at the port from any connection C_i. Time values t_k $k = 0, 1, 2\ ..$. are associated to these discrete events using an internal clock.

There is a stimulation level $s_i[k]$ (at time t_k) associated to every connection C_i, $i \in \{1\ ..\ m\}$. All m stimulation levels in

[1]Notice that we are aware that several models for the control of attention have been proposed in the literature that are much more appropriate for this specific task. In this paper we use the control of gaze as an example to demonstrate and validate the proposed mechanism.

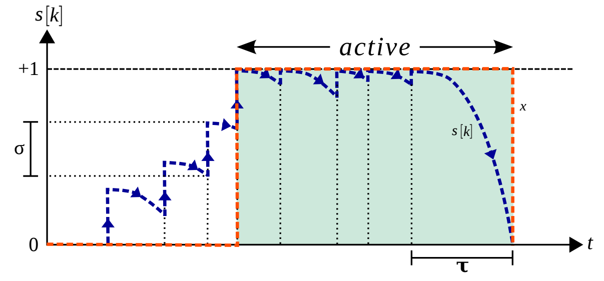


Fig. 2. Computation of the stimulation and activation levels for each connection. $s[k]$ is the stimulation level at time t_k. σ is stimulation gain and τ is damping time. Stimulation is accumulated every time a new message arrives and it continuously decays over time. The connection is in active state once cumulative stimulation reaches the threshold and until it gets completely discharged.

the port are updated at every instant k according to an exponential decay rule. In addition, the stimulation level $s_l[k]$ that corresponds to the channel l which has received the message, is increased by summing the corresponding stimulation gain σ_l, formally:

$$s_i[k] = \begin{cases} s_i[k-1] \cdot \left(1 - e^{\frac{\lambda\left(t_k - t_{k-1} - \tau_i\right)}{\tau_i}}\right) & \forall i \in \{1\ ...\ m, i \neq l\}, \\ s_i[k-1] \cdot \left(1 - e^{\frac{\lambda\left(t_k - t_{k-1} - \tau_i\right)}{\tau_i}}\right) + \sigma_i & i = l, \end{cases} \tag{1}$$

where λ and τ_i together define the decay constants of the exponential function. Next, s_i is saturated to be within $[0,1]$.

Equation (1) formulates the calculation of $s_i[k]$. When $s_i[k]$ reaches the threshold 1.0, the connection C_i is in *active* state until it gets completely discharged and decays to zero. To simplify notation we drop dependence from k and define the activation value x_i as:

$$x_i = \begin{cases} \textbf{true} & \text{if } C_i \text{ is } \textit{active}\ , \\ \textbf{false} & \text{otherwise.} \end{cases} \tag{2}$$

B. Representation of the rules

As we have mentioned previously, an active connection C_i (i.e. its $x_i = \textbf{true}$) has the opportunity to be selected by the arbitrator based on the selection rule specified by Φ_i. The rule simply provide the necessary constraint in term of activation values $\mathbf{x}$ for the connection C_i to be selected by the port arbitrator [2].

In the example from Figure 1, if data from connection C_1 of `Face Detector` is selected by the arbitrator at `Gaze Control`, the robot will gaze at the face. To gaze at a detected object, the port arbitrator should select the connection C_2 to receive the object position data from `Object Detector`. Suppose that in the example in Figure 1, we want the robot to track an object (continuously gaze at the object) when it appears in the view of the robot. This means that connection C_2 should be selected when it is in active state (i.e. $x_2 = \textbf{true}$), formally: $\Phi_2 = x_2$.

[2]A rule consistency validation is performed during the design time to ensure the rules specified in the port arbitrators do not contain contradiction.

Imagine now we also want to track the face of a person, but only if there is no object in the scene. In other words, connection C_1 should be selected if active, but only if C_2 is NOT active. Therefore the corresponding rule should be specified for C_1 and added to the arbitrator:

$$\Phi_1 = x_1 \wedge \neg x_2$$

In another scenario, suppose that we want the robot to gaze at an object if there is also a person in the scene, in terms of connections this means that connection C_2 should be selected if both C_1 and C_2 are active:

$$\Phi_2 = x_2 \wedge x_1$$

In this case, we are not interested in tracking the face. Therefore we add the following rule:

$$\Phi_1 = \textbf{false}$$

This specifies that data from connection C_1 is never delivered to the `Gaze Control`.

C. Selection Mechanism

Using the above equations, the selection mechanism is straightforward. When data arrives from connection C_i to an input port, the corresponding arbitrator has to decide whether to accept or discard it. First using equations 1 and 2 the activation values (x_i ... x_m) are updated. The rule specified in Φ_i is structured in Binary Decision Diagram (BDD) [21]. The arbitrator, then, evaluates the rule and if the constraints specified by the rule is satisfied, the index i of the current connection is given to the multiplexer which in turn opens the corresponding channel to deliver data from C_i to the behavior. Otherwise the data is discarded (In Figure 1, this "winner" connection is indicated by h). Notice that when the rules Φ_i are specified, a consistency check ensures that only a single connection can be active at a time.

IV. Parameters tuning and design policy

The selection mechanism using connection parameters allows for creating various complex scenarios using simple primitive behaviors. The stimulation gain σ and the damping time τ are used together to control the reactiveness and persistency of the output (the effect of the behavior). As an example consider a behavior which checks the sensors of the fingertips of a robot and sends events whenever the robot touches an object. If these events are used by another behavior for collision avoidance in a safety context, a higher value for σ should be chosen to obtain a prompt reactive behavior. Alternatively, if the output is used to stimulate a less time–critical behavior (i.e a grasping action), smaller value of σ is more preferable to collect enough evidence from the fingers before grasping the object. Stimulation gain σ should be used together with damping time τ. It should be clear that if stimulation rate (the elapsed time between two consecutive spikes) is larger than damping time τ (i.e. data infrequently arrives with a large delay), the stimulation level s_i can never reach the threshold.

V. Experimental validation

In the following section we present an experiment with the iCub humanoid robot. The main goal of the experiment is to demonstrate that our port-arbitrated mechanism allows for i) coordinating different distributed behaviors which compete to control the robot's actuators, ii) breaking down a complex system in subsystems that are configured, tested individually and finally combined together and iii) implementing a system that is reactive to the changes in the environment. We call this behavior "Take and return". The robot should perform a series of actions: (A) look for an object, (B) reach for the object, (C) grasp the object, (D) look for a person, (E) approach the person and (F) release (return) the object.

A. First experiment: Take an object

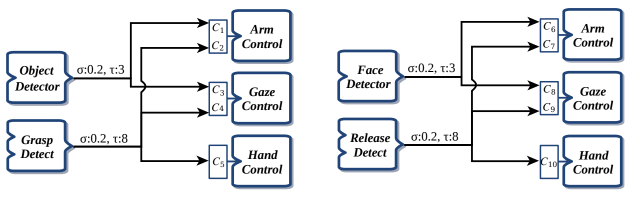
In the first experiment we combine some simple behaviors to build a system which allows the robot to take an object. The user shows a known object to iCub and the robot tracks it with the eyes and grasps it by the hand. Figure 3(a) represents the behaviors and the configuration of the connections. For the sake of brevity, non–arbitrated connections are not shown in the figure (e.g. camera inputs). `Object Detector` receives streamed image frames from the robot cameras and produces the 3D position of the object when detected. `Gaze Control` and `Arm Control` receive a 3D position in the robot root frame and respectively control the head of the robot to gaze at the target and move the hand of the robot to the target position. `Grasp Detect` monitors the positions of the object and the hand to determine when they are close enough and issue a request to grasp. `Hand Control` controls opening and closing of the hand upon receiving release or grasp command.

As shown in Figure 3(a), the output of `Object Detector` is connected to `Arm Control` and `Gaze Control` which causes the robot to track and attempt to reach for the object. The stimulation level $\sigma = 0.2$ and damping time $\tau = 3$ indicate that tracking and reaching should be started if there are enough events from `Object Detector` (i.e. at least 5 events within 3 seconds). Simultaneously, `Grasp Detect` checks if the object is graspable and if this is the case, it generates a grasp command to `Hand Control`. Table I represents the arbitration rules associated to each connection from Figure 3(a).

Constraint $\Phi_1 = x_1 \wedge \neg x_2$ implies that data from connection C_1 should be selected if C_2 is inactive. $\Phi_2 = \textbf{false}$ specifies that data from `Grasp Detect` will be never delivered to the `Arm Control`. The same rules are applied for C_3 and C_4 in the arbitrator of `Gaze Control`. These are used to inhibit commands from `Object Detector` and therefore prevent the motion of the hand and head of the robot while grasping the object. $\Phi_5 = x_5$ implies that grasp commands from `Grasp Detect` can be freely sent to the `Hand Control` for grasp whenever the object is graspable.

TABLE I. Arbitration rules for "Take an Object"

	Arm Control	Gaze Control	Hand Control
Φ_1	$x_1 \wedge \neg x_2$	-	-
Φ_2	**false**	-	-
Φ_3	-	$x_3 \wedge \neg x_4$	-
Φ_4	-	**false**	-
Φ_5	-	-	x_5



(a) Take an object (b) Return an object

Fig. 3. Configuration of the behavior network in “Take an object” and “Return an object” scenarios.

TABLE II. ARBITRATION RULES FOR “RETURN AN OBJECT”

	Arm Control	Gaze Control	Hand Control
Φ_6	$x_6 \land \neg x_7$	-	-
Φ_7	**false**	-	-
Φ_8	-	$x_8 \land \neg x_9$	-
Φ_9	-	**false**	-
Φ_{10}	-	-	x_{10}

B. Second experiment: Return an object

In the second experiment we build another network of behaviors that allows the robot to return an object to the user (assuming it has grasped it). The configuration of the network is very similar to the previous experiment. As shown in Figure 3(b) `Face Detector` now searches for a human face in the images and, when successful, it provides its 3D position to `Arm Control` and `Gaze Control`. This causes the robot to track the face and to extend the arm towards it. `Release Detect`, generates release commands to `Hand Control` if the hand is pointing toward the face, causing the robot to release the object. Table II represents the required arbitration rules to implement the scenario. Similar to the “Take an object” scenario, during release of the object `Release Detect` inhibits the movements of the arm and the head. Notice that for simplicity this behavior assumes that the robot has grasped the object. The module that explicitly checks this condition will be added in the next section.

C. Third experiment: Take and return scenario

In the final experiment (“Take and return”), we exploit behaviors implemented previously for “Take an object” and “Return an object”. Since we want the robot to return the object only if it has previously grasped it, we first introduce a new behavior `Grasped` which combines the information from the touch sensors and the hand encoders to produce a status message when the fingers are closed and the presence of an object is detected in the hand. The output of `Grasped` should inhibit all the connections in “Take an object” and enable “Return an object”. Figure 4 represents the configuration of behaviors. The connections that were previously used in “Take an object” and “Return an object” are shown here in gray. This emphasizes the fact that it is possible to build complex behaviors incrementally using existing subsystems without modifications. This is done by adding constraints to the arbitration rules of the subsystems. For example in Figure 4 we want to inhibit tracking when the robot is holding the object. This is achieved by adding the constraints “$\neg x_{12}$” to Φ_3 in the subsystem “Take an object”. This has the effect of inhibiting data from `Object Detector` (i.e. C_3) when the output of `Grasped` (i.e. C_{12}) is active. The necessary rules for inhibiting other connections

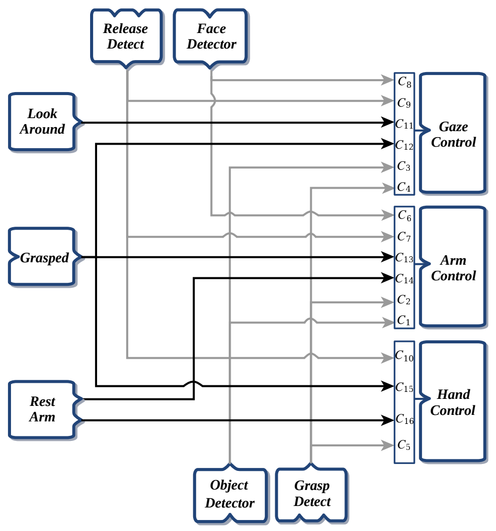


Fig. 4. Configuration of behaviors in “Take and Return” scenario.

TABLE III. ARBITRATION RULES FOR “TAKE AND RETURN”

	Arm Control	Gaze Control	Hand Control
Φ_1	$(x_1 \land \neg x_2) \land \neg x_{13}$	-	-
Φ_2	**false**	-	-
Φ_3	-	$(x_3 \land \neg x_4) \land \neg x_{12}$	-
Φ_4	-	**false**	-
Φ_5	-	-	$x_5 \land \neg x_{15}$
Φ_6	$(x_6 \land \neg x_7) \land x_{13}$	-	-
Φ_7	**false**	-	-
Φ_8	-	$(x_8 \land \neg x_9) \land x_{12}$	-
Φ_9	-	**false**	-
Φ_{10}	-	-	$x_{10} \land x_{15}$
Φ_{11}	-	$\neg(x_3 \lor x_4 \lor x_8 \lor x_9)$	-
Φ_{12}	-	**false**	-
Φ_{13}	**false**	-	-
Φ_{14}	$\neg(x_1 \lor x_2 \lor x_6 \lor x_7)$	-	-
Φ_{15}	-	-	**false**
Φ_{16}	-	-	$\neg(x_5 \lor x_{10})$

in “Take an object” to `Arm Control` and `Hand Control` are added similarly. Notice that arbitration rules are added (and removed) by specifying connection parameters to the port arbitrators and without changing the code implementing the individual modules. Table III represents the full list of required rules to implement the “Take and Return” scenario using behaviors from Figure 4.

We now add modules to put the arm in a resting position and randomly look around in search for the object. The module `Look Around` that sporadically sends random position commands to `Gaze Control` in search for objects or faces. This behavior must be clearly inhibited while “Take an object” and “Return an object” are active. To do so we add the necessary constraints to the arbitration rule Φ_{11} of connection C_{11} (i.e. $\Phi_{11} = \neg(x_3 \lor x_4 \lor x_8 \lor x_9)$). `Rest Arm` attempts to move the arm to a predefined resting position by periodically sending release commands and arm–resting–position commands to `Arm Control` and `Hand Control`. Appropriate constraints are added to Φ_{14} and Φ_{16} to inhibit this behavior when the robot is taking or returning the object.

The arbitration mechanism was implemented using YARP [1] ports. The “Take and return” scenario was then

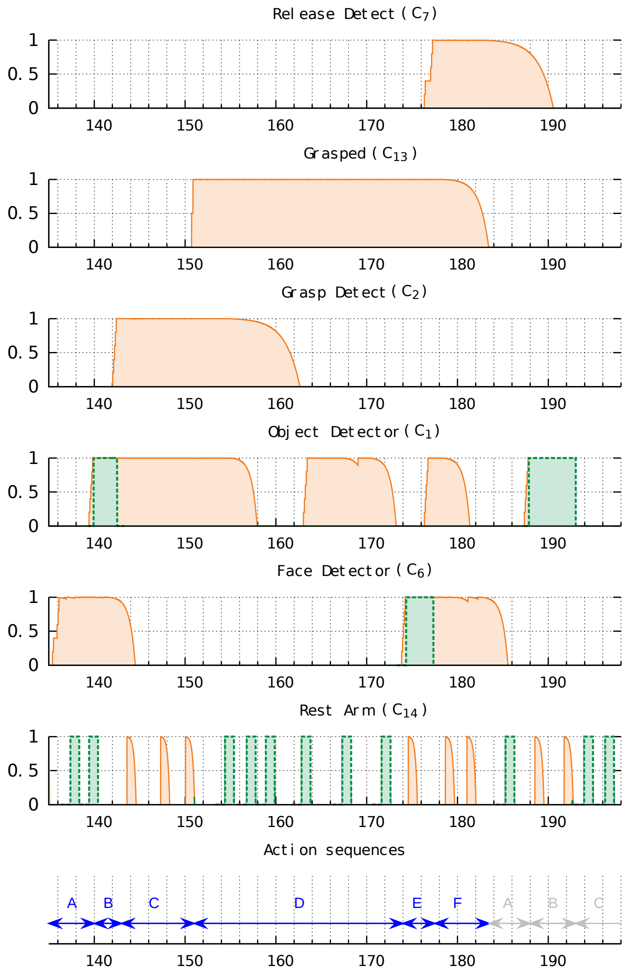


Fig. 5. Arbitration in `Arm Control` during the "Take and Return" scenario. The plots show the values of $S_i[k]$ and the selected connection by the arbitrator of all the m input connections from the modules `Release Detect`, `Grasped`, `Grasp Detect`, `Object Detector`, `Face Detector` and `Rest Arm` (see Figure 4). Horizontal axis is time t and vertical axis represents stimulation values $S_i[k]$. Stimulation are plotted in green when the corresponding connection is selected and in orange otherwise. The bottom graph shows the action of the resulting behavior, i.e. **A**: looking for an object, **B**:reaching the object, **C**:grasping, **D**:looking for a person, **E**:approaching person and **F**:releasing the object.

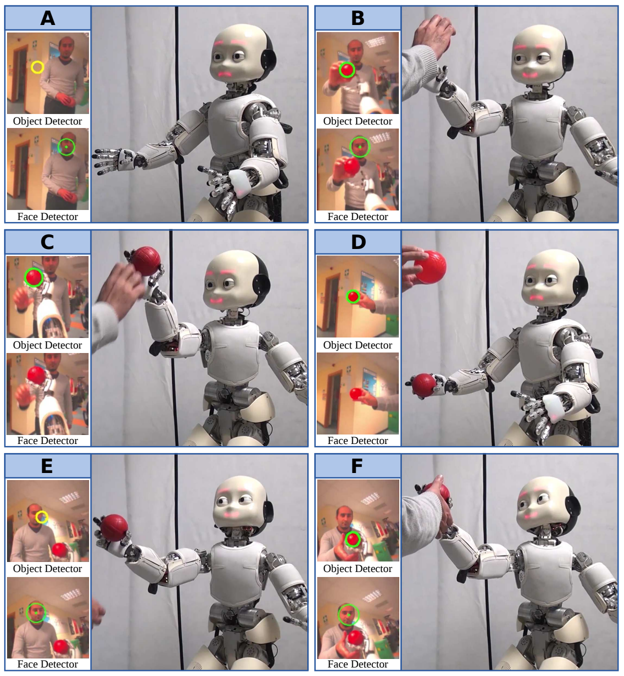


Fig. 6. iCub performing "Take and Return" scenario. In **A**: the robot is looking for the object, **B**: reaching for the object, **C**: grasping the object, **D**: looking for a person while holding the object, **E**: approaching the person and **F**: releasing the object.

tested on the iCub robot (see Figure 6) [3]. Figure 5 plots the stimulation values of all the connections to `Arm Control` during the experiment. Stimulation are plotted in green when the corresponding connection is selected and in orange otherwise. At the beginning and before $t = 140$, the hand of the robot is empty (`Grasped` is inactive) and `Rest Arm` can commands `Arm Control` and `Hand Control` to keep the robot arm in a resting position. This corresponds to the behavior A in which the robot is looking for the object. Notice that at the same time, `Face Detector` is also activated (a person enter the scene); however since the selection of `Face Detector` depends on the output of `Grasped` (i.e. $\Phi_6 = (x_6 \wedge \neg x_7) \wedge x_{13}$), the required constraints are not satisfied and data from `Face Detector` is not delivered to `Arm Control`. At $t = 140$, the person shows the object to the robot; this increases the simulation level of `Object Detector` to activation and the robot reaches for the object (this corresponds to behavior B). At $t = 142$, `Grasp Detect` is stimulated; this prevents the robot's arm from moving and at the same time sends grasp commands to `Hand Control`. This corresponds to behavior C in which the robot grasps the object. Notice that during B and C `Rest Arm` is also inhibited. At this point the robot holds the object (behavior D), `Grasped` is active. The robot cannot reach for another object. This can be noticed in the plot of the activation value C_1 corresponding to the module `Object Detector`: the latter is stimulated but its output gets inhibited by `Grasped`. At $t = 174$, the person enters again in the scene. `Face detector` is now activated and can be selected by the arbitrator because `Grasped` is active. This makes the robot move the hand towards the face (behavior E). At $t = 178$ `Release Detect` gets activated; this causes the robot to release the object (behavior F). It also inhibits commands from `Face detector` and `Rest Arm`.

During the "Take and return" experiment the behavior of the robot was tested under different conditions (e.g. by showing a face before grasping or by showing the robot another object after grasping). These tests demonstrated that the overall behavior is intrinsically reactive to the environment. All behaviors, in fact, continuously monitor the conditions under which they are activated. Unexpected situations are thus automatically handled by the network of behaviors, even if they were not explicitly considered at design time. A particular situation in the experiment explicitly demonstrated this. While the robot was returning the object, the user decided to anticipate the robot and took the object directly from the hand before the reaching command was completed. As a consequence, the

[3]The video demonstrates this behavior can be accessed on-line at: http://youtu.be/SzHRLm2z47I.

stimulation of `Grasped` decreased and prevented all behaviors in "Return an object" to run. Finally the output of `Rest Arm` were no longer inhibited and could command the arm to go back to the initial state. The system therefore went back to the initial state (A).

VI. Conclusion

This article has introduced an action selection mechanism for a network of behaviors based on port arbitration. We have illustrated that our approach allows to implement a non-trivial behavior that involves a sequence of actions. Remarkably, we have shown that the final behavior can be incrementally built as a composition of existing, simpler behaviors. Our approach is also fully distributed and minimizes the additional links required to perform arbitration. We tested the behavior in different conditions and demonstrated that the resulting behavior is intrinsically robust and reactive to unexpected changes in the environment. Perhaps more importantly, since no explicit modules are required to manage the coordination, no task dependent code was written to implement the final behavior which as a result was exclusively built out of reusable modules.

The work presented is a first one in this direction and we are aware of certain limitations. Firstly our approach requires setting a certain number of parameters and rules. In the examples reported in the paper this was in practice quite intuitive to do; however this may not scale well with the number of modules and the complexity of the behavior. To facilitate writing the rules we are studying a technique that allows extracting arbitration rules automatically from a higher level behavioral representation. Secondly, since coordination is completely distributed, it is also more difficult to characterize the current state of the system and detect or monitor the behavior of the system as a whole. This information is however present in the system and it could be made available by advertising the connection parameters (although this is not implemented in the current system).

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7 ICT) under grant agreement No. 270273 (Xperience). The authors would like to thank all the partners of the project for their valuable comments.

References

[1] L. Metta, G. Fitzpatrick, P. and Natale, "YARP: Yet Another Robot Platform," *International Journal on Advanced Robotics Systems*, vol. 3, no. 1, pp. 43–48, 2006.

[2] H. Bruyninckx, "Open robot control software: the OROCOS project," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3. IEEE, 2001, pp. 2523–2528.

[3] D. Brugali and P. Scandurra, "Component-based Robotic Engineering Part I : Reusable building blocks," vol. XX, no. 4, pp. 1–12, 2009.

[4] D. Nakhaeinia, S. H. Tang, S. B. Mohd Noor, and O. Motlagh, "A review of control architectures for autonomous navigation of mobile robots," *Int. J. Phys. Sci*, vol. 6, no. 2, pp. 169–174, 2011.

[5] R. Brooks, "Intelligence without representation," *Artificial intelligence*, vol. 47, no. 1, pp. 139—-159, 1991.

[6] B. Woolley and G. Peterson, "Real-time behavior-based robot control," *Autonomous Robots*, no. January, pp. 233–242, 2011.

[7] C. Armbrust, M. Proetzsch, B. H. Schäfer, and K. Berns, "A Behaviour-based Integration of Fully Autonomous, Semi-autonomous, and Teleoperated Control Modes for an Off-road Robot," in *Proceedings of the 2nd IFAC Symposium on Telematics Applications, Timisoara, Romania*, 2010.

[8] G. Metta, G. Sandini, and D. Vernon, "The iCub humanoid robot: an open platform for research in embodied cognition," *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pp. 50—-56, 2008.

[9] P. Maes, "How to do the right thing," *Connection Science*, 1989.

[10] T. Tyrrell, "An Evaluation of Maes's Bottom-Up Mechanism for Behavior Selection," *Adaptive Behavior*, vol. 2, no. 4, pp. 307–348, Mar. 1994.

[11] E. Hayashi, K. Ueyama, and M. Yoshida, "Autonomous action selection with motivation-based consciousness and behavior architecture of animal," *The 5th International Conference on Automation, Robotics and Applications*, pp. 294–299, Dec. 2011.

[12] P. Pirjanian, "Behavior coordination mechanisms-state-of-the-art," *Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, Tech. Rep. IRIS-99-375*, 1999.

[13] J. M. MacKenzie, Douglas C and Arkin, Ronald C and Cameron, "Multiagent mission specification and execution," *Robot colonies*, pp. 29–52, 1997.

[14] M. Scheutz and V. Andronache, "Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 34, no. 6, pp. 2377–2395, 2004.

[15] J. K. Rosenblatt, "DAMN: a distributed architecture for mobile navigation," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 339–360, Apr. 1997.

[16] T. J. Prescott, "Action selection," *Scholarpedia*, vol. 3, no. 1, p. 2705, 2008.

[17] J. Zhang, "Selection and inhibition mechanisms for human voluntary action decisions," *NeuroImage*, Jul. 2012.

[18] B. B. Werger, "Ayllu: Distributed port-arbitrated behavior-based control," in *In Proc., The 5th Intl. Symp. on Distributed Autonomous Robotic Systems*. Citeseer, 2000.

[19] M. Proetzsch, T. Luksch, and K. Berns, "Development of complex robotic systems using the behavior-based control architecture iB2C," *Robotics and Autonomous Systems*, vol. 58, no. 1, pp. 46–67, 2010.

[20] C. Kertész, "Dynamic behavior network," *Applied Machine Intelligence and Informatics*, pp. 207–212, 2012.

[21] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, pp. 677–691, 1986.