8 Control Architectures

The generation of designated robot behavior is one of the most difficult problems when designing the control system for robotic applications with many sensors and actuators. Due to the diversity of tasks an autonomous vehicle has to fulfill, the control has to be embedded into a convenient framework. The process of building up a control system should be supported by an adequate methodology to help overcoming difficulties common to complex robotic systems, e. g. ensuring secure operation, modularity, or handling a system of growing complexity. Therefore, different types of control architectures have appeared with contrary approaches for tackling the emerging problems.

A control architecture is a framework enabling a system to fulfill the following tasks:

Fusion of sensor data into logical sensors Sensor data has to be preprocessed for usage for localization, generation of obstacle maps, generation of maps for navigation and planning, object recognition, display and representation of knowledge.

Motor controller Access to the hardware has to be provided by the realization of a convenient motion control interface, e. g. velocity v and angular velocity ω.

Pilot function A Pilot realizes the path control via specification of motion commands which are required for e. g. collision avoidance, driving through narrow passages, turning in dead-end situations.

Navigation The navigator calculates accessible tracks to be traversed via the pilot function. The plan includes avoidance of obstacles and requires knowledge about the surrounding area.

Planning function A planning component generates actions and sets targets for the navigation component. This includes strategic decisions and keeping the overview concerning given tasks.

User interaction Access to the control software has to be provided by a suitable Human-Machine-Interface (HMI).

For all tasks mentioned above world knowledge has to be considered, which has to be provided adequately.

In general, control architectures in the field of robotics can be distinguished into hierarchical vs. distributed and task-oriented vs. behavior-based (see figure 8.1).

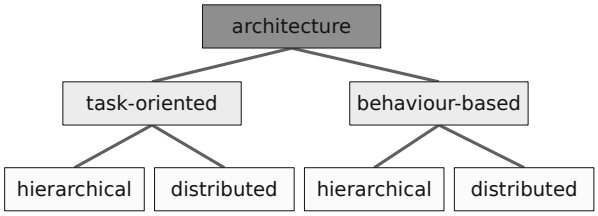

Figure 8.1 Overview over control architectures commonly used in the field of robotics

Hierarchical architectures depend on the assumption that tasks can be divided into subtasks which are arranged so that higher level components generate subgoals for lower level components. In contrast to this, distributed architectures allow the assignment of subtasks to independent components. A suitable communication mechanism is required for data transfer between the parts involved.

The distinction between task-oriented and behavior-based control architectures is reflected in the decomposition of a given task. Task-oriented architectures depend on a central world model which is manipulated and evaluated by the different components (sensing, modeling, planning, execution). The tasks of processing sensor data as well as generating control values for the robot are encapsulated into responsible components with exclusive access. Behavior-based control architectures, however, are designed by decomposing a given task into independent behaviors, each of which keeps its own compact representation of the environment which is required for task execution. Here all components have unlimited access to sensor data and to the control interface. This, however, requires mechanisms for the coordination of conflicting data.

Besides the given distinction of control architectures into task-oriented vs. behavior-based and hierarchical vs. distributed, [Mat97] emphasizes the difference of architectures in respect to the degree of deliberation, see figure 8.2. Reactive architectures not supporting knowledge storage are contrasted with deliberative architectures with an elaborate model of the world. The former supports immediate reaction on occurring situations but has

weaknesses in respect to planning tasks requiring a certain degree of memory. The latter is suited to complex tasks but has drawbacks concerning outdated knowledge, false sensor readings, or reaction time.

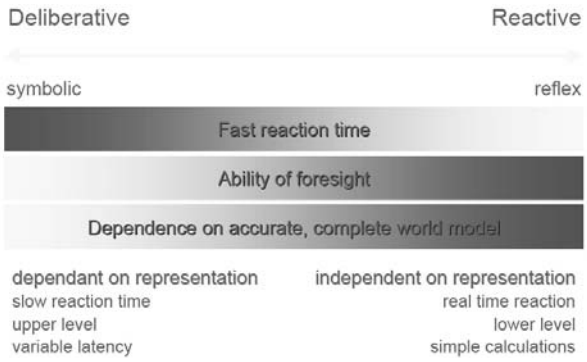

Figure 8.2 Properties of deliberative and reactive control

In order to combine the advantages of both characteristics, hybrid architectures have emerged with a lower reactive layer and a higher deliberative one. This, however, breaks the control system into two inhomogeneous parts with different characteristics. In contrast to this, behavior-based control systems store a representation of the environment which is distributed among the single components and therefore can combine reactive and deliberative components into one architectural design.

In the following, a task-oriented as well as several behavior-based control architectures are discussed in more detail.

8.1 The hierarchical task-oriented control architecture RCS

As an example of a hierarchical task-oriented control architecture, the Real-Time Control System (RCS) by James S. Albus [Alb92] is presented here. Since its beginning in the 1970s at the NIST (National Institute of Standards and Technology), the architecture has constantly advanced and been applied to several projects, e. g. the DEMO III project [LMD02].

RCS is composed of hierarchically arranged layers consisting of one or more RCS nodes. As depicted in figure 8.3, each of them has 4 modules: Behavior Generation (BG), World Modeling (WM), Sensory Perception (SP) and Value Judgment (VJ). Additionally, the Knowledge Database (KD) and the Operator Interface (OI) are provided.

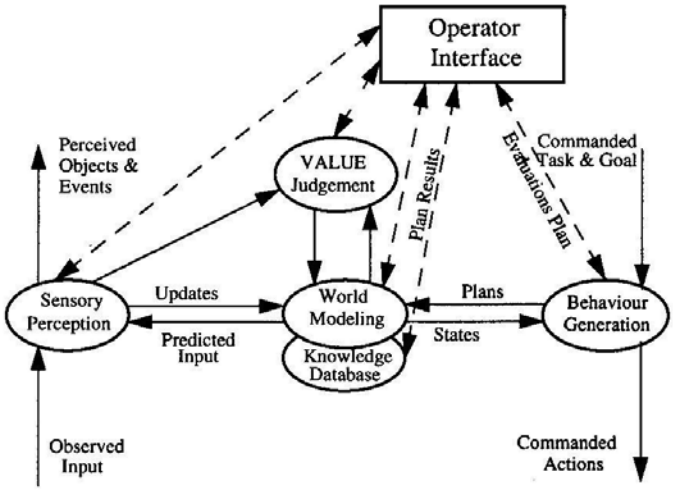

Figure 8.3 A RCS-4 node and the data flow between the included modules [Alb92]

The BG module plans and controls the actions of the system. For this purpose a complex plan is decomposed into simpler tasks by using the information provided by the higher layer as well as WM and VJ of the same layer.

The SP module processes sensor data and compares them with predictions of the WM module. Additionally, SP perceives objects, events, and situations and transfers them to the WM module.

The WM module uses information given by SP to update the Knowledge Database. Predictions of sensor values as well as the simulation of plans proposed by the BG module are additional tasks of this module.

The VJ module calculates cost, risk, and benefit of simulated plans. It distinguishes between important and irrelevant objects and events and transfers the results to the BG module.

The Knowledge Database contains data about the environment. Each of the modules has access to the KD, either directly or indirectly via the WM module.

Finally, the OI is the interface for observing or manipulating the system behavior by a human operator.

RCS supports two kinds of communication. On the one hand, communication between RCS nodes of different layers involves the transfer of new tasks to the BG module of the lower layer and newly perceived objects and events to the SP module of the higher layer. On the other hand, modules inside a RCS node communicate as depicted in figure 8.3.

This results in two main data flow directions: One top down information flow consisting of tasks and goals between the BG modules and one bottom up information flow consisting of processed sensor data. Due to the growing complexity of higher layers, the higher a layer is, the longer the cycle takes. An example for a timing diagram is depicted in figure 8.4

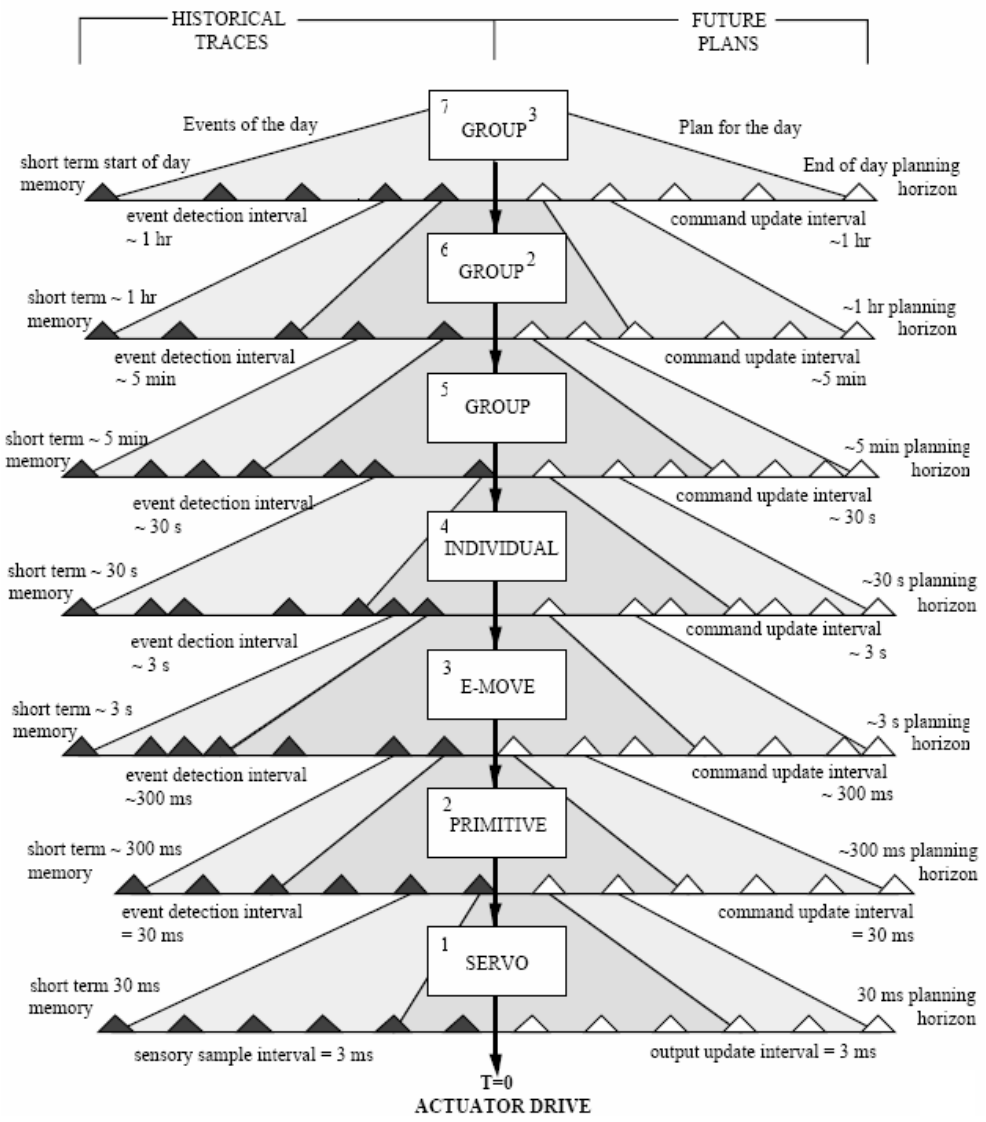

Figure 8.4 Example for the time horizon of different layers in RCS [Alb92]

Here, seven layers are covering a time range between milliseconds and days. The horizontal axis consists of historical traces (left) and future plans (right). Both of them are processed within the same time scale.

While the applicability of hierarchical task-oriented control architectures like RCS was shown in several applications, some disadvantages have emerged. At first the control depends on a consistent world model. However, this central representation of the environment is prone to errors due to false sensor readings or outdated information. Here the immediate usage of sensor data leads to a more responsive and correct behavior.

Second, the functionality of the whole system depends on the proper operation of all components. A different approach, in which malfunctions of single modules can be caught by others is the behavior-based approach, presented in the next section.

Finally, hierarchical task-oriented control architectures tend to limit the extensibility of the system as a new functionality is reflected in the change of a multitude of existing components. Therefore, the scalability of these systems is limited.

8.2 Behavior-based control architectures

The development of robotics was heavily influenced by paradigms in Artificial Intelligence (AI) like the following (Marvin Minsky [MMN55]): "[An intelligent machine] would tend to build up within itself an abstract model of the environment in which it is placed. If it were given a problem it could first explore solutions within the internal abstract model of the environment and then attempt external experiment." Therefore, knowledge representation, planning reasoning and hierarchical composition were the main fields of research reflecting the human understanding of intelligence (e. g. STRIPS [FN71], ABSTRIPS [Sac74], NOAH [Sac75]).

Inspired by observations in biology, these traditions were questioned. According to Brooks, "planning is just a way of avoiding figuring out what to do next" [Bro87]. Therefore, a shift in paradigm took place, moving from sensing and acting to behavior-based robotics where preferably simple agents show intelligence through coordinated behavior.

The motivations for this change were the following:

- Complex behavior does not necessarily arise from complex control systems.
- The real world is the best model.

- Programming should be kept simple.
- Systems should show robustness at noisy sensor readings.
- Systems should provide the possibility of incremental design.
- All calculations should be performed on-board and therefore fit the machine time available.

The motivation for the change from task-oriented to behavior-based control would be best expressed by Thomas Huxley: "The great end of life is not knowledge, but action".

The following examples of behaviors could appear in wheel-driven or legged mobile autonomous robots:

- Exploration (movement in a general direction)
- Targeted (movement in direction of attractors)
- Avoidance (avoid collision with objects and environment)
- Path following (e. g. wall, planned path, stripe)
- Posture control (balance, stability)
- Social behavior (e. g. parting, hives, flocks)
- Remote-autonomous behavior (user interaction, coordination)
- Perceptual behavior (visual search, . . .)
- Walking behavior
- Manipulator behavior, grip behavior

In contrast to task-oriented control architectures, behavior-based approaches have proven to handle emerging difficulties rather well. They do not depend on the correctness of a central world model, make it easy to incrementally add functionality while handling increasing complexity and show robustness to unknown sensor data due to an overall functionality emerging from the interaction of multiple generalizing behaviors.

Still, the problem of controlling complex robotic systems is not solved by the behavior-based paradigm alone. Rather, while helping with some common problems, behavior-based architectures introduce new difficulties. Among those is the question of how to coordinate multiple and possibly competing behaviors running in parallel and trying to act on the same actuators. Another issue is the identification of error sources in a control that shows an emergent system behavior rather than an explicitly implemented one. Also, there is the matter of how the architecture can help structuring the design process, e. g. giving support in the process of selecting the best set of behaviors and coordinating their action.

Several variants of behavior-based architectures have been developed in order to tackle the presented problems. In the following section, the Subsumption Architecture by R. Brooks [Bro86], reuse and temporal sequences of behaviors by Nicolescu and Mataric [NM00, NM02], and the iB2C[1] architecture of the Robotics Research Lab in Kaiserslautern are presented. Other behavior-based architectures include R. Arkin's works [Ark89, Ark98] on schema-based and potential field approaches, further methods by M. Mataric [Mat92, Mat97], miscellaneous fuzzy approaches [LRM94, SKR95, KS97, Ros97, SDC05], the Dynamical System Approach by Althaus/Christensen [AC02], Behavior Oriented Design [Bry01], parallel behavior execution without action selection mechanism [Ste94], or activation-based [BILM03, CA07] as well as neural-network-based architectures [FM96, Bee96].

8.2.1 The Subsumption Architecture

The first architecture implementing the presented ideas was the Subsumption Architecture developed by Rodney Brooks, MIT, 1986 [Bro86]. He proposed the alignment of behaviors along horizontal layers, with all behaviors having access to all sensor data to generate actions for all actuators. The interaction of behaviors is carried out through interdiction of inputs and overruling of outputs. The primary feedback between behaviors occurs via the environment. Using the special C derivative `interactive C` allows for system state analysis during runtime. The approach was applied to several mobile vehicles as well as to the walking machines Genghis and Attila.

This approach has many advantages. Inherently, the architecture supports running behaviors asynchronously in parallel while data connections can be prone to errors and delays. Due to the independence of the components, extending the system with sensors and behaviors is directly supported.

[1] iB2C: integrated behavior-based control

The implementation of the individual modules is kept simple and the system shows a certain robustness in case of malfunction of behavior modules or behavior levels. Finally, simultaneously achieving multiple goals can be easily implemented.

The structure of the basic elements of the Subsumption Architecture is shown in figure 8.5. Each of them has input and output lines. Via a suppressing mechanism, input signals can be replaced by the suppressing signal given into the circle. Output signals of modules can be inhibited, i. e. any output signal is blocked for the given time. Each module internally implements a finite state machine which can be reset to state NIL via the corresponding input.

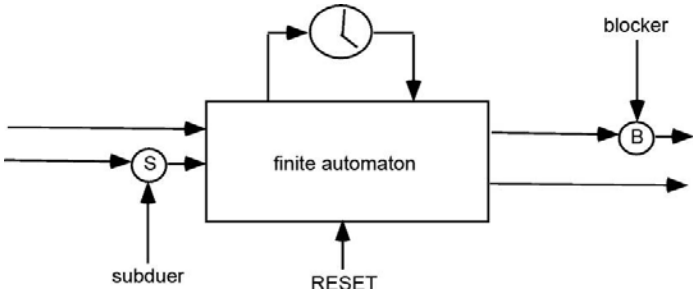

Figure 8.5 Basic module of the Subsumption Architecture

The system design follows the structure given in figure 8.6. Its control is split up into layers with higher layers subsuming the functionality of lower level layers when desired. The system can be partitioned at any layer so that the lower layers form a complete operational system. An example of the functionality of layers is depicted in figure 8.7.

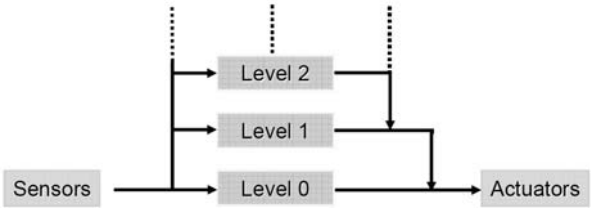

Figure 8.6 Layout of the behavior levels in the Subsumption Architecture

This architecture has proven suitable for small systems and has successfully managed navigation in an office environment with many obstacles. However, this approach tends to run into scalability problems due to limitations concerning the amount of internal representation. Also, the reuse

of components is often not possible and weaknesses occur when behaviors are added to an existing system. Therefore, further enhancements have been undertaken in the years after the introduction of Subsumption Architecture.

Figure 8.7 Example for the functionality of layers in the Subsumption Architecture

8.2.2 Reuse and temporal sequences of behaviors

The work of Nicolescu and Mataric (see [NM00, NM02]) identifies common deficiencies of behavior-based architectures: the difficulties in reusing existing behaviors for new tasks, the inability to easily realize temporal sequences of behavior activations, and the lack of support for the automatic creation of behavior networks.

The first problem is addressed by splitting up a behavior into an "abstract" and a "primitive" part, with the former constituting the interface of a behavior and the latter containing the actual functionality (see figure 8.8). The abstract part also contains checks to ensure that the preconditions of the behavior are fulfilled. So when using a behavior for different tasks with different preconditions, the primitive part can be reused and only the interface part has to be altered.

Behavior networks can be created by connecting abstract behaviors. This can either be done by hand or by an algorithm which analyzes the preconditions and effects of behaviors and uses backtracking to create a network for a specific task. Dealing with large networks containing many behaviors is facilitated by the support for grouping behaviors and thus building hierarchical networks. The architecture provides different types of connections between behaviors, allowing the creation of temporal sequences. By encoding task-specific aspects into behavior connections instead of directly into the behaviors, behaviors can be built in a more general way and thus be reused more easily.

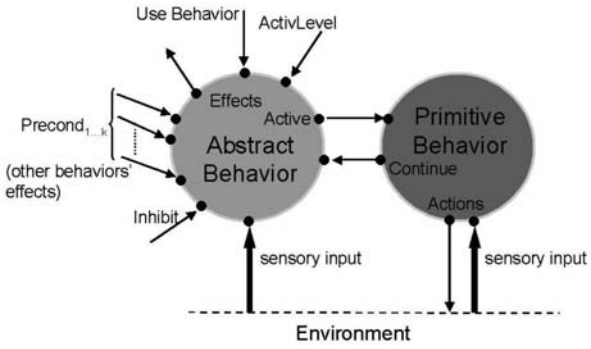

Figure 8.8 An abstract and a primitive behavior and the connection between the two [NM02].

[NM03] describes an extension of the architecture that allows robots to learn tasks from a robot or human teacher. During a demonstration phase, the learning robot makes observations and logs which of his behaviors can be used to achieve a certain situation. In a second phase, it uses these observations to build a behavior network for fulfilling the demonstrated task. So what is learned is when to use which behavior. New behaviors are not learned.

8.3 The integrated behavior-based control architecture iB2C

As an example for a behavior-based control architecture which addresses the issues mentioned before, the iB2C architecture of the Robotics Research Lab (University of Kaiserslautern) is presented here [PLB10]. The goal is to find a behavior-based architecture of modular structure with control units ranging from motor schemes up to deliberative planning layers. Common interfaces help in reusing and easily adding modules, and the arbitration mechanism allows for separating the control data from the coordination data flow.[2] The architecture is applicable to a wide range of robotic systems. It also shows design guidelines to simplify the creation of a consistent, robust and maintainable system. Last but not least, a programming framework supports the implementation by providing suitable tools for designing, debugging and

[2] Here control data is referred to as values for actuators, e. g. velocity, while the coordination data flow provides information about behavior states and is used for the internal behavior interaction of the system.

inspecting a control system of growing complexity. The proposed architecture is a further development of the behavior-based control as previously introduced in [ALBD03], where it has mainly been used to control walking machines.

8.3.1 The basic behavior module

The fundamental unit of the proposed control architecture is the behavior module (see figure 8.9) which is based on [Alb07] and [PLB05]. Each atomic behavior is wrapped into such a module with a uniform interface.

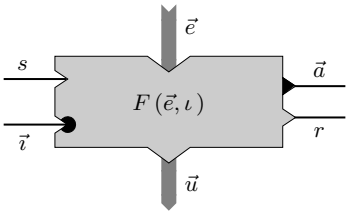

Figure 8.9 Basic iB2C behavior module

Behaviors can be described as three-tuples of the form

$$B = (f_a, f_r, F) \tag{8.1}$$

where f_a is the *activity function*, f_r is the *target rating function*, and F is the *transfer function* of the behavior. These functions generate *activity* information $\vec{a}$, a *target rating* r, and an *output vector* $\vec{u}$, respectively. Additionally, each behavior receives an *input vector* $\vec{e}$, a *stimulation* s, and an *inhibition vector* $\vec{\imath}$. In the following, these characteristics are explained in more detail.

Behaviors receive data required for fulfilling their work via the *input vector* $\vec{e} \in \mathbb{R}^m$ which can be composed of sensory data (e. g. distance measurements) or information from other behaviors (e. g. their target rating). The *output vector* $\vec{u} \in \mathbb{R}^n$ transmits data generated by the behavior (e. g. intended velocity values). This output describes the data which is used for actuator control or as input for other behaviors.

Each behavior provides standardized inputs for adjusting its relevance:

Definition 8.1 (Stimulation)**.** The stimulation $s \in [0,1]$ of a behavior B is an input determining the intended relevance of B. In this notation, $s = 0$ indicates no stimulation and $s = 1$ a fully stimulated behavior. Values between 0 and 1 refer to a partially stimulated behavior.

Stimulation can be used to adjust the influence of competing behaviors or to allow higher-level behaviors to recruit lower-level behaviors and their functionality by explicitly stimulating them. Certain behaviors require constant stimulation, e. g. safety behaviors or reflexes. These behaviors are depicted by a filled triangle at the stimulation port in the figures.

Definition 8.2 (Inhibition)**.** Each behavior can be inhibited by k other behaviors via its input $\vec{\imath} \in [0,1]^k$. The inhibition $i \in [0,1], i = \max_{j=0,\ldots,k-1}(i_j)$ of a behavior B reduces the relevance of B. Here $i = 1$ refers to full inhibition, $i = 0$ to no inhibition. Values between 0 and 1 refer to a partially inhibited behavior.

Therefore, inhibition has the inverse effect of stimulation.

Definition 8.3 (Activation)**.** The activation ι of a behavior B indicates the effective relevance of B in the behavior network. It is composed of the stimulation s and the inhibition i, with

$$\iota = s \cdot (1 - i) \tag{8.2}$$

The calculation of the outputs of a behavior is implemented by the transfer function F, the activity function f_a, and the target rating function f_r. The transfer function $F(\vec{e},\iota)$ determines the output vector $\vec{u}$, where

$$F : \mathbb{R}^m \times [0,1] \to \mathbb{R}^n, \quad F(\vec{e},\iota) = \vec{u} \tag{8.3}$$

F provides the intelligence of a behavior, calculating actions depending on input values and internal representations. This can be a reactive response to input values, but also a more complex calculation like a state machine or sophisticated algorithms. This way, both reflexive sensor-actor coupling and deliberative behaviors can be implemented (as postulated for behavior-based architectures by [Mat97]).

Each behavior provides two behavior signals that allow for deducing information about its state and its assessment of the current situation:

Definition 8.4 (Activity)**.** The behavior signal activity $a \in [0,1]$ of a behavior B represents the amount of influence of B in the current system state. $a = 1$ refers to a state where all output values are intended to have highest impact, whereas $a = 0$ indicates an inactive behavior. Values between 0 and 1 refer to a partially active behavior.

The activity a and the derived activities $\underline{\vec{a}}$ are defined by the activity function f_a with

$$f_a : \mathbb{R}^m \times [0,1] \rightarrow [0,1] \times [0,1]^q,\ f_a\left(\vec{e},\iota\right) = \vec{a} = \left(a,\underline{\vec{a}}\right)^T \tag{8.4}$$

where

$$\underline{\vec{a}} = \left(\underline{a}_0,\underline{a}_1,\ldots,\underline{a}_{q-1}\right)^T \tag{8.5}$$

with

$$\underline{a}_i \leq a \quad \forall i \in \{0,1,\ldots,q-1\} \tag{8.6}$$

The derived activities $\underline{\vec{a}}$ allow a behavior to transfer only a part of its activity to other behaviors.

Definition 8.5 (Target rating)**.** The behavior signal target rating $r \in [0,1]$ is an indicator for the contentment of a behavior. A value of $r = 0$ indicates that the behavior is content with the actual state, while $r = 1$ shows maximal dissatisfaction. Values between 0 and 1 refer to a partially content behavior.

To ensure a consistent behavior network during the development process, some principles have to be complied with. Similar to [HA01] these principles allow some basic assumptions about the structure of the control system. These are required for the analysis of system properties.

As the activation defines the upper bound of a behavior's influence, the following principle must be observed:

Principle 8.1 (Activity limitation)**.** The activity a of a behavior B is limited by the activation ι of B: $a \leq \iota$

Furthermore, if the system is in the goal state of a behavior (characterized by $r = 0$), it intends to maintain its adjusted influence. Therefore, the following principle is postulated:

Principle 8.2 (Goal state activity)**.** The activity a of a behavior B does not change in case $r = 0$ and $\iota = \text{const.}$

Usually a behavior's activity is $a = 0$ in case it is situated in its goal state, but there are cases where a constant influence is required, i. e. $a > 0$. An example is a behavior generating torque for an arm joint. If, in this case, the behavior's activity was lowered in the goal state, external forces or competing behaviors could change the adjusted joint angle.

In contrast to the influence of the activation on the activity, the target rating only depends on the input vector and the behavior-internal state. This way, the target rating is an indicator for a behavior's state assessment, leaving out external adjustments of its influence:

Principle 8.3 (Target rating independence)**.** There is no (direct, i. e. inside a behavior) influence of the activation ι on r.

As described before, behavior-based architectures do not work with a centralized world model. This is represented by the fact that actions of a behavior only depend on the input vector $\vec{e}$, their activation and the behavior-internal representation of the current situation, which can be non-existent for certain behaviors.

Example behavior Turn to object

In order to exemplify the calculation of the described behavior properties, this section describes a showcase behavior rotating a vehicle to a detected object in front. As input vector $\vec{e}$ the behavior receives the angle β to the object to be followed. The output $\vec{u}$ is a normalized rotation value rot $\in [-1,1]$. As the rotation output shall point the robot into the direction of the object, the transfer function can be defined as:

$$\text{rot} = \begin{cases} -1 & \text{if} \quad \beta < -\beta_{\max} \\ \frac{\beta}{\beta_{\max}} & \text{if} \quad -\beta_{\max} \leq \beta \leq \beta_{\max} \\ 1 & \text{if} \quad \beta > \beta_{\max} \end{cases} \tag{8.7}$$

The target rating indicates the contentment of the behavior with the current situation. As the goal is to point the vehicle into the direction of the object, the behavior becomes discontent according to the angle to the object:

$$r = h(\beta) \tag{8.8}$$

with

$$h(\beta) = \begin{cases} \frac{|\beta|}{\beta_{\max}} & \text{if} \quad |\beta| \leq \beta_{\max} \\ 1 & \text{else} \end{cases} \tag{8.9}$$

As the behavior intends to reduce the deviation to the object, its activity has to increase if the angle to the object grows. The activation ι limits a in order to meet Principle 8.1:

$$a = \iota \cdot h(\beta) \tag{8.10}$$

8.3.2 Fusion behavior module

A behavior-based system certainly is not completed with the implementation of the single behaviors. As the influence of behaviors on control values or on other behaviors interleaves, and as they can have contrary goals, their outputs must be usefully combined. This question of behavior coordination is often considered the main problem in developing such an architecture.

The behavior coordination within iB2C networks is achieved by so-called *fusion behaviors* (see figure 8.10). These are integrated in the case of competing behaviors.

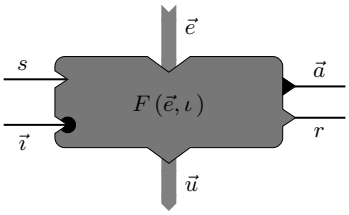

Figure 8.10 Fusion behavior module in iB2C

Fusion behaviors have the same interface as defined by the basic behavior module. For the coordination of p competing behaviors B_c, the input vector $\vec{e}$ is composed of

- the activities a_c (or the derived activities $\underline{a}_c^i$ of the vector $\underline{\vec{a}}_c$ respectively),
- the target ratings r_c, and
- the output vectors $\vec{u}_c$.

The transfer function F is the fusion function processing input values to a merged output control vector $\vec{u}$.

An example of the fusion of three competing behaviors B_c, $c \in \{0,1,2\}$ is depicted in figure 8.11. Each of the B_c is connected to the fusion behavior by its behavior signals a_c and r_c as well as the output vector $\vec{u}_c$. For clarification, the input vector of the fusion behavior is drawn separately.

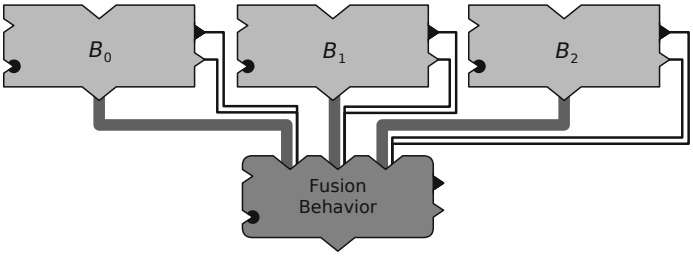

Figure 8.11 Exemplary fusion of three behavior outputs

The underlying assumption of the fusion of output values is that behaviors with a high activity deserve a higher influence on the control output than

those with a lower activity. By using the behavior signal *activity* as a means for coordinating the behaviors, the control data flow and the coordination data flow are separated.

The behavior signal calculation of fusion behaviors has to comply with the following principle:

Principle 8.4 (Fusion behavior neutrality)**.** The calculation of the activity a and the target rating r of a fusion behavior must keep the following conditions:

$$\min_c(a_c) \cdot \iota \leq a \leq \min\left(1, \sum_{j=0}^{p-1} a_j\right) \cdot \iota \tag{8.11}$$

$$\min_c(r_c) \leq r \leq \max_c(r_c) \tag{8.12}$$

This way, it is guaranteed that a fusion behavior does not inject or remove activity, as expected from a coordination component. Furthermore, there is no improvement or deterioration of satisfaction. This accounts for the fact that calculations concerning the assessment of state are only located in non-fusion behavior modules.

The following sections describe the set of fusion function implementations being used.

Maximum fusion (winner takes all)

In the case of maximum fusion the control value of the most active behavior is forwarded. Other behaviors obtain no influence. The transfer function F is defined as:

$$\vec{u} = \vec{u}_s \text{ where } s = \operatorname*{argmax}_c(a_c) \tag{8.13}$$

Activity and target rating are set according to the most active behavior:

$$a = \max_c(a_c) \qquad r = r_s \text{ where } s = \operatorname*{argmax}_c(a_c) \tag{8.14}$$

The maximum fusion implements a switching between behaviors and is suitable when a combination of control outputs leads to unwanted results.

Weighted fusion

In the case of weighted fusion the control values of the competing behaviors are weighted with the activity of the corresponding behavior. This way, a subtle gradation of coordinating behavior control outputs regarding their activity is achieved.

The transfer function F is defined as:

$$\vec{u} = \frac{\sum_{j=0}^{p-1} a_j \cdot \vec{u}_j}{\sum_{k=0}^{p-1} a_k} \tag{8.15}$$

The activity is defined as:

$$a = \frac{\sum_{j=0}^{p-1} a_j^2}{\sum_{k=0}^{p-1} a_k} \cdot \iota \tag{8.16}$$

The target rating of a fusion behavior indicates its goal to satisfy highly activated input behaviors and is calculated as follows:

$$r = \frac{\sum_{j=0}^{p-1} a_j \cdot r_j}{\sum_{k=0}^{p-1} a_k} \tag{8.17}$$

Weighted sum fusion

The weighted sum fusion is used for summing up the control values of the competing behaviors according to their activity. Applications for this fusion function are cases where several behaviors contribute to a torque of a joint or in cases where vectors are added up.

The transfer function F is defined as:

$$\vec{u} = \sum_{j=0}^{p-1} \frac{a_j \cdot \vec{u}_j}{\max_c (a_c)} \tag{8.18}$$

The activity is defined as:

$$a = \min \left(1, \sum_{j=0}^{p-1} \frac{a_j^2}{\max_c (a_c)} \right) \cdot \iota \tag{8.19}$$

The target rating calculation is the same as for the weighted fusion:

$$r = \frac{\sum_{j=0}^{p-1} a_j \cdot r_j}{\sum_{k=0}^{p-1} a_k} \tag{8.20}$$

8.3.3 Behavior interaction

Besides communication between behaviors through the environment or by using arbitrary data, the behavior interaction mainly takes place by transferring activity data between behaviors. As activity defines the relevance of behaviors and their outputs, the transfer inside the behavior network is restricted as follows:

Principle 8.5 (Stimulation/inhibition restriction)**.** Inside iB2C behavior networks a behavior B may only be stimulated or inhibited by the activity a or $\underline{a_i}$ of the vector $\vec{\underline{a}}$ of other behaviors.

This way, it is clearly defined where activity is injected into the behavior network and how it is transferred to other behaviors. Consequently, it is impossible for a behavior to gain influence without being sufficiently stimulated. The flow of activity through iB2C networks therefore allows statements about the overall system behavior.

In contrast to the activity signal, the target rating of behaviors counts as local situation assessment. It is used as abstract sensor value or for evaluating regions of dissatisfaction but is not transferred through the whole behavior network.

Due to the importance of the behavior signals activity and target rating, these values must be present everywhere inside the behavior network:

Principle 8.6 (Behavior signal availability)**.** Each control value entering a behavior network must be provided with a suitable activity and target rating value. Activity and target rating must not be dropped until control values leave the behavior network, i. e. until they are transformed to actuator commands.

This principle guarantees that for each control value an assessment of the relevance is provided. This is a key aspect which allows further processing of the data in the behavior network.

More precisely, the following sources and sinks of activity can be specified:

Sources of activity: Activity can enter the behavior network as follows:

1. Behaviors are stimulated from outside the behavior network.
2. Behaviors are constantly stimulated.
3. The activity of a behavior is used as stimulation for several other behaviors.

Sinks of activity: Activity can be reduced inside the behavior network as follows:

1. Fusion Behaviors combine several input activities to one output activity.
2. An activated behavior can emit an activity $a < \iota$ or a derived activity $\underline{a}_i < \iota$ of the vector $\vec{\underline{a}}$.
3. A behavior which is inhibited reduces the amount of activity at that place in the network.
4. If a behavior's activity output is not connected to another behavior, its activity is lost (e. g. in case a control value leaves the behavior network).

The previously defined principles allow the usage of the flow of activity for deriving each behavior's influence on other parts of the behavior network.

8.3.4 Behavior coordination

The behavior-based approach implies that several behaviors can contribute to the same control value. Therefore, the coordination of behaviors requires suitable mechanisms. This is where behavior architectures differ most. The following shows how a multitude of coordination mechanisms can be implemented in iB2C based on the uniform behavior module model (including the fusion behaviors).

A distinction of mechanisms for behavior coordination is presented in [Pir99]. Here, the first criterion distinguishes if several behaviors are *arbitrated*, i. e. one behavior or a set of them has control for a period of time, or if their *commands are fused*, i. e. a combination of control outputs of the behaviors takes place.

Arbitration makes sense when behavior actions have to be transferred without modification. The following types can be distinguished [Pir99]:

Priority-based mechanisms: Behaviors are selected according to priorities assigned to each of them (e. g. [Bro86]).

Priority-based arbitration in iB2C is implemented using inhibition of behaviors, see figure 8.12. The order of the behaviors determines the priority of each component. The maximum fusion behavior selects the most active behavior.

State-based mechanisms: Behaviors are selected in respect to the current state and the competence of behaviors for handling the situation (e. g. [KCB97]).

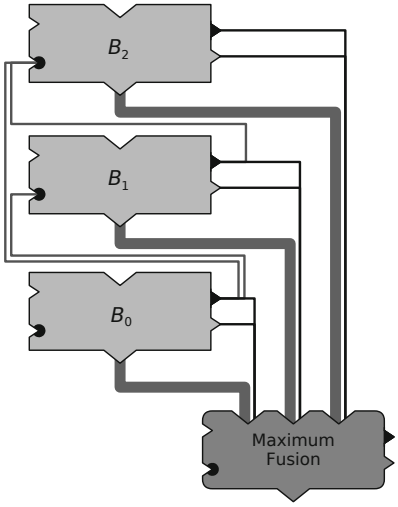

Figure 8.12 Priority-based arbitration in iB2C

State-based arbitration is realized using a behavior containing state evaluation mechanisms which stimulates action generating behaviors. Coordination takes place using a maximum fusion behavior.

If the state evaluation relies on feedback of the action generating behaviors, the activity and the target rating of the respective behaviors can be used.

Winner-takes-all mechanisms: One of the behaviors is selected as a result of a competition between them (e. g. [Mae89]).

The Winner-takes-all mechanism is directly supported in iB2C by the maximum fusion. Here, the competition between the behaviors is implemented as activity calculation.

In contrast to arbitration, command fusion supports the combination of behavior outputs. Several solutions for representing the desired commands and for determining the relevance of commands have been developed [Pir99]. Besides command fusion using the weighted sum fusion function, iB2C directly supports the superposition and voting mechanisms.

Voting: Each behavior votes for different actions. After combining them, the action receiving the highest number of votes is chosen (e. g. [PRK91]).

In iB2C, voting is implemented using a standard fusion behavior and a mapping behavior (see figure 8.13). Each behavior involved provides votes for each of the n possible options (e. g. driving directions) which

are transferred to the fusion behavior implementing the weighted fusion function. The output of the fusion behavior consists of the weighted votes for each voting option. A mapping behavior stimulated by the fusion behavior then maps the maximal option rating to a command for further processing.

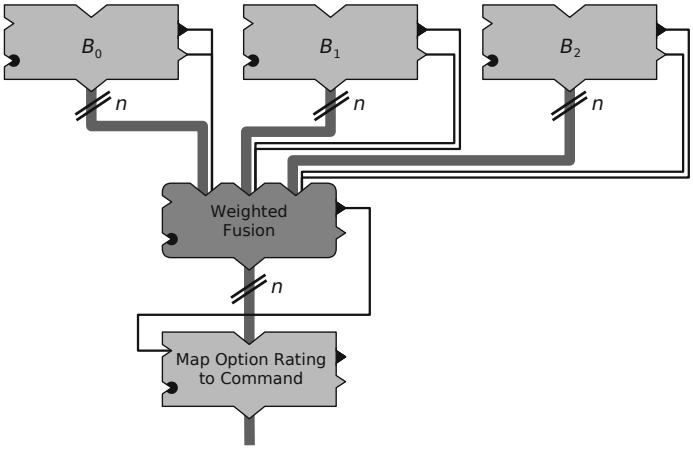

Figure 8.13 Voting mechanism in iB2C

Superposition: Behavior actions are represented as vectors which are linearly combined (e. g. [Ark87]).

Superposition in iB2C is implemented by the weighted sum fusion, where a component-wise fusion takes place with the activity representing the relative scale of each vector.

Fuzzy: Similar to voting mechanisms, here fuzzy inferencing techniques are used (e. g. [SKR95]).

Multiple objective: Also similar to voting, the desirability of actions is defined by each behavior's objective function. Coordination is carried out by looking for actions that sufficiently satisfy all objective functions by using multiple objective decision theory methods (e. g. [PHTO+00]).

As fuzzy inferencing techniques and multiple objective mechanisms implement functionality similar to voting, they are not treated here.

8.3.5 Design guidelines

Designing a control system for robotic applications requires a systematic methodology in order to cope with the complexity of sensor processing and

control data generation. In iB2C, the development begins by figuring out the relevant degrees of freedom (DOF), e. g. rotation and velocity of a vehicle, emotional actuators of a humanoid head, or joint motions of legs. Each of the DOF is divided into positive and negative direction, leading to two control data paths for every motion possibility. The conflation of the data flow is accomplished using a fusion behavior for each of the DOF. Depending on the mechanical construction, the described approach may be performed several times for each kinematic chain involved, e. g. for a pan tilt unit of a camera head or for a multitude of legs.

In order to fulfill basic safety requirements, the next step is introducing behaviors reacting on safety related sensor input (e. g. stopping or turning away a vehicle because of data provided by a proximity sensor). Each of the safety behaviors influences a DOF by using its activity output for inhibiting fusion behaviors of the layer above and by propagating a new command to a fusion behavior in the layer below. As each of the DOF is divided into positive and negative direction, behaviors can be integrated in such a way that only the supervised direction is influenced.

This procedure results in an interface for higher level behaviors and encapsulates the functionality of a safety behavior system. High-level behaviors are then added using a top-down task-oriented approach. Methods like those proposed in [Bry01], asking for *what* to do *how* and *when* and iteratively revising the structure can be applied here.

Hierarchical abstraction One advantage of the decomposition of tasks into behaviors is the low complexity of each behavior. However, the result of this approach often is a network with a large number of behaviors. In order to simplify the structure and to clarify the functionality, a hierarchical abstraction becomes necessary. In the case of iB2C this can be accomplished using behavioral groups (see figure 8.14). These are groups in the sense of the embedding programming framework[3], i. e. a collection of modules or further groups with a new interface and dedicated connections between group and modules. A behavioral group acts as a new behavior, providing the same standardized input and output signals described in section 8.3.1.

The challenge for the developer is finding sets of behaviors representing new semantic groups. One approach is to reflect the implemented decomposition in the hierarchical structure of groups. Another hint for grouping behaviors stems from the influence of multiple behaviors on a DOF. If several behaviors work in the same domain and have an influence on the same data

[3] in this case the *modular control architecture* MCA, [SAG01]

path in the network (e. g. behaviors using different sensor systems for bringing a vehicle to a halt), these behaviors are good candidates for forming a new group.

When constructing a behavior network, the designer has to question himself about the semantics behind behaviors and whether a group of behaviors separates from the individuals to form a new semantic unit. If this is the case, a new behavioral group should be introduced.

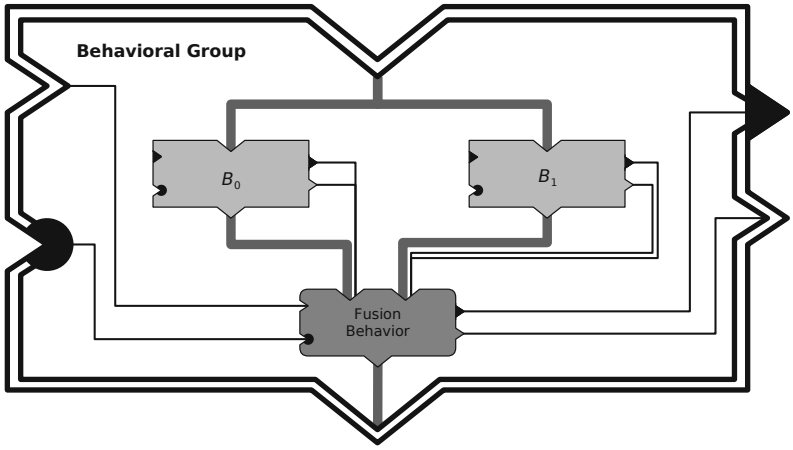

Figure 8.14 Example for a behavioral group in iB2C with a fusion behavior providing the behavior signals for the group interface.

Behavior signal usage The main challenge when coping with systems growing in complexity is making statements about the current system status. This is not only necessary for a developer trying to find out if an implemented feature works, but also for system components trying to reason about the result of a given command. In this sense it becomes invaluable having a common interface of behaviors representing their internal state in an abstract way. In iB2C, behaviors generate the behavior signals *activity* (a) and *target rating* (r) which can be used for detecting several important aspects of the system (for examples see [HBB07]):

- deadlock detection (e. g. by supervising obstacle detection behaviors),
- risk determination (e. g. by supervising slope detection behaviors)
- effort (e. g. by supervising behaviors supervising current measurement of motors)
- oscillation detection (e. g. by supervising behavior activities over time)

8.3.6 Analysis of iB2C networks

Developing robotic systems requires some kind of support for system analysis in order to tackle the complexity of the evolving structure. The aim is to accelerate system development and to reduce time spent for testing. IB2C makes extensive use of the Modular Controller Architecture (MCA). Each behavior is derived from a MCA-module with a standard interface as presented in section 8.3, and with predefined methods for the transfer function and behavior signal calculation. The behaviors are then arranged in a layered network using defined behavior interfaces and interconnections.

Therefore, the behaviors form a graph of interconnected components with a flow of activity which enables analysis using graph theory methods. Figure 8.15 gives an example of an automatically generated iB2C graph containing the flow of activity between behaviors influencing the forward and backward motion of a vehicle. Within this graph, properties like cycles as well as stimulation and inhibition successors and predecessors can be automatically identified to retrieve static information about the influence of behavior modules on the robot's behavior. This way, possible sources of oscillations inside the behavior and interconnections contradicting with the introduced principles have successfully been spotted.

During runtime of the robotic system the software can be supervised by the MCA tools MCAGUI and MCABrowser. The MCAGUI serves as the user interface for the robot which can be configured using predefined widgets and plug-ins. The tool MCABrowser lets the developer have a detailed look at the flow of data during runtime.

For iB2C, the user interface inside MCABrowser is complemented by indicators for the behavior signals. While the robot performs its tasks, a condensed view of the current distribution of activation, activity, and target rating is given using colored bars (see figure 8.16). Additionally, the flow of activity is indicated by colored edges between behavior modules. This way, sources of the current robot behavior can be easily identified and analyzed. The example depicted in figure 8.17 presents three snapshots of the behavior signal visualization. At first the robot is in an idle state where no command is to be executed (top left). Therefore, all behaviors remain inactive. Afterwards, a normal forward motion situation (top right) is indicated by a flow of activity from the top interface to the bottom interface passing through fusion behaviors of different layers. Finally, a situation where the *Forward Tactile Creep* behavior inhibits slow down behaviors while *FW Limit Creep Vel* maintains a minimal creep velocity in order to slowly move into vegetated regions is presented (bottom). This serious situation is clearly visualized by a high target rating of several behaviors as well as the massive

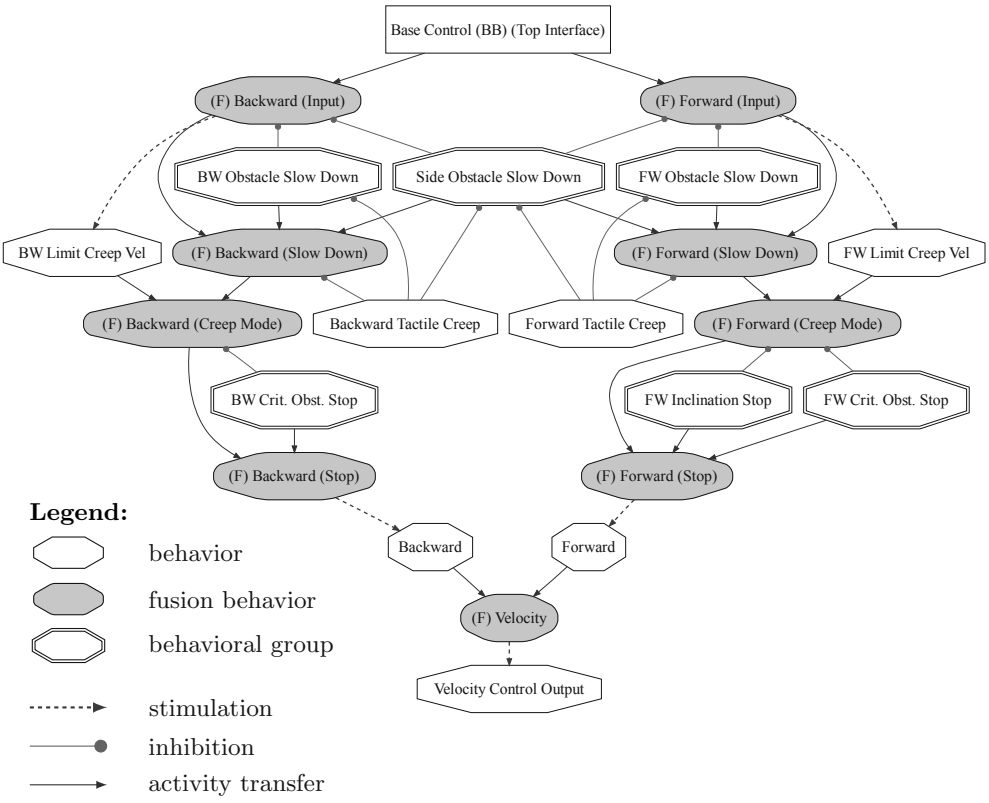

Figure 8.15 Example for an activity graph of an iB2C behavior network influencing the forward and backward motion of a vehicle. The different styles of the arrows indicate the type of interaction between behaviors, i. e. stimulation, inhibition, or activity transfer. This allows the evaluation of the activity flow through the behavior network.

inhibitory interaction between behaviors. This way, several flaws in implemented iB2C networks have been detected, e. g. behaviors showing no activity due to missing sensor information or errors in the transfer function resulting in contradictory values for activity and target rating.

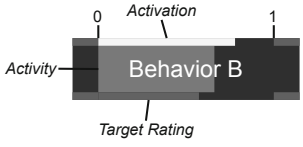

Figure 8.16 Behavior signal visualization in MCABrowser

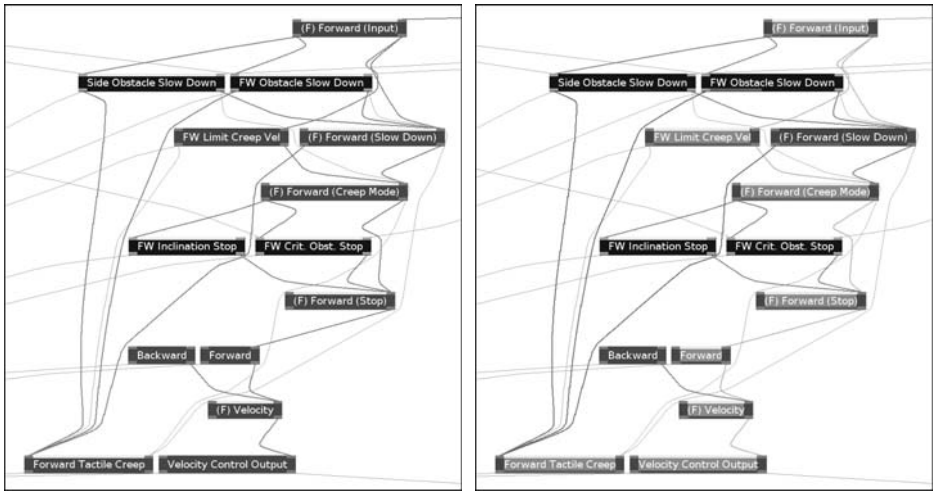

Figure 8.17 Two exemplary snapshots of the on-line behavior signal visualization in MCABrowser (Top: Normal forward motion is indicated by the course of activity through the forward behaviors. Bottom: Obstacles in the robot's proximity result in a high activity of the *Forward Tactile Creep* behavior which inhibits the slow down behaviors in order to move slowly forward)

In cases where it is necessary to guarantee certain system properties, an approach for the formal verification of a subset of behaviors can be followed as described in [PBSS07]. With this method a subset of interconnected behaviors is implemented in the synchronous language Quartz and verified concerning given properties using model checking techniques. Afterwards, code is generated which is periodically called inside a MCA module and which is proven to meet given specifications.