Abstract—Since the birth of the subsumption architecture (SA) two decades ago, several derivations have emerged from the original concept. The behavior-based architectures, one of the successors, have been successful to create simple behaviors in a system, resulting more complex behaviors by their interactions. The dynamic behavior network (DBN), introduced in this paper, does not specify a strict hierarchy between the behaviors, however, the stimulus levels and links determinate their actual states in a moment. The dynamic instantiation and destruction of the behaviors make the design and the development of the behaviors easier and the computational resources are utilized more efficiently. The concept of the DBN is presented in this paper, the common and new behavior patterns are examined as well as a practical application based on DBN is described.

I. INTRODUCTION

The artificial intelligence (AI) has a goal among others to show the levels of the human or animal intelligence by agents. These real or simulated creatures can solve different tasks, but usually, the targets demand for specific design approaches. Several theories have been used in practical applications: reactive approaches [1], [10], genetic algorithms [3], [6], [14], behavior-based programming [8], [11], [13], or machine learning based solutions [5], [12]. However, the design of a new project is not straightforward many times, since transferring the knowledge and using the accumulated experiences from the past projects is not easy. Other challenges are the growing complexity in function of the sizes of the problems, the unpredictable state transitions inside the system and the testing during the development. There is still a high interest to find new ways to improve the design process of new systems and allow fast prototyping; recent architectures implement complex behavior-based systems [8], [13] to overcome some issues.

The main goal of the dynamic behavior network (DBN), presented in this paper, is to keep the best features of the classical behavior-based systems while providing reduced development and testing complexity with algorithmic decomposition into behavior networks. The DBN introduces the dynamic creation and destruction of the behaviors to focus the attention to the actual state of the environment efficiently: the relevant behaviors are only kept alive and they compete for activation. An other emphasize is on the links between the behaviors to spread their stimulus to the connected behaviors and to determine their priorities in a temporal manner. The DBN can also take advantage of the traditional behavior patterns and its internal structure offers new designs.

The rest of this paper is organized as follows. In Section II, the background and the related work is discussed. The dynamic behavior network is described in Section III, the design patterns in Section IV, a reference implementation is presented in Section V for a quick glance, and finally, the paper is concluded in Section VI.

II. BACKGROUND AND RELATED WORK

The original topology of the behaviors in the subsumption architecture (SA) was static [2]; they received sensory input, decided on their own about activation and the hierarchical suppression links could stop the activation if a higher level behavior was going to be activated. Inhibition functions were used to avoid conflicts between behaviors, though the development of the SA systems becomes difficult after a certain complexity because the goals of the behaviors interfere on higher abstraction levels.

The ethology inspired, behavior-based cognitive architecture, JDE+ [4], uses schemas as building blocks. The decentralized action selection is similar to the dynamic behavior network, but after the ancestor schemas activate their children, they keep a control over them until the desired goal is reached. Contrary to this mechanism, the behaviors are equal in the topology of the DBN, the weights of the links and the history of the activation patterns determinate the priorities between the behaviors, but this hierarchy can change each moment. The schemas in JDE+ on the same hierarchy level check their neighborhood to select the appropriate action, but the behaviors in a DBN are independent and their actions depend on the environment states rather than querying other behaviors in the network.

A common restriction of the SA and the JDE+ systems is the hierarchy of the behaviors: a higher level task is broken down into smaller behaviors on multiple levels. This decomposition is efficient for individual goals, but the subgoals of different higher level goals interfere on lower levels and it is hard to find any resolutions to this problem. The DBN has the same issue when the decomposition goes to lower granularities, but the author of this study believes that an abstraction (controller) layer between the behaviors and the actuators can handle this problem (claiming the control of the same actuator by concurrently activated
behaviors) to some extent, however, this solution is out of scope of this paper.

The behavior network (BN), invented by Pattie Maes [7], used motivational competition and selection of a winning behavior based on activation and inhibition dynamics, contrast to the hierarchical and preprogrammed control structures of the early works. The parallelism is similar in BN and DBN, but the first has three kinds of links between the behaviors (predecessor, successor or conflictor) while the latter has one link type. A disadvantage of the behavior networks is the high number of links between the behaviors, which grows with the size of the problem quickly. The BN also has some global tunable parameters, which increases the need of the human control.

The dynamic behavior networks have some unique features, different from the earlier variations of the behavior-based systems. The number of the existing behaviors in a moment is minimal, it is a set of the static behaviors and the dynamic behaviors created temporary during run-time. Any behavior can create new behaviors and these instances are not tracked by their “parents”. The links between the behaviors distribute their stimuli and inhibitions. The activation of a behavior depends on its own self-stimulus and the received stimuli/inhibition through the incoming links (Fig. 1). The behaviors in the DBN have a loose interdependence compared to the classical systems. A deeper look on the topology and functional description of the dynamic behavior networks follow in the next sections.

III. DYNAMIC BEHAVIOR NETWORK

There is no strict definition for the behavior in the literature. The behavior in the dynamic behavior network is a computational unit with one or more goals. The DBN can contain two types of behaviors: static or dynamic. The static behaviors exist during the whole run of a system while the dynamic behaviors are instantiated at some point and their lifetime is limited. The higher level organization of the behaviors is the subnetwork. The definition of the internal mechanism of the network is detailed in the subsections.

A. Behavior

A behavior can be described in a four-tuples form:

\[ B = (L,C,T,A) \]  \hspace{1cm} (1)

where \( L \) is a lifetime duration limit and it is infinity for static behaviors \((L = \infty)\). \( C \) contains the conditions for the self-stimulus, the finished and the failed states. \( T \) is the time to reach the maximal self-stimulus and \( A \) is a set of action sets associated with the behavior. \( L, C, T \) and \( A \) are unique for each behavior.

A complete DBN system is composed of \( N \) behaviors:

\[
 DBN_{system} = \{ B_0, B_1, \ldots, B_{N-1} \} \hspace{1cm} N > 0
\]

\[
 B_s = \{ B_0, B_1, \ldots, B_{M-1} \} \hspace{1cm} 0 < M \leq N
\]

\[
 B_d = \{ B_M, B_{M+1}, \ldots, B_{N-1} \}
\]

where the \( DBN_{system} \) denotes a network with static \( (B_s) \) and dynamic behaviors \( (B_d) \).

Each behavior \((B_i \in DBN_{system})\) can have preconditions to gain stimulus or conditions to become finished or failed:

\[
 C = \{ C_o, C_s, C_f, C_{fa} \}
\]

\[
 C_o = C_{oa} \lor \ldots \lor C_{oa}
\]

\[
 C_s = C_{so} \lor \ldots \lor C_{so}
\]

\[
 C_f = C_{fo} \lor \ldots \lor C_{fo}
\]

\[
 C_{fa} = C_{fao} \lor \ldots \lor C_{fa0}
\]

where \( C \) is a set of expressions. \( C_o \) contains conditions for over-stimulation, \( C_s \) for normal stimulation, \( C_f \) for finished state (the goals are reached) and \( C_{fa} \) for failed state. Any of the \( C_{o}, C_s, C_f, C_{fa} \) conditions can make the \( C_o \) expression true and the similar principle applies to \( C_{so}, C_{fo}, C_{fao} \). A practical condition in the \( C_o \) is a time limit for the activated state (the behavior fails when the limit is reached in activated state).

Based on the \( C_o \) and \( C_s \) expressions, the self-stimulus is calculated as follows:

\[
 S(t) = \begin{cases} 
 T: C_o \land \text{activated} \\
 \max(S(t-1)+\Delta t,T): C_s \\
 \max(S(t-1)-\Delta t,0): \text{otherwise} 
\end{cases} 
\]

where the behavior has a given time to reach the maximal self-stimulus level \( (T) \) when the required conditions are met. The lack of the stimulus decreases the level and the over-stimulation results an immediate increase to the maximum. The self-stimulus function of each behavior is updated based on the present knowledge from the environment every iteration.
A behavior has four possible states as a state machine: normal, activated, finished or failed (Fig. 2) and it can execute actions during certain state transitions:

$$A = [A_n, A_f, A_b]$$,  \hspace{1cm} (5)

where $A_n$ is a nonempty set of actions performed when the behavior enters in the activated state. $A_f$ and $A_b$ are the actions for the finished and the failed states, but they are optional (may be empty sets).

A behavior is in normal state until it is activated by stimulus. The actions after the activation ($A_n$) implement a certain task. The desired goals are reached when the conditions of the finished state ($C_f$) are valid and the state makes a transition to finished, otherwise failed. The finished and failed states are changed to normal in the next iteration if the behavior is not destroyed. A dynamic behavior can reach the lifetime duration limit ($L$) in normal state and it is deleted.

### B. Links

The dynamic behavior network has a few similarities to the neural networks. The behaviors have stimulus levels and they are spread to other behaviors in the network via directed links, similar to the concept of the links between the artificial neurons in the neural networks. The negative links can inhibit the connected behaviors from activation and the positive links help the connected behaviors to reach the activation. In theory, the behaviors work in parallel, but in practice, the parallelism is simulated. The computation of the stimulus levels and the distribution to the connected behaviors are done in iterations.

The directed links spread the stimulus levels from the source behaviors to the targets; the incoming stimulus from $B_i$ ($j \in \{0, \ldots, N-1\} \setminus \{i\}$) to $B_i$ behavior is calculated as follows:

$$I(t) = \sum_{j=0}^{N-1} \frac{w_{ij} \cdot S_{B_j}(t)}{T_{B_j}} \quad i \neq j, \hspace{1cm} (6)$$

where $w_{ij}$ ($j \in \{0, \ldots, N-1\} \setminus \{i\}$) denotes the associated weight of the link from $B_i$ to $B_j$. The weight is in the range [-1.0, 1.0], a negative value represents an inhibitory link, a positive an excitatory link. The zero weight ($w_{ij} = 0$) means no link between $B_i$ and $B_j$. The stimulus of the $B_i$ behavior ($S_{B_i}(t)$) is normalized by $T_{B_i}$ before it is sent to the $B_i$ behavior.

Since the dynamic behaviors do not exist all the time, the links with missing source or target behaviors are omitted in a moment. The excitatory links are enough to activate a behavior because the final stimulus level is defined by the sum of the incoming stimuli and the normalized self-stimulus:

$$S_{sum}(t) = I(t) + \frac{S(t)}{T}. \hspace{1cm} (7)$$

A behavior becomes activated if it has not been activated before and $S_{sum}(t) \geq 1$. If a behavior has been activated, it keeps the activated state until it is finished or failed, regardless of the value of $S_{sum}$ in those iterations.

In the standard behavior-based systems, the activation of a behavior depends on the environment and the task-dependent conditions. In the DBN, both generate stimuli to the behavior, but they are not direct conditions for the activation because the stimuli of the connected behaviors are taken into account in the final stimulus level.

The link concept of the DBN has an other strong point because it supersedes multiple behavior design principles. Fig. 3 shows some behaviors with links:

1. $B_x$ and $B_y$ are in conflict mutually. Whenever $B_x$ has some stimulus, it blocks $B_y$ from activation and vica versa.
2. Regarding $B_x$ and $B_z$, the inhibitory link makes $B_x$ higher priority than $B_z$.
3. $B_y$ has an inhibitory link to $B_z$, and an excitatory link to $B_x$. Simultaneous stimulus to these behaviors determines the priority order $B_x \rightarrow B_y \rightarrow B_z$, from the point of view of $B_x$.
4. Regarding $B_x$, $B_z$ and $B_w$, the excitatory link from $B_z$ to $B_w$ is stronger than $B_z$ to $B_x$. The priority is $B_x \rightarrow B_z \rightarrow B_w$ in this case.
Like the example shows, the net of the weighted links can be interpreted as conflicts or priorities. The weights are defined by human supervision empirically and learning algorithm for the weights has not been defined yet.

C. Subnetworks

In the starting phase of a DBN system, permanent behaviors ($B_d$) are instantiated and they can create dynamic behaviors after their activations. A master behavior and its instantiated (slave) behaviors are a subnetwork. The smallest subnetwork is a static behavior, which does not create new behaviors. A subnetwork is a higher level building block of the DBN and it can embed other subnetworks to solve tasks.

The goal of the master behavior is completed by the subnetwork and this goal is decomposed into the lower granularity goals of the instantiated behaviors.

The subnetwork is inactive unless the dynamic behaviors are created by the master. When the goals are finished or failed, the slave behaviors are destroyed and the subnetwork goes back to sleep from the activated state.

The slave behaviors do not live in the sleeping state of the subnetwork. This fact spares computational resources and the non-existing behaviors can not interfere with other behaviors from different subnetworks. The next advantage of this approach is the easier behavior design compared to the classical behavior-based systems, since the behaviors only take into account their own subnetwork and the other active subnetworks at the same time. When the master behavior is destroyed, the slave behaviors are also destroyed automatically to avoid dangling behaviors (existing dynamic behaviors from an inactive subnetwork).

D. Arbitration

The self-stimulus levels of the behaviors are not cleared between the iterations, and along with the dynamic subnetworks, these properties act as behavior coordination and decentralized action selection. It is a mixture of two traditional approaches: priority-based and state-based action selection inasmuch as the world state and the spread stimulus between the behaviors have influence on the behavior activation together (7).

Traditionally, an agent has some abstract goals and it selects some behaviors to achieve the targets in an optimal way. In the DBN, there are no abstract, system-wide goals, each behavior has its own, independent goal and the activation pattern of the behavior network determinates the priority of these goals dynamically.

As long as the stimulus of a behavior reaches the activation level, it is activated automatically, winnertakes-all or similar arbitration techniques are not applied in the DBN system.

IV. BEHAVIOR PATTERNS

The behavior-based systems can use special development means during the design process [8]. The following subsections examine the usage of the classical behavior patterns with DBN and new patterns are introduced.

A. Traditional Patterns

The inhibition pattern, inhibiting behaviors on other levels, is shown on the first example in Fig. 4. $A'$ and $A''$ are on the same level, but $A'$ is a higher granularity behavior. Under normal conditions, $A'$ would stimulate the $A''$ behavior, however, the $A'$ behavior can inhibit the stimulation of the $A''$ by inhibiting $A'''$.

The degrees of freedom (DOF) access pattern is about manipulating the same actuator by multiple behaviors with different goals. In the second example of the Fig. 4, $B'$ creates $B''$ and $B'''$ behaviors during its finishing actions and those manipulate the same actuator after their activation with different goals.

The group pattern is directly related to one of the strengths of the DBN, the problem decomposition into a behavior subnetwork. The second example in Fig. 4 can be analyzed from an other point of view. $B'$ can be a static behavior, the master of the $B'_{-}B''_{-}B'''$ subnetwork. After $B'$ is activated, $B''$ and $B'''$ are instantiated. $B'$ keeps its activated state until $B''$ and $B'''$ achieve the goal of the subnetwork. Because $B'$ is activated while the subnetwork is active, the normal/inactive and active periods of $B'$ and the subnetwork are equivalent. With other words, $B'$ behavior encapsulates the subnetwork states as a state machine.

While dealing with partitioning an action into more

---

**Fig. 4.** Traditional behavior patterns with dynamic behavior subnetworks. I. Example for the inhibition pattern. II. Example for the DOF access and the group pattern. III. Example for the combined stimulation pattern.
components, the combined stimulation pattern comes into the picture. The last example in Fig. 4 have an action divided into three parts ($D'_1$, $D'_2$, $D'_3$) and variant combinations of the stimulation from $C'_1$, $C'_2$, $C'_3$ to $D'_1$, $D'_2$, $D'_3$. Namely, $C'_1$ and $C'_2$ affect $D'_3$; $C'_2$ affects $D'_1$ and $C'_1$, $C'_3$ affect $D'_2$. Different activation combinations of the $C'_i$ behaviors stimulate different combinations of the $D'_i$ behaviors.

B. New Patterns

The dynamic behavior network offers new patterns derived from its differing structure compared to the other behavior-based systems.

The static and the dynamic behaviors have distinct natures what is reflected in the static/dynamic pattern. Static behaviors may be designed to listen to the environment constantly to take actions while the dynamic behaviors react in their limited lifetime.

The subnetwork pattern is about building a subnetwork with the specified characteristics in Subsection III/C. The subnetworks are beneficial because of the problem decomposition and the decreased computational resources. From design point of view, the dynamic behavior usage reduces the need for links between multiple subnetworks and it makes the design of the slave behaviors easier. An interesting case is a master behavior that can create more subnetworks. This so-called super-master behavior usually manages subnetworks with similar scopes (for example robot body related subnetworks).

Beyond the activated state, the behaviors can create new behaviors in their finishing or failing actions ($A_{fi}$, $A_{fa}$). If the behaviors are considered as state machines, the activation actions ($A_c$) of two behaviors (e.g. $B_1$, $B_2$) can be two steps of an algorithm where the $B_2$ step must be executed after the $B_1$ step fails/finishes. Translating these steps into DBN terminology, when $B_1$ fails/finishes, $B_2$ is created and activated by unconditional over-stimulation immediately. This assignment makes possible to rewrite algorithms into behavior networks as equivalent state machines and the method is the algorithmic decomposition pattern.

After introducing the DBN concepts and going through the behavior patterns, a reference implementation is shown in the next section.

V. Reference Implementation

AIBO is a discontinued robot dog brand of the Sony. The ERS-7 model has a good hardware, which is still a notable, fast embedded system nowadays. The proprietary programming environment of AIBO, the Open-R [9] has a great real-time scheduler, but the Sony provided only the low-level control of the robot, hence a new AI should implement everything from scratch.

The on-board program of the AIBO+ project based on Open-R implements some basic goals with dynamic behavior networks. Currently, 9 static behaviors manage 7 subnetworks. The length of this paper is limited, therefore, only three closely related subnetworks have detailed description in the next subsection and a closer look is taken on the performance on AIBO in the last subsection.

A. Subnetworks for Body Responses

There are touch sensors at three areas of the robot: chin, crown and back. AIBO can react to the strokes with body responses, but it is a challenge to have more areas stroked at the same time.

Three subnetworks have been constructed (subnetwork pattern) to deal with the situation according to the three body areas (Fig. 5). The $BodyStroked$ is a static super-master behavior and all other behaviors are dynamic in its subnetworks. When a certain body part is being stroked, the appropriate behaviors ($BackStroked$, $ChinStroked$ or $CrownStroked$) are created respectively. A sound effect is played and an animation is shown on the face LEDs by the $BodyStroked$, the $BackStroked$ does movements with the head, the $CrownStroked$ stops any ongoing head movements and instantiates the $ChinStartRubbing$ behavior. Because the behaviors have conflicting actions on the same actuators, there are inhibition links to define a priority order: $CrownStroked \rightarrow ChinStroked \rightarrow BackStroked$ and the body responses can not interfere while stroking more areas simultaneously.

The $Chin$ subnetwork implements subtle details of the head movements while the chin is stroked. The $ChinStroked$ behavior is an example for the group

![Fig. 5. Three subnetworks (Back, Chin, Crown) for body reactions when stroking the robot. The dashed arrows show the instantiation of new behaviors when the parent behavior transits to a state. The normal arrows with weights are inhibitory links.](image-url)
pattern; it keeps the activated state until the stroking is stopped and it guards the active subnetwork against other behaviors with inhibitory links to avoid the movements of the head by other subnetworks.

The rest of the Chin subnetwork is straightforward, the algorithmic decomposition pattern was used to partition the head movements into behaviors. The ChinStartRubbing moves the head a bit ahead to the start position of the rubbing. The ChinRubbed, as the next in the chain, pushes the head straight ahead and back continuously unless the palm is detected on a side of the head by the forces in the neck joints. The head can be pushed to ahead/ left (ChinRubbingLeft) or ahead/right (ChinRubbingRight) against the palm in these situations to make better impression in the human observer about the situation awareness and the subtle reactions of the robot during the interactions. When the stroking is stopped at some point, the momentary active rubbing behavior fails and the ChinRubbingFinished is instantiated to move the head back to the original position.

B. Performance

The AiBO+ is written in C++, therefore, the dynamic behavior network is executed on the target system natively. Since the number of the static behaviors is kept at lowest, the activation links have minimal impact on the performance.

AiBO is equipped with an RM-7000 (RISC, MIPS) processor clocked to 576 Mhz. The 9 static behaviors generate about 1.6 % CPU load with 30 Hz iteration frequency and it does not change substantially when dynamic behaviors are instantiated in certain conditions.

VI. Conclusion

The links are integral part of a dynamic behavior network and comprise the knowledge of the network along with the existing behaviors. The simple principles of the DBN (links, subnetworks, stimulus distribution, dynamic behaviors, algorithmic decomposition) have the advantage to build subnetworks easily, decompose the problems into smaller blocks and reuse the previous works with a modular approach. The new concept was reviewed with common behavior design patterns and new DBN specific patterns were recognized.

The minimal processor usage of the static behaviors makes the dynamic behavior network a reasonable candidate for real-time purposes on robots with restricted resources.

A practical example was presented with three subnetworks to manage actuators in response to strokes on the robot. The goals were accomplished with common and newly introduced behavior patterns. More demonstration videos of the open-source AiBO+ are available on the internet.

The forthcoming work will include studies on more complex dynamic behavior networks because the current implementation does not take advantage of the possible slight values of the link weights. The presented DBN lacks any learning capability what can be a major extension in the future to minimize the human supervision, however, it is not clear yet, if the learning should happen on the level of the links or inside the behaviors.

ACKNOWLEDGMENT

Thanks to my family to tolerate my hobby research at home and my company to support my efforts.

REFERENCES


1AiBO+ demonstrations: http://youtu.be/qMzw4UIAQd8 and http://youtu.be/hyw8z_hL7lo