final version in:
International Conference on Robotics and Automation, ICRA, IEEE Press, 2001

Efficient Scheduling of Behavior-Processes on Different Time-Scales

Andreas Birk and Holger Kenn
Vrije Universiteit Brussel, Artificial Intelligence Laboratory,
Pleinlaan 2, 1050 Brussels, Belgium

Abstract

In behavior-oriented robotics, the control of a system is distributed over various processes or behaviors running in virtual parallel. But the assignment of processing power to different processes is a non-trivial task. Existing approaches to this problem like rate-monotonic scheduling focus on the fulfillment of deadlines, i.e., upper time-bounds. For behavioral control, the periodicy of processes is also of interest. Here, a novel scheduling algorithm for behavioral processes at different time-scales is introduced, which ensures time-optimal and periodically balanced execution.

1 Introduction

Behavior-oriented robotics has matured in the last 15 years from a scientific critic of "classical" AI [Bro86, Bro91, Ste91] into a wide robotics field [CA98] including a range of applications [Bir98]. As pointed out in [Ste94], the notion of behavior is used within a wide range of interpretations. Nevertheless, the common property of all behavior-oriented systems is that the overall control of the system is distributed over various processes running in virtual parallel. A major technical problem with this approach is that it needs *scheduling*, i.e., a scheme to assign processing power to different behavior-processes such that they seem to run in virtual parallel.

Existing behavior-oriented programming languages like the subsumption architecture [Bro86, Bro90] or motor schemas [Ark87, Ark92] came out of early scientific work in the field of behavior-oriented robotics. Accordingly, they did not incorporate any considerations on efficiency or software-engineering, forcing the user to do a lot of hand-tailoring for each particular application, especially in respect to scheduling. As a consequence, these languages are not widely distributed. Instead, the complete software environment for every behavior-oriented project around the globe is usually developed from scratch.

Scheduling is a major research topic in a completely different scientific area, namely the field of real-time systems [BW97, Mel83, You82], which provides a wide range of solutions to the problem. One approach for behavior-oriented robotics accordingly is to use a real-time operating-system and programming-language and to build the control-behaviors on top. The problem is that standard real-time approaches, especially the widely used rate-monotonic scheduling [LL73, LSD89], focus on the fulfillment of deadlines, i.e., upper time bounds for the execution of the processes. For control, it is in addition of interest that behavior-processes are executed as regular as possible, i.e., the periodicity plays a significant role.

Here, a novel scheduling algorithm, the so-called B-scheduling, is presented which handles behaviors running on different time-scales represented through so-called exponential effect priorities. These special priorities allow our special algorithm to achieve both desired features for behavior-scheduling, namely time-optimal and periodically balanced execution.

2 Behaviors and Time-Scales

Behavioral processes in general can span very different time-periods. The pulse-width-modulation (PWM) as speed-control has for examples to operate for some DC-motors in the 20 kHz range, i.e., on a time-basis of $5 \cdot 10^{-5}$ seconds. A behavior monitoring batteries in contrast operates on a scale of minutes. Some adaptive or learning behaviors could operate on much higher scales like hours or even days.

We hypothesize that in general it is desirable to span several orders of magnitude of time-periods. A linear priority scheme is not suited for this. Therefore, so-called *exponential effect priorities* are introduced here. The idea is that for each increase in a priority-value by one, the periodicity is halved.

In the remainder of this paper the following naming conventions are used: the set of processes: $\mathcal{P}=\{p_0, ..., p_{N-1}\}$, the priority-value of process p_i: $pv[p_i]$, the set of processes with priority k or the k-th priority class: PC_k, and the highest used priority-value: $maxpv$.

The exact semantic of a priority-value $pv[p_i]$ of process p_i within *exponential effect priorities* is:

- $pv[p_i] = 0 \iff p_i$ is executed with the maximum frequency f_0
- $pv[p_i] = n \iff p_i$ is executed with the frequency f_n which is half the frequency of the previous priority-class, i.e., $f_n = f_{n-1}/2$

3 The Chores of Scheduling

For solving the task of finding a suitable order of execution of the processes, we use a *cyclic executive scheduling* approach [BW97]. This means there is a so-called *major cycle*, which is constantly repeated. The major cycle consists of several so-called *minor cycles*. Each minor cycle is a set of processes, which are executed when the minor cycle is activated.

The general problem of finding a suitable schedule within this approach is NP-hard as it can be reduced to the Bin-Packing-problem in a straightforward manner. We present an extremely efficient, namely linear-time algorithm, which is based on the restriction to exponential-effect-priorities. As motivated above, we do not see this as a limitation, but even as a feature.

Before the algorithm for behavior scheduling or *B-scheduling* is presented, an example is considered to illustrate the problems involved in scheduling. Figure 1 shows a simple algorithm, which schedules behaviors based on their priorities. The major cycle is simply a loop proceedings in rounds. The minor cycle simply executes all processes of priority-class PC_k in every round which is a multiple of 2^k.

The major problem with this algorithm is illustrated in figure 2. Assume there is a single process $p0.1$ with priority 0 and n processes $p1.i$ with priority 1. So, $\#PC_0 = 1$ and $\#PC_1 = n$. The first minor cycle consists of $p0.1$. As S_1 executes all processes of a priority-class together, the second minor cycle includes all processes with priority 0 and priority 1 , i.e., this minor cycle has $n + 1$ processes. From a naive viewpoint, we can simply say that the processes are badly distributed.

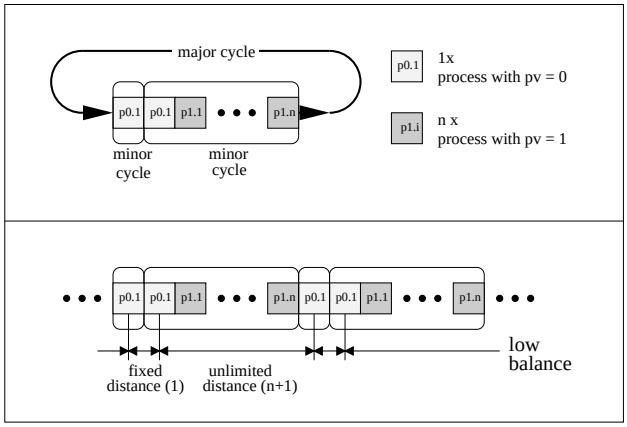

Figure 2: The simple scheduler S_1 leads to a so-called unbalanced execution. One minor cycle can consist of a single process $p0.1$ while a second minor cycle contains unlimited many other processes. Hence, the execution of $p0.1$ is not evenly spread.

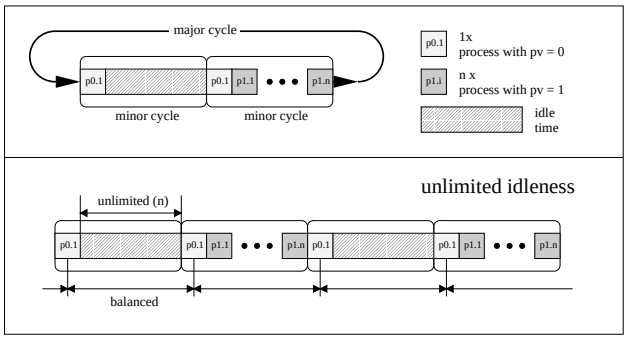

Figure 3: Adding idle-time to balance the schedule made by S_1 can lead to an unlimited waste of time.

```
1  /* Execute the Major Cycle */
2  for(round = 0; round < n_mic; round = round + 1) {
3          /* Execute the Minor Cycle */
4          ∀p_id ∈ P: {
5                  if(round modulo 2^pv[p_i] == 0) {
6                          execute p_id
7                  }
8          }
9  }
```

Figure 1: A simple scheduler S_1 which is very inefficient.

In a more formal approach, the so-called *balance* of a schedule S is defined as

$$balance(S) = \min \frac{\min\ dist(p_i, x)}{\max\ dist(p_i, y)}$$

with $dist(p_i), z)$ is the number of processes which are executed between start of the execution of p_i in cycle z and its next execution in cycle $z + 2^{pv[p_i]}$. If the balance is one then the schedule manages an equidistant spreading of every process over the cycles. If the balances is close to zero than there is at least one process which is very unevenly executed.

A small balance is very bad. As illustrated in the above example, a process with low priority-value, i.e., a process which should be executed very often, has to wait for an unbounded time-period. This is also expressed by the balance of S_1 which is in this case:

$$balance(S_1) = \frac{1}{n} = 0 \text{ for } n \to \infty$$

The balance of S_1 can be improved by adding idle-time as illustrated in figure 3. This way, the balance can even be tuned to reach the optimum of one. But this is bought at the cost of an unlimited waste of time. The *idleness* as the sum of idle-times in a major cycle is now unbounded.

In general, a schedule S is *time-optimal* if and only if the idleness is zero.

4 Properties of B-Scheduling

The workload WL within a major cycle can be computed as the sum of the occurrences of each process, i.e.,:

$$WL = \sum_{0 \le i \le maxpv} \#PC_i \cdot 2^{maxpv-i}$$

The number n_{mic} of minor cycles per major cycle is determined by the highest priority-value $maxpv$ as the process or the processes with this priority has/have to be executed once per major cycle. It follows that the average number av of processes per minor cycle has to be

$$av = WL \ / \ n_{mic} \text{ with } n_{mic} = 2^{maxpv}$$

For an even distribution of the workload, the actual number of processes in a minor cycle has to be equal to the average number av. Unfortunately, av is not necessarily an integer. Therefore, we define

$$perfect = \lceil av \rceil \quad \text{and} \quad dirty = \lfloor av \rfloor$$

A so-called *perfect minor cycle* has perfectly many processes, whereas the number of processes in a *dirty minor cycle* accordingly is dirty. A *bad minor cycle* includes more than $perfect$ or less than $dirty$ many processes. B-scheduling computes a schedule S_B such that

1. S_B is time-optimal
2. the major cycle consists only of perfect and dirty minor cycles
3. the processes are distributed over the cycles in an optimal manner, i.e., p_i is executed in cycle $c + 2^{pv[p_i]}$ if and only if p_i is executed in cycle c

```
1  /* Initialization */
2  /* computing the initial wait-values for each process p_id */
3  quicksort(P)
4  pc = 1
5  start = 0
6  n_slots = 1
7  ∀i ∈ {0, ..., maxpv − 1} : {
8          start = 2 · start
9          n_slots = 2 · n_slots
10         ∀id with pv[p_id] = pc : {
11                 wait[p_id] = reverse((start + id) modulo n_slots)
12         }
13         start = (start + #{p_id | pv[p_id] = pc}) modulo n_slots
14         pc = pc + 1
15 }
```

Figure 4: The initialization of B-scheduling.

```
1  /* Execute the Major Cycle */
2  for(round = 0; round < n_mic; round = round + 1) {
3          /* Execute the Minor Cycle */
4          id = 0
5          done = 0
6          while( (done < perfect) ∧ (id < #P) ) {
7                  if(wait[p_id] == 0) {
8                          execute p_id
9                          wait[p_id] = 2^pv[p_id]
10                         done = done + 1
11                 }
12                 id = id + 1
13         }
14         ∀p_id ∈ P : if(wait[p_id] > 0) : wait[p_id] = wait[p_i] − 1
15 }
```

Figure 5: The execution of a B-schedule.

It follows from properties 2 and 3 that S is well balanced as

$$balance(S_B) = \frac{dirty + 1}{perfect + 1}$$

$$= 1 \text{ for } av \to \infty$$

The worst-case balance of S_B is 1/2 when only two processes are used and one is more frequent than the other. In general, the balance becomes better the more work-load is handled in each minor cycle. It can be shown that any time-optimal schedule S has to consist of a set of dirty and perfect cycles. Hence, S_B provides the best balance that is possible.

5 The Heart of B-Scheduling

Figure 4 and figure 5 show the critical parts of B-scheduling in a pseudo-code. An important variable in both parts is $wait[p_{id}]$. It specifies for each process p_{id} how long it has to wait in number of cycles until it is executed again. During the execution of a B-schedule (figure 5), wait is constantly decremented in each cycle. When a process p_{id} is executed, its wait $wait[p_{id}]$ is set to $2^{pv[p_{id}]}$. Therefore, the execution of p_{id} is spread evenly other the minor cycles in the major cycle.

The dynamic execution part of a B-schedule (figure 5) is more or less straightforward. The "real magic" is done in the static initialization of the *wait*-values (figure 4). Note that the initial value of $wait[p_{id}]$ determines in which minor cycle p_{id} will be executed for the first time. So, computing suited initial waits produces a B-schedule. Note, that the number of *wait*-values is equal to the number of processes $\#\mathcal{P}$. So, the complete schedule which is of size $O(2^{\#\mathcal{P}})$ is represented in a single variable in each processes, i.e., in the overall size $O(\#\mathcal{P})$.

Before discussing the initialization of the *wait*-values in more detail, a special command from figure 4 has to be explained. The `reverse()` is used to reverse the bit-order of a binary number. More concretly, let $B_n = [b_0, ..., b_{n-1}]$ and $R_n = [r_0, ..., r_{n-1}]$ denote two binary numbers, each represented as array of bits b_i, respectively r_i. The function `reverse`() is then defined as:

$$\texttt{reverse}(B_n) = R_n \text{ with } r_i = b_{n-i}$$

The main idea when computing suited initial *wait*-values is as follows. Imagine a set $\mathcal{S}$ of natural numbers with a cardinality equal to a power of 2. Let

minor cycle number	processes within the cycle
0	p1.1 p1.3 p3.1
1	p1.2 p2.1
2	p1.1 p1.3 p4.3
3	p1.2 p2.2
4	p1.1 p1.3 p4.1
5	p1.2 p2.1
6	p1.1 p1.3
7	p1.2 p2.2
8	p1.1 p1.3 p3.1
9	p1.2 p2.1
10	p1.1 p1.3
11	p1.2 p2.2
12	p1.1 p1.3 p4.2
13	p1.2 p2.1
14	p1.1 p1.3
15	p1.2 p2.2

Table 2: A simple example of a major cycle computed with B-scheduling. The notation $pX.Y$ denotes process number Y within priority-class PC_X. Note that there is no straight-forward distribution of dirty and perfect cycles, i.e., minor cycles which consist in this example of either two or three processes.

name	p1.1	p1.2	p1.3	p2.1	p2.2	p3.1	p4.1	p4.2	p4.3
$pv[]$	1	1	1	2	2	3	4	4	4
$2^{pv[]}$	2	2	2	4	4	8	16	16	16
wait	0	1	0	1	3	0	4	12	2

Table 1: A set of processes with their priority-values $pv[]$, their according waiting-time between executions, and their initial wait values calculated with the algorithm shown in figure 4. The wait values lead to the schedule shown in table 2 when the B-scheduler 5 is invoked.

$S(start, d)$ denote a sequence which begins at the number *start* and "jumps" further to numbers x which are distance d away, i.e., $x = (k \cdot d)\texttt{modulo}\#\mathcal{S}$ with $k \in \boldsymbol{N}$. When *start* and d are powers of 2, S is called harmonic. It holds that for each harmonic list S, we can create two harmonic lists S_1 and S_2 such that $S = S_1 \cup S_2$, namely:

- $S_1 = S(start, 2 \cdot d)$
- $S_2 = S(start + d/2, 2 \cdot d)$

The overall set $\mathcal{S}$ can be expressed as $S(0, 1)$. It can recursively be divided in smaller lists and sublist.

When computing the initial *wait*-values, the goal is to distribute processes such, that the minor cycles are equally filled up. Each execution process of class PC_k can be seen as a list $S(start, 2^{maxpv-k})$ of minor cycles. The first value for *start* is zero, i.e., the first slot in the first minor cycle is used. The distance d is $2^{maxpv-pv[p_0]}$. From then on, further lists can be computed. The difficulty is to keep track of the *start* position. Especially, so-to-say left-overs, i.e., empty lists not used up by class PC_{k-1}, have to be used when the class PC_{k-1} is handled.

Table 1 shows as an example a set of processes with their priority-values $pv[]$, their according waiting-time $2^{pv[]}$ between executions, and their initial wait values calculated with the algorithm shown in figure 4. The interested reader can try to find a time-optimal, well balanced schedule of the processes (of course without using the pre-computed wait-values). The time-optimal, well balanced schedule computed by B-scheduling is shown in table 2.

6 Conclusion

Behavior-oriented robotics is based on the distribution of control over various processes running in virtual parallel. Scheduling these processes, i.e., assigning processing power to them, is a non-trivial task. Existing approaches from the field of real-time systems mainly focus on the fulfillment of deadlines. For control, it is also important that behavior-processes are executed as regular as possible in addition. Here, the novel algorithm of B-scheduling is presented which handles behaviors running on different time-scales, represented through so-called exponential effect priorities. B-scheduling ensures both desired properties for behavior-control, namely time-optimal and periodically balanced execution of processes.

References

[Ark87] R. C. Arkin. Motor schema based navigation for a mobile robot. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 264–271. 1987.

[Ark92] R. C. Arkin. Cooperation without Communication: Multiagent Schema-Based Robot Navigation. *Journal of Robotic Systems*, 9(3):351–364, 1992.

[Bir98] Andreas Birk. Behavior-based Robotics, its scope and its prospects. In *Proc. of The 24th Annual Conference of the IEEE Industrial Electronics*. IEEE Press, 1998.

[Bro86] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. In *IEEE Journal of Robotics and Automation*, volume RA-2 (1), pages 14–23. apr, 1986.

[Bro90] Rodney A. Brooks. The Behavior Language; User's Guide. Technical report, Massachusetts Institute of Technology, A.I. Lab., 1990.

[Bro91] Rodney Brooks. Intelligence without reason. In *Proc. of IJCAI-91*. Morgan Kaufmann, San Mateo, 1991.

[BW97] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages*. Addison-Wesley, 1997.

[CA98] Ronald C. Arkin. *Behavior-Based Robotics*. The MIT Press, 1998.

[LL73] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, jan, 1973.

[LSD89] I. Lehoczky, L. Sha, and Y. Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In IEEE Computer Society Press, editor, *Proceedings of the Real-Time Systems Symposium - 1989*, pages 166–171. IEEE Computer Society Press, Santa Monica, California, USA, dec, 1989.

[Mel83] Mellichamp. *Real-Time Computing*. Van Nostrand Reinhold, New York, 1983.

[Ste91] Luc Steels. Towards a theory of emergent functionality. In Jean-Arcady Meyer and Steward W. Wilson, editors, *From Animals to Animats. Proc. of the First International Conference on Simulation of Adaptive Behavior*. The MIT Press/Bradford Books, Cambridge, 1991.

[Ste94] Luc Steels. The artificial life roots of artificial intelligence. *Artificial Life Journal*, 1(1), 1994.

[You82] SJ Young. *Real Time Languages*. Ellis Horwood, 1982.