Behavior Trees with Stateful Tasks

Klöckner, Andreas

The behavior tree formalism as introduced recently to the application of mission management of unmanned aerial vehicles does provide for internal memory of mission plans. This is an important drawback for even simple plans such as waypoint sequences, because the information about visited waypoints must be stored outside of the plan execution engine. In this paper, two approaches are presented in order to provide tasks with states inside behavior trees: The first allows to embed regular state machines in a specialized behavior tree task. The second provides new memory and reset tasks in order to store information directly in the tree. Both approaches are shown to solve the waypoint following plan and promise to be applicable to a much broader range of mission management problems.

Keywords: Behavior Trees, State Machines, Mission Management

Behavior Trees with Stateful Tasks

Andreas Klöckner

Abstract The behavior tree formalism as introduced recently to the application of mission management of unmanned aerial vehicles does provide for internal memory of mission plans. This is an important drawback for even simple plans such as waypoint sequences, because the information about visited waypoints must be stored outside of the plan execution engine. In this paper, two approaches are presented in order to provide tasks with states inside behavior trees: The first allows to embed regular state machines in a specialized behavior tree task. The second provides new memory and reset tasks in order to store information directly in the tree. Both approaches are shown to solve the waypoint following plan and promise to be applicable to a much broader range of mission management problems.

1 Introduction

Current research aims at developing a number of different capabilities for unmanned aerial systems (UASs). Research groups engage in fields such as collision avoidance, formation flying or physical interaction with the environment. These capabilities do not only grow more and more diverse, but also integrate a number of low-level skills of the systems.

Additionally, practitioners seek to use UASs for an increasing range of different missions. Solar platforms e. g. are supposed to fly non-stop for several days and receive different missions as specified by the user during the flight. The operating cost of such a system is mainly determined by the personnel needed to operate the aircraft. Current solar aircraft constantly require multiple crew members to monitor the aircraft, weather, and traffic conditions.

Andreas Klöckner
DLR German Aerospace Center, Institute of System Dynamics and Control, Münchener Str. 20, D-82234 Oberpfaffenhofen, e-mail: andreas.kloeckner[at]dlr.de

In order to more efficiently deliver industrially relevant missions, the number and workload of the personnel must be decreased. This can be achieved by increasing the autonomous functions of the systems, which handle all the capabilities of the aircraft. Nonspecialist crew members must additionally be able to specify all conceivable missions targeted with the UAS. A scalable, intuitive, and flexible technique is thus needed for UAS autonomy.

Behavior trees were introduced to solve this challenge [5, 10]. They organize capabilities of the UAS in a tree of increasingly complex behaviors by using a standardized and simple interface. The formalism was first introduced for steering non-player characters in computer games [4]. It is argued that behavior trees combine a number of advantageous properties of state machines, scripting, and planning techniques [1].

However, behavior trees have a fundamental disadvantage: Behavior trees continuously adapt to changing input signals and do not have internal memory. They do thus not inherently provide means to implement behaviors requiring such internal state. This is especially important for UAS missions. These typically include behaviors such as following waypoint sequences, during which the fact of having reached a waypoint is only asserted by the sensor signals for limited duration. Mission plans implementing waypoint sequences thus require states to remember the visited waypoints. Unfortunately, this cannot be implemented easily in the standard behavior tree framework.

Researchers have invented ad-hoc solutions to this problem by describing specialized versions of the canonical behavior tree building blocks. While these are very practical solutions, they usually confound the logics of e. g. sequence with the memory introduced into the system. This paper thus describes two more general approaches in order to increase the system's modularity. The contributions of the paper are as follows:

- The formalism of conventional behavior trees is introduced in Sec. 2 and the shortcomings of state-free behavior tree implementations is shown at hand of a simple waypoint-following example in Sec. 3.
- In order to remedy this shortcoming two generic solutions are presented: Sec. 4 integrates state machines into regular behavior trees. In Sec 5, new task types handling memory within the behavior tree formalism are introduced.
- Both approaches are evaluated in the concluding Sec. 6.

2 Behavior Trees

Similar to hierarchical finite state machines, behavior trees use a hierarchy of operational modes to structure a complex mission. In a behavior tree, tasks are used instead of states and mode switches are triggered by internal statuses instead of external events.

A task is self-contained and goal-directed: it can be executed without a further framework in order to achieve a goal. A UAS mission e. g. would be composed of basic tasks, such as flying towards waypoints and probing the state of the aircraft. These basic tasks are composed in a tree structure to arbitrary complexity using generic composite tasks.

In order for the tasks to be modularly interchangeable, all tasks have the same interface to their parent tasks: All tasks report a status to their parent node, which can be either Running, Success, or Failure in the most basic behavior tree implementation. The parent nodes can activate and deactivate their children depending on their internal logics. A complete mission plan is executed by activating the root node of the tree.

The basic task types as mentioned above can be classified as actions and conditions. They actually interact with the aircraft's systems and determine their status based on custom implementation. Composite tasks are more generic and determine their status based on their internal logic and the statuses of their children. The most common composite tasks are selectors and sequences. A very basic behavior tree system thus provides the following four types of tasks:

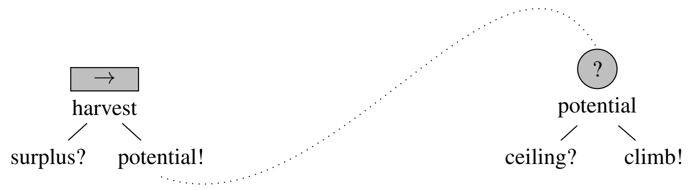
Actions provide interfaces to the aircraft system in order to change the its environment. They typically send low-level commands to the autopilot or payloads.

Conditions are used to test properties of the environment with boolean-valued functions. Examples are probing for minimal altitudes or a sufficient energy status. Conditions are specialized actions, because they cannot have Running status.

Selectors try to execute their children according to their priority and return Success, if one of their children is successful. Selectors are typically used to provide several alternatives to achieve a common goal. When comparing to logic, selectors can be regarded as an OR-operator.

Sequences activate all of their children one after another and return Success, only if all of their children are successful. They describe a series of tasks in order to achieve higher-level goals. Sequences correspond to the logical AND-operator.

Figure 1 demonstrates using behavior trees for the simple example of harvesting energy with a solar aircraft. The sequence in Fig. 1a activates its second child, if a surplus of energy is available. The second child consists of a selector to provide two strategies for maximizing the potential energy (see Fig. 1b). The first ensures that a given ceiling altitude is not exceeded and the second commands the aircraft to climb to that altitude.



(a) A sequence is used to describe the harvesting strategy by gathering potential energy, if a surplus of energy is asserted.

(b) The selector maximizes the potential energy either by holding a ceiling altitude or by climbing.

Fig. 1: Simple tasks can be connected hierarchically in a behavior tree in order to describe the energy harvesting strategy of a solar-powered aircraft. See [7].

Note that the statuses within a behavior tree are continuously evaluated. Each composite task reacts immediately to changes in its sub-tasks. This is opposed to the semantics of e. g. state machines, which retain once active states unless specific changes in the inputs trigger a change of states. Behavior trees do not include such internal memory by default in order to provide the modular logics as outlined above. This makes them very reactive, but prevents implementing behavior requiring internal memory.

Extensions to the basic notion of a behavior tree laid out here are proposed in the literature. Additional composite task types are e. g. parallel tasks used to execute multiple children simultaneously. Semaphores can be used to guard shared resources and loops repeat a task multiple times. Millington's textbook [9] provides a more detailed overview of behavior trees and common additional task types.

3 Behavior Trees in Mission Management

Taking the technology from the computer game industry, behavior trees were introduced to the UAS community to modularize control and mission management systems [10, 5]. Behavior trees provide a number of advantages for these application compared to the state-of-the-art technology of finite state machines.

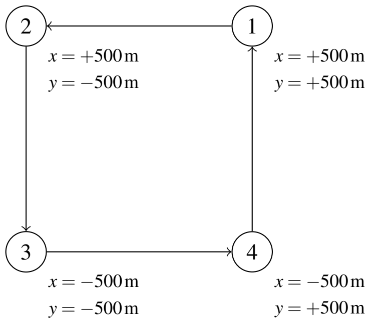
The growing number of capabilities provided by a UAS makes it difficult to maintain mission plans built as state machines for versatile aircraft, since every change requires rewiring wide parts of the mission plan. Behavior trees provide superior scalability in this case because of the standard interface of all tasks and because of the implicit switch logics. This makes it possible to modularly add, remove, and exchange arbitrary tasks at arbitrary locations in the tree without the need for global changes to the plan.

Additionally, the goal-directed semantics of behavior trees provide a very intuitive way of reading and building mission plans. Since each task can be used to achieve a sub-goal, higher-level goals can easily be composed by combining these tasks. Reading a behavior tree is intuitive on this very detailed level, but also on a high abstraction, where the actual leaf nodes are hidden from the user. This feature makes it also easy to provide an intuitive library of re-usable building blocks.

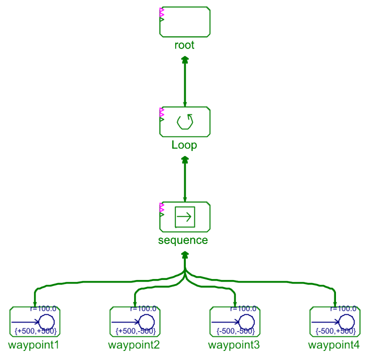
The close proximity of the basic composite tasks and logical operators additionally provides for means of validating mission plans [6, 2]. There are also approaches to behavior tree analysis building on translations to other formalisms such as hybrid dynamical systems [8].

Despite the conceptional advantages of behavior trees, there are certain things that cannot be done with standard behavior tree implementations. In particular, behavior trees in their canonical formulation do not have internal states and do not allow for proper initialization and termination of tasks. Behavior tree sequences e. g. restart a higher-priority sub-task, whenever it does not return Success anymore. This makes it hard to implement actual sequences such as simple waypoints in behavior trees.

Figure 2a shows a simple, rectangular waypoint plan to be flown by a UAS. This plan cannot easily implemented as a behavior tree because of the missing internal state. A naïve behavior tree implementation is shown in Fig. 2b. It contains the four waypoints as instances of a waypoint task taken from a task library in a sequence. A loop decorator is intended to repeat the waypoint sequence, once it has completed.



(a) The example mission consists of four waypoints arranged in a rectangular flight plan. The plan is to be flown by a small research UAS.

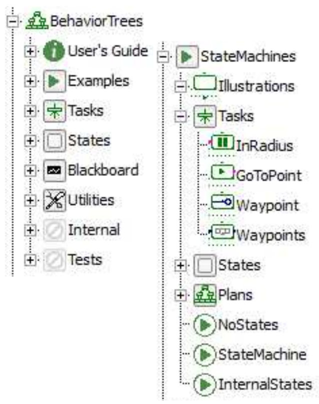


(b) An implementation using state-free behavior trees will prove unable to describe a waypoint plan, such as shown in Fig. 2a.

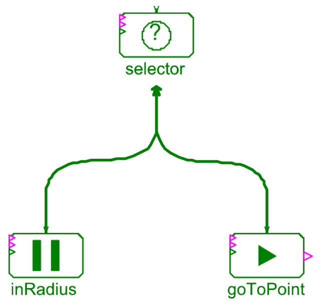
Fig. 2: Common segments of UAS missions consists of a set of waypoints to be visited by the UAS. These missions must be representable with a behavior tree for practical applications. This example shows the four waypoints used as a demonstration of this required capability.

The behavior tree is built using the Modelica BehaviorTrees library [7]. The library provides a generic framework to graphically compose behavior trees (see Fig. 3a). It also contains extensions to the behavior trees formalism such as the additional Accept status for tasks, which can be activated (see [5]). Application specific libraries can be derived easily in an object-oriented fashion. The waypoint task in Fig. 3b is e. g. composed of a condition to assert a proximity of 100 m to the waypoint and a steering task commanding a heading angle to the UAS. The signals are exchanged with the UAS model through embedded blackboard components.

As mentioned before, the implementation with standard behavior tree tasks fails to complete the mission as intended. Figure 4 shows the results of a simulation with a simplified UAS model. The UAS correctly approaches the first waypoint and then activates the second waypoint task. However, as soon as the proximity radius of the first waypoint is left by the UAS, the plan re-engages the first waypoint task. This leads to the UAS flying in circles at the boundary of the first waypoint's proximity radius.

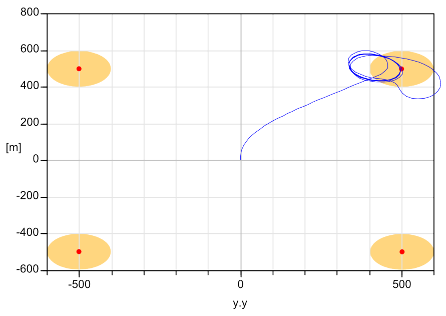


(a) The Modelica BehaviorTrees library provides a general behavior trees framework. Task libraries as e. g. for this paper can be derived easily.

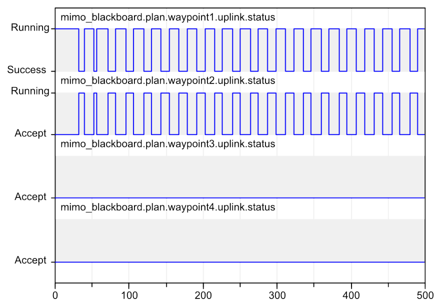


(b) The waypoint following task is implemented with a selector. It checks, if the UAS is inside a proximity radius of 100 m to the waypoint, and approaches the waypoint otherwise.

Fig. 3: All plans in this paper are built with the Modelica BehaviorTrees library [7].



(a) The UAS flies to the first waypoint and then starts to move around this waypoint in an infinite circle.



(b) The behavior is caused by re-activating the first waypoint, when the second leads the UAS out of the proximity circle of the first waypoint.

Fig. 4: The state-free conventional behavior tree is unable to implement the intended mission correctly, because the state-free sequence falls back to higher priority subtasks, when their success conditions are no longer fulfilled.

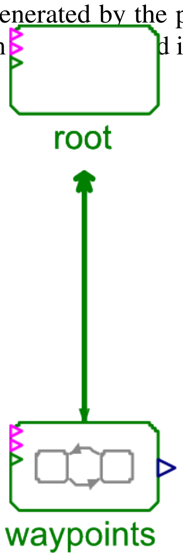
4 Embedding state machines inside behavior trees

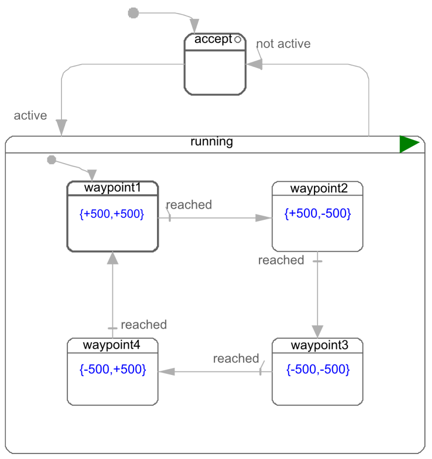
For state transitions such as between waypoints, the typical engineering approach is to use state machines. However, using state machines to model an entire mission plan means to abandon the superior modularity of behavior trees. It is therefore desirable to allow for a systematic integration of state machines inside behavior trees.

In order to allow for proper initialization and termination of tasks, a more comprehensive status cycle was already introduced in prior work [5, 7]. This modification effectively embeds a state machine in every task allowing to detect activation and deactivation of the task. Since regular behavior tree actions may contain arbitrary custom code, it is an obvious extension to also allow arbitrary state machines inside a behavior tree task. A similar approach integrates both formalisms in a third execution environment [11].

Figure 5 presents such an embedded state machine for the application described above. It simply consists of the four waypoint states. Each of the waypoint states generates a steering signal for the UAS simulation. The state machine is modeled by the synchronous elements of the Modelica language [3]. It could thus be exchanged by an arbitrary state machine described in the Modelica framework.

The waypoint states in Fig. 5b are hierarchically embedded in a *running* state. This hierarchy layer is used to generate correct behavior tree statuses for passing them up the tree. Another state, *accept* is used to allow the state machine to be stopped by the tree logic. States for generating any other allowed status are also provided in the BehaviorTrees library. The top-level states are switched using the active flag generated by the parent tasks in the tree. The tree is shown in Fig. 5a together with d interface to the UAS.



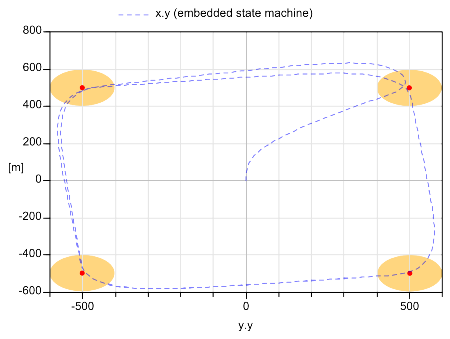


(a) A specialized behavior tree task is introduced in order to embed a regular state machine inside the behavior tree. The mission plan is additionally equipped with blocks used to communicate the state machine's commands with the aircraft model.

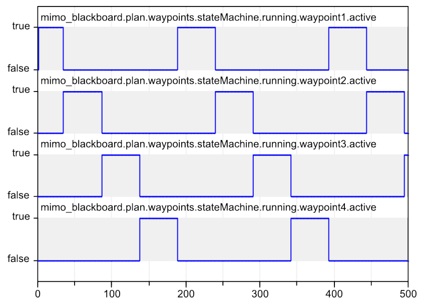
(b) The embedded state machine consists of the four waypoint following states and top-level states related to the behavior tree interface. The top-level states generate a valid status to be used in the behavior tree. They are switched by the behavior tree's active flag.

Fig. 5: In order to allow states in a behavior tree, it is combined with the common technique of state machines.

The resulting trajectory, when simulating the mission plan from Fig. 5 with the UAS model, is shown in Fig. 6. The UAS now correctly follows the prescribed waypoints in a square of each 1000 m height and width. Figure 6b also shows the active states during the simulation. The states iterate the four waypoints as intended. It should be noted that the Modelica state machine formalism only allows for one transition per clock cycle. Therefore, the waypoint plan remains in its *accept* state for one cycle at the beginning of the simulation. In the example, the state machine is clocked with an interval of 1 s.



(a) The UAS repeatedly follows the four waypoints as intended on a square with each 1000 m height and width.



(b) The state machine switches between the four waypoint states as expected. The state machine takes one clock cycle to start.

Fig. 6: The mission plan of integrated behavior trees and state machines is able to correctly follow the intended mission plan. The state is stored exclusively in the embedded state machine.

5 Stateful tasks for behavior trees

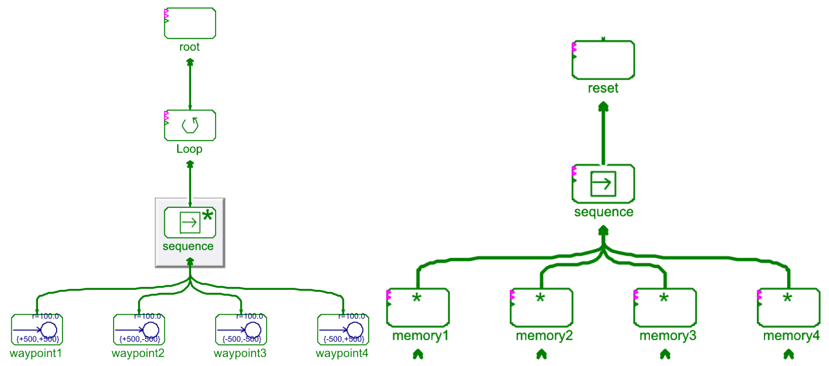
Although the solution described above provides for the desired functionality, it breaks the modularity of continuously using behavior trees. For planning purposes it is more convenient to provide the necessary states inside the behavior tree formalism. Several authors thus describe special variants of the common composite tasks (selectors and sequences) with stateful behavior. These variants are e. g. called “sequence*” [8] and exist in addition to the regular state-free variants.

Since these extensions mix standard nodes such as a sequence with a memory behavior, their modularity can be improved by separating the two behaviors. To this end, this paper introduces a memory and a reset decorator:

Memory decorators remember their status, after their child has entered a successful or failed status. They thus modify the return status of their child task such that it is prevented from being reactivated by the behavior tree.

Reset tasks send a new reset signal to their child task, when they are activated. This signal is distributed through all composite nodes to the underlying memory tasks. The reset task can thus be used to reset the memory tasks located in their child branch.

These new tasks can be used together with a regular sequence to model the behavior of previously introduced sequence* tasks (see Fig. 7b). However, the memory and reset tasks can be used in a more arbitrary way, thus allowing more flexible behaviors. In order to continuously evaluate the first child in Fig. 7b, the first memory task could e. g. be removed. The composed sequence* task can now be used in order to allow actual sequences in a behavior tree (see Fig. 7a).

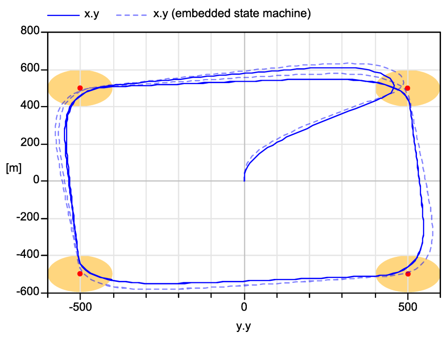


(a) The conventional sequence task is replaced in the behavior tree mission plan by a stateful sequence* task.

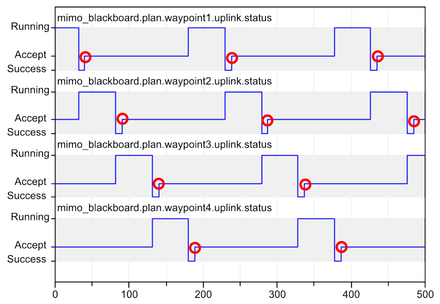
(b) The sequence* task consists of a regular sequence task with subordinate memory tasks and a superior reset task.

Fig. 7: Stateful behavior tree tasks are introduced in the waypoint plan as the sequence*, memory and reset tasks.

Figure 8 shows the results of a simulation with the mission plan defined in Fig. 7. The sequence of waypoints is correctly followed repeatedly as intended. Figure 8b additionally shows the statuses of the four waypoint tasks. It can be seen, that the tasks return Accept after the proximity radius of the waypoint is left. These events are marked with red circles in Fig. 8b. However, the superior memory task remembers the Success status returned previously and prevents re-activating the waypoint tasks. Only, when the loop task restarts the complete sequence at about $t = 180\,\mathrm{s}$, all memory tasks are reset and allow to steer towards the first waypoint again.



(a) The UAS follows the four waypoints as expected. The dashed line reproduces results using an embedded state machine for comparison.



(b) The waypoint statuses change between Accept, Running, and Success. Reactivation is prevented by memory tasks in the sequence* task.

Fig. 8: Using the stateful tasks effectively solves the waypoint following mission plan, while still providing for a modular behavior tree framework.

6 Conclusions

In the present paper, the technology of behavior trees was introduced and the need for stateful tasks within behavior trees was motivated. Two solutions were proposed in order to introduce such stateful tasks into the behavior tree formalism. The first integrates state machines within behavior trees and the second introduces the new memory and reset tasks.

Both solutions provide for accurate means to implement stateful behaviors such as waypoint plans with behavior trees. An example waypoint plan is successfully followed with both plans. The solutions are completely compatible to the existing BehaviorTrees library [7] and Modelica state machines [3].

The first solution allows to integrate arbitrary state machines into arbitrary behavior trees. This makes available the full power of well-understood state machines to the mission designer. However, this technique forces the state machine to be clocked. It thus introduces delays in the control flow. Additionally, the introduced time events might prove problematic for complex UAS models and long-term mission simulation.

The new task types of the second solution use the same interface as all other behavior tree tasks. They can thus literally introduce states in any place of a regular behavior tree. This approach even increases the modularity of stateful tasks in a behavior tree over the first approach. However, the theoretical properties of the technique such as termination or dead-locks have not yet been investigated as is the case for well-studied state machines.

Future research will target using the new approaches for industrially relevant applications. More detailed extensions of the behavior tree formalism will need to be developed when facing new challenges. In parallel, the theoretical properties of the approaches will need to be investigated in order to prove their applicability to the high standards of the aircraft industry.

References

1. Alex J. Champandard. Behavior Trees for Next-Gen Game AI. In *Game Developers Conference*, Lyon, France, 3-4 Decembre 2007. URL: `http://aigamedev.com/insider/presentation/behavior-trees/`.
2. M. Colledanchise, A. Marzinotto, and P. Ögren. Performance analysis of stochastic behavior trees. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3265–3272, May 2014. `doi:10.1109/ICRA.2014.6907328`.
3. Hilding Elmqvist, Fabien Gaucher, Sven Erik Mattsson, and Francois Dupont. State machines in modelica. In Martin Otter and Dirk Zimmer, editors, *Proceedings of 9th International Modelica Conference*, number 76 in Linköping Electronic Conference Proceedings, pages 37–46, Munich, Germany, September 3-5 2012. Modelica Association and Linköping University Electronic Press. ISBN: 978-91-7519-826-2. ISSN: 1650-3686. eISSN: 1650-3740. `doi:10.3384/ecp1207637`.
4. Damián Isla. Handling complexity in the Halo 2 AI. In *Game Developers Conference*, 2005. URL: `http://www.naimadgames.com/publications/gdc05/gdc05.doc`.
5. Andreas Klöckner. Behavior Trees for UAV Mission Management. In Matthias Horbach, editor, *INFORMATIK 2013: Informatik angepasst an Mensch, Organisation und Umwelt*, volume P-220 of *GI-Edition-Lecture Notes in Informatics (LNI) - Proceedings*, pages 57–68, Koblenz, Germany, 16-20 September 2013. Gesellschaft für Informatik e.V. (GI), Köllen Druck + Verlag GmbH, Bonn. ISBN 978-3-88579-614-5. URL: `http://www.gi.de/service/publikationen/lni/`.
6. Andreas Klöckner. Interfacing Behavior Trees with the World Using Description Logic. In *AIAA Guidance, Navigation and Control Conference*, Boston, MA, 19-22 August 2013. AIAA. AIAA 2013-4636. `doi:10.2514/6.2013-4636`.
7. Andreas Klöckner. The Modelica BehaviorTrees Library: Mission Planning in Continuous-Time for Unmanned Aircraft. In Hubertus Tummescheit and Karl-Erik Årzén, editors, *Proceedings of the 10th International Modelica Conference*, number 96 in Linköping Electronic Conference Proceedings, pages 727–736, Lund, Sweden, March 10-12 2014. Modelica Association and Linköping University Electronic Press. ISBN: 978-91-7519-380-9. ISSN: 1650-3686. eISSN: 1650-3740. `doi:10.3384/ECP14096727`.
8. A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren. Towards a unified behavior trees framework for robot control. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5420–5427, May 2014. `doi:10.1109/ICRA.2014.6907656`.
9. Ian Millington and John Funge. *Artificial intelligence for games*. Morgan Kaufmann, Burlington, MA, 2nd edition, 2009. ISBN 978-0-12-374731-0.
10. Petter Ögren. Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees. In *AIAA Guidance, Navigation and Control Conference*, Minneapolis, Minnesota, 13 - 16 August 2012. AIAA. AIAA 2012-4458. `doi:10.2514/6.2012-4458`.
11. Ismael Sagredo-Olivenza, Marco Antonio Gómez-Martín, and Pedro A González-Calero. Un modelo integrador de máquinas de estados y árboles de comportamiento para videojuegos. In David Camacho, Marco Antonio Gómez-Martın, and Pedro Antonio González-Calero, editors, *Proceedings 1st Congreso de la Sociedad Espaola para las Ciencias del Videojuego (CoSECivi)*, volume 1196 of *Workshop Proceedings*, Barcelona, Spain, June 24 2014. Sun SITE Central Europe (CEUR). ISSN 1613-0073. URL: `http://ceur-ws.org/Vol-1196/cosecivi14_submission_27.pdf`.