AIAA 2013-4636

Interfacing Behavior Trees with the World Using Description Logic

Andreas Klöckner*

In order to rigorously analyze mission plans, they have to be translated into a tractable formalism. This paper proposes to use the description logic *A*ttributive *L*anguage with *C*omplements and concrete *D*omains $\mathcal{ALC}(\mathcal{D})$ as the input formalism for a mission plan based on behavior trees. The interface is described using the safety circuit of an exemplary unmanned aerial vehicle. The system is executed using a three-degree-of-freedom simulation model. Results indicate that the system can be used as a first step towards verification of mission plans with formal methods. In a more efficient variant, the behavior tree mission plan executes sufficiently fast to be deployed in actual flight computers.

I. Introduction

Unmanned aerial vehicles (UAVs) are evolving into multi-purpose vehicles and incorporate a great range of capabilities for a multitude of missions. Different platforms such as helicopters, fixed-wing airplanes and airships exist, which can incorporate complex systems such as aerial manipulators, computer vision and solar energy systems. Also the missions envisioned with modern UAVs grow more complex and autonomous than e.g. taking photographs with remotely piloted vehicles. The aircraft must be capable of handling situations like adverse weather conditions, changing mission objectives or system failures.

This growing complexity of UAV systems calls for integrating technologies on all associated fields. Modular simulation models of all systems are used to assist in engineering decisions.[1] Generic autopilot hardware[2] and software[3] can be employed for different platforms easily. Also mission management systems must be provided with a flexible, modular, and scalable system for integrating different capabilities of the UAVs.[4]

In order to guarantee the safety of UAV operations, mission engineers should additionally be able to formally validate mission plans against mission objectives and possible states of the environment. This validation should especially ensure, that a plan always provides a suitable action to take. Additionally, a plan's success or failure should coincide with the defined goal of a mission.

The first step towards this formal validation is to separate the mission plan from complex simulation models and formalize it in a way, which is suitable for automated validation. This paper proposes to use behavior trees as the mission planning formalism and description logic to formalize the interface between the mission plan and the simulation models.

Behavior trees have recently been introduced into the discussion about integrative control architectures for UAVs.[4] Especially when a high number of capabilities need to be integrated in a single vehicle, the clear and modular scheme of behavior trees provides a promising framework. In comparison to conventional finite state machines, behavior trees feature better scalability with the growing numbers of possible capabilities to be integrated.[5] An extended implementation of behavior trees for UAV mission management has been proposed recently.[6]

Despite the clear interface between the nodes of a behavior tree, the interfaces of behavior trees with the world are not as clearly defined and remain hard-coded solutions. Additionally, assumptions on world and sensor models such as the definition of critical situations must be hard-coded in the sensor queries. For analysis and design of mission plans based on behavior trees, a standardized interface provides better control over the interactions between the UAV, the world and the plan.

Since the conditions used for switching tasks within behavior trees are boolean signals and because of the basic behavior tree building-blocks being closely related to logical operators, logic is a natural choice

*Research Associate, German Aerospace Center, Institute of System Dynamics and Control, Münchner Straße 20, D-82234 Oberpfaffenhofen-Weßling.

for formalizing the inputs of behavior trees. Using logic as an input interface overcomes the disadvantages mentioned above. It allows to clearly and modularly separate different models, such as the definition of critical situations or sensor uncertainties. These are formally defined as general sentence of knowledge about the world. The information retrieval is then just another module in the world knowledge base. With a suitable logic, even complex queries can be formulated, such as requesting that more than one airport shall be in the UAV's range.

This paper proposes to use description logic for this purpose. Description logic in its many variants is a well established and deeply investigated formalism for knowledge representation systems. A comprehensive overview is available as a handbook.[7] The formalism is for example used in the Web Ontology Language in the framework of the semantic web. It is especially suited for the description of complex worlds, because it applies the open world assumption, i.e. it avoids making assertions about the world without explicit knowledge. The complexity of answering description logic queries is collected for many variants[8] and there are efficient reasoners readily available. The formalism can further be extended with a notion of time and has been used previously for robot action planning.[9]

Formalizing mission plans in the way proposed in this paper allows for reasoning about plans and is a step towards formal verification of plans.[10] If actions are also formalized in this way, the design of plans[11] is facilitated. Logical inference can be used to assist in plan optimization and generation by e.g. deriving possible actions.

The paper is organized as follows: Section II gives an overview on behavior trees for UAV mission management. Section III describes the description logic $\mathcal{ALC}(\mathcal{D})$ used in this work and the basic reasoning services it allows. In Sec. IV, the proposed combination of behavior trees and description logic is explained using a simple example. Results from simulation are shown in Sec. V. The paper concludes with a summary and an outlook in Sec. VI and VII.

II. Behavior Trees with Transient Tasks

Behavior trees have been developed for managing complex non-player characters in computer games. They have first been described for HALO 2 in 2005.[12] Since then, they have evolved into a well established framework for artificial intelligence in computer games. There are comprehensive introductions to the field.[5,13] In this work, the behavior trees introduced in a previous paper[6] are used to model mission plans. This introduction is based on the original paper.

The behavior tree formalism describes complex behavior as a hierarchical organization of elementary building blocks in a tree. These *tasks* are enabled and disabled by persisting statuses and by the switch logic implicitly coded in the tree structure. This property describes the main difference to the discretely triggered and explicitly modeled state transitions of Hierarchical Finite State Machines (HFSMs).

Each task is self-contained, in that it describes a complete behavior including its preconditions and the evaluation of its outcome. Tasks are also goal-directed, because the evaluation states whether or not the task's goal is achieved. The simplest tasks in a UAV mission management system are e.g. querying and setting autopilot modes, such as altitude hold.

These simple tasks are the interface of the behavior tree to the world. They are found at the leaf nodes of the behavior tree. If a task is composed of sub-tasks, it is called a composite task. These tasks form the intermediate layers of the tree. Almost arbitrary behaviors can be formed by arranging atomic and composite tasks in the tree structure.

While the interface of the behavior tree to the world is composed of arbitrary custom code, the internal interface between a composite task and its sub-tasks is highly standardized. Execution of the tree is started at the top-most task, the root of the tree. Each composite task queries the status of its sub-tasks by executing them. The process is called *ticking* the sub-tasks. Based on the statuses of the sub-tasks, the composite task determines its own status. The process is repeated iteratively until the tick reaches the leaf nodes of the tree. The status of a task can be either *Success*, *Failure*, or *Running*.

A basic behavior tree has two kinds of leaf nodes: Actions and conditions. The processing of these two basic tasks is shown in Fig. 1. Each *action* tries to change the state of the world by e.g. engaging autopilot modes, switching payloads, or taking photographs. It returns Success (✓), if the action is successfully completed, and Failure (↯), if the action is unsuccessfully aborted. During execution of the action, it has the status Running (▶). A *condition* is a specialized action that never enters Running status. It is used to query information from the world's state and it immediately returns either Success or Failure, when ticked.

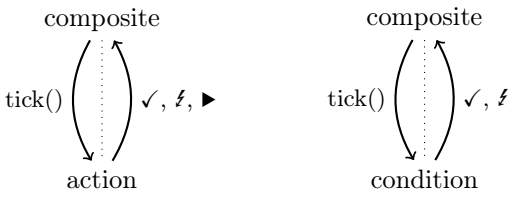


Figure 1. When an action is ticked, it can return either Success (✓) or Failure (↯), depending on whether it is completed successfully or aborted unsuccessfully. It returns Running (▶) during execution. The condition cannot enter Running status. It is used to query the state of the world.

The most common composite tasks of behavior trees are sequences and selectors. *Sequences* are goal-directed series of pre-conditions and sub-tasks. They execute all their sub-tasks in the specified order and return Success only, if all of them are completed successfully. Sequences thus draw an analogy to the logical AND operator. A *selector* is a list of prioritized tasks, suitable to ensure that a common goal is achieved. The selector dynamically selects the sub-tasks to be executed by ticking its sub-tasks in order until one of them returns Success or Running. It thus draws the analogy to the logical OR operator. The process of ticking these composite nodes is illustrated in Fig. 2. Thick arrows show a sample processing, dashed arrows show all possible alternatives.

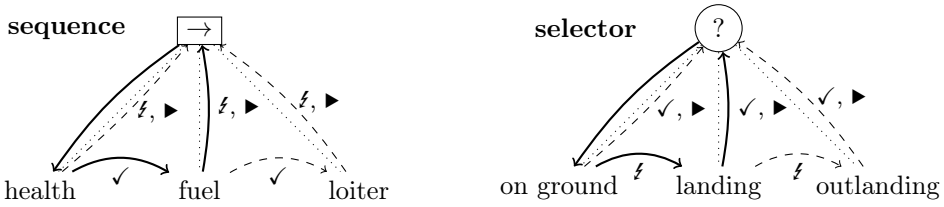


Figure 2. The example sequence executes the loiter action only, if both preconditions of being healthy and having fuel are satisfied. The exemplary case of the aircraft being out of fuel is shown with thick arrows. The example selector aims at bringing the aircraft safely to the ground by either checking for a previous landing, landing regularly, or performing an outlanding. The exemplary case of an ongoing regular landing is depicted with thick arrows.

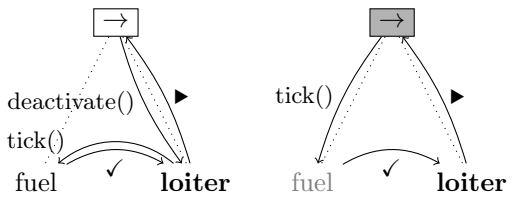
Additional composite tasks can be conceived by implementing different switch logics. Typical applications include e.g. parallel task to execute several sub-tasks in parallel. Loops are often used to repeat the same sub-task several times. The variety is only limited by the standard interface of a task.

This unified interface of ticking and a limited list of return codes allows to connect arbitrary sub-tasks to composite tasks. It makes behavior trees simple to understand and flexible to extend. Adding a new sub-task especially requires substantially fewer new links as compared to HFSMs. Relying on persisting statuses additionally makes initialization of a behavior tree a simple task. The close relation to logics can be exploited for formal validation of the mission plans.

For use in mission management, the basic scheme of behavior trees is extended with the introduction of entry and exit hooks similar to finite state machines. These are executed, when the task enters or leaves the Running state respectively. The task is said to be activated or deactivated. However, typically only one sub-task of a composite node can be in the Running state. This may lead to repeated deactivation and re-activation of the sub-tasks in order to tick e.g. preconditions. In order to guarantee chatter-free activation, deactivation and interruption, the behavior tree formalism is thus extended by the notion of transient tasks.[6] They do not enter a Running state and thus do not need to be activated in order to check them.

As stated earlier, conditions never enter the Running state. They are thus inherently transient. If a composite node only ticks transient nodes during a specific execution cycle, it also behaves transiently and propagates the transient behavior upstream towards the root node. This makes it possible to tick whole sub-trees without the need to activate them. This extension provides for more robust execution of behavior trees for use in UAV mission management. The extension is illustrated in Fig. 3 for re-ticking a sequence with a currently running action. The transient pre-condition is printed in gray and does not need to be activated in order to tick it. The modified composite node is indicated by a gray filling and the currently running action is printed in bold font.

Figure 4 shows an exemplary behavior tree as it might be employed for steering a solar-powered high-altitude aircraft on a communication relay mission. The plan contains a longitudinal and a lateral steering logic in parallel, several safety checks and strategies for energy harvesting and coping with adverse winds.



(a) chattering (b) transient tasks

Figure 3. The chattering introduced by unconditionally activating ticked sub-tasks is shown using the example of a loiter action, which is only executed, if the aircraft has sufficient fuel. It is repeatedly deactivated and reactivated with each tick. A transient pre-condition remedies this problem by being checkable without interruption of the loiter action.

III. The Description Logic $\mathcal{ALC}(\mathcal{D})$

The formalism of description logics has developed out of frame-based systems, terminological systems and concept languages. It has since then been developed into a well understood and thoroughly researched family of logical languages. The Description Logic Handbook[7] provides a comprehensive overview of the development and current state of description logics. In this work, the description logic variant $\mathcal{ALC}(\mathcal{D})$ is used.[14]

Description logics describe knowledge using the terminology of *concepts* and *roles*, defining abstract object classes and their relations. They can be regarded as general knowledge about the world. The semantics of the concept and role names are provided by the *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. In the interpretation, the set $\Delta^{\mathcal{I}}$ consists of all objects in the world. The function $\cdot^{\mathcal{I}}$ maps every concept name to a subset of $\Delta^{\mathcal{I}}$ and every role name to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. These semantics are summarized in Table 1 for the concept name C and the role name R.

Table 1. The interpretation $\mathcal{I}$ of description logics consists of the set $\Delta^{\mathcal{I}}$, containing all objects in the world. The function $\cdot^{\mathcal{I}}$ maps concept names C to a subset of $\Delta^{\mathcal{I}}$ and role names R to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Names	Interpretation
C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

Instances of objects in the world are called *individuals*. Knowledge about individuals is stored as *assertional axioms*. The collection of these axioms is called the *ABox*. The two most common assertional axioms are shown in Table 2 along with their semantics: The concept assertion states that the individual a belongs to the class of objects described by the concept C. The role assertion states that the two individuals a and b have the relationship defined by the role R.

Table 2. The most common pieces of knowledge about individuals are the concept assertion and the role assertion. They declare the individuals to be instances of the respective concept and role interpretations.

Assertion	Description	Semantics
$a\colon C$	concept assertion	$a \in C^{\mathcal{I}}$
$(a, b)\colon R$	role assertion	$(a, b) \in R^{\mathcal{I}}$

The family of description logics can have different degrees of expressiveness depending on which types of operators are allowed in complex concept descriptions. The description logic used in this work is called $\mathcal{A}$ttributive $\mathcal{L}$anguage with $\mathcal{C}$omplements and concrete $\mathcal{D}$omains: $\mathcal{ALC}(\mathcal{D})$. The syntax and semantics of the constructs introduced by the basic description logic $\mathcal{AL}$ and the two extensions are given in Table 3. The letters C and D denote concepts, the letters R and S denote roles. Predicates in the concrete domain are denoted P. In this work, the predicates P are linear, real-valued equality and inequality constraints of arbitrary arity. In $\mathcal{ALC}(\mathcal{D})$, no complex role descriptions are available. For a detailed description of the listed semantics see the Description Logic Handbook.[7]

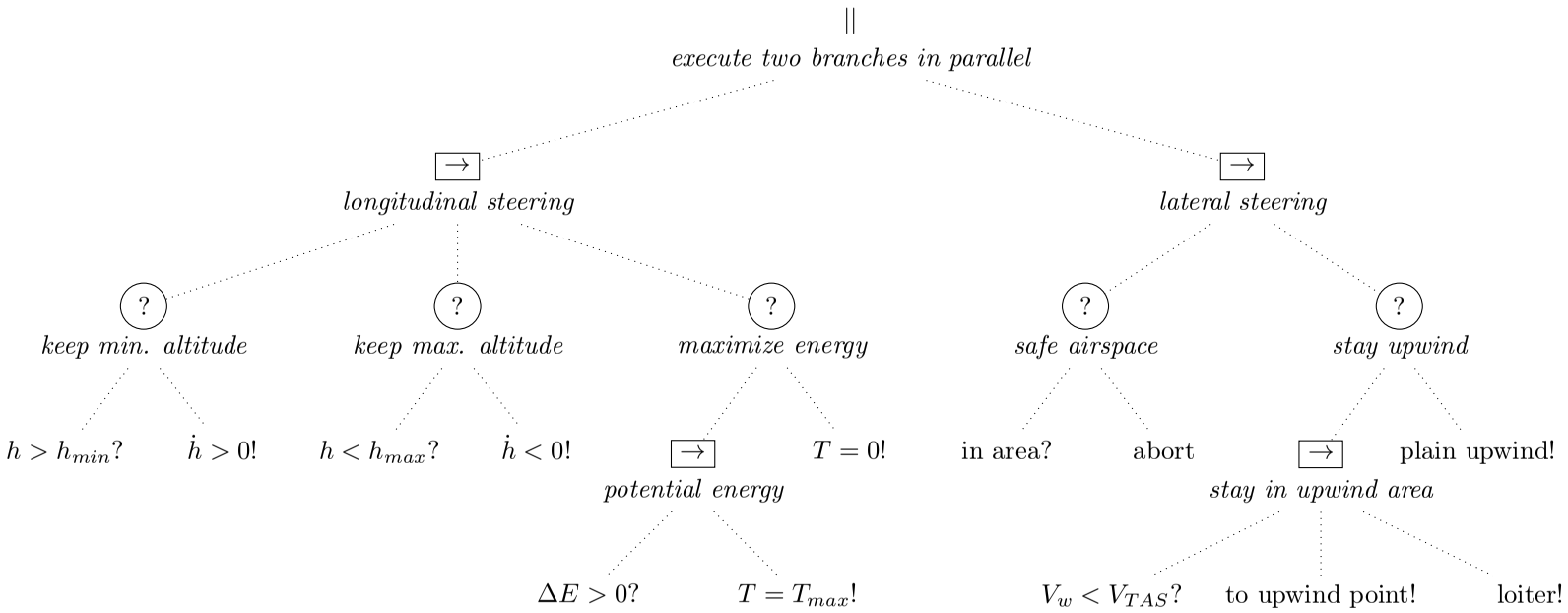


Figure 4. A more complex behavior tree plan than used in this work can be used for steering a solar-powered high-altitude aircraft on a communication relay mission. It includes two parallel branches for longitudinal and lateral steering logics. The longitudinal steering includes two safety altitude objectives and a plan to maximize the harvested energy. The lateral plan contains a safety layer to abort the mission, if the aircraft leaves safe airspace. Additionally, it contains a strategy to stay in a location as upwind as possible.

Table 3. The basic description logic $\mathcal{AL}$ defines the top and the bottom concepts. The intersection and union concept operators are interpreted as the equivalent set operators. The value restriction $\forall R.C$ describes the set of objects, for which all R-successors are instances of the concept C. The existential quantification $\exists R.C$ describes all objects with at least one R-successor belonging to the concept C. The $\mathcal{C}$omplements extension introduces concept negation. The concrete $\mathcal{D}$omain extension introduces real-valued predicates P for the value restriction and existential quantification. The predicates P can be of arbitrary arity. This table shows the case for binary predicates, such as $b + c < 2$.

Concepts	Interpretation	Description	Variant
$\top$	$\Delta^{\mathcal{I}}$	top concept	$\mathcal{AL}$
$\bot$	$\emptyset$	bottom concept	$\mathcal{AL}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	intersection	$\mathcal{AL}$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	union	$\mathcal{AL}$
$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall b\ (x,b) \in R^{\mathcal{I}} \Rightarrow b \in C^{\mathcal{I}}\}$	value restriction	$\mathcal{AL}$
$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists b\ (x,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$	existential quantification	$\mathcal{AL}$
$\exists R$	$(\exists R.\top)^{\mathcal{I}}$	limited quantification	$\mathcal{AL}$
$\neg C$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$	complement	$\mathcal{C}$
$\forall (R,S).P$	$\{x \in \Delta^{\mathcal{I}} \mid \forall b,c\ (x,b) \in R^{\mathcal{I}} \wedge (x,c) \in S^{\mathcal{I}} \Rightarrow P(b,c)\}$	predicate restriction	$(\mathcal{D})$
$\exists (R,S).P$	$\{x \in \Delta^{\mathcal{I}} \mid \forall b,c\ (x,b) \in R^{\mathcal{I}} \wedge (x,c) \in S^{\mathcal{I}} \Rightarrow P(b,c)\}$	predicate restriction	$(\mathcal{D})$

For convenience, abstract knowledge about concepts can be stored separately as *terminological axioms* The collection of these axioms is called the *TBox*. The commonly allowed terminological axioms are shown in Table 4 along with their semantics. The semantic distinction between atomic concepts A and complex concepts C and D is exploited in the reasoning algorithms.

Table 4. The common axioms about abstract concepts allow defining an atomic concept A by a complex concept description, declare two complex concepts as equivalent, and declare a concept C to be included in a concept D.

Axiom	Description	Semantics
$A \doteq C$	definition	$A^{\mathcal{I}} = C^{\mathcal{I}}$
$C \equiv D$	equivalence	$C^{\mathcal{I}} = D^{\mathcal{I}}$
$C \sqsubseteq D$	inclusion	$C^{\mathcal{I}} \subset D^{\mathcal{I}}$

The basic reasoning task needed to query information from the knowledge base is the consistency check. An ABox is said to be *consistent*, if there is an interpretation satisfying the given assertions. In doing this check, description logic reasoners employ the open world assumption: If a concept C and an individual a are mentioned in the knowledge base but there are no assertions $a\colon C$ nor $a\colon \neg C$, the algorithm does not implicitly assume one of the assertions. This is in contrast to other frameworks, in which the absence of $a\colon C$ implies $a\colon \neg C$. The open world assumption especially allows reasoning about incomplete world descriptions.

Most relevant reasoning tasks can be reduced to checking consistency of an ABox. The consistency check is efficiently performed using tableaux algorithms. The work presented in this paper employs a variant of the tableaux algorithm described by Baader and Hanschke,[14] which explicitly incorporates the constructs described earlier. The structure of tableaux algorithms is outlined here briefly.

Tableaux algorithms start with a given ABox and expand it according to a table of consistency preserving rules. Such a rule is for example the AND-rule, which adds $a\colon C$ and $a\colon D$ to the ABox, if $a\colon C \sqcap D$ is asserted. There are also non-deterministic rules, such as the OR-rule, which adds either $a\colon C$ or $a\colon D$ to the ABox, if $a\colon C \sqcup D$ is asserted. Rules can also generate new individuals, such as the EXIST-rule, which adds a new assertion $(a,x)\colon R$, if $a\colon \exists R$ is asserted. Figure 5 depicts the expansion of an exemplary ABox according to these rules.

Because of non-deterministic rules, the expansion is not linear, but a tree of expansions is created, until no more rules are applicable at the leaf nodes. The ABoxes at the leaf nodes are said to be *complete*. The consistency check then amounts to finding a leaf node ABox, which does not contain obvious contradictions called *clashes*. The most common clash is an ABox asserting both $a\colon C$ and $a\colon \neg C$. Another prominent clash is caused by contradictory constraints on the concrete domain. The latter is checked using linear programming satisfiability checks. If one clash-free leaf node is found, it is a proof for the original ABox's assertions to be satisfiable. The original ABox is then called consistent.

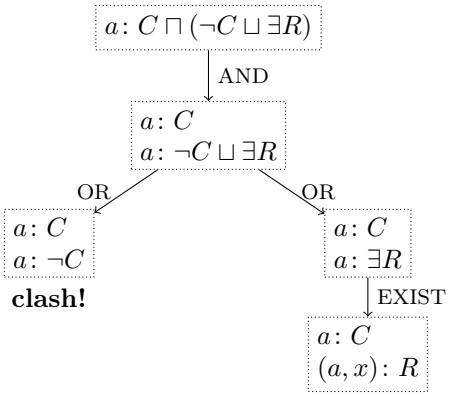


Figure 5. The expansion of an ABox using tableaux algorithms yields a tree of ABoxes with complete ABoxes at its leafs. The example employs the AND-, OR-, and EXIST-rules. While one leaf ABox contains an obvious clash, the other is clash-free. The given ABox is thus consistent because a feasible interpretation is found.

IV. Interfacing Behavior Trees and Description Logics

In order to control a UAV simulation model with a mission plan based on the behavior trees introduced in Sec. II, the plan needs to be provided with inputs from the simulation model. This section outlines a scheme to use the description logic described in Sec. III as a formalized interface, instead of directly checking sensor readings with custom implementation.

The formalism is explained using the safety program of an exemplary electric UAV. The main mission of the UAV is to fly along a given set of waypoints. In addition to this main plan, the UAV is equipped with a superior safety objective. When the battery state-of-charge approaches a critical level, the UAV shall activate a fallback plan and return to its base.

The behavior tree implementation for this safety precaution is shown in Fig. 6. The additional safety layer is introduced by the root selector node containing the safety plan as well as the main plan. The safety plan has a higher priority than the main plan. It contains a guard condition in order to only execute the fallback plan, if the battery level is critical. This condition is checked with every tick of the plan and provides a quick reaction in the critical case. In nominal conditions the main plan is executed.

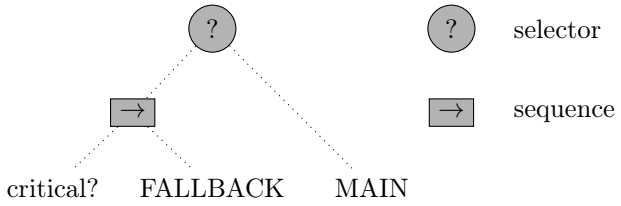


Figure 6. The example used in this section employs the safety layer of an electric UAV. The UAV is meant to return to its base, if the battery state-of-charge reaches a critical level. This condition will be queried through an abstracted logical interface.

The mission plan retrieves information about the world through two intermediate layers of abstraction as depicted in Fig. 7. The autopilot directly controls the aircraft simulation model based on sensor information. It additionally generates assertions in the description logic world model, which is used to reason about the current state of the world. Such information is retrieved by the behavior tree mission plan using queries. It is then used by the behavior tree to directly generate switching commands for the autopilot modes. These commands are currently not passed through the description logic world model.

The plan's state-of-charge condition is checked by querying information from a world model. In this example, the world includes the notion of a UAV with its associated sensor output and a notion of a critical battery state-of-charge. The battery state-of-charge is considered to be critical, if it is not sufficient to continue the main mission but the UAV can still return safely to its base.

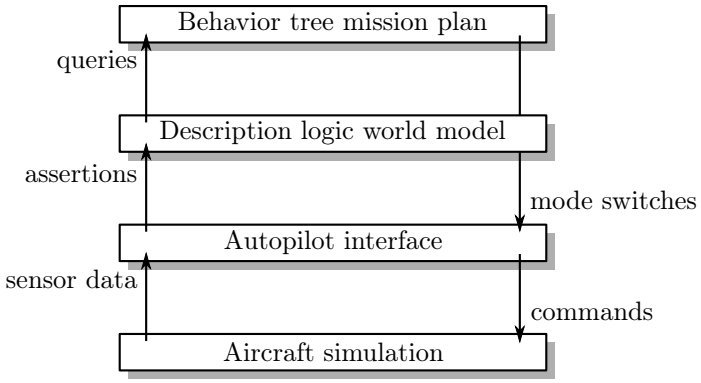


Figure 7. The interface between the mission plan and the aircraft simulation contains two intermediate layers of abstraction. First, the autopilot interface converts sensor readings into assertions about the world. Second, the description logic world model is used to query this information in a formalized way.

The first part of the world model is the TBox. It contains the general knowledge about the world. A UAV is defined by the description logic concept UAV. It is assumed to provide two sensor outputs: The current state-of-charge and the distance to its base station. The information is connected to the UAV concept by the roles *soc* for the state-of-charge and r for the distance to the UAV's base. For convenience, the additional role *range* is defined, which describes the range of the UAV with its current state-of-charge.

$$UAV \doteq \exists soc \sqcap \exists r \sqcap \exists range$$

In order to connect the state-of-charge with the range of the UAV, another terminological axiom is introduced. This is done using a top inclusion axiom. It introduces predicate restrictions, such that for all individuals with a state-of-charge and a range, their range is equal to the state-of-charge divided by a mean battery consumption per distance traveled c.

$$\top \sqsubseteq \forall(range, soc).(range = soc/c)$$

The concept $CRITICAL$ defines the notion of a critical battery charge level. It describes the case, where the range of the UAV is less than the distance to its base station. A safety margin of 500 m is also included.

$$CRITICAL \doteq \forall(range, r).(range \leq r + 500)$$

The second part of the world model is the ABox, which contains information about individual instances of the concepts and roles defined earlier. For the example considered here, it is only necessary to create one individual instance *me* of the concept UAV. The individual *me* is used by the sensor models to assign sensed data to the aircraft instance. At this point it becomes clear, that the formalism is suited for multiple instances of aircrafts, as for example in a coordinated mission scenario.

$$me\colon UAV$$

During the simulation, the autopilot interface adds additional assertions to the knowledge base. These assertions assign concrete values to the roles *soc* and r. The assertions can be exact, if the sensor is assumed to be reliable. This variant is used for the distance to the UAV's base station r.

$$me : \forall(r).(r = 1000).$$

The sensor assertions can also include uncertainties, if the sensor is assumed to have a known confidence interval. The state-of-charge in this example is contaminated with a measurement error of 1 %. To add more clarity to the knowledge base, the sensor uncertainty could also be modeled as a separate concept, thus requiring the sensor to simply assert its actual reading into the knowledge base.

$$me :\forall(soc).(soc <= 0.5 + 0.01)$$
$$me :\forall(soc).(soc >= 0.5 - 0.01)$$

In a final step, the conditions used in the behavior tree mission plan are translated into description logic queries. This is done by formulating equivalent assertions, which can be checked against the knowledge base. In the example above, the condition is translated into querying, if the UAV *me* is an instance of the $CRITICAL$ concept.

$$me : CRITICAL$$

There are two ways of querying this condition. One could ask, if the knowledge base allows for the condition to hold. This is done by adding the condition to the knowledge base and checking the overall ABox's consistency. The other way is to ask, if the knowledge base entails the condition. That is, the condition can be logically deduced from the knowledge base. This is done by adding the negated condition to the knowledge base and checking for inconsistency. Both variants require a consistent ABox to start with.

The two variants produce different results because of the open world assumption: Assume an empty baseline ABox. An arbitrary condition $a : C$ is then allowed but not entailed. For checking conditions in the mission plan, entailment is thus queried, because incomplete world models or missing sensor data shall not trigger actions unless explicitly requested.

V. Simulation Results

The interface introduced in the previous section is applied to a test case. The mission is executed using a three-degrees-of-freedom aircraft model. The mission plan is evaluated only once every five seconds, because the custom Matlab implementation of the description logic reasoner is not optimized and takes about one second to evaluate the fallback plan's condition. All other conditions in the plan directly query the sensor data for this reason. With custom code directly querying sensor signals, the behavior tree can be executed in real-time at 30 Hz.

In Fig. 8, the statuses of the relevant mission tasks are shown as functions of time. The statuses are shown as Idle (·), Activating (△), Running (▶), Success (+) and Failure (○). At the beginning of the simulation, the condition of the fallback plan is evaluated to Failure and the fallback plan is not activated. Instead, the main plan is activated. About 600 s in the simulation, the battery state-of-charge becomes critical. The corresponding condition evaluates to Success and causes the mission plan to abort its main plan and activate the fallback plan. When finally the base station is reached, the fallback plan evaluates to Success and stops the simulation. Figure 8 also shows the resulting trajectory of the UAV.

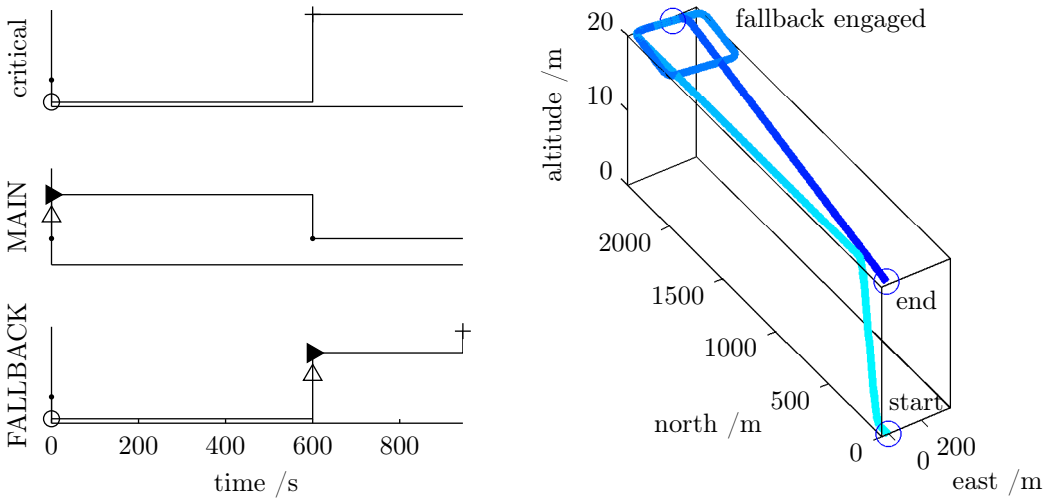


Figure 8. The evolution of the task statuses shows that first the main plan is activated. After 600 s, the battery state-of-charge becomes critical and the fallback plan is invoked. The response to this event can also be seen in the UAV's trajectory.

VI. Conclusions

A mission management system based on behavior trees is presented. It uses the description logic variant $\mathcal{ALC}(\mathcal{D})$ as interface to the world. A working implementation of the system is shown using a test case.

The presented system allows complex queries about the world to be used as input conditions to a behavior tree mission plan. This is done using a formal notation, for which the reasoning complexity is well understood. The combination of clear mission plan and interface formalisms provides for more rigorous analysis of mission plans. Simple support for uncertainty is given by using confidence intervals. This is the first step towards formal verification and design of such mission plans.

The formalized interface cannot be used in an online mission management system, because the computational burden is too high. Behavior tree mission plans using custom code without this formalization as interfaces, however, can be deployed efficiently to actual flight computers. The presented interface provides means for guaranteeing the integrity of these plans.

VII. Outlook

In the current state, the formalism will allow for instantaneous verification, such that a plan could be proven to always execute a proper action. With a formalization of also actions and the dynamics of the world and the aircraft, it could even be examined whether or not a plan is guaranteed to succeed. To this end, temporal extensions of description logics[15,16] could be used. The possible inference mechanisms could also assist in plan optimization and generation by advising possible actions for specific situations.

Immediate improvements of the presented interface can be made either on the side of performance or expressiveness. Integrating available reasoners or improved algorithms will speed up the run-time of the description logic part of the system. Added expressiveness can be valuable by allowing more constraint classes on the real numbers. Uncertainty could be handled by an appropriate extension.[17] In addition, description logic can also be used as a blackboard mechanism for communication between different behavior tree tasks.

References

[1]Klöckner, A., Leitner, M., Schlabe, D., and Looye, G., "Integrated Modelling of an Unmanned High-Altitude Solar-Powered Aircraft for Control Law Design Analysis," *CEAS EuroGNC*, edited by Q. Chu, B. Mulder, D. Choukroun, E.-J. van Kampen, C. de Visser, and G. Looye, Vol. Advances in Aerospace Guidance Navigation and Control – Selected Papers of the Second CEAS Specialist Conference on Guidance, Navigation and Control, Springer Berlin Heidelberg, Delft, The Netherlands, 10-12 April 2013 2013, pp. 535–548.

[2]Laiacker, M., Klöckner, A., Kondak, K., Schwarzbach, M., Looye, G., Sommer, D., and Kossyk, I., "Modular scalable system for operation and testing of UAVs," *American Control Conference*, IEEE, Washington, DC, 17-19 June 2013, pp. 1462–1467, ISBN: 978-1-4799-0176-0.

[3]Kastner, N. and Looye, G., "Generic TECS based autopilot for an electric high altitude solar powered aircraft," *CEAS EuroGNC*, CEAS, Delft, The Netherlands, 10-12 April 2013 2013.

[4]Ögren, P., "Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees," *AIAA Guidance, Navigation and Control Conference*, AIAA, Minneapolis, Minnesota, 13 - 16 August 2012, AIAA 2012-4458.

[5]Millington, I. and Funge, J., *Artificial intelligence for games*, Morgan Kaufmann, Burlington, MA, 2nd ed., 2009.

[6]Klöckner, A., "Behavior Trees for UAV Mission Management," *Informatik 2013 – 43rd annual German conference on informatics*, Gesellschaft für Informatik, Koblenz, Germany, 16-20 September 2013, accepted for publication.

[7]Baader, F., *The description logic handbook: theory, implementation, and applications*, Cambridge Univ Pr, 2003.

[8]Zolin, E., "Complexity of reasoning in Description Logics," 2011, Online at `http://www.cs.man.ac.uk/~ezolin/dl/`; accessed 28-October-2011.

[9]Artale, A., Franconi, E., et al., "Representing a robotic domain using temporal description logics," *AI EDAM*, Vol. 13, No. 2, 1999, pp. 105–117.

[10]Humphrey, L., "Model Checking UAV Mission Plans," *AIAA Modeling and Simulation Technologies Conference*, AIAA, Minneapolis, Minnesota, 13 - 16 August 2012, AIAA 2012-4723.

[11]Lim, C., Baumgarten, R., and Colton, S., "Evolving behaviour trees for the commercial game DEFCON," *Applications of Evolutionary Computation*, 2010, pp. 100–110.

[12]Isla, D., "Handling complexity in the Halo 2 AI," *Game Developers Conference*, 2005.

[13]Champandard, A. J., "Behavior Trees for Next-Gen Game AI," *GDC*, 2007.

[14]Baader, F. and Hanschke, P., *A scheme for integrating concrete domains into concept languages*, Citeseer, 1991.

[15]Artale, A. and Franconi, E., "A survey of temporal extensions of description logics," *Annals of Mathematics and Artificial Intelligence*, Vol. 30, No. 1, 2000, pp. 171–210.

[16]Lutz, C., Wolter, F., and Zakharyashev, M., "Temporal description logics: A survey," *Temporal Representation and Reasoning, 2008. TIME'08. 15th International Symposium on*, IEEE, 2008, pp. 3–14.

[17]Qi, G., Pan, J., and Ji, Q., "Extending description logics with uncertainty reasoning in possibilistic logic," *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, 2007, pp. 828–839.