A Sequence and Supervisory Control System for Onboard Mission Management of an Unmanned Helicopter

Florian-M. Adolf*
German Aerospace Center (DLR), D-38108 Braunschweig, Germany

OVERVIEW

This paper presents a solution to the onboard mission management problem for UAVs. Inspired by successful approaches from the mobile robotics domain, the proposed architecture achieves hybrid control by combining ideas from the behavior-based paradigm and a three-layered high-level control architectures. Two main components, a Sequence Control System and a Supervisory Control System, form the Mission Management System and implement the essential components of this architecture. It is open with respect to its interfaces to other onboard components (e.g. the obstacle avoidance system). Events and commands sent by a remote operator or an onboard component can be integrated into the system in a plug-and-fly fashion. An operator can control the UAV at different levels of autonomy, ranging from common joystick control to execution of prearranged missions. To assure the safe operation of the system, it is characterized by multiple techniques. First, the event-based decision logic is modeled as a State Chart model for each component. Second, a truth table of valid external commands assures that only permitted commands are accepted by the event handling. Finally, a grammar-based plausibility check avoids illegal behavior commands within a mission plan. The system is implemented onboard the UAVs of the ARTIS program. It is validated in unit tests, and tested in software-in-the-loop and hardware-in-the-loop simulations.

1 INTRODUCTION

The ARTIS[6] program at the DLR Institute of Flight Systems is working on innovative mission management concepts for UAVs. One research aspect involves the design and implementation of decision making components for onboard mission management. The ultimate goal is to employ an onboard system that allows different levels of UAV autonomy for the low-altitude domain (e.g. urban areas).

This paper presents the Mission Management System that forms the executive part of an architecture which aims on combining the advantages of the behavior-based paradigm and a layered architecture. First, we develop a set of elementary movements that allow the UAV to perform certain movements like hovering to a location or turning around a point. Second, a 3T architecture[3] defines the framework for the mission management in order to handle system complexity while being flexible and fast enough for updates and extensions. The highest level of UAV autonomy is implemented by a deliberative layer that comprises offline and online planning algorithms in order to solve high level tasks (e.g. a search mission). A skill layer stores the elementary behaviors that can be used by the deliberative layer's planning algorithms.

These system features require a safe coordination between multiple events. Therefore, the Mission Management System's decision logic is modeled using UML State Chart diagrams which can be verified using abstract tests (e.g. path coverage or event sequences).

The safe and robust operation plays a major role for this onboard system. Consequently, certain safety precautions are built into the Mission Management System.

The software, generated from the State Chart model, is validated against certain functional scenarios using Unit Test techniques. Moreover, the implemented code is validated in both software-in-the-loop and hardware-in-the-loop simulations.

In the following chapters, the underlying problems of onboard mission management, embedded high level architectures and their implementation issues are discussed. Then the design of the ARTIS Mission Management System is presented, followed by discussions on the implemented system and future work.

2 PROBLEM DESCRIPTION

For many UAVs, remote joystick control and planning missions is performed by an operator at a ground control station [10] [5]. The operator either directly commands the UAV to a location using a joystick (remote control) or a position command. With an onboard world model and path plannning capabilities, more "intelligence" is brought to the onboard system such that an operator might issue complex commands directing the vehicle to fly back to base or search for an object.

This implies different abstraction levels within the onboard system such that levels of autonomy can be achieved. The level of autonomy at which an operator commands the UAV might vary over time, depending on the situation and task. Moreover, the design and implementation of different levels of control necessitates provisions for operational safety

*PhD student, Institute of Flight Systems, Systems Automation Department, florian.adolf{at}dlr.de.

and certain user requirements. In particular, the system operator must remain "in the loop" at all levels of autonomy. Also, the operational environment is characterized by events that can occur in an unknown order and at sporadic time instances. It must implement input checks for syntactical plausibilities and even semantic correctness, wherever possible.

There are several functional requirements with respect to predictability of decisions and traceability of internal states at execution time. For example, the system has to react appropriately when the safety pilot switches between manual and computer-based flight control. The system has to be extensible for new flight maneuvers, planning algorithms for onboard mission planning, and interfaces for "smart" system components like an obstacle avoidance system.

In reference to the system's autonomy level, it has higher authority over the flight control system for which the established design process has to remain untouched. The interface for data exchange with the flight control system demands real-time algorithms.

Finally, testing the system is mission critical throughout its development stages and before its deployment. High level control systems can embed several different kinds of techniques, such as approximation algorithms or classical planning from the field of artificial intelligence. Hence, the testing strategy has to cope with both user and saftey requirements, as well as, characteristics of these different components.

3 RELATED WORK

Onboard Mission Management has gained increased attention in recent years [22] [8] [17] [21]. More self-reliance and decision making autonomy for UAVs poses questions regarding suitable decision making architectures, system modeling techniques, and system validation and verification methods. Therefore, the ARTIS program considers answers to these questions to be major challenges in research for high level control of UAVs.

To solve the complex problems that arise, different high level control architectures have been proposed, and some of them have been deployed on UAV systems. Knowledge-based systems establishing the concept of a cognitive process as decision making entity were presented in the UAV domain[11] [19]. Some other concepts are based on the behavior-based paradigm, [26] which means that a set of elementary behaviors (so-called skills, such as movement primitives) is combined in such a way that a new emergent behavior is created. Others prefer layered architectures [9] comprising distinct system modes, also known as hybrid control [7]. At this point, for the ARTIS program, it remains uncertain which architectural concept to choose. Since the control architecture is a mission critical design decision, these architectural concepts will be discussed with respect to the UAV domain and the requirements discussed in chapter 2.

Using knowledge-based systems, classical AI spent over five decades trying to model human-like intelligence. Inspired by these systems, several research projects seek to produce a human-like thinking process (also known as cognition) in order to achieve high level control in UAV decision making systems[11] [19]. A commonly used cognitive architecture is implemented in SOAR[13]. Since real-time properties are one crucial design aspect for a UAV decision making system, a real-time derivate of SOAR, Hero-SOAR, would be a suitable implementation. However, there are major implementation issues related to cognitive production systems[16]. First, "chunking", a pattern matching technique, might be hard to confine with respect to execution time and memory usage. Second, real-time reflexive actions (a direct connection of a sensor to an actuator) invoke a high-variance of unpredictable AI techniques. Furthermore, problems were experienced when trying to effectively coordinate and mediate reflexive behaviors with the overall deliberative behavior of the system. If the reflexive actions can bypass the normal deliberation mechanisms, it may be difficult or impossible for the deliberation processing to reason about and affect the real-time reaction. Hence, the architecture for any UAV decision making system should particularly focus on "embedding real-time in AI" rather than "embedding AI in Real-time"[16]. Moreover, a principle shortcoming of the cognitive approach is the emphasis on representation at a high, symbolic level. This yields to control strategies that may make conceptual sense to a human designer but the intelligence in such systems belongs to the designer. Additionally, it is questionable whether humans deploy a complex thinking process for every intended behavior rather than think in a more reactive way[2].

These disadvantages were addressed by the behavior-based control with the Subsumption Architecture[24] [4], which did not necessarily seek to produce cognition. It rather uses a hierarchy of fast reactive loops where each loop is capable of executing a distinct behavior and higher reactive loops modify the behavior of lower ones. The concept of arbitration allows to automatically select among behaviors, and the so-called action-oriented perception frames the perceptual input according to the task. Some approaches interconnect elementary behaviors and superposition them which results in a new, emergent system behavior. The ultimate goal in many behavior-based approaches is to enable robot learning techniques such that a system can "learn" which behaviors must be compiled together in order to achieve a goal. Since the world is an implicit model, the behavior-based approach shows advantages with respect to design simplicity, implementation elegance and inherently more robustness against failures. Behaviors use little or even no internal state variable to model the environment and thus are less computing-intensive than deliberate counterparts. Hence, a behavior-based system qualifies for the high level architecture as it enables "AI in Real-time"[16]. Admittedly, one of the side effects in many approaches is that they produce black-box systems. When behaviors are interconnected it is almost impossible to explain the sys-

tem behavior. Moreover it is hard to achieve a notion of optimality[18]. In a practical sense, debugging a behavior-based system is known to be hard, since the overall behavior emerges from the interaction of many layers of asynchronous control.

UAVs are currently supposed to be semi-autonomous, remotely guided, assistant systems rather than anthropoid, autonomous systems. One of the key requirements of having several levels of system autonomy cannot be achieved with solely deliberate nor reactive architectures. Deliberate architectures relate ”autonomy” to human-like intelligence and rational acting, whereas reactive architectures consider it as a system’s ”ability to act independently in the real world environment” [14]. Thus, for UAVs it is worthwhile to achieve a combination of the advantages of a knowledge-based and a behavior-based architecture. Current trends in the robotic development created architectures that combine both ideas into one system.

The goal of the TCA[23] is to combine deliberation and reactivity in such a way that the system detects changes in its environment and makes appropriate responses. For example, deliberative aspects of TCA can be used to plan moves based on certain constraints, whereas the reactive components detect and handle deviations arising from sensor and actuator uncertainty. TCA provides designers with control constructs for developing behaviors and software utilities for implementing the necessary control decisions. However, this architecture does not itself provide behaviors for a particular task.

Inspired by the Subsumption Architecture[24] [4] and empirical observations, the 3T architecture[3] separates intelligent control into three interacting layers (or tiers). The first layer comprises a set of so-called reactive skills, that are behaviors (control laws) tightly coupled with environment through sensor readings and actuators. Skills make ”simple-world” assumptions; such as, the sensor input is valid and desired goal can be achieved. In order to accomplish specific task, the sequencer on the second layer assembles an appropriate task network of skills by activating and deactivating respective skills. When more than one skill is active, they form a so-called task network. The third layer is the deliberative layer, which comprises a planner that reasons about goals, resources and timing constraints with well known AI techniques. The centralized sequencer of this architecture does not allow concurrency between all behaviors and modules in the system. Also, this architecture does not differ between skills that perform control commands and abstract skills (e.g. a skill activating lower skills).

4 MISSION MANAGEMENT SYSTEM

The advantages and the disadvantages of the architectures discussed before, demand a thorough consolidation into one embedded architecture for onboard mission management. The major requirements with respect to real-time execution, predictable system behavior and the need for different levels of (operational) autonomy result in the following design decisions:

- The embedded system architecture must be separated into interacting layers, to enable the implementation of deliberate and reactive approaches. This leaves room for a behavior-based reactive layer and allows several kinds of AI techniques in the deliberative layer(s).
- The layered architecture chosen for this hybrid control problem is the 3T architecture. Compared to the TCA, it offers a more flexible way of modularization, centralizes the execution of actions and does not rely on interacting skills.
- The behavior-based paradigm, as a bottom-up strategy for intelligent systems, is worth being considered for the reactive layer, since it enables real-time execution and relatively simple behavior development. This paradigm allows a way to compile elementary problem solutions (e.g. moving to a position) into a library of behaviors.
- Known shortcomings of the behavior based approach with respect to online learning, behavior interaction and arbitration techniques, are eliminated intentionally.
- When behavior interaction and abstract behaviors are not available in the reactive layer, the discussed disadvantages of the 3T approach can be neglected.

As a result, the design concept for the ARTIS onboard mission management system combines a 3T architecture with ideas from the behavior-based paradigm. In the next sections the first component of this architecture, the Sequence Control System, is described. It implements the sequencing layer’s executive component and a set of basic behaviors in the reactive layer.

The SMission Management System’s embedded system is described from three points of view. In order to highlight a system wide context, Figure 1 describes the component organization from an implementation point of view. The illustrations in Figure 2 outline how 3T’s principle high level control decomposition boils down to the actual application for ARTIS. Finally, Figure 3 shows the decision logic which is implemented as an UML State Chart model.

4.1 Embedded Architecture

Two basic prerequisites of the proposed architecture in Figure 2 are implemented in the Sequence Control System. It comprises a library of basic movement behaviors located at the reactive layer, followed by an executive component of the sequencing layer. The Supervisory Control System implements higher level behaviors and influences the Sequence Control System’s mission execution when an operator on ground issues a high level command or when the human control is not available. Figure 1 shows that the Sequence Control System forms the interface to the flight

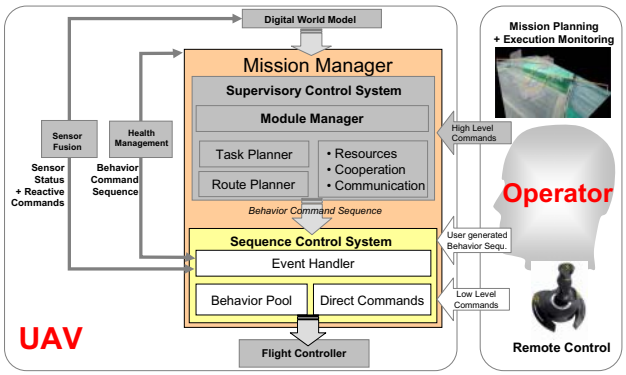

FIG. 1: The ARTIS Mission Management System shown in the context of the overall high level system architecture.

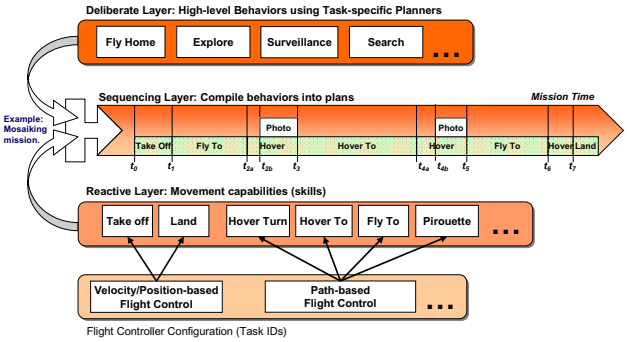

FIG. 2: ARTIS' Sequence Control System implements the executive component of the sequencing layer of this 3T architecture. The reactive layer contains basic movement behaviors that interface with the flight control system. The Sequencing Layer shows a mosaicking mission compiled from the behavior set.

control system. It is the unifying building block of the onboard Mission Manager and takes inputs from various sources like the ground control station, vision computer or supervision component.

Furthermore, neither the deliberative layer nor the skills alone can handle all situations optimally. The Sequence Control System provides the necessary glue logic and is the place to store procedural knowledge that neither fits at the deliberative layer nor at the skill layer. For ARTIS, this procedural knowledge comprises some of the safety requirements for flight tests mentioned in chapter 2. For example, when the safety pilot switches between manual or computer aided control, the system must stop producing actuator commands and reset its onboard components into a defined stand-by state.

Figure 2 demonstrates which behaviors are located at the skill layer. Depending on which flight controller configuration is used, there are two basic behavior groups. The first group outputs direct position and velocity commands to the flight controller. The take-off and the landing behaviors use this controller configuration, where they command a fixed position for the horizontal layer (x and y axis) and a velocity command on the z axis. These behaviors contain only two state variables, namely the x and y position

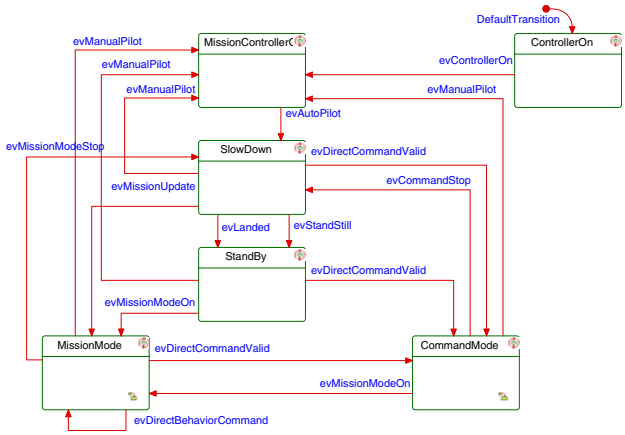

FIG. 3: The UML State Chart Model of the Sequence Control System comprises two composite states: State "Mission Mode" for mission plan execution and "Command Mode" for direct operator commands or commands from onboard components.

they have to maintain. As such, these behaviors are "reflexive" behaviors in the original sense of the behavior-based paradigm[24].

The second group is the largest set of behaviors. These produce trajectory-based control commands which comprise two tangential trajectory angles and the deviation from this tangent. At this time, the behavior library contains behaviors to wait at a position ("Wait For"), turn on the spot ("Hover Turn"), to fly along a linear trajectory toward a location ("Hover To"), to fly around a point along a horizontal circular trajectory ("Pirouette"), and to perform fast forward flights along an arbitrary trajectory ("Fly To").

The deliberate layer shows examples of complex behaviors. These will alter existing missions or even create new missions. In this context, mission planning will output the list of sequential behavior commands shown in Figure 2. However, the compilation does not necessarily take place onboard. In fact, at present, missions are compiled by the ground control station software and sent to the Sequence Control System. However, there's still a necessity for a Supervisory Control System when the ground operator issues a complex behavior command, e.g. to let the UAV fly to a (predefined) home location.

4.2 Event-based Model

As discussed in Chapter 2, the Mission Management System is exposed to a number of potentially concurrent events. Hence, the specification and implementation of the system is modeled as an event-based system. [15] The majority of event-based systems are modeled using an industry-wide standard notation, UML[1]. It supports the object oriented design pattern and provides dynamic modeling techniques such as state charts, sequence diagrams and activity diagram. State charts have been extensively studied such that abstract testing techniques allow to verify a model

[1] Version 1.2, as defined by OMG (http://www.uml.org/)

(semi-)automatically. Also, there exists good software tool support which eases the development process significantly. Moreover, there are tools that provide code generators such that the implemented code is directly derived from the state chart-based specification. Otherwise, it is likely that specification and implementation begin to diverge over time.

Thus, the Mission Management System is modeled as UML (currently v1.2) State Charts. Basically, State Chart diagrams (also known as State Machines in UML 2) are finite automatons with a finite set of states where exactly one state is active at a time. They depict the dynamic behavior based on its response to events, showing how the model reacts to various events depending on its current state. Events can trigger a transition into another state, where so-called guards are the condition that must become true in order to traverse along the transition. The guards on similar transitions leaving a state must be consistent (deterministic) with one another.

The UML model of the Mission Management System's Sequence Control System is shown in Figure 3. It has two hierarchical levels where the top level models the procedural flow for a safe operation. The two composite states, "Mission Mode" and "Command Mode", model mission plan processing and direct command execution respectively.

Every state of the top level has a transition to the "Mission Controller Off" to handle a manual control event, such that the Sequence Control System stands by in an idle state. If the system is in computer-based flight mode (auto mode), another idle state "Stand By" lets the UAV hover at its current position when the state was entered; including a position on the ground. The state "Slow Down" is necessary to assure a smooth changeover into "Stand By" regardless of the flight maneuver currently being executed. In case the GCS operator commands the UAV to stop, a transition from every auto mode state assures that the command is executed. Among events certain priorities exist. For example, an event switching to manual mode is more important than a stop command and requires processing. The order in which the event check is performed accounts for this obligation.

The state mission mode contains the actual library of behaviors. There are no transitions among behaviors which assures that not more than one behavior can be active at a time. This is a major requirement in order to overcome emergent behaviors caused by interactions. For each behavior there exists a termination condition which transits into the command parser "Parse Command". Basically this state grabs behavior commands from an existing mission plan. It issues an event for traversing into the appropriate state. If the mission plan is processed, "Mission Mode" is finished.

The "Command Mode" can be entered from any auto mode state. It is relatively complex decision to determine whether a direct command is permitted or not. First, compared to behaviors in "Mission Mode", every direct command may not have a termination condition. For example, one instantaneous velocity command from the GCS cannot reach a target condition. For such cases, a quasi infinite behavior is bounded in execution time, such as the time passed since a GCS velocity command was received.

Second, it depends on which data is coming on what input channels. This specific problem is addressed with the static truth table.This table defines the direct input data channels for the GCS, vision computer and FLARM[2]. The columns represent the data values, either boolean or numeric, for position, joystick, pattern position, stereo camera-based avoidance position, or a FLARM-based avoidance velocities. It names all valid, conflict free combinations explicitly. At runtime this table allows checks of every incoming command.

Likewise the command checks for these low-level direct commands, missions defined as sequence of behavior commands have to be understood as one single command. Although, for the system it is not possible to reason about the sense of a mission plan, it is possible to implement a plausibility check. These checks are implemented using the EBNF grammar[12] which ignores malformed behavior sequences that cannot be matched against it. EBNF grammars are formal grammars of logical production rules that comprise nonterminal symbols, similar to place holders or variables, and terminal symbols, which are generated by the grammar. Grammars are widely used to implement compilers for programming languages, and, as such, a behavior sequence is a sequential program. The grammar implemented in this context defines production rules for each behavior command or task flag, such as a "hover to" or "tracker off". Its root is defined by "<mission>" which basically enforces every mission to start with a take-off behavior and end with a land. It implements some basic parameter checks, for instance for the "hover and wait" command where the waiting time must not be a negative number. Furthermore, the grammar allows a check for any necessary preconditions for specific behaviors. For example, the trajectory planner for the forward flight needs at least three of its behavior commands.

4.3 System Testing

The overall complexity of the Mission Management System's software implementation imposes a thorough test strategy. The testing process has to account for user requirements ("scenarios"), as they were mentioned in Chapter 2, and must address theoretical issues that can occur in finite state machine models and State Charts, respectively. Basically, there are four testing stages during the development of the onboard mission management system:

1. Abstract tests assure that the model is free of principle errors.
2. Unit Tests assess the functionality of individual software component or a composite component.

[2] An integrated device for collision avoidance, http://www.flarm.ch/

3. Simulation provides the infrastructure for operational tests that assess the overall system integration.

4. Flight Tests provide a practical proof of concept for the real world application.

In the modeling stage a set of errors can occur. Some relate to potentially isolated or unreachable states, as well as, missing or erroneous triggers and guards. An even more fundamental problem is the theoretically infinite set of sequences that have to be tested in an abstract manner. That is, regardless of the meaning and function behind events, states and transitions, the tests must traverse through the model even if a certain test would make no sense from a practical point of view.

Abstract tests relate to Model-based testing[25]. In the case of the Mission Management System, the test model can be derived from the State Chart model. UML state charts represent a constructive model[20], such that tests can make use of its internal structure (known as white-box testing). These abstract tests can be divided into three groups, namely the path coverage, transition coverage and state coverage tests. For each of these groups, a set of event chains is generated and executed. Once a full coverage of all possible combinations is reached, the tests are completed. However, there is no defined final state and states can have loops, as such infinitely long test chains for path coverage tests result. Therefore, a relaxation for the path coverage criteria is implemented such that loops must be passed only once. This is a strong and feasible criteria[25].

Inspired by some of the best practices in XP[27], the user requirements are translated into scenarios that are implemented as Unit Tests. The tests focus solely on the observed behavior (known as black-box testing). XP's test-first practice allows an "inside-out" strategy where each component, for example a trajectory interface, is tested before a higher component is tested that makes use of this interface. Technically, the tests implement certain heuristics. For example, if the GCS does not send new velocity commands, a time out must trigger an appropriate event. Besides these procedural checks, some behaviors can be tested using Unit Tests, such as the landing behavior, which must stop descending when a certain level of lateral or longitudinal velocity is exceeded. Other scenarios intentionally produce illegal commands which the system must ignore. For instance, when the UAV stands on the ground any direct command must be ignored, except the take-off command. Furthermore, the system always has to check against a shortfall of minimum ground distance, which can be tested easily using simulated data. Essentially, these unit tests also serve as so-called regression tests[1]. It is possible to find "local" bugs, that is, test whether changes introduced new bugs. Then they can "unmask" previously existing bugs that were not detected before. Finally, they potentially uncover unintended side effects with "remote" tests, where the change of one part breaks another part of the software.

Before every flight test, the fully integrated Mission Management System software is tested in simulation. For ARTIS, SITL and HITL simulations are available.The SITL simulation implements a number of simplifications for the sake of reducing the computational load on desktop PCs. Whereas the HITL simulation uses the original onboard computer hardware and realistic sensor models. Moreover, the SITL simulation runs in a Simulink model which does not enforce real-time code execution. Further, most state chart events are based on thresholds, time outs and distinct sensor values. With respect to the Sequence Control System, the major difference between SITL and HITL simulations is, that sensor values, timing conditions and event sequences differ. Logic tests in abstract and unit testing cannot check the correctness of the effective interface to real valued state estimates and sensor data. These issues are addressed during the operational tests in these simulations. Predefined functional tests are executed manually, then missions that provide a full state coverage are executed.

5 FUTURE WORK

In theory many high level control problems of the UAV domain, like path planning or performing a single maneuver, are considered to be solved. However, integrated architectural concepts can rarely be observed in flight tests. That is why the focus is on the difficulty of bridging theory and practice, which was the goal in this paper and will be an important future research aspect within the ARTIS program. In order to achieve this goal, a number of research directions have been identified. In the short term, the onboard mission management concept will be extended with complex behavior modules enabling a significantly higher level of abstraction for UAV control. Moreover, work is underway to explore alternative modeling techniques, such as Petri Nets. Additionally, behaviors for the deliberate layer are developed that allow an operator to perform tasks like object search and tracking, exploration or flying back to a base.

6 CONCLUSION

This paper presents a solution to the onboard mission management problem of UAVs. It is implemented in the unmanned helicopters used by the ARTIS program and successfully validated, verified and flight tested. This behavior-based Mission Management System, embedded into a 3T architecture is a feasible approach and has advantages over the discussed approaches. The UAV is controlled by a robust, deterministic system that can handle various events and commands coming from multiple sources. These sources can either be system components or a remote operator. The presented (semi-)automatic test process makes testing manageable and allows test execution in a repeatable way.

Both, the Sequence Control System and the Supervisory Control System, execute the output of the 3T architec-

ture's behaviors compositions which allows different levels of UAV autonomy. The behavior-based aspects of the proposed architecture enable real-time properties through a direct linkage between sensors and actuators, even for complex maneuvers. New behaviors can be added to the behavior library such that functional extensibility is facilitated ("plug-and-fly"). Since some concepts of solely behavior-based approaches, like arbitration or robot learning, are not eligible for UAV control, they have not been implemented in the architecture. The State Chart model assures an event-based behavior with deterministic, predictable actions. The decision logic can be verified using abstract tests like state coverage, path coverage and transition coverage tests. The level of abstraction in the State Chart model allows a convenient way of mission execution and execution monitoring. The Mission Management System implements several operational safety features. First, an operator can overrule autonomous actions anytime. Second, the system provides a stand by state which also serves as a fall-back state in case of errors. Third, the user input is checked against plausibility rules using a truth table and an EBNF grammar. Regardless of which behavior is executed or what operator command the system is executing, the system allows no shortfall of a minimum ground distance, thus avoiding accidental ground strikes. Furthermore, it rejects a malformed mission plan, that is a plan which could harm the helicopter during mission execution. By design the Mission Management System keeps the operator in the loop. He has the possibility to intercept any running behavior by different kinds of commands, ranging from direct velocity commands (e.g. for remote joystick control), direct position commands, a single behavior (e.g. land) or a complex behavior command sequence. Furthermore, at this time, high-level tasks like mission planning remain operator tasks. It is possible to implement an "onboard operator" in case an operator on ground is not available.

However, the control architecture may reach its limits when the centralized sequencer needs to implement concurrency (e.g. multiple deliberate behaviors need to be active). It does not allow concurrency between all behaviors and modules in the system. Moreover, known shortcomings of State Chart models are inherent to the current system. First, State Charts need constraints to restrict semantics on the input data since these cannot be modeled. Second, UML use cases are user examples and as such inherently incomplete. A formal specification of the functional requirements is be desirable. State Chart models provide only a limited support for temporal design aspects.

REFERENCES

[1] Hiralal Agrawal, Joseph Robert Horgan, Edward W. Krauser, and Saul London. Incremental regression testing. In *ICSM*, pages 348–357, 1993.

[2] Philip E. Agre and David Chapman. Pengi: an implementation of a theory of activity. In *Computation & intelligence: collected readings*, pages 635–644, Menlo Park, CA, USA, 1995. American Association for Artificial Intelligence.

[3] R. Peter Bonasso, James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Marc G. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2/3):237–256, April 1997.

[4] R. S. Brooks. A robust layered control system for a mobile robot. pages 204–213, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[5] Joerg S. Dittrich, Florian Adolf, Augusto Langer, and Frank Thielecke. Mission planning for small uav systems in uncertain environments. In *2nd European Micro Aerial Vehicle Conference*, Braunschweig, Germany, July 2006.

[6] Joerg S. Dittrich, Andreas Bernatz, and Frank Thielecke. Intelligent systems research using a small autonomous rotorcraft testbed. In *2nd AIAA "Unmanned Unlimited"*, number AIAA-2003-6561, San Diego, CA, September 2003.

[7] M. Egerstedt, X. Hu, and A. Stotsky. A hybrid control approach to action coordination for mobile robots. In *Proceedings of IFAC 99 14th World Congress*, Beijing, China, Jul 1999.

[8] Patrick Fabiani. The ressac autonomous air vehicle project: Challenges and demonstration platforms. In *5th ONERA-DLR Aerospace Symposium*, June 2003.

[9] Michael Freed, Pete Bonasso, K. Michael Dalal, Will Fitzgerald, Chad Frost, and Robert Harris. An architecture for intelligent management of aerial observation missions. In *Infotech@Aerospace*, Arlington, Virginia, 26-29 September 2005.

[10] Lucas Heitzmann-Gabrielli. Mission planning and dynamic avoidance for groups of autonomous agents. Trabalho de Graduacao, Instituto Tecnologico de Aeronautica, Sao Jose dos Campos, Brasil, Nov 2005.

[11] R. Hill, J. Chen, J. Gratch, P. Rosenbloom, and M. Tambe. Intelligent agents for the synthetic battlefield: A company of rotary wing aircraft. In *Innovative Applications of Artificial Intelligence (IAAI-97)*, 1997.

[12] ISO-14977. Information technology — syntactic metalanguage — extended bnf. 2001. International Organization for Standardization, ISO/IEC 14977.

[13] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artif. Intell.*, 33(1):1–64, 1987.

[14] P. Makowski. Survey on architecures and frameworks for autonomous robots, November 2004.

[15] Gero Muehl, Ludger Fiege, and Peter Pietzuch. *Distributed Event-Based Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[16] David John Musliner, James Hendler, Ashok K. Agrawala, Edmund H. Durfee, Jay K. Strosnider, and C. J. Paul. The challenges of real-time AI. Technical Report CS-TR-3290, 1994.

[17] A. Piquereau, V. Fuertes, and P. Fabiani. Autonomous flight and navigation of vtol uavs. In *5th ONERA-DLR Aerospace Symposium*, June 2003.

[18] P. Pirjanian. The notion of optimality in behavior-based robotics. In *Journal of Robotics and Autonomous Systems*, 1999.

[19] H. Putzer and R. Onken. COSA – a generic cognitive system architecture based on a cognitive model of human behavior. *Cognition, Technology and Work*, 5, 2003.

[20] Bernhard Rumpe. *Agile Modellierung mit UML : Codegenerierung, Testfaelle, Refactoring (Xpert.press)*. Springer, November 2004.

[21] Daniel P. Schrage and Eric N. Johnson. The georgia tech vtol uav testbed - the gtmax, as an open system for uav research and development. In *59th Annual Forum of the American Helicopter Society*, Phoenix, AZ, May 2003.

[22] David Hyunchul Shim. *Hierarchical Control System Synthesis for Rotorcraft-based Unmanned Aerial Vehicles'*. PhD thesis, University of California, Berkeley, 2000.

[23] Reid Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1), February 1994.

[24] D. Toal, C. Flanagan, C. Jones, and B. Strunz. Subsumption architecture for the control of robots. In *IMC-13, Limerick*, 1996.

[25] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[26] Lora G. Weiss. Intelligent collaborative control for uavs. In *Infotech@Aerospace*, Arlington, Virginia, 26-29 September 2005.

[27] Graham Wright. Achieving iso 9001 certification for an xp company. In *XP/Agile Universe*, pages 43–50, 2003.