VERMILLION: VERifiable MultIagent Framework for DependabLe and AdaptabLe AvIONics

Rajanikanth N. Kashi [*]
Dayananda Sagar University, Bangalore, Karnataka, 560068

Meenakshi D'Souza[†]
International Institute Of Information Technology - Bangalore, Bangalore, Karnataka, 560100

The aerospace industry is continuously looking for improvements in operational efficiency and performance of systems in major programs like NextGen and SESAR. In its quest to do so, the industry is turning to Intelligent Adaptive Systems as a possible solution in many areas. These areas cover both the ground and air-side operations. However, the nature of the domain imposes expectations of safety, correctness, and guarantees of behavior from such systems. Meeting these expectations simultaneously, finally leading to certified products poses many challenging problems, especially on the air-side operations. While the use of Intelligent Adaptive systems holds much promise, decision makers and users of these technologies need to see more exemplar systems meeting all these expectations. A research gap is perceived when the cycle of requirements, system design, verification and validation is examined, paving the need for correctness and guarantees of specifications in early stages of a complex adaptive avionics systems. To this end, we present a framework VERMILLION that is targeted for a broad class of avionics systems, engineered for both short and long term system behaviors, resilient, real-time decision making, establishing trust on the way to certification, and above all is amenable to analysis using formal methods. At the heart of the framework is the BDI model with suitable extensions and equipage for incorporating learning, safety, determinism, and real time response. The abstracted formal model is specified using the Z language. In support of the principles, concepts, and methodology of this framework, we present two case studies, one in the area of flight management system and the other in the area of UAV Mission Management, where this framework has been successfully employed.

Keywords: Self Adaptive Software, Avionics, Multiagent Systems, BDI Model, Verification and Validation, Z models, CTL specifications, Model Checking, Learning, Scheduling, Self-* Properties

[*]Associate Professor, Dept. of Computer Science and Engineering, Dayananda Sagar University - Bangalore, Campus 3, Kudlu Gate, Hosur Main Rd, Bengaluru, Karnataka 560068, India.

[†]Associate Professor, Software Engineering, International Institute Of Information Technology - Bangalore, 26/C, Hosur Rd, Electronics City Phase 1, Bengaluru, Karnataka 560100, India.

I. Introduction

Adaptivity is a mechanism of obtaining foreseeable behavior from a system under consideration when there are non-deterministic activities or events occurring in the environment of the system and affecting the system under consideration. The natural evolution of many systems leads to complexity due to many factors including size, behavioral changes of constituent elements (sometimes non-linear), imprecise causality, and systems dynamics. Therefore, there is considerable research in the area of adaptive systems. There are several practical examples of these systems in diverse areas like engineering (aircraft systems), economics (markets), management (organizations), cities (urban planning), environment (ecosystem and biosphere, environmental law), and education (intelligent tutoring, adaptive hypermedia), to name a few. Across the globe, the steady increase in air traffic has fuelled the need for complex avionics systems. Intelligent and autonomous systems, operations and decision making, human integrated systems, and networking and communications has been identified as one of the five research and technology areas that requires high priority work to be done for conquering many of challenges in this area [1]. The trend in modern cockpits has been to introduce more automation to assist the pilots and this burden is taken up in a large amount by the software. Also, the seemingly imminent induction of Unmanned Air Vehicles into the civilian airspace throws up a unique set of challenges with automation and control aspects. We quote some excerpts on the subject:

- "Developing certifiable V&V methods for highly adaptive autonomous systems is one of the major challenges facing the entire field of control science, and one that may require the larger part of a decade or more to develop a fundamental understanding of the underlying theoretical principles and various ways that these could be applied" [2]
- "Advanced system attributes (on-board intelligence and adaptive control laws) will be required to accommodate emerging functional requirements. This will increase the size and complexity of control systems beyond the capability of current V&V practices." [3]

This paper deals with the area of Adaptive systems for avionics. Avionics systems manifest both physical systems engineering (analytical models) and computational systems engineering (computational models). In this paper we focus our attention on the computational aspects of such systems while keeping the analytical aspects in background. Considerations of performance, safety, and efficiency of avionics systems are prime drivers for researching adaptive systems in the context of these systems and the environments in which they operate. Since the computational elements assume significant importance, the term "Self Adaptive Software" [4] is more apt in the context of our work and will be at the foundational core of such adaptive avionic systems. Several sub-areas of interest in Self Adaptive Software have emerged: Basic approaches for construction, concepts, detailed methodologies, requirements, architecture, autonomy, measures and assessments, verification and validation [4]. Our work with case studies have tackled some of the challenges related to:

- requirements [5] - what to adapt, what are high-level goals, and how to deal with uncertainty for requirements in the context of the environment.
- architecture [6], [5] - mechanizing control loops through software engineering, modeling for engineering effective adaptation, reference architectures.
- construction [6], [5] - engineering runtime assurance (safe behavior), timeliness, and controlled autonomy.
- evaluative measures [6], [5] - obtained by simulation testing, and
- Verification and validation [7], [5] - formal methods, model checking.

While there have been many developments to advance the state of the art in developing adaptive systems for avionics that includes flight tests on experimental aircraft [8], [9], [10], [11], and [12], we are yet to come across working adaptive avionics systems in regular widespread use. Organizations like NASA and FAA recognize that more work is needed on languages and tools for specifying and analyzing architectures and designs of complex distributed hard real-time systems [13, 14]. The primary goal of this research is therefore to explore and possibly come up with representative methods that will further our understanding in this important area. A secondary goal is to facilitate and enable human computer integration in the area of adaptive avionics systems [15], since effective working of the human in the adaptive loop is of primary importance. In this direction we choose suitable case studies that demonstrate our methods.

At the fundamental level, every adaptive system has an adaptation loop comprising of the detection, decision making, acting and monitoring processes. In order for this adaptation loop achieve our concerns of improved efficiency, enhanced performance, and effective method of handling uncertainty, we need 'intelligence' to be fabricated within this infrastructure to obtain an 'intelligent adaptive avionics' system. A powerful model that has a basis from all of these fields is the BDI (Belief Desire Intention) model of agency [16] that blends human-like psychological and philosophical behavior with the ability to be formulate mathematical computational aspects. Although many implementations of the BDI model can be found in different non-avionics applications [17], [18], [19], and [20], a rigorous method of application of the same for adaptive avionics systems with due considerations for using it in a safety-critical systems domain (avionics) is the main subject focus of this paper. At the top level, a multi-agent approach is chosen since agents lend autonomy, embody the analysis and reasoning mechanism, efficient task accomplishment via learning, and useful in dynamic and unknown environments. Therefore, a multi-agent system designed on the BDI model of agency is conceived and various aspects are described using the formal Z language [21]. Considerations from the use of an approach for engineering self adaptivity would be the following.

DM Model of Agency: Several techniques exist to engineer self-adaptivity [22]. We have chosen an agent based approach with BDI (Belief-Desire-Intention) model of agency [16, 23].

- In the general BDI model, *Beliefs* are facts about the agents environment and may also represent some internal assumptions. *Beliefs* are generated by a belief generation function that operates on *Percepts* received from the environment and internal status.
- A set of *Options* or *Desires* is generated by a desire generation function that either works from the *Beliefs* or directly from *events* received by the agent.
- These *Desires* are then deliberated based on some apriori mechanism or a rule-set that chooses one of the desires which is designated an *Intention*. The *Intentions* are generally pushed onto an intention stack which is used during further processing.
- Simultaneously a set of events is also collected in a event queue. A *Plan* corresponding to the intention and the event at the top of the event queue is chosen for performing actions by the agent.

A classic BDI style interpreter provided in [24] is reproduced below:

BDI Interpreter

```
initialise-state();
repeat
options := option-generator(event-queue);
selected-options := deliberate(options);
update-intentions(selected-options);
execute();
get-new-external-events();
drop-successful-attitudes();
drop-impossible-attitudes();
end repeat
```

After the agent executes the plan, events on the event queue are looked up once again and decisions about intentions and goals are taken and the entire cycle repeats. Note that the terms *Goals* and *Desires* are used interchangeably.

Identification of metrics and measures for evaluating intelligent systems is a need of the hour. With this in mind, measures based on the self-* properties [25] have been proposed. At the architectural level, simulation is a proven approach to validate the proposed models. Certification of adaptive avionics software is a huge challenge. Towards this direction, we articulate how our framework is well positioned utilizing formal methods with due considerations flowing from important standards like ARP4754A, ARP4761, and DO-178C.

The contributions of this paper are as follows:

1) We provide a generalized description of VERMILLION an acronym for Verifiable MultIagent Framework for DependabLe and AdaptabLe AvIONics, targeted for adaptive systems in avionics after eliciting the drivers from the problem domain, primarily based on the multi-agent system concept.
2) Theoretical underpinnings for constructing and assessing an adpative avionics system using BDI multiagent systems as the base model.
3) Abstract specifications are provided in a formal language that describe elements of the model. This model may be used as a reference specification from which minor variations can be derived for most adaptive avionics applications. A niche aspect is the exposition of the integration of the adaptation loops within the aircraft control system loops which are so very characteristic of avionic systems. With a view to provide plug and play characteristics for embedding intelligence, the classical BDI model is extended to provide hooks to incorporate adaptivity via a choice of learning methods and algorithms.
4) We also provide a quick overview and concepts that enable verification and validation of the proposed framework instantiations in individual applications. Since certification is a prime consideration, techniques that are set in place early in the framework development cycle are delineated. The section also includes measures for assessing goodness with respect to adaptivity of the system.
5) In order to provide concrete examples that demonstrate application of the framework two case studies on air-side operations are provided. One of them has a potential for extension to ground-side application too.

The rest of the paper is organized as follows: Section II provides an outline of the problem domain and drivers leading to the formulation of the VERMILLION framework. Section III deals with verification and validation in the context of the VERMILLION framework. Section IV provides an overview of the VERMILLION framework while section V captures the VERMILLION framework specifications using the Z notation. Section VI explains the dynamic parts of the model. Section VII describes two case studies where the framework has been applied. Section VIII documents information on related work in this area with attendant discussions. Section IX captures our summarized conclusions.

II. Problem domain outline and VERMILLION framework drivers

The typical broad mission profile of a civil aircraft consists of the following phases: taxiing, takeoff, climb, cruise, descent, terminal area approach, and landing. Military aircraft has similar profiles except that the cruise phase could differ in terms of the mission being operated (Eg. engage in bombing, reconnaissance or combat). An UAS (Unmanned Aircraft System) may have additional phases like pre-flight phase and post-flight phase. Across these common flight phases, one can then generalize and perform a high-level operational functions decomposition analysis that yields the following top level functions: ’Aviate’, ’Navigate’, ’Communicate’, and ’Mitigate’. The Aerospace Recommended Practice (ARP) 4754A [26] provides guidelines for system development process (development assurance) encompassing areas of aircraft level requirements, allocation of requirements, development of system architecture, allocation of requirements to hardware-software, and implementation. Using a hierarchical decomposition process, the aircraft level functions are allocated to systems which in turn drives the generation of the systems architecture. The systems architecture then drives the definition of items which are pieces of hardware and software which house the important system functions. ARP 4754A introduces the concept of IDAL (Item Design Assurance Levels) and relate to these system and hardware equipment. These items eventually find their manifestations as Flight Control System (FCS), Flight Management System (FMS), Collision Avoidance System (CAS), Navigation System (NS), Communication Link Management System (CLMS), Mission Manager System (MMS), Payload Management System (PMS) and an Emergency Management System (EMS). Within each of these items a further functional decompostion yields a second level set of sub-functions. As an example, for the FMS, these would be path planning in the presence of weather, traffic, and terrain constraints, automatic waypoint navigation, cruise level management, takeoff, and landing phase management functions. Closely entwined is the system safety assessment process which evaluates the functions and sub-functions in the backdrop of systemic failures that could occur across the systems and assesses the level of rigor and special safety tests related to software and safety-critical functions. A more detailed treatment of this is provided in the next Section II.A.

A. Avionics Self Adaptive Software

From our framework point of view, what is important is the recognition of the Design Assurance Levels (DALs - A: Catastrophic, B: Hazardous, C: Major, D: Minor, E: No effect) that categorize the effect of software failure for many of these functions. With this identification, necessary requirements on monitoring, independence, and fault handling is imposed on the system under consideration. The software development assurance guidelines associated with these functions then drive the following aspects for avionics self-adaptive software:

- Safety requirements need to be unconditionally met.
- A deterministic behavior is expected in the context of functionality, resources and time.
- Systems need to meet timing constraints imposed on them.
- Systems should be amenable to certification.
- Adaptation considerations need to be examined from the perspective of self-* properties (Major Level hierarchical view consisting of self-configuration, self-healing, self-optimization, self-protection) [25].

B. Avionics Self Adaptive Software Drivers

Drivers are considerations that are in-between requirements and design. [27] identify two broad categories: (a) the first category are a subset of requirements that shape the architecture and may include functional requirements, quality attributes, and constraints (b) the second category that depend on the system built (for a particular domain), design objectives, and concerns. We start out with the first step in the development of an adaptive system: Requirements

Capture. Requirements may be expressed in the form:

$$Requirement = \{RqFPart\} + \{RqNFPart\}_p + [RqFAdPart] + [RqNFAdPart]_q \tag{1}$$

{*RqF Part*} denotes a functional part; {*RqNF Part*} is the traditional non-functional part and buckets the timeliness (performance), deterministic aspect, and safety aspect which are of prime importance for avionics software. [*RqFAd Part*] addresses the adaptive part of the functional part {*RqF Part*} while [*RqNFAd Part*] denotes the non-functional part of the adaptation [*RqFAd Part*]. The use of { } braces indicates the mandatory part while [ ] braces indicate optionality. The subscripts p and q denote that there can be multiple non-functional parts associated with the respective functional parts

D1 We are primarily interested in providing a method wherein we want guarantees for system specifications in early design stages and therefore we resort to expressing them formally using logic like Computation Tree Logic (CTL) [28]. Some of these CTL formulae will contain both functional and adaptive parts. The methods of abstraction and model checking (described in section III) is used to ensure that these specifications are correct for the system design model.

D2 A decision is made about the top level function or a sub-function of this top level function that is allocated to a single aircraft system or a set of sub-systems (identified as FCS, FMS, CAS, NS, CLMS, MMS. . .) and will be the subject of adaptation.

Consider a simplified layered model for avionics shown in Fig. 1. If F denotes the set of all top level functions {F1, F2, F3,. . . Fn. . . } defined for an avionics system, then a Fn belonging to this set is chosen for adaptation. If Fn is mapped onto multiple systems across the layers, then Fn is a composition of all of these functions existing at different layers {FL_1, FL_2, FL_3. . . } (shown in the right side of the figure) or if Fn is mapped onto multiple systems in a single layer, then Fn is a composition of all of these functions existing in a single layer across different subsystems FL_{nS1}, FL_{nS2}, FL_{nS3}. . . (shown in the middle part of the figure).

Fn is then composed of FnF part and FnAd part, the functional and adaptive parts. The FnF part and FnAd parts are then identified with the {RqF Part} and [RqFAd Part] expressed within the formal requirements. The {RqF Part} (functional), {RqNF Part} (non-functional), [RqFAd Part] (functional adaptive), [RqNFAd Part] (non-functional adaptive) are allocated across a multi-agent system comprising a set of Agents AG = {AG1, AG2, AG3, . . . } using principles of logical cohesion, separation of concerns, and functional coherence. An agent is a computational system that interacts with one or more counterparts or real-world systems with key features of autonomy, reactiveness, pro-activeness, social abilities citeHexmoor

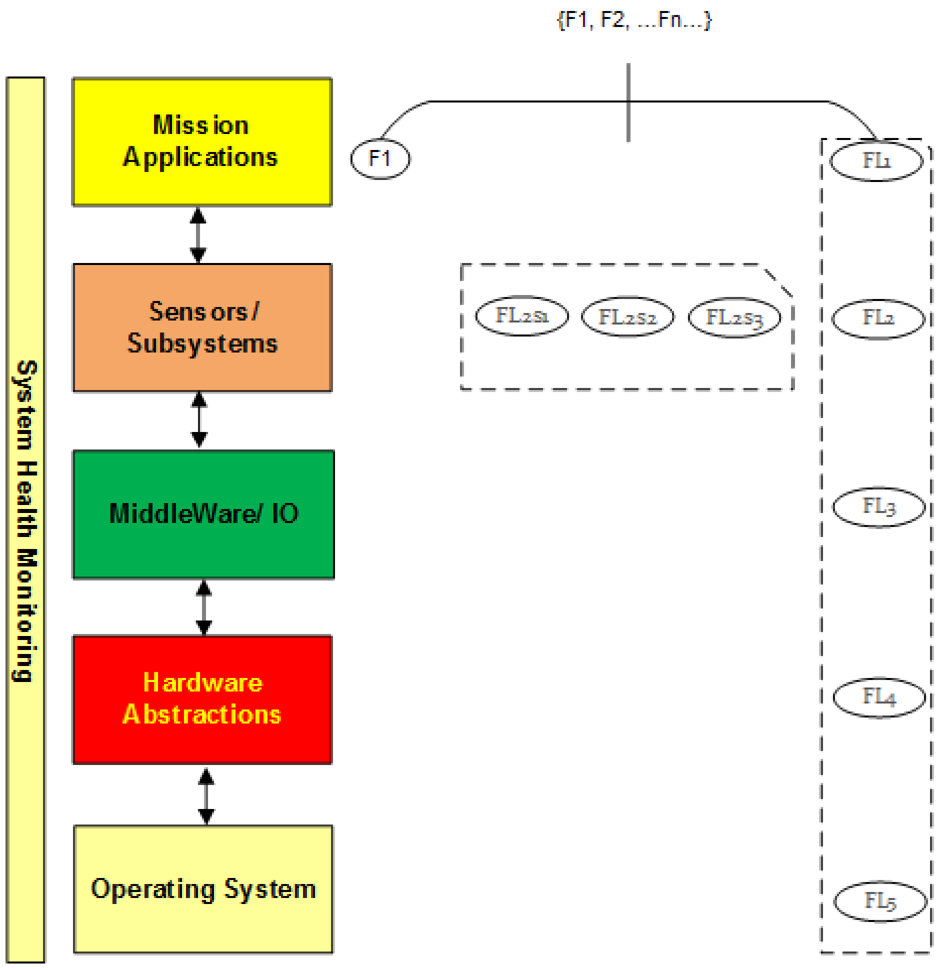


Fig. 1 A layered avionics architectural model

Considerations from Self Adaptive Software theory would require us to view the adaptation mechanisms in the context of avionics domain to cater for:

D3 Self-Configuration: Provide mechanisms to marshal or decommission software agents based on the adaptation context (E.g. Provide a service for a degraded mission) or choose a software entity that employs different algorithm within an agent (Eg: When a learning method is producing a result that is not converging for a sufficiently long time).

D4 Self-Healing: Provide for handling errors and faults during the adaptation process. This implies providing methods to enable each component or an agent involved in the process considering the environment in which it operates. For example, in the case of an UAS, which experiences a loss of communication link, self healing is related to how well the communication component adapts to regain the communication link. As another example, in the case of an adaptive flight planning system, it may be how well the weather agents are selected (voted) in case some of them fail, still maintaining continuity of service.

D5 Self-Optimization: In order to answer the fundamental question of whether the adaptive system in the context of avionics provides significant benefits, adaptivity by way of learning can be employed to improve overall performance. Measures like throughput, utilization, and workload will always need to be met.

D6 Self-Protection: Viewed from the environment perspective, safe performance can be resorting to mitigation measures for impending situations or events that could lead to vehicle eventualities that prove to be major, hazardous or even catastrophic in the context of safety criticality. Viewed from the perspective of software entities, this could be protection against malevolent attacks.

Considering the avionics domain itself [29], some drivers for VERMILLION would be aspects related to the following:

D7 Safety: It is preferable to keep the functional aspects distinct and separate from adaptive aspects and therefore an external approach would be the choice. When Adaptivity is engineered for optimization and safety issues arise, then safety requirements are given priority over everything else. Entities that address long term strategic response should preferably be separate from those that provide short term tactical response. Both these different kinds of entities need to be monitored by external means. In this context, the BDI model should explicitly recognize and categorize safety beliefs and have a mechanism to prioritize tactical responses. Therefore, in order to meet safety requirements a design solution that incorporates specific types of domain agents that handle safety related aspects should be engineered. Failure conditions should be monitored and trapped by other specific agents or entities.

D8 Deterministic Response: Determinism may be regarded from three viewpoints: functionality, resources, and time. In the context of functionality, we provide for three focus points which we believe, if tackled properly, will to a large extent, mitigate non-determinism.

1) The first focus point is the function that maps beliefs to goals in the BDI model of agency [16]. An uncertain input (percept) that is exogenous (from the environment) or endogenous (within the adaptive system) is always mapped to produce a definitive goal. However a downstream decision making process (decision-making process) like the intention generation mechanism within the BDI model will either elect to keep this goal or drop it, possibly based on a learning algorithm.
2) The second focus point will be the decision making or the reasoning process itself, which may introduce non-determinism. A non-convergent learning process may be evaluated by the outputs that it produces (temporally also) or may be monitored by observing some output variables from the system itself (variance of the output or estimate of the variance of the adaptive system outputs or even the probability density function of adaptive system outputs during the learning process). The jitter associated with the output of the decision process needs to be carefully considered. For example, in the case of path planning, there should not be a frequent change of the next waypoint (destination) choice when a decision has to be made.
3) The third focus point is the amount of autonomy that is provided to the agents: a restricted autonomy with a master in control is necessary. Non-deterministic resource usage may occur when dynamically allocated memory for stack usage may creep in as a result of using adaptive functions/components at the application level (though this is a low level aspect generally handled by the operating system). A consequence of this is that we need to have a finite goal set and dis-allow dynamic resource usage within our agent framework implementation. The aspect of determinism with respect to time is dealt with in the next paragraph.

D9 Real-timeliness: The aircraft domain requires guarantees w.r.t responses from the system, since responses from the system have meaning and correctness only in the context of a finite time duration from the occurrence of an event. Therefore this has an influence on the degree of autonomy allowable for the adaptive system. In many cases, the order of execution of pieces of software (agents) imposes a restriction on the autonomy aspect of agents. For example, during an adaptive path planning function a disturbance to the safe flight of an aircraft may require suitable responses from the control systems responsible for keeping the aircraft platform safe and stable. This may require responses from pieces of software controlling the aircraft platform more urgently than those responsible for the path planning function. In this context, the BDI model should explicitly recognize and categorize timeliness beliefs. Also, in case plan executions of agents are bound by timeliness constraints, it is desirable to elicit performance metrics related to real timeliness. For example, in case of an UAS, the amount of time from the detection of a failure of a rotor to the initiation or completion of the response may be specified. Timeliness constraints would need to be engineered for the adaptation mechanisms and monitored to provide failure recovery states when the adaptations fail. Update rates and Refresh rates on input data need to be considered in the context of belief generation for the BDI model. Update rates for the outputs from the agents will need due consideration in context of the overall aircraft control/operating loops.

D10 Byzantine Fault Tolerance: Engineering adaptive systems involves processing probably distributed across processing entities. When these processing entities of the system fail in an arbitrary fashion trying to produce incorrect or inconsistent outputs, the system must be resilient to such failures [30]. Handling these kinds of failures requires analysis and identification of the zones and processing entities where the system is susceptible to such failures and appropriate mitigation measures need to be employed.

III. Verification and Validation

Although verification and validation forms the most important components of certification, traditional methods of verification and validation may not be completely applicable in the context of adaptive systems since one of the fundamental approaches is to examine results of tests performed on such a system, which will produce non-repeatable outputs [14]. One of the apprehensions is whether the adaptive system will not endanger the safety of the airborne platform. Therefore, for civil commercial aircraft SAE ARP4761 [31], describes the safety assessment process, and supports the development process described by ARP4754A [26] mentioned earlier. While development assurance process and safety assessment process are covered by ARP4754A and ARP4761 respectively, the design assurance processes are covered by DO-178B/C [32, 33] and DO-254 [34]. In this context, validation of requirements and their allocation (guided by the 4754 process and supported by the 4761 process) occurs on the left side of a V-process, while verification of those requirements happen progressively on the right hand side of the V-process as shown in Fig. 2, adapted from [35]. It is to be noted that the outer arm of the left hand side V-process is representative of the ARP4754 while the inner arm of the left hand side V-process is representative of the ARP4761.

A. Requirements and guiding standards

Requirements are progressively flowed down from the top (aircraft level) to the bottom (system and sub-systems level) in a systematic manner. The right arm of this V-process is governed by the DO-178B/C [32],
[33] and DO-254 [34]. Since probabilistic assessment cannot be made about software, development assurance level guidelines are provided by DO-178B/C. While DO-178B/B refers to certification (legal acknowledgement that the equipment compiles with requirements), it is closely related to verification since that is one of the important means

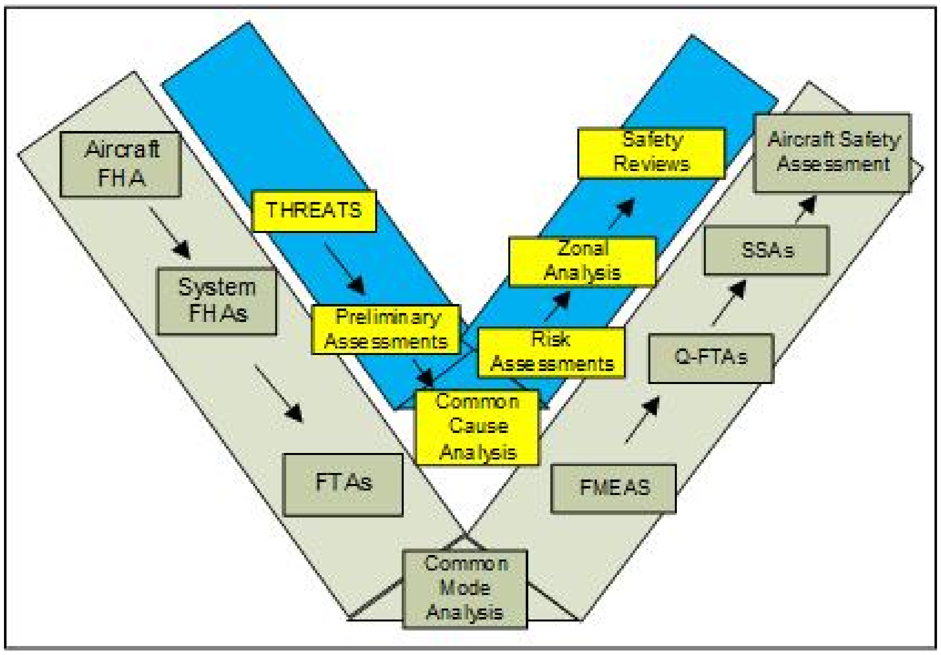


Fig. 2 V&V of Avionics Requirements Process - a top level perspective [35]

leading to certification. Novel methods for verification assume importance and formal methods supplement DO-333 of DO-178C [36] provides guidelines enabling the use of mathematical techniques for requirements verification. Therefore, in consideration of the above, the basic functional requirements and safety requirements are developed with guidance from ARP4754 and ARP4761, while verification of those requirements are guided by DO-178C/DO-333. These standards do not discuss about evaluating adaptive requirements and therefore we have proposed a method to evaluate them using test runs performed by simulation which is outlined in [7] and

[5]. Note that simulation plays a major role in the verification of adaptive systems. The DO-178C process can be visualized with the activities shown in Fig. 3. Since each of formal methods have their own strengths and relative difficulty level, their amenability to application in the DO-178C process may be specific [37]. We have used model checking to cover some objectives of high-level requirements versus system requirements compliance (first level from the top in Fig. 3), some objectives of low-level requirements/architecture versus high-level requirements (second level from the top in Fig. 3), and also some objectives of source code compliance with low-level requirements (third level from the top in Fig. 3). The dotted lines are indicative of the objectives while solid arrows indicate development flow down.

B. Design and prototype implementation

After requirements have been allocated to appropriate agents in our multi-agent based adaptive system, the multi-agent system designed using the framework provided in section V can be implemented using any of the agent oriented languages or general purpose programming languages which concretize the structures and behavior specified

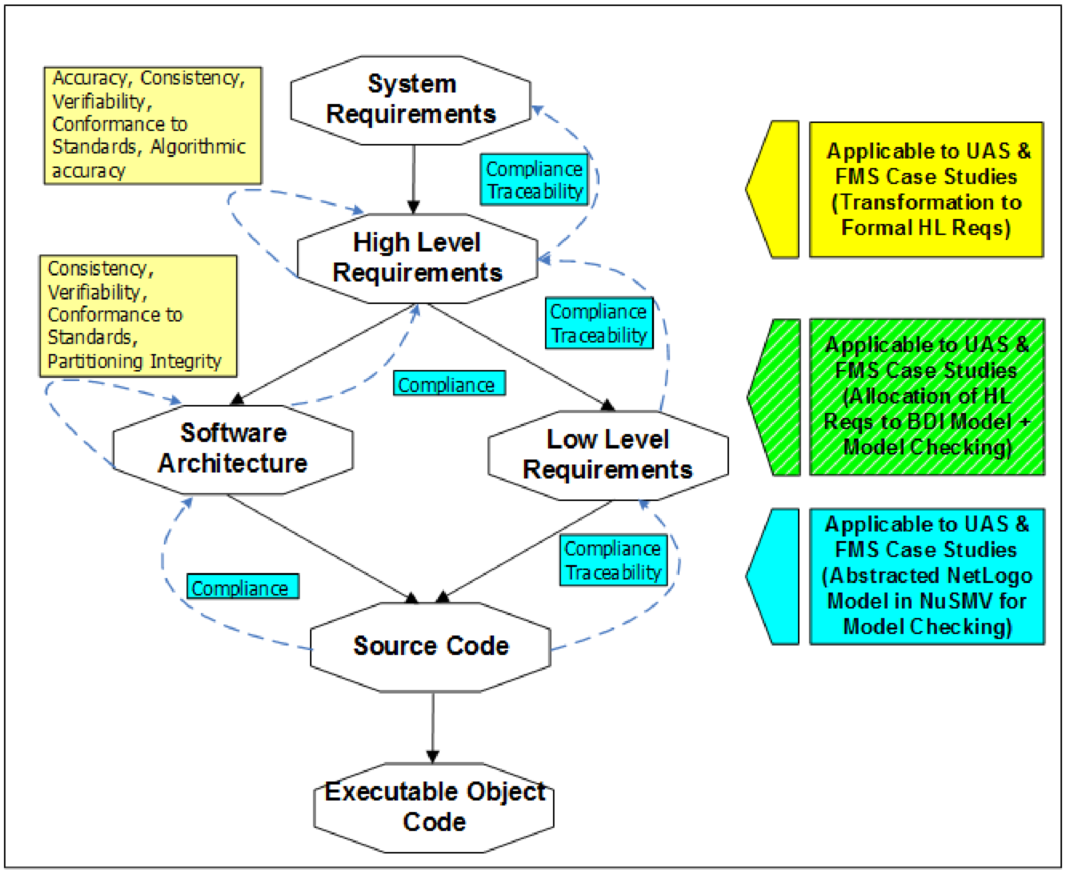

Accuracy, Consistency, Verifiability, Conformance to Standards, Algorithmic accuracy
System Requirements
Compliance Traceability
High Level Requirements
Consistency, Verifiability, Conformance to Standards, Partitioning Integrity
Compliance Traceability
Compliance
Software Architecture
Low Level Requirements
Compliance
Compliance Traceability
Source Code
Executable Object Code
Applicable to UAS & FMS Case Studies (Transformation to Formal HL Reqs)
Applicable to UAS & FMS Case Studies (Allocation of HL Reqs to BDI Model + Model Checking)
Applicable to UAS & FMS Case Studies (Abstracted NetLogo Model in NuSMV for Model Checking)

Fig. 3 V&V of Avionics Requirements Process - use of Model Checking within the DO-178C Guidelines

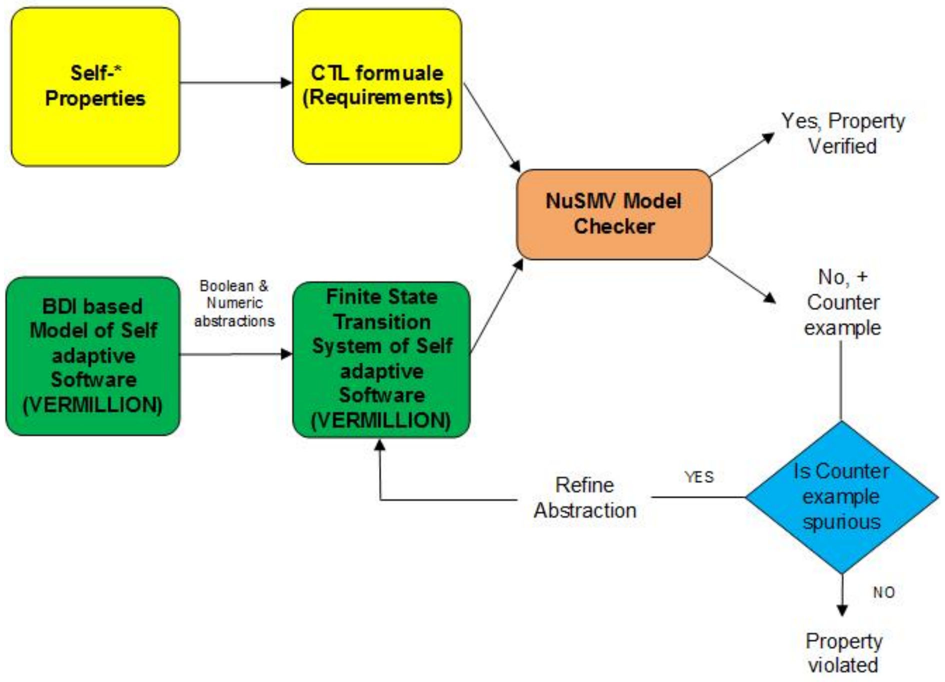

Self-* Properties
CTL formuale (Requirements)
Yes, Property Verified
NuSMV Model Checker
BDI based Model of Self adaptive Software (VERMILLION)
Boolean & Numeric abstractions
Finite State Transition System of Self adaptive Software (VERMILLION)
No, + Counter example
Refine Abstraction
YES
Is Counter example spurious
NO
Property violated

Fig. 4 Abstraction and model checking based self-adaptive software formal verification

in VERMILLION. It may be mentioned that parts of agent implementations that do not rely on non-deterministic or non-converging learning algorithms need to be crafted with appropriate coverage considerations (statement coverage, decision coverage, MC/DC) based on the criticality level of the software that is mechanized by these agents. The notions of what happens to failures within these plans or when an agent or a group of agents fail needs due consideration. Two of our case studies presented in section VII have been implemented in NetLogo [38].

C. Formal verification and validation

The implementations are then abstracted to finite state transition systems containing Boolean and few integer variables. The abstractions are performed manually using the following principles:

- Data abstraction techniques [39]
- Cone of influence reduction techniques [39]
- Predicate abstraction techniques [40], [41] and [42]

The primary consideration in generating the finite state transition systems is ensuring that the abstractions are sound, i.e. it should admit all behaviors of the BDI based VERMILLION model. The requirements will embody

self-* properties and are specified as CTL formulae [28]. CTL is a branching temporal logic that helps specify system properties. The uniqueness of this branching time logic is that at each moment in time there appear several different possibilities of future, leading to the notion of a tree of states rather than a single path or a unique future. There are two important conceptualizations in CTL: the state formula (hold at a state) and the path formula (hold along a path). CTL formulae have temporal operators (X, F, G, U denoting respectively neXt state, some Future state, all future states (Globally), Until) that must be immediately preceded by a path quantifier (A, E denoting along All Paths, Exists along at least one path). These temporal operators are used to specify precisely about properties of some or all computations that start in a state. The requirements specified as CTL formulae and the finite state transition system described by the CTL predicates are passed to a model checker, which in our case is the NuSMV model checker [43]. This is diagrammatically shown in the Fig 4. The model checking [44] results in both of our case studies revealed property violations which led to design modifications in the context of learning adaptation convergence. The modification was the introduction of recovery states in the transition system model which was then carried over to the implementation.
Validation is concerned with the determination that the requirements specified are correct and complete. Adaptive systems pose challenges in this context, for example the specification that a flight planning system shall provide an optimized path presents significant challenges. Therefore we provide methods to address these aspects in sections VII.A.3 and VII.B.3. Fault tolerant aspects pose another area of challenge since it is not possible to build a system that will account for all possible byzantine faults [30]. In this context, we have suggested some measures to mitigate such faults in section VI.E. Therefore, as we can assess, methods are needed to evaluate adaptive systems and in this context and we believe that by measuring some of the Self-* properties, combined with simulation, analysis, and testing, we are on a firmer ground for validating these systems.

IV. VERMILLION BDI framework

A. BDI for Avionics

Although the BDI model of agency provides the underpinning principle, this model of agency needs to be examined in the context of its usage for the avionics domain. Table 1 details the differentiators between the general BDI model and the BDI for avionics. However, in both cases, *Beliefs* can map to predicates and temporal formula while *Goals* invariably are well represented by temporal logic formulae. Therefore, these mappings would remain the same whether we consider a system to be representative in the avionics domain or not.

B. VERMILLION BDI

Fig. 5 shows the architecture of VERMILLION, a BDI based multi-agent system which takes into consideration the drivers mentioned in section II.B and provides a foundational overview of the framework. The drivers applicable to the appropriate elements are placed in ellipses tagged to the elements. The VERMILLION framework provides four

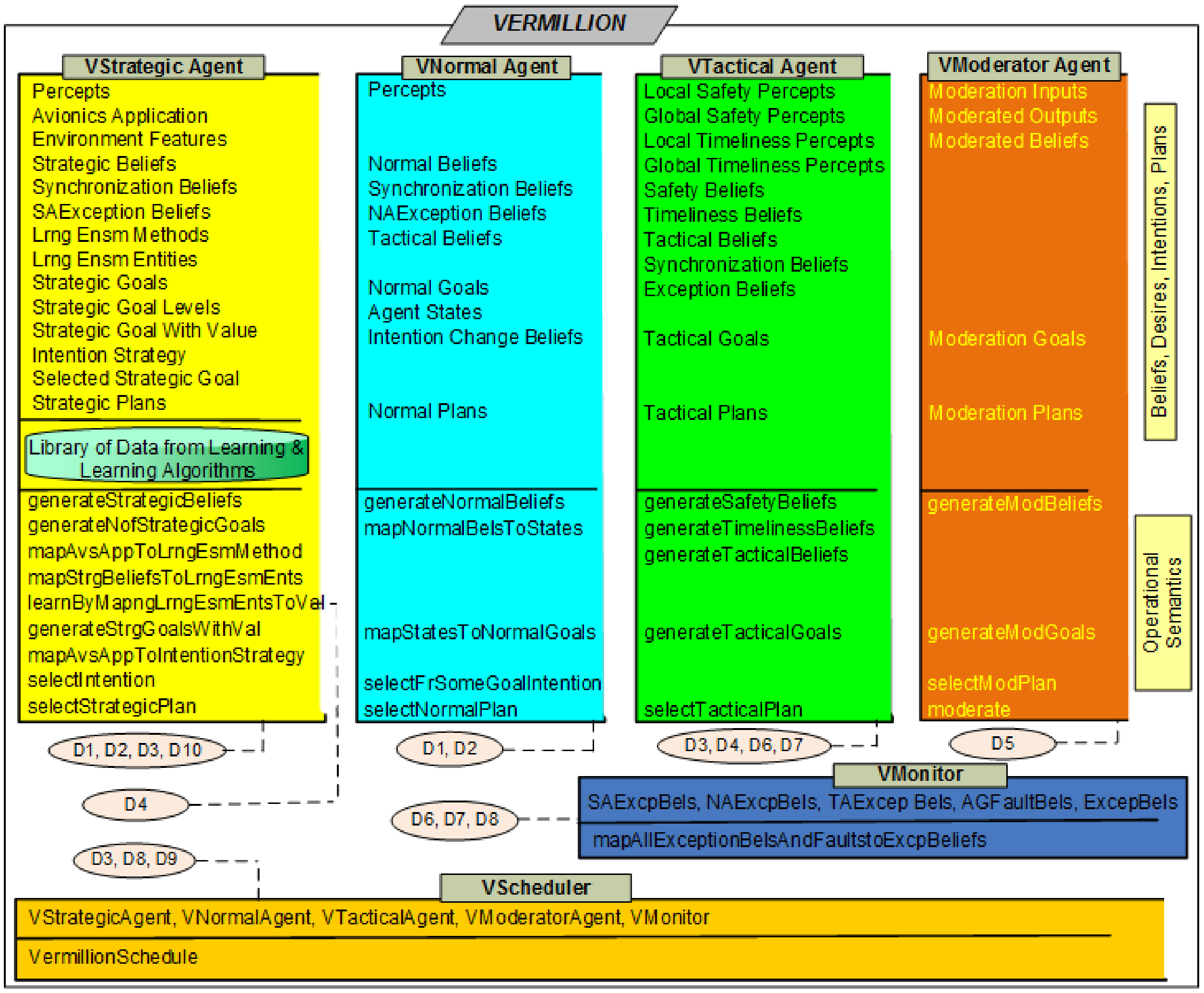


Fig. 5 VERMILLION BDI model

Table 1 Measures enabling Validation of Non-Functional Requirements, AFPS

Difference	General BDI	Avionics BDI
1	there is a notion of first trapping events and then finding plans that match events in the event queue along with beliefs that the agent has	For an avionics system, many events need to be considered and appropriate decision taken based on a notion of priority
2	in a generalized BDI model there is a notion of an intention stack (as stated earlier in section I), since a plan may contain sub-goals that require further planning	For an avionics system, this concept of accommodating a tree of sub-goals and plans needs to be bounded and kept to a minimum
3	A generalized classical BDI model does not directly segregate long term planning and tactical planning	For an avionics system, segregation of functions based on tactical events and strategic events is essential
4	There is a notion of a tree of goals, subgoals and plans unfolding on this tree	For an avionics system, the number of goals and subgoals need to be bounded and the structure of the plans would need attention from the viewpoint of determinism and real-timeliness and hence the notion of *further planning* within a plan needs restrictions

different types of agents: *Vermillion Strategic Agents* (VSA), *Vermillion Tactical Agents* (VTA), *Vermillion Normal Agents* (VNA), *Vermillion Moderator Agents* (VMA). The framework also provides for *Vermillion Monitors* (VMon), and a *Vermillion scheduler* (VSch).

- VSAs address the [*RqFAd Part*] and [*RqNFAd Part*] (see equation 1). The VSA mechanizes behavior that requires long term strategic response. The VSA provides the ability to adapt different kinds of learning based on the application and houses the repository of learning data and the various learning algorithms. Depending on the application and the learning algorithm, the intention strategy may be chosen.
- VTAs address {*RqF Part*} and {*RqNF Part*} (see equation 1) for exclusively meeting the safety requirements. The VTA provides behavior that requires short term and emergency responses due to detection of critical events. These events are obtained through percepts and mapped to safety and timeliness beliefs. The union of safety and timeliness beliefs are termed 'Tactical Beliefs'. The Tactical Goal Generation function maps the Tactical Beliefs to Prioritized Goals, since critical events need to be examined and prioritized before they are acted upon.
- VNAs ensure meeting the {*RqF Part*} and {*RqNF Part*} which are imposed by the system functional requirements that are specified during the normal description of a system. The VNA mechanizes behavior that provides responses catering to normal functions. The VNA operates mainly by generating Agent States using Normal Beliefs. Most

states are mapped to goals in a straight forward manner with the exception of a few which may require intention generation.

- VMAs supplement the VSAs when a moderation function is required amongst a set of agents. VMAs are optional and not every application would necessarily need them. One mechanism by which VMAs may provide moderation is via negotiations, mechanized via different protocols.
- VMons are exclusively meant as a support system for built-in fault tolerance 'shadowing' all other agent types, and Vermillion Scheduler (VSch) that address determinism and realtimeliness concerns. Fault tolerance is engineered by recording and acting on failures caught through Exception Beliefs. VMons report failures to appropriate VTAs for failure responses. One specific VTA is allocated the responsibility to provide response in case of Failures observed in all other VTAs.
- VSch determines the timelines for execution of VSAs, VNAs, VTA, VMAs and VMons, relaxing the autonomy characteristic of agents. Generally, scheduling strategies can be categorized into static and dynamic scheduling strategies. A number of points have been raised about why static scheduling is less efficient than dynamic scheduling in the avionics domain [45]. However, in the avionics domain, where an adaptive system is being incorporated, the advantage of determinism offered by static scheduling and overhead associated with dynamic scheduling [46] cannot be overlooked, we have leaned towards the static scheduling approach. However, the framework does not preclude one from using a dynamic scheduling approach and this can easily be accommodated.

V. Modeling VERMILLION with Z Notation

Our goal is to define a reference architectural framework that can be seamlessly integrated for an adaptive avionics systems with minimal tailoring and yet at the same time be pliable for analysis. Since certification is an important aspect in the domain, the approach should aim to satisfy most objectives if not all, required by DO-178C. At the heart of such a reference architecture will be the need for a language that can capture model specifications derived from all of the adaptive system requirements captured in CTL. We looked at a number of candidates, considering the broad categories of algebraic and model based specifications and eventually chose Z [21]. Section VIII provides more discussions on aspects related to this choice. Z is a formal specification language and provides a number of advantages even though it does not directly provide support for timing aspects and concurrency – it is domain independent, provides tool support, helps capture model constructs precisely, and obtain refinements that are easily tractable. Most importantly, it provides the freedom to specify the model at a high level, independent of an implementation, yet succinctly capture details needed at a reference architecture level. Z has foundations from typed set theory and first-order logic. At the core is a structure called a schema that separates declarations and predicates. Data Invariants, States, Operations, Invariants, Pre-conditions, and Postconditions are all elements of the Z-language. These characteristics lend itself well for certification as laid out in DO-333 supplement of DO-178C. Adaptation via learning mechanisms can be abstracted

mucH like a black box and therefore the base Z models can be extended to include Z models of learning [47] . In simple terms, Z is amenable to specifying a library of learning algorithms [48]. More details of Z are provided in [21].
We now provide the specifications of VERMILLION in Z, first describing the static aspects of the model in subsections V.A through V.L and how the dynamical aspects come alive in section VI for some important functions. The specifications provided in this section have been specified and syntax checked using the CZT standalone tool [49].

A. Terms, Atoms, Solutions

Our definitons of the base elements are modeled along lines similar to those specified in dMARS [50] but those elements that are specific to VERMILLION will provide significant extensions.
We start out by defining the *given sets*:

[*PredSym, FunSym*]

[*Events*]

[*LocalSafetyEvents, GlobalSafetyEvents, LocalTimelinessEvents, GlobalTimelinessEvents*]

[*ModerationTypes*]

[*EnvironmentFeatures*]

[*ActionSym*]

where *FunSym* represents function symbols while *PredSym* represents the predicate symbols. The different kinds of events are segregated based on the type of events and this accounts for *LocalSafetyEvents, GlobalSafetyEvents, LocalTimelinessEvents, GlobalTimelinessEvents*. However, they all can be assumed as subsets of the more general *Events*. The *ModerationTypes* is needed to abstract the different types of moderation that may be employed when multiple agents are collectively involved in solving a problem. *EnvironmentFeatures* are used to map entities in the environment to learning beliefs that are used to enable learning for strategic agents. The actions that an agent performs in compliance with a plan are facilitated by *ActionSym*.

We would also need the following to represent the convergence aspect of a learning algorithm:

ConvergeStatus::= Converge | *Diverge*

The notion of a Boolean would be defined as follows:

Boolean::= True | *False*

Since we are dealing with a real time system, the notion of a *TimeInterval* becomes necessary:

TimeInterval

$interval : \mathbb{P}\,\mathrm{T}$

$begin, finish : \mathrm{T}$

$interval = begin \mathrel{..} finish$

In VERMILLION, a Term can be one of the following: (a) constant (b) variable or (c) a function symbol applied to a sequence of terms.

$Term ::= const \ll Const \gg \mid var \ll Var \gg \mid fun \ll FunSym \times \mathrm{seq}_1\ Term \gg$

where *Const* and *Var* are a set of constants and variables. We also define *Safety Term*s, *Timeliness Term*s, *Learning Term*s, *Exception Term*s, *Agent Fault Term*s, *Synchronization Term*s, and *State Term*s that are needed during elaboration of various agents:

$SafetyTerm ::= const \ll SConst \gg \mid var \ll SVar \gg \mid fun \ll FunSym \times \mathrm{seq}_1\ SafetyTerm \gg$

$TimelinessTerm ::= const \ll TConst \gg \mid var \ll TVar \gg \mid fun \ll FunSym \times \mathrm{seq}_1\ TimelinessTerm \gg$

$LearningTerm ::= const \ll LConst \gg \mid var \ll LVar \gg \mid fun \ll FunSym \times \mathrm{seq}_1\ LearningTerm \gg$

$ExceptionTerm ::= const \ll EConst \gg \mid var \ll EVar \gg \mid fun \ll FunSym \times \mathrm{seq}_1\ ExceptionTerm \gg$

$AgentFaultTerm ::= const \ll AFConst \gg \mid var \ll AFVar \gg \mid fun \ll FunSym \times \mathrm{seq}_1\ AgentFaultTerm \gg$

$SynchronizationTerm ::= const \ll SyncConst \gg \mid var \ll SyncVar \gg \mid fun \ll FunSym \times \mathrm{seq}_1\ SynchronizationTerm \gg$

$StateTerm ::= const \ll Const \gg \mid var \ll Var \gg \mid fun \ll FunSym \times \mathrm{seq}_1\ Term \gg$

$ModerationTerm ::= const \ll Const \gg \mid var \ll Var \gg \mid fun \ll FunSym \times \mathrm{seq}_1\ ModerationTerm \gg$

We also define atoms which are predicate symbols as follows:

NAtom

head : *PredSym*

terms : seq*Term*

SAtom

head : *PredSym*

safetyterms : seq*SafetyTerm*

TAtom

head : *PredSym*

timelinessterms : seq*TimelinessTerm*

LAtom

head : *PredSym*

learningterms : seq*LearningTerm*

EAtom

head : *PredSym*

exceptionterms : seq*ExceptionTerm*

AFAtom

head : *PredSym*

agentfaultterms : seq*AgentFaultTerm*

SyncAtom

head : *PredSym*

syncterms : seq*SynchronizationTerm*

ModAtom

head : *PredSym*

modterms : seq*ModerationTerm*

A solution is the output or answer from an agent and can be defined as:

Solution::=const≪Const≫ | *var≪Var≫*

B. Beliefs and Goals

The base definition of *Beliefs* is similar to the one provided in dMARS [50]. However, we define new kinds of *Beliefs* that are relevant for the avionics domain. *Beliefs* are defined from the base atoms and defined as belief formula as given by:

NormBelForm::=pos≪NAtom≫ | *not≪NAtom≫*

SafetyBelForm::=pos≪SAtom≫ | *not≪SAtom≫*

TimelinessBelForm::=pos≪TAtom≫ | *not≪TAtom≫*

TacticalBelForm::=SafetyBelForm ∪ *TimelinessBelForm*

LearningBelForm::=pos≪LAtom≫ | *not≪LAtom≫*

ExceptionBelForm::=pos≪EAtom≫ | *not≪EAtom≫*

AgentFaultBelForm::=pos≪AFAtom≫ | *not≪AFAtom≫*

SynchronizationBelForm::=pos≪SyncAtom≫ | *not≪SyncAtom≫*

ModerationBelForm::=pos≪ModAtom≫ | *not≪ModAtom≫*

The belief fromulae can contain both constants and variables.

A *Goal* in VERMILLION is a temporal formula which is a specific state of the belief formula. This specific state is said to be attained when the VERMILLION agents take action or changes in the environment of the agents lead to the belief formulae attaining these distinct values. We define a *Goal* as:

Goal::=≪NormalGoal≫ | *≪StrategicGoal≫* | *≪TacticalGoal≫* | *≪ModerationGoal≫*

and

NormalGoal::=achieve ≪NormalBelForm≫ | *query ≪NormalBelForm≫*

StrategicGoal::=achieve ≪LearningBelForm≫ | *query ≪LearningBelForm≫*

TacticalGoal::=mustAction ≪TacticalBelForm≫ | *query ≪TacticalBelForm≫* | *mustAction ≪ExceptionBelForm≫*| *query ≪ExceptionBelForm≫*

ModerationGoal::=achieve ≪ModerationBelForm≫ | *query ≪ModerationBelForm≫*

We see that goals are of four types: *Normal*, *Strategic*, *Tactical*, and *Moderation*; *achieve* is an operator that triggers agent actions so that the temporal formula associated with normal, strategic or moderation goals are attained. *Query* is another operator that obtains the current state of the temporal formula associated with the goal. *mustAction* is yet another operator that guarantees a specific set of actions taken by tactical agents. Note that *Tactical Goals* are always achieved since not doing so would possibly lead to violation of safety or realtimeliness constraints. while the *achieve* and *Query* operators are the same as those defined in dMARS, the *mustAction* operator is unique to VERMILLION. In the context of a *Strategic Agent*, the following would become necessary:

StrategicGoalWithValue

strg : *StrategicGoal*

stgv : $\mathbb{R}$

C. Intentions

In VERMILLION, Strategic and Normal Agents are associated with Intentions. Intentions are committed goals. Three different committment strategies are identified: blind committment wherein an intention is maintained until a goal is fully achieved, single minded committment wherein an intention is maintained until a goal is fully achieved or it is assessed some time later that it is impossible to achieve, open minded committment wherein an intention is maintained until a goal is fully achieved or it is assessed some time later that it is impossible to achieve or it is no longer a goal. These different committment strategies may be implemented using different mechanisms. For example, *Mechanism_2* in the *IntentionMechanism* below, may represent an algorithm that enforces open minded committment. This algorithm may use the method of examining goal value associated with every goal and keeping that goal as an intention until it is within a range of bounded values. Therefore, we can define committment strategies as a list of algorithms that provide such strategies:

IntentionMechanism::={Mechanism_1, Mechanism_2, Mechanism_3 . . . }

Note that we do not specify the algorithms, but only state the fact that they exist and group them in a set.

D. Agent Set, Agent Phases, and Agent States

We define the VERMILLION agent entities as follows:

VermillionAgentsEntities::= VSA | *VNA* | *VTA* | *VMA* |*Vmon*

The formulation of different kinds of agents in VERMILLION leads to different notions to reason about the behavior for each kind of agent. We use the term *Phase* to encompass the macro behavior of any kind of agent within VERMILLION, using BDI as the guiding principle. The phase applicable to VSAs, VNAs, VTAs, VMAs, will be defined as follows:

VermillionAgentPhases::= Agent_Configuration | *Percept_Formation* |
Belief_Updation | *Desire_Generation* | *Intention_Selection* | *Plan_Execution*| *BLOCKED*

On may easily identify the above agent phases with parts of the basic BDI Interpreter cycle provided in section I.

We use the term *State* to characterize the behavior of any agent within the phase. However the notion of a state is more relevant in the case of a Normal Agent since states are mapped to goals. Therefore, we define Normal Agent states as follows:

NormalAgentStates::={StateTerm}

E. Actions and Events

Actions in VERMILLION are similar to those in dMARS
[50]. However, we donot distinguish between external and internal actions. Actions are like procedure names which can bring out changes in the agents environment or update beliefs of an agent.

AgAction

actionName : *ActionSym*

terms : seq*Term*

Although at the gross level both dMARS
[50] and VERMILLION have *Events*, the conceptualization is slightly different in VERMILLION. *Events* are either sensory inputs received from the environment or those internally generated within agents during execution of *Plans*. Whether these are external or internal, our abstractions of *Events* and its subsets as provided in Section V.A would remain unaltered and will encompass these distinctions also.

F. Plans

Plans are the artifacts in VERMILLION that are responsible for carrying out an Agent's Desires and Intentions. Every *Desire* or *Intention* is associated with a *Plan* or set of *Plans*. Although *Plans* in VERMILLION are notionally similar to that in dMARS [50], the structure is tailored keeping determinism and real timeliness aspects in focus. The invocation element of the *Plan* is similar but the structure of the body as a tree containing intermediate states connected by *Goals* and *Actions*, is done away with. Instead, there are only intermediate states and *Actions*. *Goals* are mapped to *Plans* using the invocation element. There are four kinds of *Plans* in VERMILLION and each of these types are associated with the Agent Types. All VERMILLION *Plan* types have a very simple structure: an invocation condition, a plan body, and exception handlers that will be invoked if any of the steps within the *Plan* generate active exception belief formulae and those exception can be handled within the plan. If the exception cannot be handled at the plan level, they are propagated to the agent. With this general exposition, the following definitions are self-explanatory.

STRATEGIC PLANS are associated with the Strategic Agents which are defined later.

The components of a *Strategic Plan* are defined as follows:

$SP\_Trigger ::= SelectGoal \langle\langle StrategicGoal \rangle\rangle$

$SP\_Branch ::= ExtOrIntSAgAction \langle\langle AgAction \rangle\rangle$

$SP\_Body ::= End \langle\langle State \rangle\rangle \mid Fork \langle\langle State \times \mathbb{P}\,(SP\_Branch \times SP\_Body) \rangle\rangle$

$SP\_Status ::= NoExcep \mid Excep$

The *Strategic Plan* can be defined as:

$$\begin{array}{|l}
\hline
\textit{StrategicPlan} \\
inv : SP\_Trigger \\
body : SP\_Body \\
status : SP\_Status \\
\hline
\end{array}$$

When the strategic agent executes a strategic plan, it may result in the plan being executed successfully without any exception or there may be an exception raised indicating unsuccessful plan execution. The exception can be handled by a pre-defined exception handler or it may be propagated up to the next higher level. The functions that map the exceptions are provided below:

$SelectSPExcepHdlr : ExceptionBelForm \rightarrowtail SPExcepHdlr$

$SPExcepPropagate : ExceptionBelForm \rightarrowtail SPPropagatedExcp$

SPExcepHdlr is the set of Exception Handlers defined for most SP exceptions, while *SPPropagatedExcp* is the set of SP exceptions for which there are no handlers.

ExecuteStrategicPlanExcep

$\Delta VermillionStrategicAgent$

$\Delta StrategicPlan$

$excep? : SP\_Status$

$spexception? : \mathbb{P}\, ExceptionBelForm$

$(excep = Excep \land (vsa\_excep\_beliefs' = spexception) \land$

$((spexception \in \operatorname{dom} SelectSPExcepHdlr) \lor$

$(spexception \in \operatorname{dom} SPPropagatedExcp))$

ExecuteStrategicPlanNoExcep

$\Delta VermillionStrategicAgent$

$\Delta StrategicPlan$

$strg\_beliefs? : \mathbb{P}\, LrngBelForm;$

$excep? : SP\_Status$

$spexception? : \mathbb{P}\, ExceptionBelForm$

$(excep = NoExcep \land (spexception \mapsto \emptyset) \land$

$(strg\_beliefs' = strg\_beliefs \cup strg\_beliefs?)$

Therefore, we can define the *ExecuteStrategicPlan* as:

$ExecuteStrategicPlan \mathrel{\widehat=} ExecuteStrategicPlanExcep \lor ExecuteStrategicPlanNoExcep$

NORMAL PLANS are associated with the Normal Agents which are defined later.

The components of a *Normal Plan* are defined as follows:

$NP\_Trigger ::= SelectGoal \langle\!\langle NormalGoal \rangle\!\rangle$

$NP\_Branch ::= ExtOrIntNAgAction \langle\!\langle AgAction \rangle\!\rangle$

$NP\_Body ::= End \langle\!\langle State \rangle\!\rangle \mid Fork \langle\!\langle State \times \mathbb{P}\,(NP\_Branch \times NP\_Body) \rangle\!\rangle$

$NP\_Status ::= NoExcep \mid Excep$

```
NormalPlan
inv : NP_Trigger
body : NP_Body
status : NP_Status
```

When the normal agent executes a normal plan, it may result in the plan being executed successfully without any exception or there may be an exception raised indicating unsuccessful plan execution. The exception can be handled by a pre-defined exception handler or it may be propagated up to the next higher level. The functions that map the exceptions are provided below:

$SelectNPExcepHdlr : ExceptionBelForm \pfun NPExcepHdlr$

$NPExcepPropagate : ExceptionBelForm \pfun NPPropagatedExcp$

NPExcepHdlr is the set of Exception Handlers defined for most SP exceptions, while *NPPropagatedExcp* is the set of NP exceptions for which there are no handlers.

```
ExecuteNormalPlanExcep
ΔVermillionNormalAgent
ΔNormalPlan
excep? : NP_Status
npexception? : ℙ ExceptionBelForm
----------
(excep = Excep ∧ (vna_excep_beliefs′ = npexception) ∧
((npexception ∈ dom SelectNPExcepHdlr) ∨
(npexneption ∈ dom NPPropagatedExcp))
```

ExecuteNormalPlanNoExcep

$\Delta VermillionNormalAgent$

$\Delta NormalPlan$

$normal\_beliefs? : \mathbb{P}\, NormBelForm;$

$excep? : NP\_Status$

$npexception? : \mathbb{P}\, ExceptionBelForm$

$(excep = NoExcep \land (npexception \mapsto \emptyset) \land$

$(normal\_beliefs' = normal\_beliefs \cup normal\_beliefs?)$

Therefore, we can define the *ExecuteNormalPlan* as:

$ExecuteNormalPlan \mathrel{\widehat=} ExecuteNormalPlanExcep \lor ExecuteNormalPlanNoExcep$

TACTICAL PLANS are associated with the Tactical Agents which are defined later.

The components of a *Tactical Plan* are defined as follows:

$TP\_Trigger ::= SelectGoal \langle\langle TacticalGoal \rangle\rangle$

$TP\_Branch ::= ExtOrIntTAgAction \langle\langle AgAction \rangle\rangle$

$TP\_Body ::= End \langle\langle State \rangle\rangle \mid Fork \langle\langle State \times \mathbb{P}\, (TP\_Branch \times TP\_Body) \rangle\rangle$

$TP\_Status ::= NoExcep \mid Excep$

TacticalPlan

$inv : TP\_Trigger$

$body : TP\_Body$

$status : TP\_Status$

When the tactical agent executes a tactical plan, it may result in the plan being executed successfully without any exception or there may be an exception raised indicating unsuccessful plan execution. The exception can be handled by a pre-defined exception handler or it may be propagated up to the next higher level (scheduler). The functions that map the exceptions are provided below:

$SelectTPExcepHdlr : ExceptionBelForm \rightarrowtail TPExcepHdlr$

$NPExcepPropagate : ExceptionBelForm \rightarrowtail NPPropagatedExcp$

TPExcepHdlr is the set of Exception Handlers defined for most TP exceptions, while *TPPropagatedExcp* is the set of TP exceptions for which there are no handlers.

ExecuteTacticalPlanExcep

$\Delta VermillionTacticalAgent$

$\Delta TacticalPlan$

$excep? : TP\_Status$

$tpexception? : \mathbb{P}\, ExceptionBelForm$

$(excep = Excep \wedge (vta\_excep\_beliefs' = tpexception) \wedge$

$((tpexception \in \operatorname{dom} SelectTPExcepHdlr) \vee$

$(tpexneption \in \operatorname{dom} TPPropagatedExcp))$

ExecuteTacticalPlanNoExcep

$\Delta VermillionTacticalAgent$

$\Delta TacticalPlan$

$tactical\_beliefs? : \mathbb{P}(SafeyBelForm \cup TimelinessBelForm);$

$excep? : TP\_Status$

$tpexception? : \mathbb{P}\,\mathbb{P}\, ExceptionBelForm$

$(excep = NoExcep \wedge (tpexception \mapsto \emptyset) \wedge$

$(tactical\_beliefs' = tactical\_beliefs \cup tactical\_beliefs?)$

Therefore, we can define the *ExecuteTacticalPlan* as:

$ExecuteTacticalPlan \mathrel{\widehat{=}} ExecuteTacticalPlanExcep \vee ExecuteTacticalPlanNoExcep$

MODERATION PLANS are associated with the Moderator Agents which are defined later.

The components of a *Moderation Plan* are defined as follows:

$MP\_Trigger ::= SelectGoal \langle\langle ModerationGoal \rangle\rangle$

$MP\_Branch ::= ExtOrIntMAgAction \langle\langle AgAction \rangle\rangle$

$MP\_Body ::= End \langle\langle State \rangle\rangle \mid Fork \langle\langle State \times \mathbb{P}\ (MP\_Branch \times MP\_Body) \rangle\rangle$

$MP\_Status ::= NoExcep \mid Excep$

ModerationPlan

$inv : MP\_Trigger$

$body : MP\_Body$

$status : MP\_Status$

When the moderation agent executes a moderation plan, it may result in the plan being executed successfully without any exception or there may be an exception raised indicating unsuccessful plan execution. The exception can be handled by a pre-defined exception handler or it may be propagated up to the next higher level. The functions that map the exceptions are provided below:

$SelectMPExcepHdlr : ExceptionBelForm \nrightarrow MPExcepHdlr$

$MPExcepPropagate : ExceptionBelForm \nrightarrow MPPropagatedExcp$

MPExcepHdlr is the set of Exception Handlers defined for most MP exceptions, while *MPPropagatedExcp* is the set of MP exceptions for which there are no handlers.

ExecuteModerationPlanExcep

$\Delta VermillionModerationAgent$

$\Delta ModerationPlan$

$excep? : MP\_Status$

$mpexception? : \mathbb{P}\, ExceptionBelForm$

$(excep = Excep \land (vma\_excep\_beliefs' = mpexception) \land$

$((mpexception \in \operatorname{dom} SelectMPExcepHdlr) \lor$

$(mpexneption \in \operatorname{dom} MPPropagatedExcp))$

ExecuteModerationPlanNoExcep

$\Delta VermillionModerationAgent$

$\Delta ModerationPlan$

$moderation\_beliefs? : \mathbb{P}\, ModerationBelForm$

$excep? : MP\_Status$

$mpexception? : \mathbb{P}\, ExceptionBelForm$

$(excep = NoExcep \land (mpexception \mapsto \emptyset) \land$

$(moderation\_beliefs' = moderation\_beliefs \cup moderation\_beliefs?)$

Therefore, we can define the *ExecuteModerationPlan* as:

$ExecuteModerationPlan \mathrel{\widehat=} ExecuteModerationPlanExcep \ \lor\ ExecuteModerationPlanNoExcep$

The above plan definitions will be used later in the BDI model definitions of the individual agents.

G. Additional Parameters related to Adaptivity

In order to make the VERMILLION extensible, we need to identify the various avionics applications [51] that can be hosted on the framework. Also, we need to identify the various methods of adaptivity and their associated support mechanisms. Learning [52] is one of the powerful ways to engineer adaptivity in agents [53], [54]. Therefore the following sets become essential for VERMILLION strategic agent operations:

$AvionicsApplications ::= Flight\_Management \mid Communication\_Management \mid Collision\_Avoidance \mid \ldots$

$LearningEnsembleMethods ::= State\_Values \mid State\_Action\_Pairs \mid Neural\_Weights \mid \ldots$

LearningEnsembleEntities == {sv1..svn, sap1..sapn, nwt1..nwtn}

LearningAlgorithmLibrary ::= Algorithm_1 | Algorithm_2 | Algorithm_3 | Algorithm_Others

LearningAlgorithmSteps ::= AlgorithmStepSet_1 | AlgorithmStepSet_2 | AlgorithmStepSet_3 | AlgorithmStepSet_Others

LrngEnsmEntsWithValue

lensent : *LearningEnsembleEntities*

lensentval : $\mathbb{R}$

The *LearningEnsembleMethods* abstracts the overall mechanism by which the learning structures will be accomodated and associated with the learning algorithm. *LearningEnsembleMethods* will dictate the actual set of entities in *LearningEnsembleEntities* that form the structure which hold values during the learning process. *LearningAlgorithmLibrary* provides the actual learning algorithm from : Reinforcement, Supervised, and Unsupervised modes of learning while *LearningAlgorithmSteps* abstracts the steps involved when a particular learning algorithm is chosen. We would like to explicitly mention that all of these are not specified using Z schemas. We just list them as abstractions corresponding to actual algorithms that are available. We believe that this suffices for our verification using model checking.

H. Vermillion Strategic Agent (VSA)

- ***Beliefs***: The VSA operationalizes behavior that requires long term strategic response. In the VSA, some percepts from sensors are mapped to strategic beliefs with the function *genrteStrgBeliefs*. The function *mapAvsAppToLrngEsmMethod* is used create the connection between an avionics application and the learning method. In order for the learning mechanism to work, the function *mapStrgBeliefsToLrngEsmEnts* links the strategic beliefs to the actual learning structural entities that will be used by the learning algorithms to mechanize adaptivity via the learning process. A key aspect that may be used is mapping a feature of the environment captured by the beliefs to the learning structural entities. Learning invariably takes place by manipulation of values associated with the learning structural entities. The mapping of these values to the learning structural entities is accomplished by the function *learnByMapngLrngEsmEntsToVal* and the learning process is said to be effected in each cycle when this function is invoked.
- ***Desires***: In order to bound the number of distinct goals that the strategic agent pursues, the function *genrteNofStrgGoals* is used to arrive at the number of goals; this is akin to segregating the different desires at the top level. Since beliefs are already appropriately mapped by the learning structural entities, it suffices to map the beliefs to Goals, with the value attached to these groupings of beliefs, and this is accomplished by *genrteStrgGoalsWithVal* function.

VermillionStrategicAgent

$vsa\_agent\_operating\_phase : \mathbb{P}\, VermillionAgentPhases$
$percepts : \mathbb{P}\, Events; \quad spercepts : \mathbb{P}\, Events$
$env\_features : \mathbb{P}\, EnvironmentFeatures$
$strg\_beliefs : \mathbb{P}\, LearningBelForm$
$adapt\_converge\_belief\_status : \mathbb{P}\, ConvergeStatus$
$sync\_beliefs : \mathbb{P}\, SynchronizationBelForm$
$vsa\_excep\_beliefs : \mathbb{P}\, ExceptionBelForm$
$lrng\_esm\_methods : \mathbb{P}\, LearningEnsembleMethods$
$lrng\_esm\_entities : \mathbb{P}\, LearningEnsembleEntities$
$seqLrnEsSnts : \mathrm{seq}\, LearningEnsembleEntities$
$lrng\_esm\_entities\_with\_val : \mathbb{P}\, LrngEnsmEntsWithValue$
$adapt\_converge\_status : \mathbb{P}\, Boolean$
$curr\_tm : \mathbb{P}\, \mathrm{T}; \quad adpt\_conv\_tst\_tm : \mathbb{P}\, \mathrm{T}$
$adpt\_conv\_tm : \mathbb{P}\, TimeInterval$
$strg\_goal : \mathbb{P}\, StrategicGoal\ strg\_goal\_level : \mathbb{P}\, \mathbb{N}$
$strg\_goal\_with\_value : \mathbb{P}\, StrategicGoalWithValue$
$intention\_strgy : \mathbb{P}\, IntentionMechanism$
$strg\_plan : \mathbb{P}\, StrategicPlan; \quad unhndld\_vsa\_excep : \mathbb{P}\, Boolean$

$spercepts \subset percepts$
$\forall\, sp : spercepts \bullet (\exists\, stb : strg_beliefs \bullet genrteStrgBeliefs(sp) = stb)$
$\forall\, numOfGoals : strg\_goal\_level \bullet$
$(\exists\, stg : strg\_goal \bullet genrteNofStrgGoals(numOfGoals) = stg)$
$\forall\, a : AvionicsApplications \bullet$
$(\exists\, lem : lrng\_esm\_methods \bullet mapAvsAppToLrngEsmMethod(a) = lem)$
$\forall\, stb : \mathrm{seq}\, strg\_beliefs \bullet$
$(\exists\, lee : \mathrm{seq}\, lrng\_esm\_entities \bullet mapStrgBeliefsToLrngEsmEnts(stb) = lee)$
$\forall\, lee : \mathrm{seq}\, lrng\_esm\_entities;\ lem : lrng\_esm\_methods \bullet$
$(\exists\, leev : \mathrm{seq}\, lrng\_esm\_entities\_with\_val \bullet$
$learnByMapngLrngEsmEntsToVal(lee, lem) = leev)$
$\forall\, stg : \mathrm{seq}\, strg\_goal;\ lee : \mathrm{seq}\, lrng\_esm\_entities \bullet$
$(\exists\, stgv : \mathrm{seq}\, strg\_goal\_with\_value \bullet genrteStrgGoalsWithVal(stg, lee) = stgv)$
$\forall\, a : AvionicsApplications \bullet$
$(\exists\, is : intention\_strgy \bullet mapAvsAppToIntentionStrategy(a) = is)$
$\forall\, is : intention\_strgy;\ stgv : strg\_goal\_with\_value;\ synb : sync\_beliefs \bullet$
$(\exists\, stg : strg\_goal \bullet selectIntention(is, stgv, synb) = stg)$
$\forall\, stg : strg\_goal \bullet (\exists\, stp : strg\_plan \bullet selectStrgPlan(stg) = stp)$
$[PREDICATES]$
$\forall\, stb : strg\_beliefs;\ sp : spercepts;\ stgv : \mathrm{seq}\, strg\_goal\_with\_value;$
$sstg : \mathrm{seq}\, strg\_goal;\ lee : \mathrm{seq}\, lrng\_esm_entities;\ sstb : \mathrm{seq}\, strg\_beliefs \bullet$
$stb \in \{genrteStrgBeliefs(sp)\} \Rightarrow stgv \in \{genrteStrgGoalsWithVal(sstg, lee)\}$
$\wedge\ lee \in \{mapStrgBeliefsToLrngEsmEnts(sstb)\} \qquad [SPRD1]$
$\forall\, numOfGoals : strg_goal\_level;\ stg : \mathrm{seq}\, strg\_goal;\ lee : \mathrm{seq}\, lrng\_esm\_entities;$
$stgv : \mathrm{seq}\, strg\_goal\_with\_value \bullet$
$numOfGoals = \#genrteStrgGoalsWithVal(stg, lee) \qquad [SPRD2]$
$\forall\, ct : curr\_tm;\ acbs : adapt\_converge\_belief\_status;$
$actstmSTART : adpt\_conv\_tst\_tm;\ actstmEND : adpt\_conv\_tst\_tm;$
$acchk : adpt\_conv\_tm;\ ac : adapt\_converge\_status \bullet$
$((actstmSTART - actstmEND) \geq (acchk.finish - acchk.begin))$
$\wedge\ (acbs = Converge) \Rightarrow (ac = True) \qquad [SPRD3]$
$\forall\, stp : strg\_plan;\ vb : vsa\_excep\_beliefs \bullet$
$stp.status = SPExcep \Rightarrow SelectSPExcepHdlr = \emptyset \wedge SPExcepPropagate \neq \emptyset [SPRD4]$

- ***Intention***: The function *mapAvsAppToIntentionStrategy* allows flexible intention incorporation within the strategic agent by allowing the mapping of an avionics application to the intention strategy to be put in place. The intention strategy is then used by the *selectIntention* function to choose the goal that the agent will commit to.
- ***Plans***: Once the goal is chosen, the *selectStrgPlan* function selects the plan to be executed. Any exception beliefs that could not be handled within a plan are raised to the agent and recorded within the *vsa_excep_beliefs*. These exceptions are handled by the combined use of the monitors and tactical agents and is described later.

Invariants for VSA The first predicate SPRD1 states that all strategic beliefs will be mapped to strategic goals. The second predicate SPRD2 states that the number of goal levels will always be equal to the number of strategic goals generated. The third predicate SPRD3 states that agent strategy operations will always be tested for convergence. The fourth predicate SPRD4 states that when plans are executed, an unhandled exception will be propagated up to the agent.

I. Vermillion Normal Agent (VNA)

The VNA mechanizes behavior that provides responses catering to normal functions. The VNA operates mainly by generating Agent States using Normal Beliefs. The normal beliefs are obtained from the *generateNormalBeliefs* function, while the *mapNormalBeliefsToStates* function maps those beliefs to states. The State Goal Generation function identified by *mapStatesToNormalGoals* maps the Agent States to Agent Goals while the Normal Goal-Plan Mapping function identified by *selectNormalPlan* maps a goal to a plan as in the case of VSA. In the case of VNA, there may be some goals that require an Intention Selection function *selectForSomeGoalsIntention* that uses the *Intention Change Beliefs* to decide whether the selected goal should be changed; the decision is based on a belief or a combination thereof. Both VSA and VNA consider *sync_beliefs* that will drive their respective Goal Generation functions to produce agent inhibited behaviors when Tactical Beliefs emerge.

Invariants for VNA The first predicate NPRD1 states that all normal beliefs will be mapped to normal goals. The second predicate NPRD2 states that the number of combinations of the beliefs will always equal the number of goals generated. The third predicate NPRD3 states that when plans are executed, an unhandled exception will be propagated up to the agent.

VermillionNormalAgent

$vna\_agent\_operating\_phase : \mathbb{P}\, VermillionAgentPhases$
$percepts : \mathbb{P}\, Events$
$npercepts : \mathbb{P}\, Events$
$normal\_beliefs : \mathbb{P}\, NormBelForm$
$intention\_change\_beliefs : \mathbb{F}\, NormBelForm$
$sync\_beliefs : \mathbb{P}\, SynchronizationBelForm$
$vna\_excep\_beliefs : \mathbb{P}\, ExceptionBelForm$
$normal\_goal : \mathbb{P}\, NormalGoal$
$some\_normal\_goal : \mathbb{F}\, NormalGoal$
$agent\_state : \mathbb{P}\, NormalAgentStates$
$normal\_plan : \mathbb{P}\, NormalPlan$

$npercepts \subset percepts$
$intention\_change\_beliefs \subseteq normal\_beliefs$
$\forall\, np : npercepts \bullet$
$(\exists\, nb : normal\_beliefs \bullet generateNormalBeliefs(np) = nb)$
$\forall\, nb : \mathrm{seq}\, normal\_beliefs \bullet$
$(\exists\, as : agent\_state \bullet mapNormalBeliefsToStates(nb) = as)$
$\forall\, as : agent\_state;\ synb : sync\_beliefs \bullet$
$(\exists\, ng : normal\_goal \bullet mapStatesToNormalGoals(as, synb) = ng)$
$\forall\, icb : intention\_change\_beliefs;\ synb : sync\_beliefs \bullet$
$(\exists\, sng : some\_normal\_goal \bullet selectForSomeGoalsIntention(icb, synb) = sng)$
$\forall\, ng : normal\_goal;\ sng : some\_normal\_goal \bullet$
$(\exists\, np : normal\_plan \bullet selectNormalPlan(ng, sng) = np)$
$[PREDICATES]$
$\forall\, nb : normal\_beliefs;\ np : npercepts;\ nsb : \mathrm{seq}\, normal\_beliefs;$
$as : agent\_state;\ ng : normal\_goal;\ synb : sync\_beliefs \bullet$
$nb \in \{generateNormalBeliefs(np)\} \wedge as$
$\in \{mapNormalBeliefsToStates(nsb)\} \Rightarrow$
$ng \in \{mapStatesToNormalGoals(as, synb)\} \qquad [NPRD1]$
$\forall\, nb : \mathrm{seq}\, normal\_beliefs;\ as : agent\_state;\ synb : sync\_beliefs \bullet$
$numOfStateElements(mapNormalBeliefsToStates(nb)) =$
$numOfNGoalElements(mapStatesToNormalGoals(as, synb)) \ [NPRD2]$
$\forall\, np : normal\_plan;\ vn : vna\_excep\_beliefs \bullet$
$np.status = NPExcep \Rightarrow SelectNPExcepHdlr = \emptyset \wedge$
$NPExcepPropagate \neq \emptyset \qquad [NPRD3]$

J. Vermillion Tactical Agent (VTA)

Tactical Agents cover responses required from a tactical viewpoint - those that require attention from safety, timeliness, and fault response perspectives. Tactical agents need to examine percepts locally (those that may be under the purview of a tactical agent) and those that may manifest in a global fashion (affecting a group of agents or the system as a whole).

VermillionTacticalAgent

$vta\_agent\_operating\_phase : \mathbb{P}\, VermillionAgentPhases$
$percepts : \mathbb{P}\, Events$
$local\_safety\_percepts : \mathbb{P}\, Events$
$global\_safety\_percepts : \mathbb{P}\, Events$
$local\_timeliness\_percepts : \mathbb{P}\, Events$
$global\_timeliness\_percepts : \mathbb{P}\, Events$
$safety\_beliefs : \mathbb{P}\, SafetyBelForm$
$timeliness\_beliefs : \mathbb{P}\, TimelinessBelForm$
$tactical\_beliefs : \mathbb{P}\, TacticalBelForm$
$sync\_beliefs : \mathbb{P}\, SynchronizationBelForm$
$vta\_excep\_beliefs : \mathbb{P}\, ExceptionBelForm$
$exception\_beliefs : \mathbb{P}\, ExceptionBelForm$
$tactical\_goal : \mathbb{P}\, TacticalGoal$
$tactical\_plan : \mathbb{P}\, TacticalPlan$

$local\_safety\_percepts \subset percepts$
$global\_safety\_percepts \subset percepts$
$local\_timeliness\_percepts \subset percepts$
$global\_timeliness\_percepts \subset percepts$
$\forall\, lsp : local\_safety\_percepts;\ gsp : global\_safety\_percepts \bullet$
$(\exists\, sb : \mathrm{seq}\, safety\_beliefs \bullet generateSafetyBeliefs(lsp, gsp) = sb)$
$\forall\, ltp : local\_timeliness\_percepts;\ gtp : global\_timeliness\_percepts \bullet$
$(\exists\, tb : \mathrm{seq}\, timeliness\_beliefs \bullet$
$generateTimelinessBeliefs(ltp, gtp) = tb)$
$\forall\, sb : \mathrm{seq}\, safety\_beliefs;\ tb : \mathrm{seq}\, timeliness\_beliefs \bullet$
$(\exists\, tacb : \mathrm{seq}\, tactical\_beliefs \bullet generateTacticalBeliefs(sb, tb) = tacb)$
$\forall\, tacb : \mathrm{seq}\, tactical\_beliefs;\ synb : \mathrm{seq}\, sync\_beliefs;$
$eb : \mathrm{seq}\, exception\_beliefs \bullet$
$(\exists\, tg : tactical\_goal \bullet$
$generateTactGoalsAndSynBeliefs(tacb, eb) = (tg, synb))$
$\forall\, tg : tactical\_goal \bullet$
$(\exists\, tp : tactical\_plan \bullet selectTacticalPlan(tg) = tp)$
$[PREDICATES]$
$\forall\, tacb : \mathrm{seq}\, tactical\_beliefs;\ sb : \mathrm{seq}\, safety\_beliefs;\ tb : \mathrm{seq}\, timeliness\_beliefs;$
$eb : \mathrm{seq}\, exception\_beliefs;\ tg : tactical\_goal \bullet$
$tacb \in \{generateTacticalBeliefs(sb, tb)\} \Rightarrow$
$tg \in \mathrm{dom}\{generateTactGoalsAndSynBeliefs(tacb, eb)\} \qquad [TPRD1]$
$\forall\, tp : tactical\_plan;\ vt : vta_{e}xcep\_beliefs \bullet$
$tp.status = TPExcep \Rightarrow$
$SelectTPExcepHdlr = \emptyset \wedge TPExcepPropagate \neq \emptyset \qquad [TPRD2]$

These percepts spread across the safety and timeliness categories are mapped to beliefs by the *generateSafetyBeliefs* and *generateTimelinessBeliefs* functions. An example of a local safety percept would be the distance of say a UAV Target with respect to the ownship. The belief so generated will belong to a Collision Avoidance Agent. On the other hand, a low fuel/power condition affecting the continued safe flight of the airborne platform is viewed as a global safety percept, and the belief so generated will affect all agents. The function *generateTacticalBeliefs* examines all safety and timeliness beliefs and generates a prioritized set of beliefs termed the tactical beliefs. The prioritization is necessary to generate tactical goals that have a relative sense of urgency amongst them. From the previous examples cited, the belief generated in context of collision avoidance would have a higher priority for action compared to the low fuel/power condition belief. Since tactical agents are supposed to cover up for fault conditions that are propogated at agent level, the function *generateTactGoalsAndSynBeliefs* examines exception beliefs also to generate goals that will respond to tactical events while simultaneously actioning fault responses. This may involve generating suitable synchronization beliefs that are shared with other agents in appropriate scenarios.The function *selectTacticalPlan* maps a goal to a plan as in the case of VSA and VNA.

Invariants for VTA The first predicate TPRD1 states that all tactical, synchronization, and exception beliefs will be mapped to tactical goals. This is because the VTAs are supposed to handle all exception beliefs raised at the agent levels in all other types of agents. The second predicate TPRD2 states that when plans are executed, an unhandled tactical exception will be propagated up to the scheduler.

K. Vermillion Moderator Agent (VMA)

Moderator agents provide consistency when more than one agent provides a solution to a problem or provide partial solutions to a problem, and those solutions need to be arbitrated or mediated amongst the group, so that it is in general agreement with all the members of the group. The moderator is not needed for every application domain. Moderator Agents map the *moderation_percepts* to *moderation_beliefs* using the *generateModerationBeliefs*. As an example, these percepts can be configuration inputs (say for example negotiation or voting, and the sources to moderate) for the specific avionics application or they can indicate the weightages to be given for outputs from a set of agents to be considered. Therefore the *generateModerationBeliefs* can be designed to generate beliefs which have specific semantics. The function *generateModerationGoalsWtSources* can then arrive at the goals and sources to moderate using the *moderation_beliefs*. As an example, a one-shot negotation between the agent set may be the goal, with the sources for moderation identified. Once the goal is set up, the specifics of the moderation process is governed by the function *selectModerationPlan*. For the negotiation example, this plan could set up the specifics of the moderation and the order of negotiation. With the moderation plan in place, the function *generateModeratedOutputs* produces the outputs from the inputs provided to the moderation process.

Invariants for VMA The predicate MPRD1 stipulates that the moderation should be determinsitic and complete within a finite amount of time. The second predicate MPRD2 states that when plans are executed, an unhandled moderation exception will be propagated up to the agent to be later actioned by an appropriate VTA.

VermillionModeratorAgent

$vma\_agent\_operating\_phase : \mathbb{P}\, VermillionAgentPhases$
$moderation\_percepts : \mathbb{P}\, ModerationTypes$
$moderation\_beliefs : \mathbb{P}\, ModerationBelForm$
$sync\_beliefs : \mathbb{P}\, SynchronizationBelForm$
$modr\_agreemen\_succ\_belief\_status : \mathbb{P}\, Boolean$
$modr\_succ\_status : \mathbb{P}\, Boolean$
$curr\_tm : \mathbb{P}\, \mathrm{T}$
$modr\_succ\_tst\_tm : \mathbb{P}\, \mathrm{T}$
$modr\_succ\_tm : \mathbb{P}\, TimeInterval$
$moderation\_inputs : \mathbb{P}\, Solution$
$moderated\_outputs : \mathbb{P}\, Solution$
$vma\_excep\_beliefs : \mathbb{P}\, ExceptionBelForm$
$moderation\_goal : \mathbb{P}\, ModerationGoal$
$moderation\_sources : \mathbb{P}\, VermillionAgentSet$
$moderation\_plan : \mathbb{P}\, ModerationPlan$

$\forall\, mp : moderation\_percepts \bullet$
$(\exists\, mb : moderation\_beliefs \bullet generateModerationBeliefs(mp) = mb)$
$\forall\, mb : moderation\_beliefs;\ ms : \mathrm{seq}\, moderation\_sources;$
$synb : sync\_beliefs \bullet$
$(\exists\, mog : moderation\_goal \bullet$
$generateModerationGoalsWtSources(mb, synb) = (mog, ms))$
$\forall\, mog : moderation\_goal \bullet$
$(\exists\, mopl : moderation\_plan \bullet selectModerationPlan(mog) = mopl)$
$\forall\, mopl : moderation\_plan;\ mip : moderation\_inputs \bullet$
$(\exists\, mop : moderated\_outputs \bullet$
$generateModeratedOutputs(mopl, mip) = mop)$
$[PREDICATES]$
$\forall\, ct : curr\_tm;\ msb : modr\_agreement\_succ\_belief\_status;$
$mststmSTART : modr\_succ\_tst\_tm;\ mststmEND : modr\_succ\_tst\_tm;$
$mschk : modr\_succ\_tm;\ ms : modr\_succ\_status \bullet$
$((mststmSTART - mststmEND) \geq (mschk.finish - mschk.begin))$
$\wedge\, (msb = True) \Rightarrow (ms = True) \qquad [MPRD1]$
$\forall\, mp : moderation\_plan;\ vmeb : vma\_excep\_beliefs \bullet$
$mp.status = MPExcep \Rightarrow$
$SelectMPExcepHdlr = \emptyset \wedge MPExcepPropagate \neq \emptyset \qquad [MPRD2]$

L. Vermillion Monitor (VMon)

It is imperative that avionics systems monitor the software health. The VERMILLION monitors exist to monitor exceptions propagated in all types of agents: VSAs, VNAs, VMAs, and even VTAs. These are exceptions that could not be handled by the respective agents' plans and hence need external means of support (support external to the agents) and are caught by the *vsa_exception_beliefs*, *vna_exception_beliefs*, *vta_exception_beliefs*, and *vma_exception_beliefs*. There are prevalent conditions wherein the agents are not able to detect that they are faulty and external probing and mitigation is required. These types of errors are caught by the *agent_fault_beliefs* by the function *monitorAgentFaults*. With the complete list of exceptions gathered, the function *mapAllExcpBlfsAndFltsToExcpBlfs* generates the final list of *exception_beliefs*. These *exception_beliefs* are acted upon by the VTAs to provide the appropriate fault responses.

$$
\begin{array}{l}
\textit{VermillionMonitor} \\
\hline
vsa\_excep\_beliefs : \mathbb{P}\, ExceptionBelForm \\
vna\_excep\_beliefs : \mathbb{P}\, ExceptionBelForm \\
vta\_excep\_beliefs : \mathbb{P}\, ExceptionBelForm \\
vma\_excep\_beliefs : \mathbb{P}\, ExceptionBelForm \\
agent\_fault\_beliefs : \mathbb{P}\, AgentFaultBelForm \\
vsas : \mathbb{P}\, VermillionStrategicAgent \\
vnas : \mathbb{P}\, VermillionNormalAgent \\
vtas : \mathbb{P}\, VermillionTacticalAgent \\
vmas : \mathbb{P}\, VermillionModeratorAgent \\
exception\_beliefs : \mathbb{P}\, ExceptionBelForm \\
\hline
\forall\, v : vsas;\ n : vnas;\ t : vtas;\ m : vmas \bullet \\
(\exists\, afb : agent\_fault\_beliefs \bullet monitorAgentFaults(v, n, t, m) = afb) \\
\forall\, vsaeb : vsa\_excep\_beliefs;\ vnaeb : vna\_excep\_beliefs \\
vtaeb : vta\_excep\_beliefs;\ afb : agent\_fault\_beliefs \\
vmaeb : vma\_excep\_beliefs \bullet \\
(\exists\, eb : exception\_beliefs \bullet \\
mapAgExpBlfsNFltsToExBlfs(vsaeb, vnaeb, vtaeb, vmaeb, afb) = eb) \\
\hline
\end{array}
$$

VI. VERMILLION Dynamics

The VERMILLION system operations come alive by an activated scheduler that invokes individual agents in a deterministic fashion. To this extent, the agent autonomy aspect is relaxed since we wish to mitigate problems related to race conditions and deadlocks. All agents will invoke operations following the BDI cycle, where belief generation precedes desire generation and intention selection. However, there are certain agent operations that are applicable in other agent phases. The dynamic specifications provided in this section have been specified and syntax checked using the CZT standalone tool [49].

A. VSA Dynamics

We start by defining the most important operations of the VSA.

genrteStrgBeliefs

$\Delta VermillionStrategicAgent$

$\Xi VermillionTacticalAgent$

$\Xi VermillionNormalAgent$

$\Xi VermillionModeratorAgent$

$strg\_in\_percepts? : \mathbb{P}\, Events$

$mapPerceptsToStrBeliefs : \mathbb{P}\, Events \twoheadrightarrow \mathbb{P}\, LearningBelForm$

$strg\_in\_percepts? \cap npercepts = \emptyset \wedge$

$strg\_in\_percepts? \cap global\_safety\_percepts = \emptyset \wedge$

$strg\_in\_percepts? \cap local\_timeliness\_percepts = \emptyset \wedge$

$strg\_in\_percepts? \cap global\_timeliness\_percepts = \emptyset \wedge$

$getVSAoperatingPhase(vsa\_agent\_operating\_phase) \neq BLOCKED \wedge$

$getVSAoperatingPhase(vsa\_agent\_operating\_phase) = Belief\_Updation$

$strg\_beliefs' = strg\_beliefs \cup mapPerceptsToStrBeliefs(strg\_in\_percepts?)$

$vsa\_agent\_operating\_phase' =$

$\qquad setVSAoperatingPhase(Desire\_Generation)$

The *genrteStrgBeliefs* is operational only when the *agent_operating_phase* indicates that it is time to update the beliefs in the BDI cycle and the VSA has not been blocked by actions of a VTA due to exception handling. The strategic percepts would have already been updated by the sensors in the *Percept_Formation phase*. The strategic percepts are mapped to strategic beliefs by a surjective function.

The *mapStrgBeliefsToLrngEsmEnts* is operational only when the
agent_operating_phase indicates that it is *Agent_Configuration* and can be performed manually. This function maps the strategic beliefs to entities which are appropriate collections of structures that enable learning used for adaptation. A two stage process is employed, wherein in the first stage the raw strategic beliefs are mapped to sequences and then this sequence of beliefs is mapped to the learning structures by a surjective function. The strategic beliefs which map appropriate features in the agents environment are a good choice.

$$
\begin{array}{l}
\textit{mapStrgBeliefsToLrngEsmEnts} \\
\hline
\Delta \textit{VermillionStrategicAgent} \\
\textit{strg\_beliefs}? : \mathbb{P}\, \textit{LearningBelForm} \\
\textit{seqLrnEsSnts} : \mathrm{seq}\, \textit{LearningEnsembleEntities} \\
\textit{maprawStrBelstoSeqStrBels} : \mathbb{P}\, \textit{LearningBelForm} \rightarrow \\
\qquad\qquad\qquad\qquad \mathrm{seq}\, \textit{LearningBelForm} \\
\textit{mapStrBelsLrnEsEnts} : \mathrm{seq}\, \textit{LearningBelForm} \twoheadrightarrow \\
\qquad\qquad\qquad\qquad \mathrm{seq}\, \textit{LearningEnsembleEntities} \\
\hline
\textit{getVSAoperatingPhase}(\textit{vsa\_agent\_operating\_phase}) \neq \textit{BLOCKED} \wedge \\
\textit{getVSAoperatingPhase}(\textit{vsa\_agent\_operating\_phase}) = \\
\qquad\qquad\qquad\qquad \textit{Agent\_Configuration} \\
\textit{seqLrnEsSnts}' = \textit{seqLrnEsSnts} \cup \\
\textit{mapStrBelsLrnEsEnts}(\textit{maprawStrBelstoSeqStrBels}(\textit{strg\_beliefs}?)) \\
\hline
\end{array}
$$

The *learnByMapngLrngEsmEntsToVal* is one of the main operations by which the actual adaptation is obtained. The function is operational during desire generation phase (sub phase 1) when the *agent_operating_phase* indicates *Desire_Generation* and the VSA has not been blocked by actions of a VTA due to exception handling. In the first step a learning algorithm is chosen from a library and mapped to a specific method to mechanize the algorithm. In the second step, the learning algorithmś steps are extracted. The specific method obtained in the first step is then used choose the learning structures. In the last step, the learning algorithm or function operates via the learning steps, on the initial values assigned to the learning structures and generates new values which are again mapped to the learning structures; the values are representative of the adaptation by learning mechanism. The specific learning algorithms are not specified in Z.

$learnByMapngLrngEsmEntsToVal$

$\Delta VermillionStrategicAgent$

$lrng\_algorithm? : \mathbb{P}\, LearningAlgorithmLibrary$

$algo\_steps : LearningAlgorithmSteps$

$selectLrngAlgoAndMapMethod : \mathbb{P}\, LearningAlgorithmLibrary \rightarrow \mathbb{P}\, LearningEnsembleMethods$

$mapLrngAlgoWithAlgoSteps : \mathbb{P}\, LearningAlgorithmLibrary \rightarrow LearningAlgorithmSteps$

$selectLrngEsmEntsForLrngEsmMethod : \mathbb{P}\, LearningEnsembleMethods \rightarrow \mathbb{P}\, LearningEnsembleEntities$

$updateLrngEsmEntsWithValUseAlgoSteps : LearningAlgorithmSteps \times \mathbb{P}\, LearningEnsembleEntities \rightarrow \mathbb{P}\, \mathrm{seq}\, LrngEnsmEntsWithValue$

$getVSAoperatingPhase(vsa\_agent\_operating\_phase) \neq BLOCKED \land$

$getVSAoperatingPhase(vsa\_agent\_operating\_phase) = Desire\_Generation$

$lrng\_esm\_methods' = selectLrngAlgoAndMapMethod(lrng\_algorithm?)$

$algo\_steps = mapLrngAlgoWithAlgoSteps(lrng\_algorithm?)$

$lrng\_esm\_entities' = selectLrngEsmEntsForLrngEsmMethod(lrng\_esm\_methods)$

$\mathrm{seq}\, lrng\_esm\_entities\_with\_val' = updateLrngEsmEntsWithValUseAlgoSteps(algo\_steps, lrng\_esm\_entities)$

The *lrng_esm_entities* which have captured the necesssary information for adaptation are stringed to the appropriate goals using the *genrteStrgGoalsWithVal* function. This function is operational during desire generation phase (sub phase 2) when the *agent_operating_phase* indicates *Desire_Generation* and the VSA has not been blocked by actions of a VTA due to exception handling. With this function, goals are now mapped to a value and are available for further processing.

genrteStrgGoalsWithVal

$\Delta VermillionStrategicAgent$

$\Delta StrategicGoalWithValue$

$strg\_goal? : \mathbb{P}\, StrategicGoal$

$tempStrgGoalwithVal : \mathbb{P}\, StrategicGoalWithValue$

$lrng\_esm\_entities\_with\_val : \mathbb{P}\, LrngEnsmEntsWithValue$

$mapInputStrgGoal : StrategicGoal \twoheadrightarrow \mathrm{seq}\, StrategicGoalWithValue$

$mapInputLrnEsEnts : \mathrm{seq}\, LrngEnsmEntsWithValue \twoheadrightarrow$

$\qquad \mathrm{seq}\, StrategicGoalWithValue$

$getVSAoperatingPhase(vsa\_agent\_operating\_phase) \neq BLOCKED \land$

$getVSAoperatingPhase(vsa\_agent\_operating\_phase) =$

$\qquad Desire\_Generation$

$\forall\, s : strg\_goal? \bullet$

$(\exists\, t : \mathrm{seq}\, tempStrgGoalwithVal \bullet mapInputStrgGoal(s) = t)$

$\forall\, l : \mathrm{seq}\, lrng\_esm\_entities\_with\_val \bullet$

$(\exists\, t : \mathrm{seq}\, tempStrgGoalwithVal \bullet mapInputLrnEsEnts(l) = t)$

$strg\_goal\_with\_value' = tempStrgGoalwithVal$

$vsa\_agent\_operating\_phase' =$

$\qquad setVSAoperatingPhase(Intention\_Selection)$

The *selectIntention* forms the last part of the BDI cycle before the plan attached to the goal is dispatched for execution. This function is operational after desire generation phase when the *agent_operating_phase* indicates *Intention_Selection* and the VSA has not been blocked by actions of a VTA due to exception handling. The function uses a particular intention strategy to choose the goal that will become the intention.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\, \mathit{selectIntention}\, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta \mathit{VermillionStrategicAgent} \\
\mathit{intention\_strgy} : \mathbb{P}\, \mathit{IntentionMechanism} \\
\mathit{strg\_goals}? : \mathbb{P}\, \mathrm{seq}\, \mathit{StrategicGoal} \\
\mathit{sync\_beliefs}? : \mathbb{P}\, \mathit{SynchronizationBelForm} \\
\mathit{selectGoalBasedOnIntStrgy} : \mathbb{P}\, \mathrm{seq}\, \mathit{StrategicGoal} \nrightarrow \mathbb{P}\, \mathit{StrategicGoal} \\
\hline
\mathit{getVSAoperatingPhase}(\mathit{vsa\_agent\_operating\_phase}) \neq \mathit{BLOCKED} \wedge \\
\mathit{getVSAoperatingPhase}(\mathit{vsa\_agent\_operating\_phase}) = \\
\qquad\qquad\qquad\qquad \mathit{Intention\_Selection} \\
\mathit{sync\_beliefs}? \in \emptyset \Rightarrow \mathit{strg\_goal}' = \\
\qquad\qquad \mathit{selectGoalBasedOnIntStrgy}(\mathit{strg\_goals}?) \\
\mathit{vsa\_agent\_operating\_phase}' = \\
\qquad\qquad \mathit{setVSAoperatingPhase}(\mathit{Plan\_Execution}) \\
\hline
\end{array}
$$

The function *selectStrgPlan* is very straightforward and is not provided here.

B. VNA Dynamics

Now, looking up the VNA, the behavior of function *generateNormalBeliefs* is very similar to its counterpart *genrteStrgBeliefs* of VSA, and therefore we do not delinate it, but instead provide the important functions *mapNormal-BeliefsToStates* and *mapStatesToNormalGoals* of VNA as follows:

mapNormalBeliefsToStates

$$\Delta VermillionNormalAgent$$
$$normal\_beliefs? : \mathbb{P}\,\mathrm{seq}\, NormBelForm$$
$$mapNBelstoNStates : \mathbb{P}\,\mathrm{seq}\, NormBelForm \twoheadrightarrow \mathbb{P}\, NormalAgentStates$$

$$getVNAoperatingPhase(vna\_agent\_operating\_phase) \neq BLOCKED \land$$
$$getVNAoperatingPhase(vna\_agent\_operating\_phase) = Desire\_Generation$$
$$agent\_state' = mapNBelstoNStates(normal\_beliefs?)$$
$$vna\_agent\_operating\_phase' = setVNAoperatingPhase(Desire\_Generation)$$

mapStatesToNormalGoals

$$\Delta VermillionNormalAgent$$
$$agent\_state? : \mathbb{P}\, NormalAgentStates$$
$$sync\_beliefs? : \mathbb{P}\, SynchronizationBelForm$$
$$mapStateToGoal : \mathbb{P}\, NormalAgentStates \twoheadrightarrow NormalGoal$$

$$getVNAoperatingPhase(vna\_agent\_operating\_phase) \neq BLOCKED \land$$
$$getVNAoperatingPhase(vna\_agent\_operating\_phase) = Desire\_Generation$$
$$\forall\, vnas : agent\_state? \mid sync\_beliefs? = \emptyset \bullet$$
$$(\exists\, ngwosynbs : normal\_goal' \bullet mapStateToGoal(agent\_state?) = ngwosynbs)$$
$$\forall\, vnas : agent\_state? \mid sync\_beliefs? \neq \emptyset \bullet$$
$$(\exists\, ngwsynbs : normal\_goal' \bullet ngwsynbs = \emptyset)$$
$$vna\_agent\_operating\_phase' = setVNAoperatingPhase(Intention\_Selection)$$

while the *mapNormalBeliefsToStates* function is self-explanatory, the function *mapStatesToNormalGoals* examines whether there are any synchronization beliefs and if there are, then there is no normal goal generated, since there are tactical goals to be generated by the VTA which have to be acted upon more urgently. If there are no synchronization beliefs, the function uses *mapStateToGoal* to map a distinct state to a goal. The *selectNormalPlan* function is similar to selectStrategicPlan and is not provided here to for succinctness.

C. VTA Dynamics

We now turn our attention to the VTA and provide the functions *generateTacticalBeliefs* and *generateTactGoalsAndSynBeliefs*.

generateTacticalBeliefs

$\Delta \mathit{VermillionTacticalAgent}$

$\mathit{safety\_beliefs}? : \mathbb{P}\, \mathit{SafetyBelForm}$

$\mathit{timeliness\_beliefs}? : \mathbb{P}\, \mathit{TimelinessBelForm}$

$\mathit{getVTAoperatingPhase}(\mathit{vta\_agent\_operating\_phase}) \neq \mathit{BLOCKED} \wedge$

$\mathit{getVTAoperatingPhase}(\mathit{vta\_agent\_operating\_phase}) =$
$\qquad \mathit{Belief\_Updation}$

$\mathit{vta\_agent\_operating\_phase}' =$
$\qquad \mathit{setVTAoperatingPhase}(\mathit{Desire\_Generation})$

generateTactGoalsAndSynBeliefs

$$\Delta VermillionTacticalAgent$$
$$tactical\_beliefs? : \mathbb{P}\,\mathrm{seq}\,TacticalBelForm$$
$$exception\_beliefs? : \mathbb{P}\,\mathrm{seq}\,ExceptionBelForm$$
$$sync\_beliefs : \mathbb{P}\,SynchronizationBelForm$$
$$mapInputTactBelsToGoal : \mathbb{P}\,\mathrm{seq}\,TacticalBelForm \times \mathbb{P}\,\mathrm{seq}\,ExceptionBelForm \twoheadrightarrow \mathbb{P}\,TacticalGoal$$
$$generateSyncBels : \mathbb{P}\,TacticalGoal \rightarrow \mathbb{P}\,SynchronizationBelForm$$

$$getVTAoperatingPhase(vta\_agent\_operating\_phase) \neq BLOCKED \land$$
$$getVTAoperatingPhase(vta\_agent\_operating\_phase) = Desire\_Generation$$
$$tactical\_goal' = mapInputTactBelsToGoal(tactical\_beliefs?, exception\_beliefs?)$$
$$sync\_beliefs' = generateSyncBels(tactical\_goal)$$
$$vta\_agent\_operating\_phase' = setVTAoperatingPhase(Plan\_Execution)$$

The former function combines the safety and timeliness beliefs which are themselves prioritized, to produce a final prioritized set of tactical beliefs. The latter function generates the tactical goals which may be enqueued if necessary. Tactical goals generation involves acting on both tactical beliefs which the VTA traps along with exception beliefs provided by the VMon.

D. VMA Dynamics

The most important function within the VMA is the *generateModerationGoalsWtSources* which operates in the *Desire_Generation* phase. This function chooses the appropriate moderation goal based on the moderation beliefs; if there are synchronization beliefs, the moderator agent elects to supress the goal generation in order to allow the VTAs to perform the more urgent and necessary actions. For a dependable system, the moderator can act as a voter and this is discussed in the next subsection. Therefore, the type of goal and sources for moderation become important. We do not describe the *generateModeratedOutputs* function since it is straightforward and similar to its counterparts in other agents. The inputs to the moderation process may be partial solutions generated by other agents.

generateModerationGoalsWtSources

$\Delta VermillionModeratorAgent$

$moderation\_beliefs? : \mathbb{P}\, ModerationBelForm$

$sync\_beliefs? : \mathbb{P}\, SynchronizationBelForm$

$moderation\_goal : \mathbb{P}\, ModerationGoal$

$mapModBelsToGoal : \mathbb{P}\, ModerationBelForm \nrightarrow ModerationGoal$

$mapModBelsToSources : \mathbb{P}\, ModerationBelForm \nrightarrow$

$\qquad \mathrm{seq}\, VermillionAgentSet$

$getVMAoperatingPhase(vma\_agent\_operating\_phase) \neq BLOCKED \wedge$

$getVMAoperatingPhase(vma\_agent\_operating\_phase) =$

$\qquad Intention\_Selection$

$\forall\, vmabls : moderation\_beliefs? \mid sync\_beliefs? = \emptyset \bullet$

$(\exists\, mgwosynbs : moderation\_goal' \bullet$

$\qquad mapModBelsToGoal(moderation\_beliefs?) = mgwosynbs)$

$\forall\, vmabls : moderation\_beliefs? \mid sync\_beliefs? = \emptyset \bullet$

$(\exists\, mswosynbs : \mathrm{seq}\, moderation\_sources' \bullet$

$\qquad mapModBelsToSources(moderation\_beliefs?) = mswosynbs)$

$\forall\, vmabls : moderation\_beliefs? \mid sync\_beliefs? \neq \emptyset \bullet$

$(\exists\, mgwsynbs : moderation\_goal' \bullet mgwsynbs = \emptyset)$

$vma\_agent\_operating\_phase' =$

$\qquad setVMAoperatingPhase(Plan\_Execution)$

We donot decsribe the *monitorAgentFaults* and *mapAllExcpBlfsAndFltsToExcpBlfs* functions of the VMon since they are simple and provide a straightforward mapping of their input elements to their corresponding output elements.

E. Mitigating Byzantine Failures

A dependable system must handle byzantine failures [55]. These failures are a result of Byzantine Faults that are manifested in the system when there are distributed elements in the system which need to have agreement to solve a common problem. Since VERMILLION is targeted for dependable general class of avionics systems, it needs to address means of mitigating byzantine failures [30].

Byzantine fault Is any fault presenting different symptoms to different observers [56].

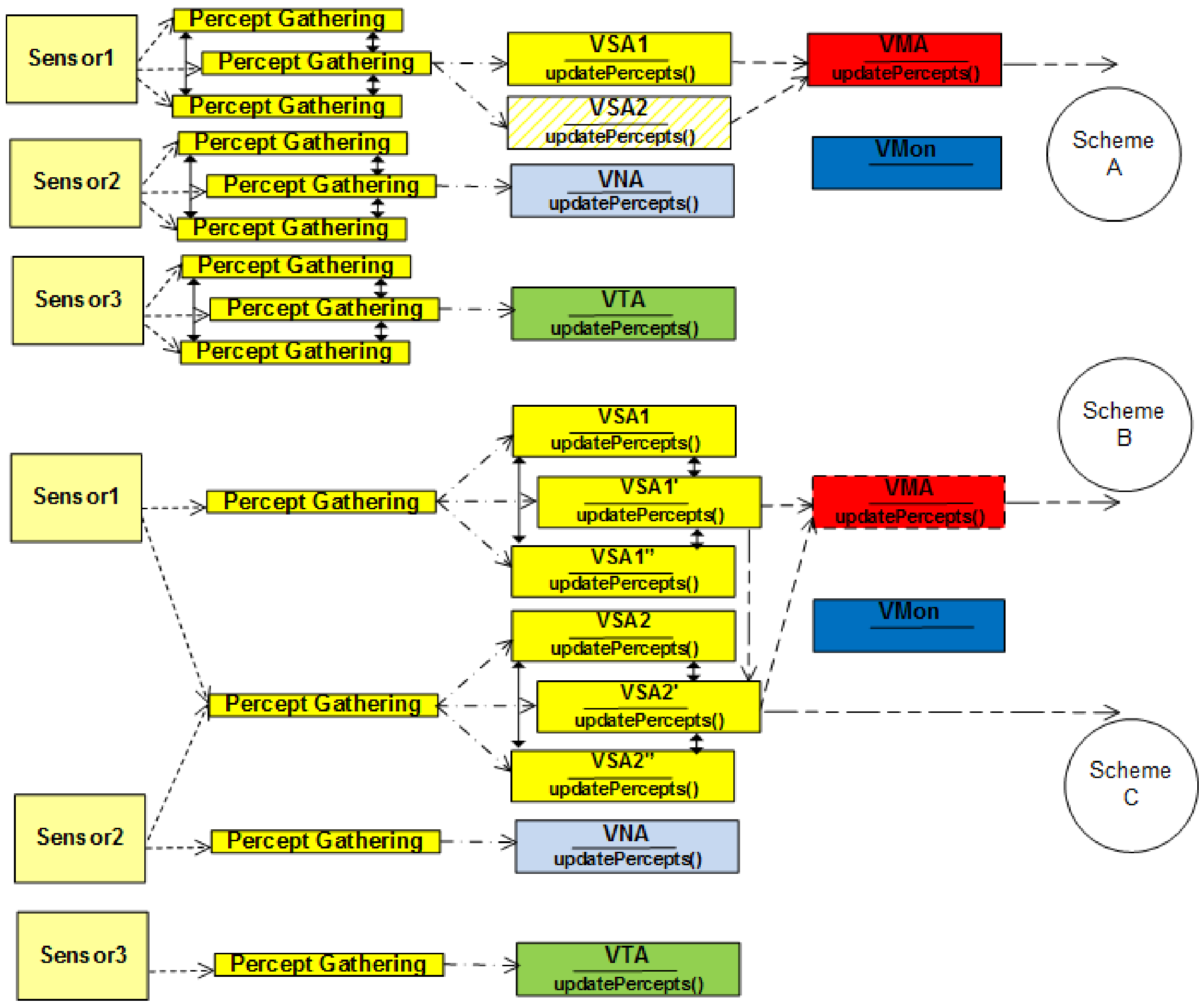


Fig. 6 Byzantine Fault Resilience within the VERMILLION framework

Byzantine failure Is the loss of a system service due to a Byzantine fault in systems that require consensus[56].

The traditional method of handling byzantine failures is by replication of the processing units and passing multiple messages between them. A system is Byzantine failure tolerant for ’m’ faults when the following requirements are satisfied:

- There are at least 3m + 1 processing nodes
- m + 1 rounds of messages are exchanged between them
- Processing nodes are synchronized within a known skew

If however, additional considerations of signed messages being exchanged and verification of the signed messages is performed by the participating entities, then first requirement in the above list reduces to [30]:

- There are at least 2m + 1 processing nodes

We first assume that this system tolerates 1 Byzantine fault per agent focus. Handling additional faults can be done but this can become prohibitively expensive. There are three cases to consider, depending on where and how the byzantine faults appear in VERMILLION. However since byzantine faults are 'consensus requirement' dependent, the major effect is on the VSAs involved in consensus building. This is better explained with the help of a diagram. Shown in Fig. 6 are architectural locations where Byzantine faults can appear. In the top portion of the figure (Scheme A)is shown the interface between the sensors or input devices and the function which updates the percepts in VSAs, VNAs, and VTAs. These faults are mitigated by replicating the percept gathering units and satisfying the message passing requirements provided above. In the middle portion of Fig. 6 is shown the case (Scheme B) where one VSA is producing a byzantine fault and a VMA acts as a voter (assumed byzantine fault free) and this necessitates replication of the VSAs: an underlying assumption is that there are two VSAs (VSA1 and VSA2) providing adaptation for two different system level functions. Notice that the replicated agents are the single and double dashed versions. In the bottom portion of the Fig. 6 is shown the case (Scheme C) where one VSA is producing a byzantine fault and one of the VSAs produces the final system level output. This case also necessitates replication of the VSAs except that the moderator agent is not included.

F. The Control Loops and the Vermillion Scheduler

All Avionics systems are basically part of a bigger aircraft control system. The avionics systems are associated with multiple smaller control systems [57] [58] that function seamlessly within the aircraft control system. Therefore, we now delineate the agent existence and functioning within these constituent control systems. The control systems are analogous to the control loops that operate them. Figure 7 shows the architecture of a generic control loop arrangement which is used in the VERMILLION framework. Our description and Z-specification of the control loops is along the lines of a scheduler proposed in [59]. The specifications provided in this section have been verified and syntax checked using the CZT tool [60].

The Nested Control Loops Each of the loops indicated in the figure above is specified by a schema defining a "Time Period" for the loop , a "Computation Time" allocated for the loop and also a "Deadline" within which the loop components have to complete execution. An important characteristic of the loops is that the middle and outer loop periods will be a simple multiple of the innermost loop period.

InnerFlightControlLoopTimes The innermost loop is used for basic guidance and motion control of the aircraft. This loop mechanizes the physical laws and aerodynamic constraints. We require that all computations in this loop to be less than or equal to the deadline constraint which in turn needs to be less than or equal to the period of this loop.

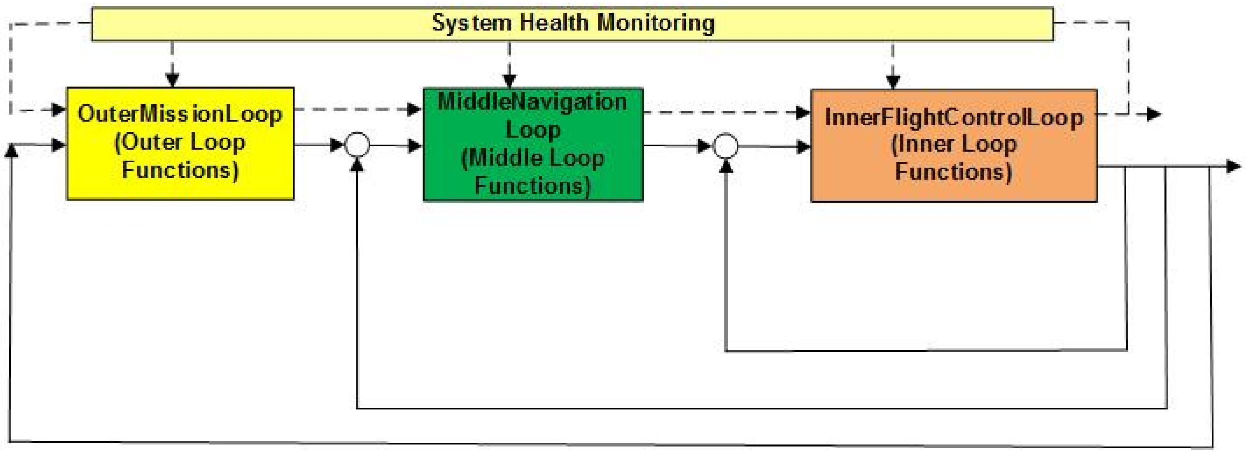

System Health Monitoring
OuterMissionLoop (Outer Loop Functions)
MiddleNavigation Loop (Middle Loop Functions)
InnerFlightControlLoop (Inner Loop Functions)

Fig. 7 The Avionics Control Loops embedding the VERMILLION framework

InnerFlightControlLoopTimes

TimeInterval

t_i : T	[*period*]
c_i : T	[*computation time*]
d_i : T	[*deadline*]

$0 < c_i \leq d_i \leq t_i$

MiddleNavigationLoopTimes The middle loop exists to manage navigation of the aircraft, flight plan, and threat avoidance. This loop mechanizes the physical laws and aerodynamic constraints. Again, we require that all computations in this loop be less than or equal to the deadline constraint which in turn needs to be less than or equal to the period of the loop.

MiddleNavigationLoopTimes

InnerFlightControlLoopTimes

TimeInterval

$n : \mathbb{Z}$ [*Cycle Multiplier*]

$t_m : \mathrm{T}$ [*period*]

$c_m : \mathrm{T}$ [*computation time*]

$d_m : \mathrm{T}$ [*deadline*]

$t_m = n * (t_i)$

$0 < c_m \leq d_m \leq t_m$

OuterMissionLoopTimes The outer loop, exists to manage overall mission and the payloads. Again, we require that all computations in this loop be less than or equal to the deadline constraint which in turn needs to be less than or equal to the period of the loop.

OuterMissionLoopTimes

InnerFlightControlLoopTimes

TimeInterval

$p : \mathbb{Z}$ [*Cycle Multiplier*]

$t_o : \mathrm{T}$ [*period*]

$c_o : \mathrm{T}$ [*computation time*]

$d_o : \mathrm{T}$ [*deadline*]

$t_o = p * (t_i)$

$0 < c_o \leq d_o \leq t_o$

The Loop Processes Each of the loops have loop processes defined. These are "Vehicles" that are representatives of the various loops and perform the actual pieces of execution. The loop processes will embody the various agents defined as threads. The various loop processes define functions that allocate different types of agents (strategic, normal, and tactical) and monitors to the loops via a process. First, we define each agent thread's timeliness and storage criteria by a schema, second we define the schema for the group of threads and finally, we define the various loop processes.

AgentCarrierThreadTimesStorage

TimeInterval

$t_{pc} : \mathrm{T}$ [*period*]

$c_{pc} : \mathrm{T}$ [*computation time*]

$d_{pc} : \mathrm{T}$ [*deadline*]

$ag\_memory\_size : \mathbb{Z}$ [*Agent Thread Memory size*]

$0 < c_{pc} \leq d_{pc} \leq t_{pc}$

AgentCarrierThreadsSet

agents : *VermillionAgentsEntities*

VermillionInnerLoopProcess The inner loop allocation function maps only Vermillion Tactical Agents and Vermillion monitors to the InnerFlightControlLoop via the AgentCarrierProcesses. This is because the agents within this loop need to provide hard real time response to system events. The state variable *sum_thr_times* contains the result of the sum of computation times of each *VermillionAgentsEntities*. Each agent or a monitor is allocated to a *inner_thread_set* by an *innerThreadAllocation* function. The partial function *ILThreadSchedule* maps an *AgentCarrierThreadsSet* to an *AgentCarrierThreadTimesStorage* giving us the base numerical values of various time elements for each agent or entity that form pieces of the *inner_thread_set*. Therefore, the function *computeSumofThreadComputationTimes* operates on each of the agents within the *inner_thread_set* (the domain of the function *ILThreadSchedule*) to obtain the inner loop computation time, which will be used later in the *VermillionSchedule* (defined later in this section). The first invariant ILPRD1, related to non pre-emption of an agent within the inner loop process is stated using the range of the function *ILThreadSchedule* which defines the *AgentCarrierThreadTimesStorage* and in each such occurrence, explicitly requiring that the computation time between successive agents or monitors have atleast 1 unit of time and also ensuring that there is no other agent's start time which occurs before the current agent's finish time. The second invariant ILPRD2, related to Non-colliding agents is stated by requiring that the intervals assigned to each agent donot overlap. In order to do this, the range of the function *ILThreadSchedule* is examined and with each successive occurences of *AgentCarrierThreadTimesStorage*, a condition is imposed that these successive intervals do not have anything in common. If within the inner loop, there is a requirement to force execution of agents in an order (say an engine failure processing needs precedence over collision avoidance), then this can be stated using the precedence relationship invariant, ILPRD3. This is a simple requirement stating that the finish time of an agent be less than the start time of a succeeding agent.

VermillionInnerLoopProcess

$st\_tm : \mathrm{T}$
$ilp\_memory\_size : \mathbb{Z} \qquad [Loop\ Process\ memory\ size]$
$inner\_thread\_set : \mathbb{P}\, AgentCarrierThreadsSet$
$sum\_thr\_times : \mathrm{T}$
$vta? : VermillionAgentsEntities$
$vmon? : VermillionAgentsEntities$
$ILThreadSchedule : AgentCarrierThreadsSet \rightarrowtail AgentCarrierThreadTimesStorage$
$innerThreadAllocation : VermillionAgentsEntities \times VermillionAgentsEntities \rightarrowtail \mathbb{P}\, AgentCarrierThreadsSet$
$computeSumofThreadComputationTimes : \mathrm{T} \rightarrow \mathrm{T}$

$inner\_thread\_set = innerThreadAllocation(vta?, vmon?)$
$\mathrm{dom}\, ILThreadSchedule = inner\_thread\_set$
$\forall\, ag\_thrs : inner\_thread\_set \bullet$
$\forall\, ag\_thr\_tm\_stg : AgentCarrierThreadTimesStorage \bullet$
$sum\_thr\_times =$
$computeSumofThreadComputationTimes(ag\_thr\_tm\_stg.c_{pc})$
$\qquad [Sum\ of\ all\ Individual\ Agent\ computation\ times]$
$\forall\, ag\_thrs : inner\_thread\_set \bullet$
$\forall\, ag\_thr\_tm\_stg : \mathrm{ran}\, ILThreadSchedule \bullet$
$ag\_thr\_tm\_stg.finish - ag\_thr\_tm\_stg.begin + 1 = ag\_thr\_tm\_stg.c_{pc}$
$\qquad [ILPRD1 - Non\ Pre - emption\ of\ an\ agent]$
$\forall\, ag\_thr\_1 : inner\_thread\_set;\ ag\_thr\_2 : inner\_thread\_set \bullet$
$\forall\, ag\_thr\_1\_intv : \mathrm{ran}\, ILThreadSchedule;$
$ag\_thr\_2\_intv : \mathrm{ran}\, ILThreadSchedule \bullet$
$ag\_thr\_1\_intv.interval \cap ag\_thr\_2\_intv.interval = \emptyset$
$\qquad [ILPRD2 - Non\ Collission\ of\ Agents]$
$\forall\, ag\_thr\_1 : inner\_thread\_set;\ ag\_thr\_2 : inner\_thread\_set \bullet$
$\forall\, ag\_thr\_1\_stentimes : \mathrm{ran}\, ILThreadSchedule;$
$ag\_thr\_2\_stentimes : \mathrm{ran}\, ILThreadSchedule \bullet$
$ag\_thr\_1\_stentimes.finish \leq ag\_thr\_2\_stentimes.begin$
$\qquad [ILPRD3 - Precedence\ of\ an\ Agent]$

VermillionMiddleLoopProcess The middle loop allocation function maps Vermillion Strategic Agents, Vermillion Normal Agents, and Vermillion monitors to the MiddleNavigationLoop via the AgentCarrierProcesses. Strategic Agents whose learning rates are faster and which need to learn mission constraints are allocated to this loop. Most Normal Agents which perform bulk of the mission specific activities belong to this loop. The Moderator Agent may also be part of this loop depending on how fast or slow the overall solution set is required. The invariants MLPRD1, MLPRD2, MLPRD3 are similar to their counterparts ILPRD1, ILPRD2, ILPRD3 in the *VermillionInnerLoopProcess* and have the same semantics. The invariant MLPRD4 states that within the *middle_thread_set*, mutual exclusion between a pair of agents is achieved when the intervals of any two agents under consideration do not overlap, i.e., the begin and finish times are apart.

$$
\begin{array}{l}
\textit{VermillionMiddleLoopProcess} \\
\hline
mlp\_memory\_size : \mathbb{Z} \qquad [\textit{Loop Process memory size}] \\
middle\_thread\_set : \mathbb{P}\, AgentCarrierThreadsSet \\
sum\_thr\_times : \mathrm{T} \\
vsa? : VermillionAgentsEntities \\
vna? : VermillionAgentsEntities \\
vma? : VermillionAgentsEntities \\
vmon? : VermillionAgentsEntities \\
MLThreadSchedule : AgentCarrierThreadsSet \mathrel{+\!\!\!\!\!\rightarrow} \\
\qquad AgentCarrierThreadTimesStorage \\
middleThreadAllocation : VermillionAgentsEntities \times \\
\qquad VermillionAgentsEntities \times VermillionAgentsEntities \times \\
\qquad VermillionAgentsEntities \mathrel{+\!\!\!\!\!\rightarrow} \mathbb{P}\, AgentCarrierThreadsSet \\
computeSumofThreadComputationTimes : \mathrm{T} \rightarrow \mathrm{T} \\
\hline
middle\_thread\_set = \\
\quad middleThreadAllocation(vsa?, vna?, vma?, vmon?) \\
\operatorname{dom} MLThreadSchedule = middle\_thread\_set \\
\forall ag\_thrs : middle\_thread\_set \bullet \\
\forall ag\_thr\_tm\_stg : AgentCarrierThreadTimesStorage \bullet \\
sum\_thr\_times = \\
\quad computeSumofThreadComputationTimes(ag\_thr\_tm\_stg.c_{pc}) \\
\qquad\qquad [\textit{Sum of all Individual Agent computation}] \\
\forall ag\_thrs : middle\_thread\_set \bullet \\
\forall ag\_thr\_tm\_stg : \operatorname{ran} MLThreadSchedule \bullet \\
ag\_thr\_tm\_stg.finish - ag\_thr\_tm_stg.begin + 1 = ag\_thr\_tm\_stg.c_{pc} \\
\qquad\qquad [MLPRD1 - Non\ Pre - emption\ of\ an\ agent] \\
\forall ag\_thr\_1 : middle\_thread\_set;\ ag\_thr\_2 : middle\_thread\_set \bullet \\
\forall ag\_thr\_1\_intv : \operatorname{ran} MLThreadSchedule; \\
ag\_thr\_2\_intv : \operatorname{ran} MLThreadSchedule \bullet \\
ag\_thr\_1\_intv.interval \cap ag\_thr\_2\_intv.interval = \emptyset \\
\qquad\qquad [MLPRD2 - Non\ Collission\ of\ Agents] \\
\forall ag\_thr\_1 : middle\_thread\_set;\ ag\_thr\_2 : middle\_thread\_set \bullet \\
\forall ag\_thr\_1\_stentimes : \operatorname{ran} MLThreadSchedule; \\
ag\_thr\_2\_stentimes : \operatorname{ran} MLThreadSchedule \bullet \\
ag\_thr\_1\_stentimes.finish \leq ag\_thr\_2\_stentimes.begin \\
\qquad\qquad [MLPRD3 - Precedence\ of\ an\ Agent] \\
\forall ag\_thr\_1 : middle\_thread\_set;\ ag\_thr\_2 : middle\_thread\_set \bullet \\
\forall ag\_thr\_1\_stentimes : \operatorname{ran} MLThreadSchedule; \\
ag\_thr\_2\_stentimes : \operatorname{ran} MLThreadSchedule \bullet \\
ag\_thr\_1\_stentimes.finish \leq ag\_thr\_2\_stentimes.begin \vee \\
ag\_thr\_2\_stentimes.finish \leq ag\_thr\_1\_stentimes.begin \\
\qquad\qquad [MLPRD4 - Mutual\ Exclusion\ of\ Agents] \\
\hline
\end{array}
$$

VermillionOuterLoopProcess The outer loop allocation function maps Vermillion Strategic Agents, Vermillion Normal Agents, and Vermillion monitors to the OuterMissionLoop via the AgentCarrierProcesses. The Strategic

Agents are those which have learning rates that are matched to the slowest loop rates. The Normal Agents in this loop address overall mission goals and perform payload management. The Moderator Agent can also be part of this loop. The invariants OLPRD1, OLPRD2, OLPRD3, and OLDPRD4 are similar to their counterparts MLPRD1, MLPRD2, MLPRD3, MLPRD4 in the *VermillionMiddleLoopProcess* and have the same semantics.

VermillionOuterLoopProcess

$olp\_memory\_size : \mathbb{Z} \qquad [Loop\ Process\ memory\ size]$
$outer\_thread\_set : \mathbb{P}\, AgentCarrierThreadsSet$
$sum\_thr\_times : \mathrm{T}$
$vsa? : VermillionAgentsEntities$
$vna? : VermillionAgentsEntities$
$vma? : VermillionAgentsEntities$
$vmon? : VermillionAgentsEntities$
$OLThreadSchedule : AgentCarrierThreadsSet \pfun$
$\qquad AgentCarrierThreadTimesStorage$
$outerThreadAllocation : VermillionAgentsEntities \times$
$VermillionAgentsEntities \times VermillionAgentsEntities \times$
$VermillionAgentsEntities \pfun \mathbb{P}\, AgentCarrierThreadsSet$
$computeSumofThreadComputationTimes : \mathrm{T} \rightarrow \mathrm{T}$

$outer\_thread\_set = outerThreadAllocation(vsa?, vna?, vma?, vmon?)$
$\mathrm{dom}\, OLThreadSchedule = outer\_thread\_set$
$\forall ag\_thrs : outer\_thread\_set \bullet$
$\forall ag\_thr\_tm\_stg : AgentCarrierThreadTimesStorage \bullet$
$sum\_thr\_times =$
$\qquad computeSumofThreadComputationTimes(ag\_thr\_tm\_stg.c_{pc})$
$\qquad\qquad [Sum\ of\ all\ Individual\ Agent\ computation]$
$\forall ag\_thrs : outer\_thread\_set \bullet$
$\forall ag\_thr\_tm\_stg : \mathrm{ran}\, OLThreadSchedule \bullet$
$ag\_thr\_tm\_stg.finish - ag\_thr\_tm\_stg.begin + 1 = ag\_thr\_tm\_stg.c_{pc}$
$\qquad\qquad [OLPRD1 - Non\ Pre-emption\ of\ an\ agent]$
$\forall ag\_thr\_1 : outer\_thread\_set;\ ag\_thr\_2 : outer\_thread\_set \bullet$
$\forall ag\_thr\_1\_intv : \mathrm{ran}\, OLThreadSchedule;$
$ag\_thr\_2\_intv : \mathrm{ran}\, OLThreadSchedule \bullet$
$ag\_thr\_1\_intv.interval \cap ag\_thr\_2\_intv.interval = \emptyset$
$\qquad\qquad [OLPRD2 - Non\ Collission\ of\ Agents]$
$\forall ag\_thr\_1 : outer\_thread\_set;\ ag\_thr\_2 : outer\_thread\_set \bullet$
$\forall ag\_thr\_1\_stentimes : \mathrm{ran}\, OLThreadSchedule;$
$ag\_thr\_2\_stentimes : \mathrm{ran}\, OLThreadSchedule \bullet$
$ag\_thr\_1\_stentimes.finish \leq ag\_thr\_2\_stentimes.begin$
$\qquad\qquad [OLPRD3 - Precedence\ of\ an\ Agent]$
$\forall ag\_thr\_1 : outer\_thread\_set;\ ag\_thr\_2 : outer\_thread\_set \bullet$
$\forall ag\_thr\_1\_stentimes : \mathrm{ran}\, OLThreadSchedule;$
$ag\_thr\_2\_stentimes : \mathrm{ran}\, OLThreadSchedule \bullet$
$ag\_thr\_1\_stentimes.finish \leq ag\_thr\_2\_stentimes.begin \vee$
$ag\_thr\_2\_stentimes.finish \leq ag\_thr\_1\_stentimes.begin$
$\qquad\qquad [OLPRD4 - Mutual\ Exclusion\ of\ Agents]$

Therefore, we can define the VermillionProcess as:

$$VermillionProcess \mathrel{\widehat=} VermillionInnerLoopProcess \cup VermillionMiddleLoopProcess \cup VermillionOuterLoopProcess$$

The Vermillion Scheduler consists of mapping of the set of processes that contain all the four different types of agents along with the monitors to the three control loops. This mapping is obtained via the Loop Process Schedule functions. The scheduler provides for specification of different constraints for these loop process schedules.

VermillionSchedule

$$
\begin{array}{l}
\mathit{verm\_memory\_size} : \mathbb{Z} \\
\mathit{VermillionInnerLoopProcSchedule} : \mathit{VermillionInnerLoopProcess} \pfun \\
\qquad \mathit{InnerFlightControlLoopTimes} \\
\mathit{VermillionMiddleLoopProcSchedule} : \mathit{VermillionMiddleLoopProcess} \pfun \\
\qquad \mathit{MiddleNavigationLoopTimes} \\
\mathit{VermillionOuterLoopProcSchedule} : \mathit{VermillionOuterLoopProcess} \pfun \\
\qquad \mathit{OuterMissionLoopTimes} \\
\mathit{VERMILLION\_SYSTEM} : \mathit{VermillionInnerLoopProcess} \fatsemi \\
\qquad \mathit{VermillionMiddleLoopProcess} \fatsemi \mathit{VermillionOuterLoopProcess} \\
\mathit{vilp?} : \mathit{VermillionInnerLoopProcess} \\
\mathit{vmlp?} : \mathit{VermillionMiddleLoopProcess} \\
\mathit{volp?} : \mathit{VermillionOuterLoopProcess}
\end{array}
$$

$$
\begin{array}{l}
\exists\, \mathit{vilp?} : \mathit{VermillionInnerLoopProcess};\ \mathit{max\_lp\_cmp\_tm} : \operatorname{ran} \mathit{VermillionInnerLoopProcSchedule} \bullet \\
\mathit{vilp?.sum\_thr\_times} \leq \mathit{max\_lp\_cmp\_tm.c_i} \\
\exists\, \mathit{vmlp?} : \mathit{VermillionMiddleLoopProcess};\ \mathit{max\_lp\_cmp\_tm} : \operatorname{ran} \mathit{VermillionMiddleLoopProcSchedule} \bullet \\
\mathit{vmlp?.sum\_thr\_times} \leq \mathit{max\_lp\_cmp\_tm.c_m} \\
\exists\, \mathit{volp?} : \mathit{VermillionOuterLoopProcess};\ \mathit{max\_lp\_cmp\_tm} : \operatorname{ran} \mathit{VermillionOuterLoopProcSchedule} \bullet \\
\mathit{volp?.sum\_thr\_times} \leq \mathit{max\_lp\_cmp\_tm.c_o} \\
[\mathit{SHPRD}1\ -\ \mathit{Guarantee\ Bounds\ on\ Agent\ Comp.\ times\ within\ a\ Loop}] \\
\exists\, \mathit{vilp?} : \mathit{VermillionInnerLoopProcess};\ \mathit{vmlp?} : \mathit{VermillionMiddleLoopProcess}; \\
\mathit{ilp\_intv} : \operatorname{ran} \mathit{VermillionInnerLoopProcSchedule}; \\
\mathit{mlp\_intv} : \operatorname{ran} \mathit{VermillionMiddleLoopProcSchedule} \bullet \\
\mathit{ilp\_intv.interval} \cap \mathit{mlp\_intv.interval} = \emptyset \\
\exists\, \mathit{vmlp?} : \mathit{VermillionMiddleLoopProcess};\ \mathit{volp?} : \mathit{VermillionOuterLoopProcess}; \\
\mathit{mlp\_intv} : \operatorname{ran} \mathit{VermillionMiddleLoopProcSchedule}; \\
\mathit{olp\_intv} : \operatorname{ran} \mathit{VermillionOuterLoopProcSchedule} \bullet \\
\mathit{mlp\_intv.interval} \cap \mathit{olp\_intv.interval} = \emptyset \\
\exists\, \mathit{vilp?} : \mathit{VermillionInnerLoopProcess};\ \mathit{volp?} : \mathit{VermillionOuterLoopProcess}; \\
\mathit{ilp\_intv} : \operatorname{ran} \mathit{VermillionInnerLoopProcSchedule}; \\
\mathit{olp\_intv} : \operatorname{ran} \mathit{VermillionOuterLoopProcSchedule} \bullet \\
\mathit{ilp\_intv.interval} \cap \mathit{olp\_intv.interval} = \emptyset \\
[\mathit{SHPRD}2\ -\ \mathit{Non\ collission\ of\ Processes}] \\
\exists\, \mathit{vilp?} : \mathit{VermillionInnerLoopProcess};\ \mathit{min\_lp\_cmp\_tm} : \operatorname{ran} \mathit{VermillionInnerLoopProcSchedule} \bullet \\
\mathit{min\_lp\_cmp\_tm.finish} - \mathit{min\_lp\_cmp\_tm.begin} + 1 = \\
\mathit{min\_lp\_cmp\_tm.c_i} \\
\exists\, \mathit{vmlp?} : \mathit{VermillionMiddleLoopProcess};\ \mathit{min\_lp\_cmp\_tm} : \operatorname{ran} \mathit{VermillionMiddleLoopProcSchedule} \bullet \\
\mathit{min\_lp\_cmp\_tm.finish} - \mathit{min\_lp\_cmp\_tm.begin} + 1 = \\
\mathit{min\_lp\_cmp\_tm.c_m} \\
\exists\, \mathit{volp?} : \mathit{VermillionOuterLoopProcess};\ \mathit{min\_lp\_cmp\_tm} : \operatorname{ran} \mathit{VermillionOuterLoopProcSchedule} \bullet \\
\mathit{min\_lp\_cmp\_tm.finish} - \mathit{min\_lp\_cmp\_tm.begin} + 1 = \\
\mathit{min\_lp\_cmp\_tm.c_o} \\
[\mathit{SHPRD}3\ -\ \mathit{Non\ Pre-emption\ of\ Process\ Loops}]
\end{array}
$$

The *VERMILLION_SYSTEM* is the complete set of processes composed from the set of three process groups: *VermillionInnerLoopProcess*, *VermillionMiddleLoopProcess*, and *VermillionOuterLoopProcess*. The predicate SHPRD1, specifies that the sum of the computation times of the agents assigned to a loop should be less than the computation time allocated to the loop: inner loop, middle loop, and outer loop. The predicate SHPRD2 provides the condition for non collision between the processes in the various groups using the interval attribute of *TimeInterval*. Non-preemption between a set of processes is specified by the predicate SHPRD3. The precedence constraint across the groups (loops) is automatically taken care by having the middle and outer control loops to be running at rates which are a a multiple of the inner loop.

VII. Applying the VERMILLION Framework

We provide two case studies from the experimental work that we have performed using the framework. Each case study is structured along the following lines: We provide a brief summary of the application and present the important requirements captured as CTL specifications. Next, we provide implementation details of the framework instantiation in NetLogo [38], which is a simple yet powerful multiagent system development platform
[61].The implementation for most important functions of each agent specified in Section VI are provided. A concise explanation follows for verification and validation aspects in each case study. Additional details can be found in [5–7].

A. Case Study 1: Adaptive Flight Planning System (AFPS)

Flight planning [62] is the process of producing a flight plan [63, 64] to describe a proposed aircraft flight. It involves two safety-critical aspects: fuel calculation, to ensure that the aircraft can safely reach the destination, and compliance with air traffic control requirements, to minimize the risk of midair collision. Traditional flight planning on an airborne platform is built in synchronism with an Air Traffic Management operational concept. In the ATM operational concept, an important aspect is the construction of layers wherein the top level layer addresses planning and execution responsibilities over a long planning horizon while subsequent lower layers address tactical or short term planning horizons [65]. To demonstrate our system in the operational concept of flight operations, we choose an application that is prevalent in the tactical management planning layer with a planning horizon that typically spans 10 to 20 minutes in the terminal airspace. At the core of the flight management system is a flight plan [64] that aggregates the route that the aircraft will fly along with altitudes and speeds for various segments of the flight path. If this flight plan is optimal, then significant benefits accrue [66] from the perspective of an individual aircraft while the airspace systemic efficiency also increases. Although aircraft flying in the terminal airspace are strictly bound by the ATC (Air Traffic Control) instructions and directives, we structure our hypothetical case study to allow some amount of flexibility in route planning with the assumption that this planning is on track for the concept of free flight [67]. The concept of free flight provides more autonomy to the aircraft and lessens the ATC burden of responsibilities [68, 69]. In this context we propose a hypothetical avionics adaptive flight planning system that works out a flight plan to a terminal area (for landing) by directly interacting with several other systems considering constraints involving weather, traffic, and terrain. Stringent safety requirements pertaining to safe landing need to be met by the adaptive system too. In order to demonstrate the efficacy of our approach, we do not take into account myriad other considerations involved in flight planning but these could be easily incorporated as additional constraints. An exemplar problem is that of an aircraft arriving in a terminal airspace, and navigating to any of the endpoint waypoints from where the final descent onto the airport runway takes place [70]. An aircraft arriving in the terminal area adheres to a Standard Terminal Arrival Chart (STARS) [71] on which various segments of the fight path and altitude levels that it has to adhere to are indicated. The pilot needs to evaluate clearances from the ATC and accept or reject such clearances. Decision of the pilot is influenced by conditions involving weather, approaching traffic and obstacles in the terrain along the route [72] to the runways indicated in the STARS chart. We make an assumption that the pilot is free to choose an endpoint on the STARS chart as the final destination once he enters the terminal area. The end point so chosen could change continuously based on weather, traffic and terrain constraints. To illustrate such a flight planning, consider an aircraft entering the Chennai (India) Terminal airspace at the top right corner of the STARS chart in Figure 8 around the HYDOK way-point [71]. The terminal endpoints are MM515, MM513, MM512 and MM510 and the goal is to reach any of these endpoints. The aircraft needs to navigate the intermediate way-points to reach one of these endpoints in the presence of weather, traffic, terrain, and ATC clearance constraints.

1. AFPS Formal Requirement Specifications

We start by indicating the broad requirements of the adaptive flight planning system outlined in the Figure 9. The requirements are captured formally, structured using {*RqF Part*}, {*RqNF Part*}, [*RqFAd Part*], and [*RqNFAd Part*] along the lines provided in equation 1. As an example, we show the segregation of a CTL requirement into these constituent parts, for the first two specifications.

CTL temporal logic specifications provide a sound basis to answer questions about the system behaviors using the method of model checking. We use CTL (Computation Tree Logic) [28] to capture safety specification for the system. CTL is a branching temporal logic (each moment of time has several possible futures) and the building blocks of CTL are atomic propositions or boolean variables that become true or false. The specifications for a system are captured as properties using connectives of CTL. The model of the system is also formulated as a labeled transition system [44]. Model checking refers to checking whether the requirement imposed on the system specified as a property is satisfied by the model of the system.

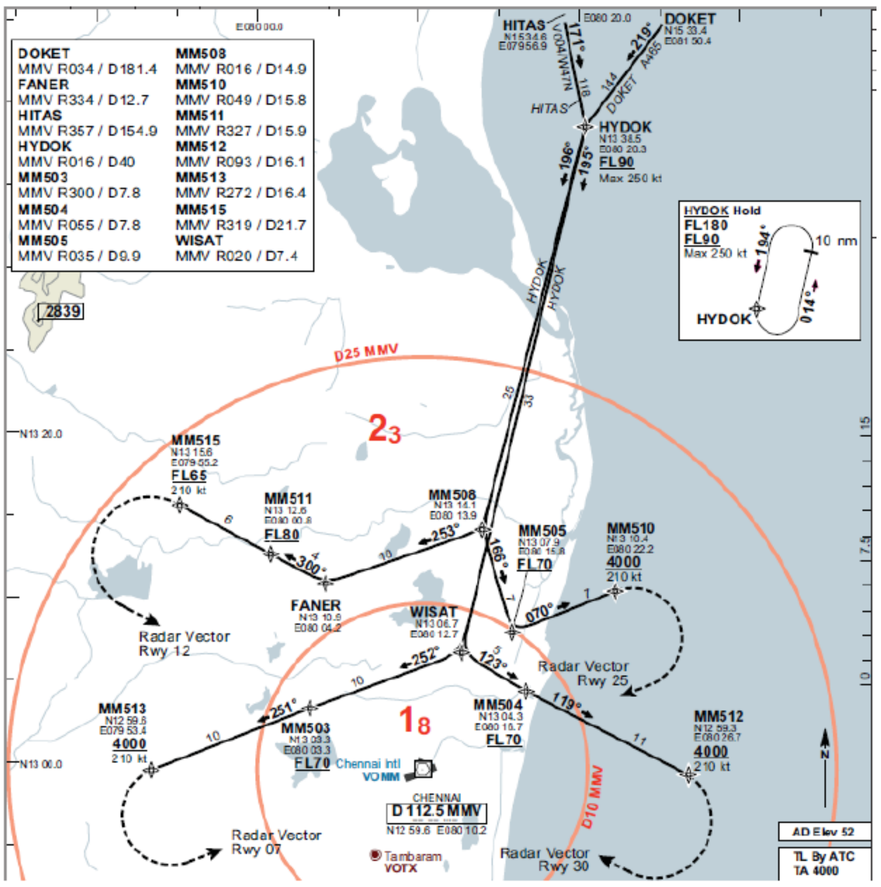


Fig. 8 Terminal Area Adaptive Flight Planning - STARS Chart

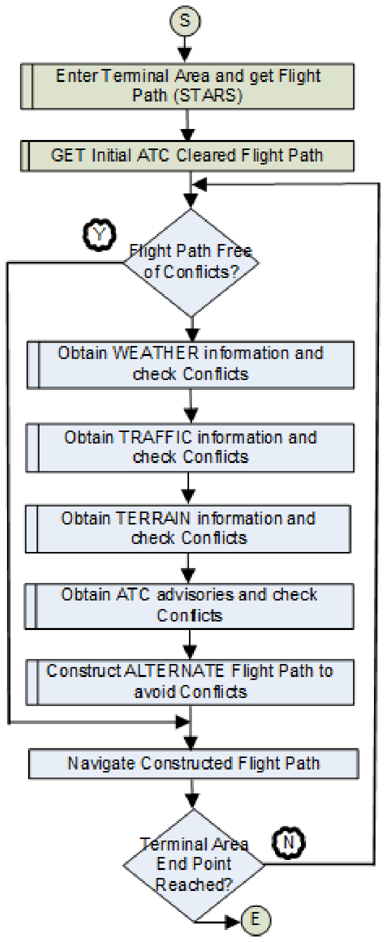

S
Enter Terminal Area and get Flight Path (STARS)
GET Initial ATC Cleared Flight Path
Y
Flight Path Free of Conflicts?
Obtain WEATHER information and check Conflicts
Obtain TRAFFIC information and check Conflicts
Obtain TERRAIN information and check Conflicts
Obtain ATC advisories and check Conflicts
Construct ALTERNATE Flight Path to avoid Conflicts
Navigate Constructed Flight Path
Terminal Area End Point Reached?
N
E

Fig. 9 Terminal Area Adaptive Flight Planning - Basic Flow

Safety and Real-timeliness Specifications: S1 The system shall find a path to the next waypoint in the terminal area that is free of conflicts from weather, traffic, and terrain constraints:

$$AG(\neg\ Wx\_Conf \wedge \neg\ Tr\_Conf \wedge \neg\ Te\_Conf \wedge ATC\_Clrnc$$
$$\wedge \neg\ FPL\_Route\_AVL) \rightarrow EF(FPL\_Route\_AVL \wedge TMBt_{nx}tl_{py}) \tag{2}$$

In the above equation, a constraint takes the form *Xx_Conf* and is a boolean predicate. *Xx* may be one of {*Wx, Tr, Te*}. $TMBt_{nx}tl_{py}$ is a boolean predicate that indicates timing requirement imposed and consists of two parts: the timing bound t_{nx} which can be one of $\{t_{5M},\ t_{3000m}\ \dots\}$, where the subscript n is indicative of the time duration and subscript x indicative of the time units (M for minutes and m representative of milliseconds); the tolerance on the timing bound expressed by tl_{py} which can be one of $\{tl_{1M},\ tl_{300m}\ \dots\}$, where the subscript p indicates the tolerance bound and subscript y indicates the time units (M for minutes and m representative of milliseconds). *FPL_Route_AVL* is a boolean predicate indicative that a *Flight Plan Route* is available. This is the result of the route search algorithm. The segregation of functional and adaptive parts for the above requirement would be:

{*RqF Part*} → ¬ *Wx_Conf*, ¬ *Tr_Conf*, ¬ *Te_Conf*, *ATC_Clrnc*, *FPL_Route_AVL*

{*RqNF Part*} → $TMBt_{nx}tl_{py}$

[*RqFAd Part*] → None

[*RqNFAd Part*] → None

Safety and Real-timeliness Specifications: S2 A route will be chosen that is optimal w.r.t each constraint that is considered, without compromising safety.

$$AG((Xx\_Conf \wedge \neg\ MDXx \wedge \neg\ LDXx \wedge \neg\ MxXxTi \wedge \neg\ (Fuel\_Remaining >$$
$$Qty\_Reqd\_Reach\_LWPT)) \rightarrow EF(HDXx \wedge \neg\ Xx\_Conf)$$

$$AG((Xx\_Conf \wedge \neg\ HDXx \wedge \neg\ LDXx \wedge \neg\ MxXxTi \wedge \neg\ (Fuel\_Remaining >$$
$$Qty\_Reqd\_Reach\_LWPT)) \rightarrow EF(MDXx \wedge \neg\ Xx\_Conf)$$

$$AG((Xx\_Conf \wedge \neg\, HDXx \wedge \neg\, MDXx \wedge \neg\, MxXxTi \wedge \neg\, (Fuel\_Remaining > Qty\_Reqd\_Reach\_LWPT)) \rightarrow EF(LDXx \wedge \neg\, Xx\_Conf) \tag{3}$$

In the above equations, *MDXx* is a Boolean predicate indicative of a Desire/Goal that provides a lower utility value (discussed in section VII.A.3) than *HDXx*, when constraint *Xx* is being resolved. *HDXx* represents the Boolean predicate indicative of a Desire/Goal that provides the highest utility value. On similar lines, *LDXx* represents the Desire/Goal with lowest utility value. *MxXxTi* is a Boolean predicate which indicates whether the maximum time for searching a solution for the constraint *Xx*, has been reached. Therefore the above specification also captures real-timeliness requirements. Also note the usage of the condition *(Fuel_Remaining > Qty_Reqd_Reach_LWPT)* in the conditional combination. *Qty_Reqd_Reach_LWPT* is a numerical value indicative of the fuel required to reach the last terminal waypoint. The segregation of functional and adaptive parts for the above requirement would be:

{*RqF Part*}	$\rightarrow$	*Xx_Conf*
{*RqNF Part*}	$\rightarrow$	None
[*RqFAd Part*]	$\rightarrow$	¬ MDXx, ¬ MDXx, ¬ LDXx
[*RqNFAd Part*]	$\rightarrow$	¬ MxXxTi

Deterministic Response Specifications: S3 The system shall issue a warning (to divert to an alternate airport) when with the fuel remaining onboard the aircraft, it appears improbable to find a route (terminal area waypoint) that is weather, traffic, and terrain constraint free. The objective of this specification is to introduce the notion of determinism for the system as a whole.

$$AG((Wx\_Conf \vee Tr\_Conflict \vee Te\_Conflict \vee \neg\, ATC\_Clrnc) \wedge (\neg\, FPL\_Route\_AVL) \wedge (Fuel\_Remaining > Qty\_Reqd\_Reach\_LWPT)) \rightarrow EX(Abandon\_Warn) \tag{4}$$

where *Abandon_Warn* is a Boolean predicate indicating that the route planning is abandoned with a warning.

Adaptation Specifications: S4 When an individual constraint is resolved and a route proposed based on the highest utility obtainable for this constraint is not accepted, the system shall choose a route that provides next highest utility (value) without compromising safety for that constraint.

$$AG((Xx\_Conf \wedge EX\neg\, Xx\_Conf \wedge \neg\, HDXx \wedge EX\neg\, HDXx) \rightarrow EG(MDXx \wedge \\ \neg\, MxXxTi \wedge (Fuel\_Remaining > Qty\_Reqd\_Reach\_LWPT))$$

$$AG((Xx\_Conf \wedge X\neg\, Xx\_Conf \wedge \neg\, MDXx \wedge X\neg\, MDXx) \rightarrow EG(LDXx \wedge \\ \neg\, MxXxTi \wedge (Fuel\_Remaining > Qty\_Reqd\_Reach\_LWPT)) \tag{5}$$

Optimal Solution / Negotiation Specifications: S5 The system shall always find a route (terminal area waypoint) that is optimal when constraints of weather, traffic, and terrain constraint are present. Optimality is given as a boolean variable indicating whether the desired optimal value has been achieved or not. Each functional subsystem allocated to handling a constraint arrives at a decision using the following ordered rule for finding the next waypoint(s) for each constraint: (Distance + Heading), (Distance), (Heading).

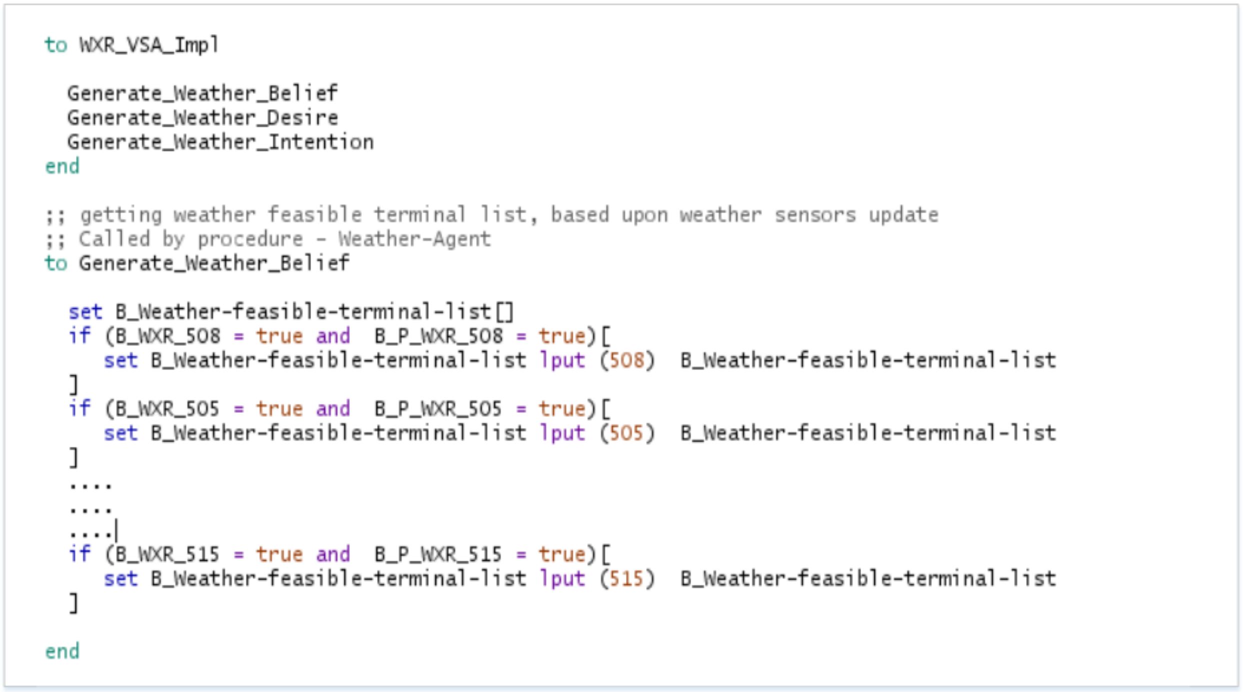
$$\begin{aligned}
&AG((Wx\_Upd \wedge (HDWx \vee MDWx \vee LDWx)) \wedge \\
&\quad (Tr\_Upd \wedge (HDTr \vee MDTr \vee LDTr) \wedge \\
&\qquad (Te\_Upd \wedge (HDTe \vee MDTe \vee LDTe) \wedge ATC\_Clrnc \wedge \\
&\qquad (\neg\, MxWxTi \wedge (Fuel\_Remaining > Qty\_Reqd\_Reach\_LWPT) \wedge \\
&\qquad\quad \neg\, FPL\_Route\_AVL)) \\
&\qquad \rightarrow EG(FPL\_Route\_AVL \wedge (HDWx \vee MDWx \vee LDWx) \wedge \\
&\qquad (HDTr \vee MDTr \vee LDTr) \wedge (HDTe \vee MDTe \vee LDTe))
\end{aligned} \tag{6}$$

2. AFPS VERMILLION Model Dynamics

Examining the whole system, we have arrived at the list below which provides a mapping of subsystems within the domain for this application to agents, applying principles of functional identification, cohesion, separation of concerns, and data exchange at the interfaces [73]. Each subsystem except for the Controller receives relevant data from its associated sensors and is responsible to provide a flight path that is free of conflict for the constraint that it handles. The Controller is then responsible to aggregate data from each of these sensors and provides an overall solution that will solve all of the constraints.

Weather Subsystem	→	Strategic Weather Agent
Traffic Subsystem	→	Strategic Traffic Agent
Terrain Subsystem	→	Strategic Terrain Agent
ATC Subsystem	→	Normal ATC Agent
Controller Subsystem	→	Moderating Negotiation Agent

The BDI model of a Strategic Agent along with its interactions is described and other strategic agents will have a similar implementation.The strategic agent maintains two kinds of beliefs about intermediate and terminal waypoints. The first is about whether a waypoint is constraint (weather/traffic/terrain) free and is available for navigation to it currently. The second is about whether a waypoint is constraint free and is available for navigation for the next (say) 5 minute window. The latter is a set of predicted beliefs. Sensors provide the 'percepts' of the current and predicted weather situation. The strategic agent directly maps these percepts onto beliefs. Feasible waypoint lists are generated by examining these base beliefs and segregated according to the desires. In our case, these are highest, moderate, and lowest desire lists corresponding to (Distance + Heading) prioritization, (Distance) prioritization, and (Heading) prioritization. The desire generation and selection operates in two stages: In the first stage it generates the partial plans (options generation function, (Opf-1)), and in the next stage chooses the 'Best Desire List' among the three using a learning algorithm (Opf-2). A reinforcement learning algorithm (TD Learning) is used to attach a utility value to each of the generated lists within the individual agents looking at 'Negotiated Final Desire List' that is provided by the Moderator Agent (Controller/Negotiation agent). Looking at the 'Negotiated Final Desire List', each Strategic Agent learns what would be the most preferred list to send to the Negotiation Agent, so that its list is accepted. Therefore, for each individual agent handling the constraint, generation of the 'Best Desire List' is equivalent to intention generation. The 'Best Desire Lists' (waypoint lists from multiple agents are reconciled within the negotiation agent which uses a simple one-shot negotiation for real-time considerations. The Negotiation Agent is formulated with two filter functions filter-Neg (for Negotiation) and Filter-Int (for Intention). Filter-Neg implements the one shot negotiation between weather, traffic, and Terrain agents, wherein the waypoint lists from each agent is compared with the other to find agreeable common waypoints. This common waypoint list forms the 'Negotiated Waypoint List'. During each execution cycle, the current waypoint to which the aircraft is heading is set to the head of the 'Negotiated List'. The Intention function (commitment) checks if there are changes brought about during an execution cycle to the negotiated list that introduces a new waypoint in the list that is currently not in it. Only when this happens the intention changes. Also, during execution of an agent cycle, temporal constraints are checked and a warning is generated if it is exceeded. The fuel remaining within the system is also tracked within a global cycle that evaluates it continuously and a warning is issued if there is an exceedance. A distance trending check (mechanism) is also evaluated to monitor whether the final solution is converging. A simple execute function navigates the current waypoint (housed within negotiation agent or



```
to WXR_VSA_Impl

  Generate_Weather_Belief
  Generate_Weather_Desire
  Generate_Weather_Intention
end

;; getting weather feasible terminal list, based upon weather sensors update
;; Called by procedure - Weather-Agent
to Generate_Weather_Belief

  set B_Weather-feasible-terminal-list[]
  if (B_WXR_508 = true and  B_P_WXR_508 = true)[
     set B_Weather-feasible-terminal-list lput (508)  B_Weather-feasible-terminal-list
  ]
  if (B_WXR_505 = true and  B_P_WXR_505 = true)[
     set B_Weather-feasible-terminal-list lput (505)  B_Weather-feasible-terminal-list
  ]
  ....
  ....
  ....
  if (B_WXR_515 = true and  B_P_WXR_515 = true)[
     set B_Weather-feasible-terminal-list lput (515)  B_Weather-feasible-terminal-list
  ]

end
```

Fig. 10   NetLogo Implementation of Beliefs Generation (Weather VSAgent)

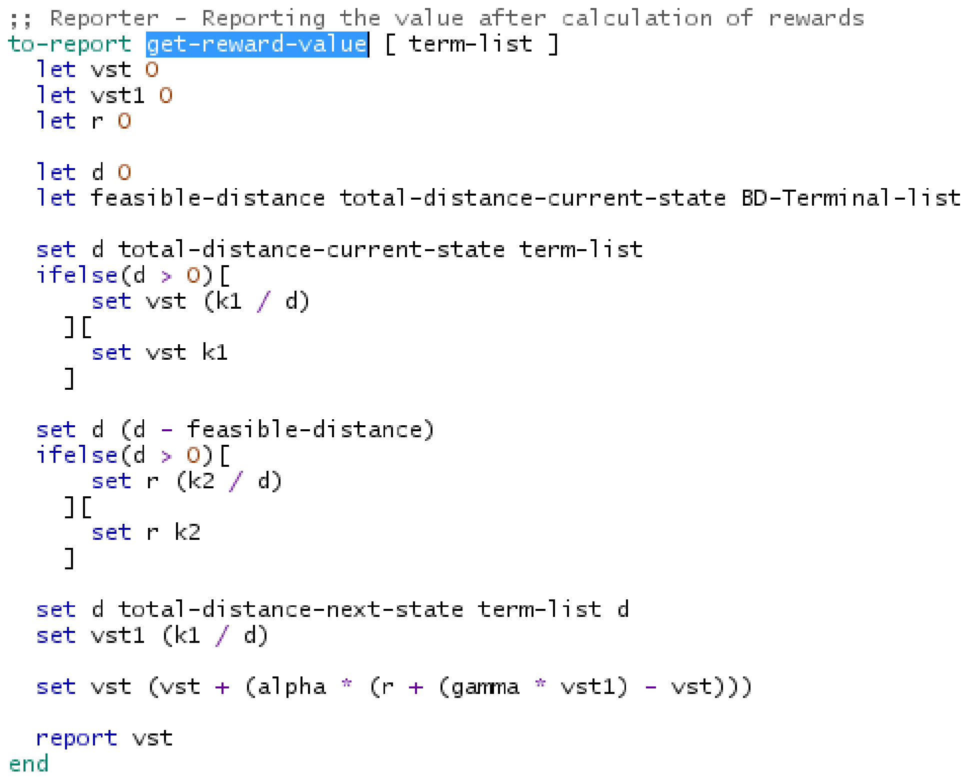
with a separate and distinct entity).

The mappings of the various VERMILLION framework operational semantics functions to their NetLogo implementation is provided below:

genrteStrgBeliefs - (See sections V.H and VI.A) Figure 10 shows how the initial feasible terminal list is generated in the "Generate_Weather_Belief" function. Also shown in the upper part of the figure is the clean implementation of an agent in terms of belief, desire, and intention generation procedures.

learnByMapngLrngEsmEntsToVal - (See sections V.H and VI.A) Figure 11 shows how the TD Learning algorithm is used in within the "get_reward_value" function that maps the learning entities vst and vst1 that hold values representative of the feature (total distance of currently generated Waypoint List from the Terminal Waypoint)that is learned from the environment.

genrteStrgGoalsWithVal - (See sections V.H and VI.A) Shown in Figure 12 are three NetLogo procedures "Generate_Weather_Desire", "get-highest-desire-list", and "reward-weather-agent" that map to the *genrteStrgGoalsWithVal* function. Notice that the first procedure calls the third to associate a value that is indicative of the relative value of desire under consideration elicited from a feature in the environment (dist-heading). Therefore by this first activity, a relative value of the highest desire (in the example) is obtained considering the environment. An actual value to this



```
;; Reporter - Reporting the value after calculation of rewards
to-report get-reward-value [ term-list ]
  let vst 0
  let vst1 0
  let r 0

  let d 0
  let feasible-distance total-distance-current-state BD-Terminal-list

  set d total-distance-current-state term-list
  ifelse(d > 0)[
      set vst (k1 / d)
    ][
      set vst k1
    ]

  set d (d - feasible-distance)
  ifelse(d > 0)[
      set r (k2 / d)
    ][
      set r k2
    ]

  set d total-distance-next-state term-list d
  set vst1 (k1 / d)

  set vst (vst + (alpha * (r + (gamma * vst1) - vst)))

  report vst
end
```

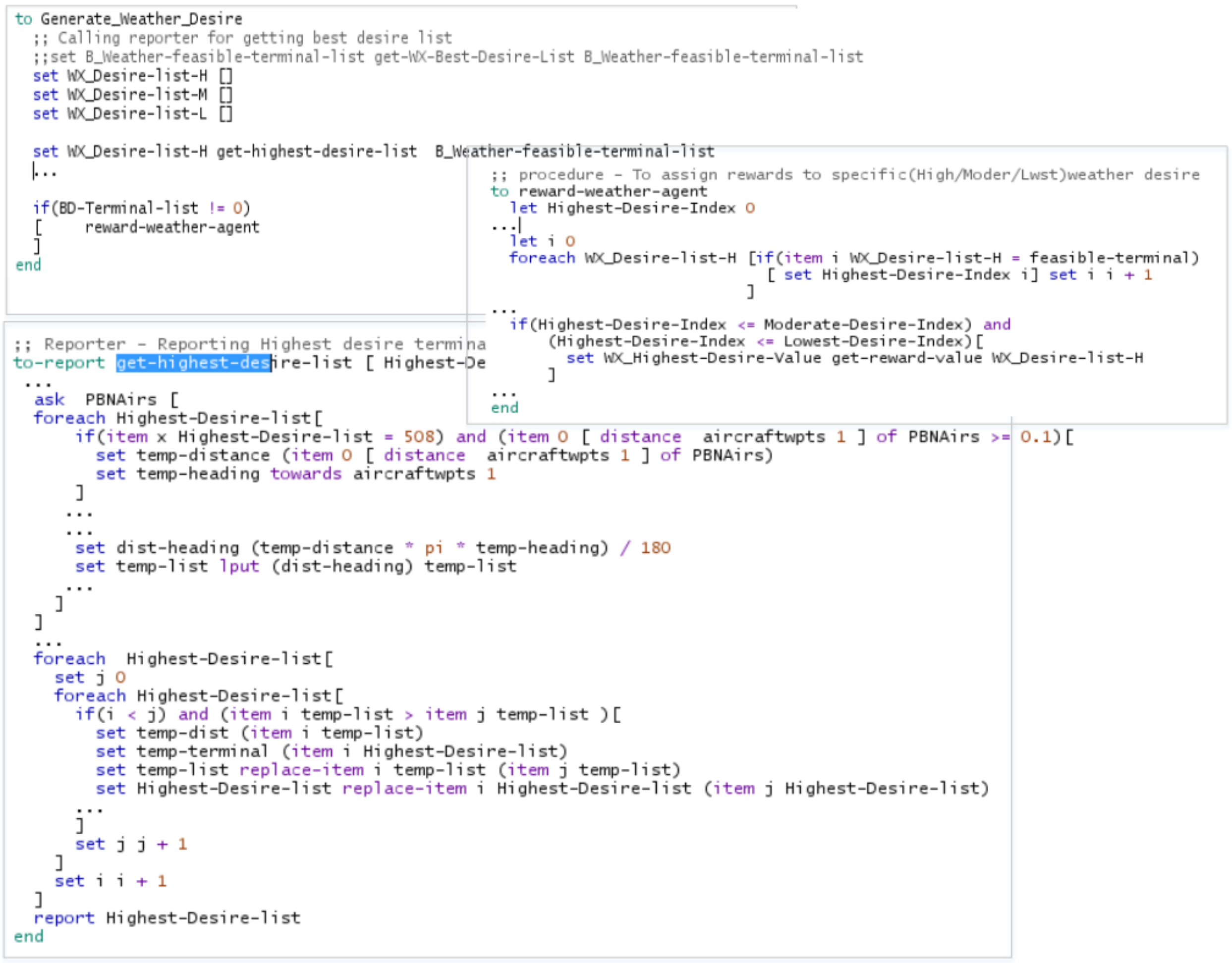
Fig. 11 NetLogo Implementation of Desire Generation SubPhase1 (Weather VSAgent)

desire is mapped onto by the second procedure "reward-weather-agent" considering the learning aspects (which came about by use of the learning algorithm).

selectIntention - (See sections V.H and VI.A) Each Agent generates its own intention and in this case operates with a single-minded commitment which is determined by the value of the desire. Figure 13 shows how the NetLogo procedure "Generate_Weather_Intention" implements this. Observe that the *WX_Highest-Desire-Value* was actually set up in the desire generation process by the "reward-weather-agent" procedure (shown in Figure 12).

mapNormalBeliefsToStates, mapStatesToNormalGoals - (See sections V.I and VI.B)Figure 14 shows the NetLogo procedures "WXR_VNA_Map_States_To_Goals" and "Wxr_VNA_Map_Beliefs_To_States". These implementations are straight-forward. These examples provide the exemplar implementations for the VNA.

generateModeratedOutputs - (See sections V.K and VI.D) Two NetLogo procedures "Negotiate_Stage_1" and "Negotiate_Stage_2" implement this function and these are shown in Figure 15. In the stage 1 negotiation, a one shot negotiation process is effected between four sources: the weather agent, traffic agent, terrain agent and ATC agent.



```
to Generate_Weather_Desire
  ;; Calling reporter for getting best desire list
  ;;set B_Weather-feasible-terminal-list get-WX-Best-Desire-List B_Weather-feasible-terminal-list
  set WX_Desire-list-H []
  set WX_Desire-list-M []
  set WX_Desire-list-L []

  set WX_Desire-list-H get-highest-desire-list  B_Weather-feasible-terminal-list
  ...

  if(BD-Terminal-list != 0)
  [     reward-weather-agent
  ]
end
```

```
;; procedure - To assign rewards to specific(High/Moder/Lwst)weather desire
to reward-weather-agent
  let Highest-Desire-Index 0
...
  let i 0
  foreach WX_Desire-list-H [if(item i WX_Desire-list-H = feasible-terminal)
                             [ set Highest-Desire-Index i] set i i + 1
                           ]
...
  if(Highest-Desire-Index <= Moderate-Desire-Index) and
      (Highest-Desire-Index <= Lowest-Desire-Index)[
        set WX_Highest-Desire-Value get-reward-value WX_Desire-list-H
      ]
...
end
```

```
;; Reporter - Reporting Highest desire termina
to-report get-highest-desire-list [ Highest-De
 ...
  ask  PBNAirs [
  foreach Highest-Desire-list[
      if(item x Highest-Desire-list = 508) and (item 0 [ distance  aircraftwpts 1 ] of PBNAirs >= 0.1)[
        set temp-distance (item 0 [ distance  aircraftwpts 1 ] of PBNAirs)
        set temp-heading towards aircraftwpts 1
      ]
     ...
     ...
      set dist-heading (temp-distance * pi * temp-heading) / 180
      set temp-list lput (dist-heading) temp-list
     ...
    ]
  ]
  ...
  foreach  Highest-Desire-list[
    set j 0
    foreach Highest-Desire-list[
      if(i < j) and (item i temp-list > item j temp-list )[
        set temp-dist (item i temp-list)
        set temp-terminal (item i Highest-Desire-list)
        set temp-list replace-item i temp-list (item j temp-list)
        set Highest-Desire-list replace-item i Highest-Desire-list (item j Highest-Desire-list)
      ...
      ]
      set j j + 1
    ]
    set i i + 1
  ]
  report Highest-Desire-list
end
```

Fig. 12   NetLogo Implementation of Desire Generation SubPhase2 (Weather VSAgent)

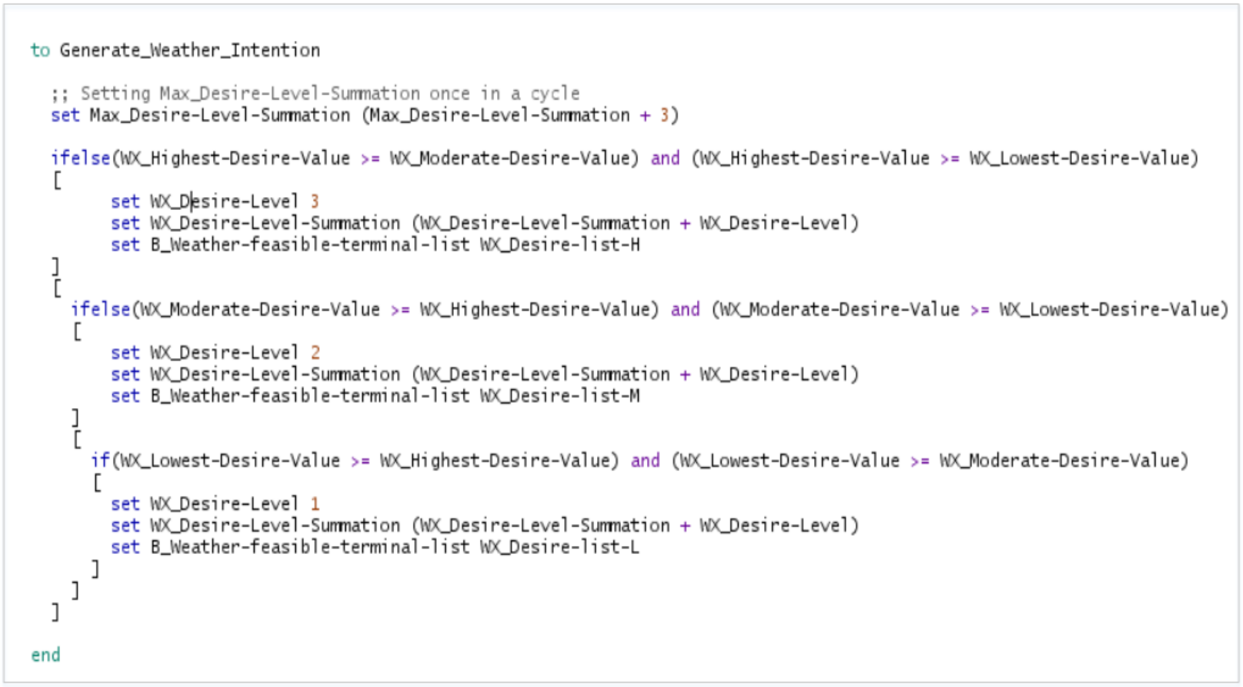

```
to Generate_Weather_Intention

  ;; Setting Max_Desire-Level-Summation once in a cycle
  set Max_Desire-Level-Summation (Max_Desire-Level-Summation + 3)

  ifelse(WX_Highest-Desire-Value >= WX_Moderate-Desire-Value) and (WX_Highest-Desire-Value >= WX_Lowest-Desire-Value)
  [
      set WX_Desire-Level 3
      set WX_Desire-Level-Summation (WX_Desire-Level-Summation + WX_Desire-Level)
      set B_Weather-feasible-terminal-list WX_Desire-list-H
  ]
  [
    ifelse(WX_Moderate-Desire-Value >= WX_Highest-Desire-Value) and (WX_Moderate-Desire-Value >= WX_Lowest-Desire-Value)
    [
      set WX_Desire-Level 2
      set WX_Desire-Level-Summation (WX_Desire-Level-Summation + WX_Desire-Level)
      set B_Weather-feasible-terminal-list WX_Desire-list-M
    ]
    [
      if(WX_Lowest-Desire-Value >= WX_Highest-Desire-Value) and (WX_Lowest-Desire-Value >= WX_Moderate-Desire-Value)
      [
      set WX_Desire-Level 1
      set WX_Desire-Level-Summation (WX_Desire-Level-Summation + WX_Desire-Level)
      set B_Weather-feasible-terminal-list WX_Desire-list-L
      ]
    ]
  ]

end
```

Fig. 13 NetLogo Implementation of Intention Selection (Weather VSAgent)

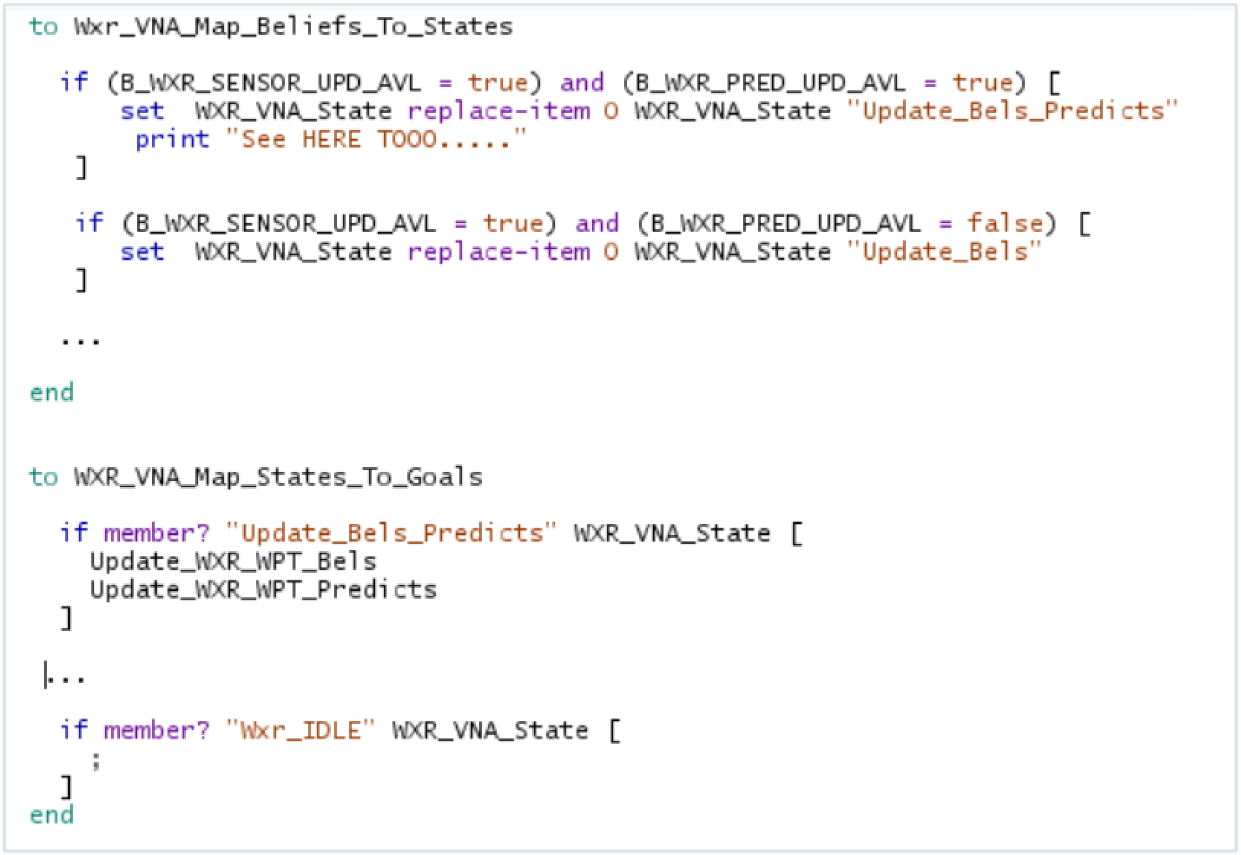

```
to Wxr_VNA_Map_Beliefs_To_States

  if (B_WXR_SENSOR_UPD_AVL = true) and (B_WXR_PRED_UPD_AVL = true) [
      set  WXR_VNA_State replace-item 0 WXR_VNA_State "Update_Bels_Predicts"
       print "See HERE TOOO....."
   ]

   if (B_WXR_SENSOR_UPD_AVL = true) and (B_WXR_PRED_UPD_AVL = false) [
      set  WXR_VNA_State replace-item 0 WXR_VNA_State "Update_Bels"
   ]

  ...

end


to WXR_VNA_Map_States_To_Goals

  if member? "Update_Bels_Predicts" WXR_VNA_State [
    Update_WXR_WPT_Bels
    Update_WXR_WPT_Predicts
  ]

 ...

  if member? "Wxr_IDLE" WXR_VNA_State [
    ;
  ]
end
```

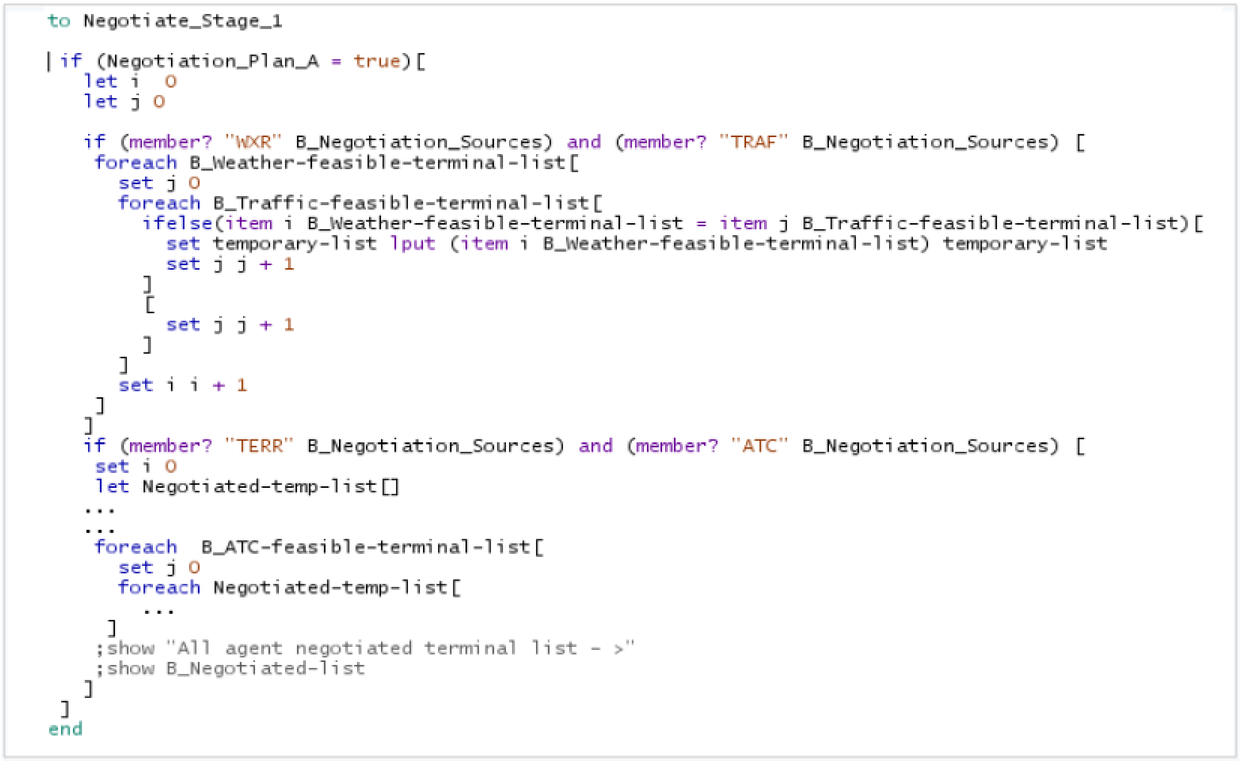
Fig. 14 NetLogo Implementation of Mapping Beliefs to States to Goals (Weather VNAgent)

Notice the decision on using Plan A which specifies the order of negotiation. In Stage 2 negotiation, a rule terminal list is used (in addition to learning, that helps locate the next terminal from the current aircraft position) to compare the waypoints in the negotiated list from the phase 1 negotiation and a final terminal list to navigate to is obtained. An additional commitment maintenance check is performed on the final terminal list. With the above, we conclude the exemplar delinations of the important functions in VERMILLION for our first case study.

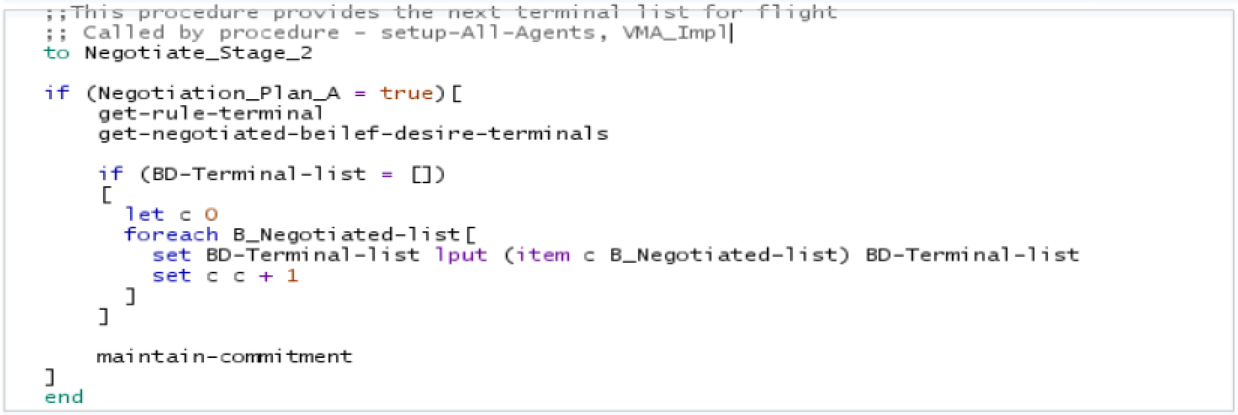
3. AFPS Verification and Validation

In accordance with the DO178C process, the formal high level requirements expressed in CTL was model checked using the NUSMV tool
[74], using a finite state model that predominantly exhibits boolean behavior and is sound with respect to the VERMILLION BDI model. The mapping of the variables in the NetLogo AFPS implementation to boolean variables in NUSMV model was performed manually using guidelines and techniques outlined in section III. Additional details with illustrated examples are provided in section VII.C. The different possible behaviors was captured and evaluated by subjecting the NUSMV model to several runs of simulation. Figure 16 shows the abstracted finite state transition system for the AFPS. Since we are interested in examining the adaptive system, there is a notion of a global flight planning transition system that takes inputs from local transition systems that correspond to the strategic agents: Weather VSA, Traffic VSA, Terrain VSA, and ATC VSA. The adaptivity incorporated as learning is abstracted to a set of desires at three levels: high, moderate, and low. A brief description of the state transition system is in order: States S0, S1, and S18 to S23 depict the abstracted transition system corresponding to the flight planning system. The flight planning system runs in a cycle of a deterministic scheduler. Inputs from weather, traffic, and terrain VSAs, in order, are fed into the flight planning system. These checks are depicted in states S2 to S16, with state labels appropriately indicating the update provided by each agent. After receiving updates from the three agents, the flight planning system checks with ATC for beliefs and generates a new set of flight plans as its desires in states S17 to S23. Negotiation of all individual partial plans is accomplished in state S20. The model also incorporates state variables for fuel remaining aboard the aircraft and distance trending checks that are required to examine if the flight planning is really converging. We now turn our attention to the NUSMV model which depicts the state transition system. The upper part in Figure 17 shows sections of the model of the transition system as described above. The lower part shows how the state transitions are described in the *TRANS* part. The safety property existant in the requirement provided by equation 4 (section VII.A.1) was checked with NUSMV tool which reported a counter example shown in Figure 18.

The counter example produced by the NUSMV was cross checked with the NetLogo implementation and this indicated a flaw in the design. The fix was incorporated by introducing a timer variable that keeps track of the time for arriving at a solution and if this time limit is exceeded, will provide an appropriate warning. A more detailed treatment of the verification aspects of this case study is available in [7].



```
to Negotiate_Stage_1

| if (Negotiation_Plan_A = true)[
    let i  0
    let j 0

    if (member? "WXR" B_Negotiation_Sources) and (member? "TRAF" B_Negotiation_Sources) [
     foreach B_Weather-feasible-terminal-list[
       set j 0
       foreach B_Traffic-feasible-terminal-list[
         ifelse(item i B_Weather-feasible-terminal-list = item j B_Traffic-feasible-terminal-list)[
           set temporary-list lput (item i B_Weather-feasible-terminal-list) temporary-list
           set j j + 1
         ]
         [
           set j j + 1
         ]
       ]
       set i i + 1
     ]
    ]
    if (member? "TERR" B_Negotiation_Sources) and (member? "ATC" B_Negotiation_Sources) [
     set i 0
     let Negotiated-temp-list[]
    ...
    ...
     foreach  B_ATC-feasible-terminal-list[
       set j 0
       foreach Negotiated-temp-list[
         ...
      ]
     ;show "All agent negotiated terminal list - >"
     ;show B_Negotiated-list
    ]
  ]
 end
```



```
;;This procedure provides the next terminal list for flight
;; Called by procedure - setup-All-Agents, VMA_Impl
to Negotiate_Stage_2

if (Negotiation_Plan_A = true)[
     get-rule-terminal
     get-negotiated-beilef-desire-terminals

     if (BD-Terminal-list = [])
     [
       let c 0
       foreach B_Negotiated-list[
         set BD-Terminal-list lput (item c B_Negotiated-list) BD-Terminal-list
         set c c + 1
       ]
     ]

     maintain-commitment
]
end
```

Fig. 15 NetLogo Implementation of Moderation Outputs Generation (Negotiation VMAgent)

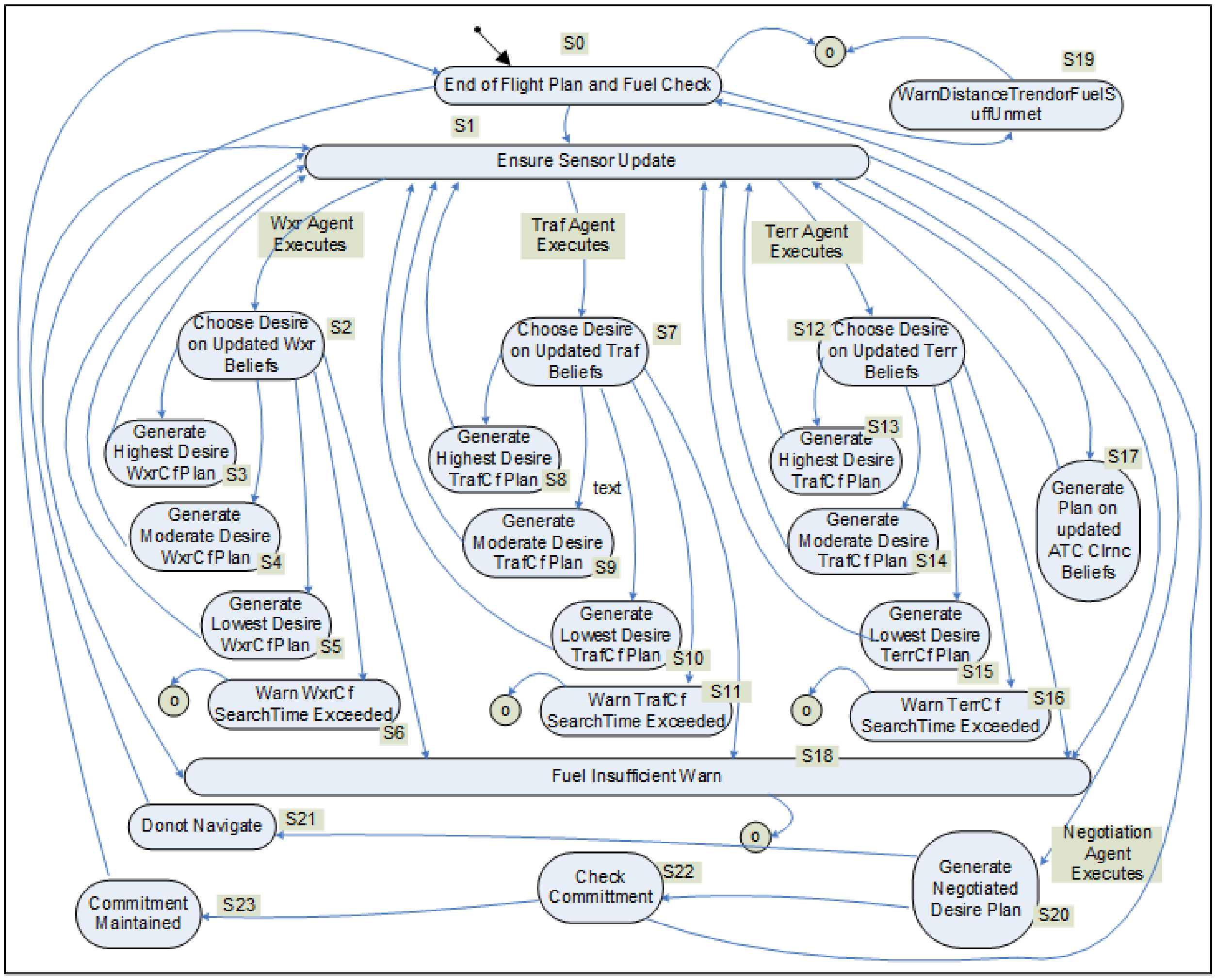
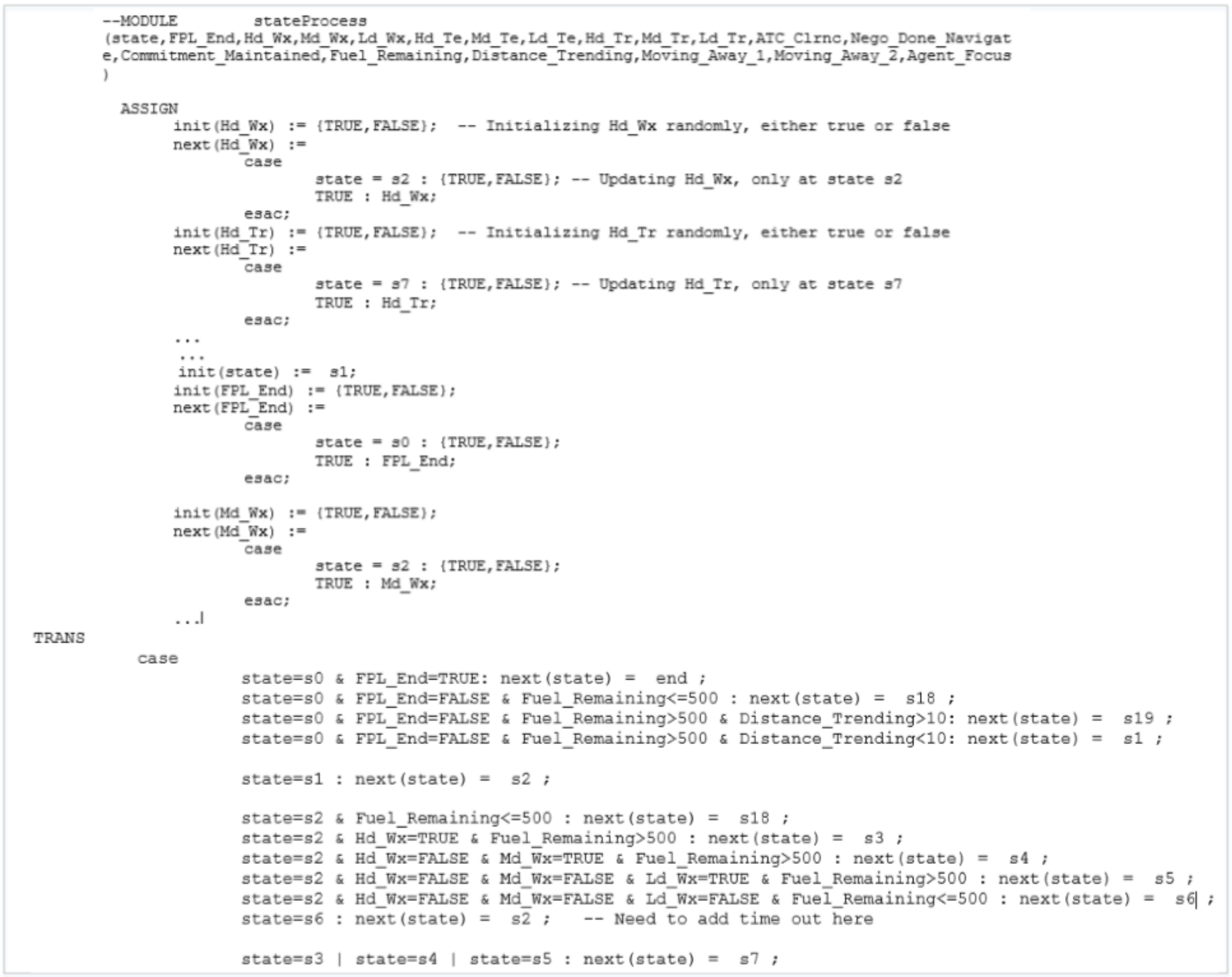
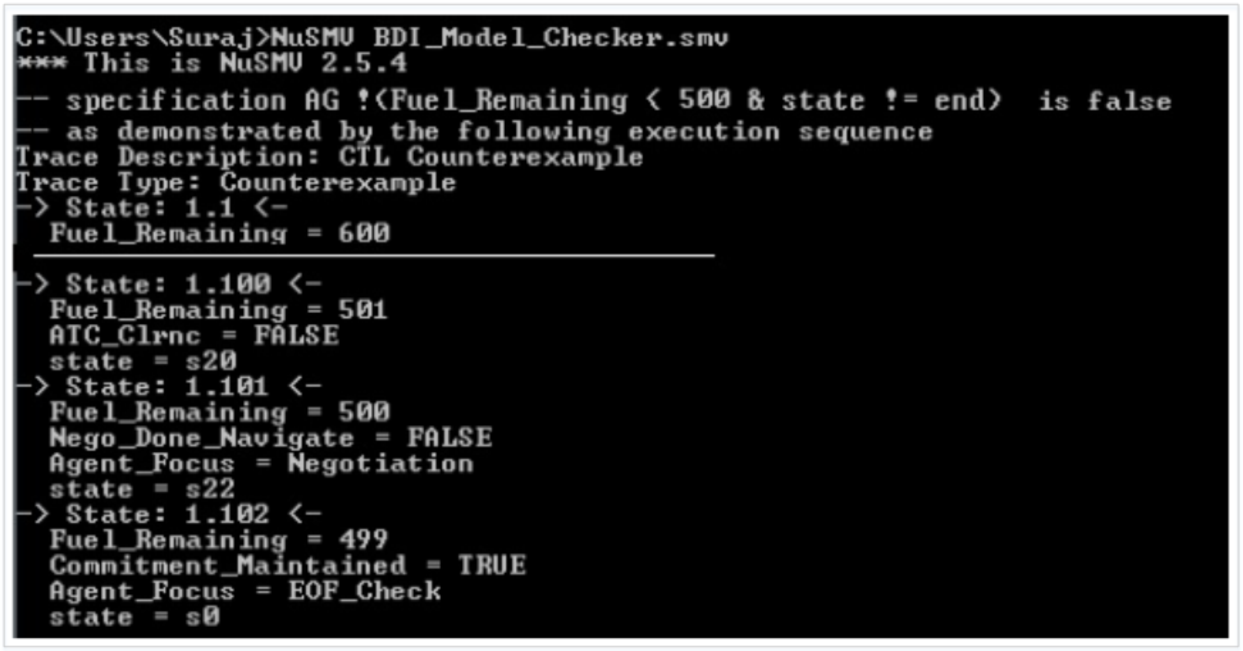

S0
End of Flight Plan and Fuel Check
S19
WarnDistanceTrendorFuelSuffUnmet
S1
Ensure Sensor Update
Wxr Agent Executes
Traf Agent Executes
Terr Agent Executes
Choose Desire on Updated Wxr Beliefs
S2
Choose Desire on Updated Traf Beliefs
S7
S12
Choose Desire on Updated Terr Beliefs
Generate Highest Desire WxrCfPlan
S3
Generate Highest Desire TrafCfPlan
S8
S13
Generate Highest Desire TrafCfPlan
S17
Generate Plan on updated ATC Clrnc Beliefs
text
Generate Moderate Desire WxrCfPlan
S4
Generate Moderate Desire TrafCfPlan
S9
Generate Moderate Desire TrafCfPlan
S14
Generate Lowest Desire WxrCfPlan
S5
Generate Lowest Desire TrafCfPlan
S10
Generate Lowest Desire TerrCfPlan
S15
Warn WxrCf SearchTime Exceeded
S6
Warn TrafCf SearchTime Exceeded
S11
Warn TerrCf SearchTime Exceeded
S16
S18
Fuel Insufficient Warn
Donot Navigate
S21
Negotiation Agent Executes
Check Committment
S22
Generate Negotiated Desire Plan
S20
Commitment Maintained
S23

Fig. 16 Abstract transition system for the AFPS, case study 1



```
        --MODULE        stateProcess
        (state,FPL_End,Hd_Wx,Md_Wx,Ld_Wx,Hd_Te,Md_Te,Ld_Te,Hd_Tr,Md_Tr,Ld_Tr,ATC_Clrnc,Nego_Done_Navigat
        e,Commitment_Maintained,Fuel_Remaining,Distance_Trending,Moving_Away_1,Moving_Away_2,Agent_Focus
        )

          ASSIGN
                init(Hd_Wx) := {TRUE,FALSE};  -- Initializing Hd_Wx randomly, either true or false
                next(Hd_Wx) :=
                        case
                                state = s2 : {TRUE,FALSE}; -- Updating Hd_Wx, only at state s2
                                TRUE : Hd_Wx;
                        esac;
                init(Hd_Tr) := {TRUE,FALSE};  -- Initializing Hd_Tr randomly, either true or false
                next(Hd_Tr) :=
                        case
                                state = s7 : {TRUE,FALSE}; -- Updating Hd_Tr, only at state s7
                                TRUE : Hd_Tr;
                        esac;
                ...
                 ...
                 init(state) :=  s1;
                init(FPL_End) := {TRUE,FALSE};
                next(FPL_End) :=
                        case
                                state = s0 : {TRUE,FALSE};
                                TRUE : FPL_End;
                        esac;

                init(Md_Wx) := {TRUE,FALSE};
                next(Md_Wx) :=
                        case
                                state = s2 : {TRUE,FALSE};
                                TRUE : Md_Wx;
                        esac;
                ...
TRANS
           case
                        state=s0 & FPL_End=TRUE: next(state) =  end ;
                        state=s0 & FPL_End=FALSE & Fuel_Remaining<=500 : next(state) =  s18 ;
                        state=s0 & FPL_End=FALSE & Fuel_Remaining>500 & Distance_Trending>10: next(state) =  s19 ;
                        state=s0 & FPL_End=FALSE & Fuel_Remaining>500 & Distance_Trending<10: next(state) =  s1 ;

                        state=s1 : next(state) =  s2 ;

                        state=s2 & Fuel_Remaining<=500 : next(state) =  s18 ;
                        state=s2 & Hd_Wx=TRUE & Fuel_Remaining>500 : next(state) =  s3 ;
                        state=s2 & Hd_Wx=FALSE & Md_Wx=TRUE & Fuel_Remaining>500 : next(state) =  s4 ;
                        state=s2 & Hd_Wx=FALSE & Md_Wx=FALSE & Ld_Wx=TRUE & Fuel_Remaining>500 : next(state) =  s5 ;
                        state=s2 & Hd_Wx=FALSE & Md_Wx=FALSE & Ld_Wx=FALSE & Fuel_Remaining<=500 : next(state) =  s6 ;
                        state=s6 : next(state) =  s2 ;   -- Need to add time out here

                        state=s3 | state=s4 | state=s5 : next(state) =  s7 ;
```

Fig. 17   NuSMV Model for Abstract transition system of the AFPS, case study 1



```
C:\Users\Suraj>NuSMV BDI_Model_Checker.smv
*** This is NuSMV 2.5.4
-- specification AG !(Fuel_Remaining < 500 & state != end)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  Fuel_Remaining = 600
________________________________________
-> State: 1.100 <-
  Fuel_Remaining = 501
  ATC_Clrnc = FALSE
  state = s20
-> State: 1.101 <-
  Fuel_Remaining = 500
  Nego_Done_Navigate = FALSE
  Agent_Focus = Negotiation
  state = s22
-> State: 1.102 <-
  Fuel_Remaining = 499
  Commitment_Maintained = TRUE
  Agent_Focus = EOF_Check
  state = s0
```

Fig. 18 Counter example in NuSMV model execution of AFPS, case study 1

Table 2 Measures enabling Validation of Non-Functional Requirements, AFPS

Measure Description	Mean Value	Standard Deviation
Co-operative Gain with		
TD Learning	2.99	7.16
Coeff. of Co-operation (Wx) [Tr] {Te}	(0.294) [0.287] {0.278}	(0.14) [0.14] {0.135}

We discuss briefly some novel methods to validate our requirements and case study application. Observing the fact that runtime offers a good window providing insights to the adaptation of the system to changes in the environment, we are motivated to examine a system from its self-* properties [25]. If one can devise a method and measure some of the self-* properties associated with the system under consideration, significant insights can be obtained that provide reassurances with respect to the adaptive system as a whole. Examination of [*RqFAd Part*] and [*RqNFAd Part*] of requirements (Equation) and associating these with the self-* properties provides a structured way to characterize the system. In the context of our adaptive flight planning system, these measures are: co-operative gains directly related to self-optimization property, Coefficient of co-operation between pairs of agents directly related to self-healing property, and Timing measures for handling critical events directly related to self-protecting property. Table 2 shows two of these measures culled out from our work
[6].

B. Case Study 2: Avionics of a small Unmanned Air System (AVsUAS)

small Unmanned Ariel Systems (sUAS) and their potential applications is seeing a tremendous growth and by 2021 the sUAS fleet is expected to be ten times the 2016 population [75]. We focus our attention to missions of a sUAS and motivated to explore the beyond line of sight operations as brought out in [76]. Several mission scenarios are outlined in [77] and [78]. We start by constructing a hypothetical mission scenario using a rotary wing aircraft operating with a quad or hexacopter configuration and flying an autonomous mission consisting of about five waypoints (A, B, C, D, E) in an inhospitable terrain. The specific mission aim could be survey photography, pipeline inspection, or a mission that involves emergency response in an uneven and inhospitable terrain or surroundings. The UAS is connected to a Ground Control Station (GCS) via a Radio Frequency (RF) control and video data link. The payload is assumed to consist of an onboard camera which feeds the video datalink. A mission is considered successful when the sUAS is launched and flies the full mission path and records the details during the data gathering mission else a degraded mission is flown. In both nominal and degraded missions, the sUAS is required to fly autonomously, navigating all the waypoints in the pre-programmed mission plan. Loss of communication link, unfavorable winds, rapidly diminishing onboard power conditions, and possibly unfavorable weather or light conditions provide the driving conditions for an adaptive UAS. In the United States, sUAS operations are governed by FAA (Federal Aviation Administration) Part 107 Small Unmanned Aircraft Regulations [79]. In the context of autonomy, the regulation has specific guidance on autonomous operations (sec. 5.2.3 in [79]) and loss of link conditions (sec. B.6.1 in [79]) when operating such systems. However, the main functions of the ariel platform would be ‘Aviate’, ‘Navigate’, ‘Communicate’, ‘Mitigate’, with due considerations from ARP4754 and ARP4761. Therefore the avionics system on board the UAS is envisaged to consist of a Flight Control System (FCS), Flight Management System (FMS), Collision Avoidance System (CAS), Navigation System (NS), Communication Link Management System (CLMS), Mission Manager System (MMS), Payload Management System (PMS) and an Emergency Management System (EMS). We hypothesize a scenario wherein the UAS loses radio link and tries to recover this link by maneuvering itself at that point in space. The UAS has a set of pre-planned maneuvers in its repertoire that it will perform to get back the link. However, these maneuvers have to be evaluated in the context of wind conditions prevailing and onboard power levels. Therefore we conceptualize an adaptive UAS that also learns such manuevers when flying repeated missions in the geographical area. The adaptation mechanism is distributed amongst three onboard avionics systems: CLMS, FMS, and MMS, to produce a "joint learning" system. It is to be noted that the overall system will still need to cater to other safety requirements while experiencing the link loss conditions. Simultaneous failure conditions like a rotor loss and impending collision with another UAS may occur, necessitating appropriate prioritized responses by other systems like FCS and CAS.

1. AVsUAS Formal Requirement Specifications

We segregate the {*RqF Part*} requirements of the AVsUAS system in the context of the adaptation for link loss into three buckets: Safety, Deterministic Response, Realtimeliness, and Adaptation requirements. Formal specifications provide a sound basis to answer questions about the system behaviors using the method of model checking. We again use CTL to capture safety, deterministic, and realtimeliness specification for the system. As an exemplar, we show the segregation of a CTL requirement into constituent parts (along the lines provided in equation 1), for the first two specifications.

Safety Specifications with timeliness: S1 The system shall ensure that ultimately there is no collision with another aircraft when flying a mission and all of the following conditions are true: (a) there is a separation loss with another aircraft (*CSpLoss*) (b) there is no loss of more than one rotor (*TwTRtrLoss*) that will destabilize the platform (c)sUAS is either navigating (*Nav*) or hovering (*Hvr*) (d) presence of a link loss (*LnLoss*) and the sUAS is attempting to regain the link (*Atmpt_Reg_Lnk*) (e) prese nce of low fuel (*LowF*) conditions

$$\begin{aligned} &AG((\neg\ EOFpl \wedge (LnLoss \wedge Atmpt\_Reg\_Lnk) \wedge CSpLoss \wedge \neg\ TwTRtrLoss \\ &\wedge (Nav \vee Hvr) \wedge LowF) \rightarrow E[(ColEvMnvr \wedge MvrRcvrTm \cup \neg\ CSpLoss]) \end{aligned} \tag{7}$$

In the above equation, *EOFpl* indicates the CTL representation of the End of flight Plan and ColEvMnvr indicates a collision evasive manuever, while MvrRcvrTm indicates the time alloted for recovering from the manuever. The segregation of functional and adaptive parts for the above requirement would be:

{*RqF Part*} → ¬ *EOFpl, LnLoss, CSpLoss,* ¬ *TwTRtrLoss,*
Nav, Hvr, LowF, ColEvMnvr, CSpLoss
{*RqNF Part*} → *MvrRcvrTm*
[*RqFAd Part*] → *Atmpt_Reg_Lnk*
[*RqNFAd Part*] → None

Deterministic Response Specifications: S2 The system shall always execute a finite number of manuvers to regain a lost link when all of the following conditions are true: (a) flight plan is not completed (EOFpl) (b) absence of low fuel

(LowF) conditions

$$AG(\neg\ EOFpl \wedge LnLoss \wedge \neg\ LowF \rightarrow A(((Atmpt\_Reg\_Lnk \leq Max\_Atmpts\_Reg\_Lnk) \wedge \neg\ LnLoss) \cup ((Atmpt\_Reg\_Lnk > Max\_Atmpts\_Reg\_Lnk) \wedge LnLoss)))) \quad (8)$$

In the above equation, *Atmpt_Reg_Lnk* indicates the boolean variable representing the fact that manuevers are being made to regain the link and *Max_Atmpts_Reg_Lnk* indicates maximum number of such manuevers. The segregation of functional and adaptive parts for the above requirement would be:

{*RqF Part*}	→	¬ *EOFpl, LnLoss,* ¬ *LowF*
{*RqNF Part*}	→	None
[*RqFAd Part*]	→	*Atmpt_Reg_Lnk*
[*RqNFAd Part*]	→	*Max_Atmpts_Reg_Lnk*

Deterministic Response Specifications: S3 The system shall fly a degraded mission (*DegMission*) by modifying the flight plan (*ModFpl*)when either of the following conditions are true: (a) presence of low fuel (*LowF*) conditions (b) irrecoverable loss of sensor dedicated for the mission (c) Occurence of a single rotor failure (*SoTRtrLoss*)

$$AG((\neg\ EOFpl \vee SoTRtrLoss \vee MiSensorLoss) \wedge (\neg\ DegMission) \rightarrow EF(X(ModFpl) \wedge DegMission)) \quad (9)$$

In the above equation, *MiSensorLoss* indicates the CTL representation of any of the sensors that are needed for completing a successful mission and in the current context, can indicate a permanent loss of the radio link subsystem. *DegMission* indicates the predicate representative of a degraded mission.

Realtimeliness Specifications: S4 The system shall ensure a degraded mission plan activation within [0.5] seconds of detection of an irrecoverable communication system sensor loss

$$\begin{aligned} &AG((\neg\ \mathit{EOFpl} \wedge \mathit{DegMission} \wedge \mathit{CommSysFail} \wedge \neg\ \mathit{DegMisPlActvnStart}) \\ &\quad \rightarrow E[\neg\ \mathit{DegMisActvnTm}\ _{E}\mathit{xprd}\ U\ \mathit{DegMisPlActvnStart}]) \end{aligned} \tag{10}$$

In the above equation, *CommSysFail* indicates the CTL representation of the permanent failure of the communication system, while *DegMisPlActvnStart* is a predicate indicating activation of the degraded mission plan. *DegMisActvnTm_Exprd* indicates the predicate representative of timing requirement constraint imposed for activating the degraded mission plan.

Adaptation Specifications: S5 When a communication link is lost, the system shall choose the best manuever to regain the link.

$$\begin{aligned} &AG((\mathit{LnLoss} \wedge X\ \mathit{LnLoss}) \rightarrow E[(\mathit{MnvrBstPosLnk} \wedge \mathit{MnvrBstPosWnd} \wedge \\ &\quad \mathit{MnvrBstPosPwr})\ U\ (\mathit{MnvrNextBestPosLnk} \wedge \\ &\quad \mathit{MnvrNextBestPosWnd} \wedge \mathit{MnvrNextBestPosPwr})]) \end{aligned} \tag{11}$$

In the above equation, *MnvrBstPosLnk*, *MnvrBstPosWnd*, and *MnvrBstPosPwr* indicate the CTL representation of the best position that is learned to regain the link in the context of link regain position, wind conditions, and power efficiency while *MnvrNextBestPos* with suffixes *Lnk*, *Wnd*, and *Pwr* are indicative of any other position in the sUAS repertoire, where link regain position, wind, and power conditions are acceptable in the next best position.

2. AVsUAS VERMILLION Model Dynamics

The AVsUAS system implementation is orchestrated by a Scheduler that invokes the following set of agents:

Flight Control System Normal Agent (FCS_VNA) and Flight Control System Tactical Agent (FCS_VTA)

Communication Link Normal Agent (CL_VNA) and Communication Link Strategic Agent (CL_VSA)

Flight Management System Normal Agent (FMS_VNA) and Flight Management System Strategic Agent (FMS_VSA)

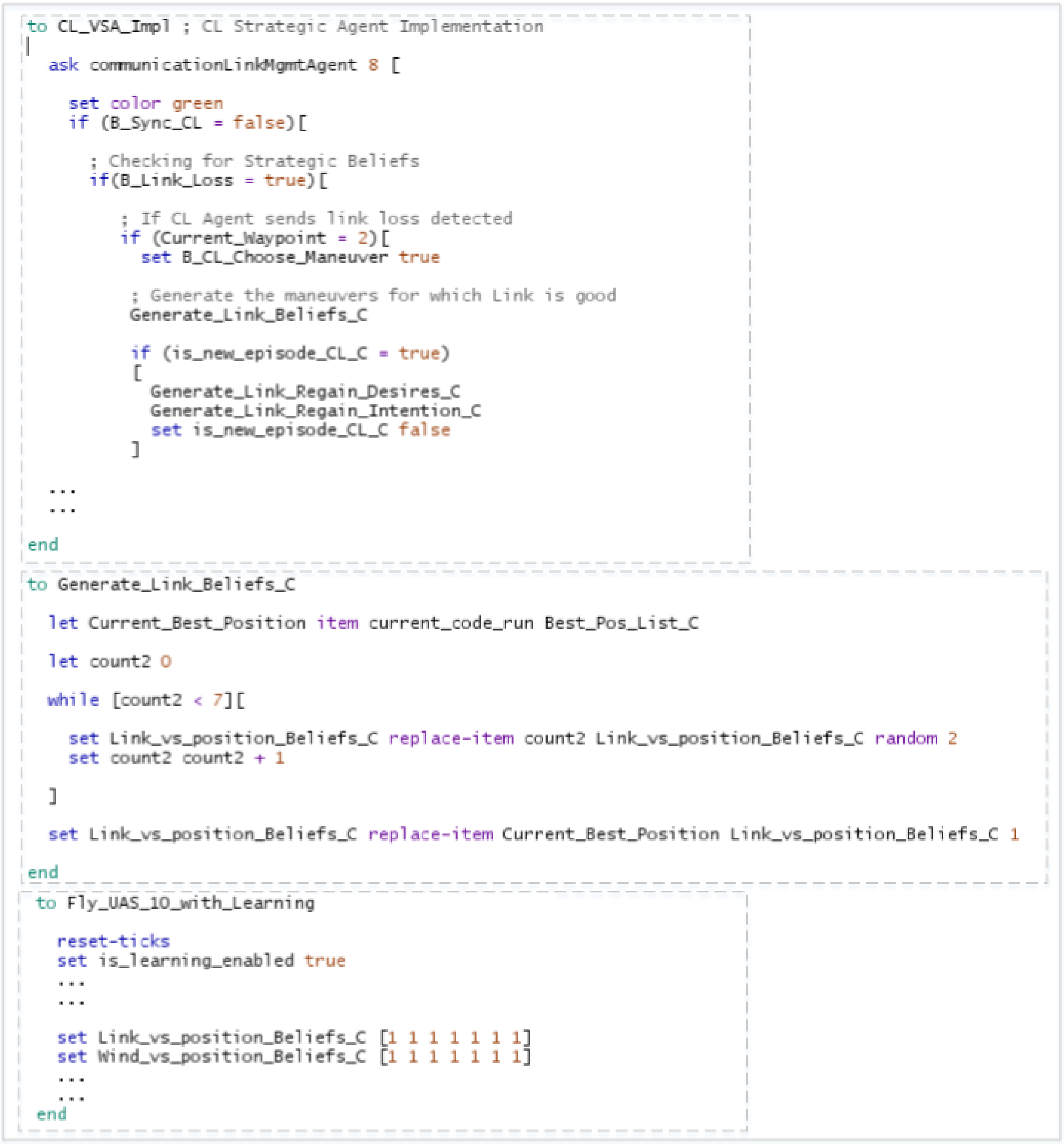
Navigation Agent (NAV_VNA)

The agents are grouped based on their association with the different flight control loops that operate at different timeliness requirements. The FCS_VNA addresses general functions involved in normal flight control: (a) Detection of abnormal events (b) execution of normal flight control laws to keep the platform stable and flying (c) execution of hovering, loitering, and altitude increase/decrease procedures. The FCS_VTA steps in for responses in case of (a) single rotor failure by providing for a recovery procedure (b)two rotor failures by providing for an emergency procedure(c) irrecoverable system faults requiring execution of emergency recovery procedure (d) procedural execution of collision avoidance maneuvers, the actual maneuvers being suggested by a collision avoidance system. The CL_VNA performs general functions related to the management of communication link like setting up of the transmit and receive channels. The CL_VSA is responsible for handling scenarios involving link loss. The CL_VSA exemplifies the BDI model and discovers the best maneuvers using a learning algorithm, to regain the link loss. The FMS_VNA is responsible for the flight management functions like sequencing the flight plan waypoints, adhering to fight plan restrictions, providing the destination waypoint co-ordinates to the NAV_VNA, and general management of mission sensors. FMS_VSA is the manifestation of the BDI model that incorporates learning by providing the best altitudes to fly for the airborne platform considering wind related aspects in our case study. The NAV_VNA navigates the airborne platform by continuously providing the navigation co-ordinates to the FCS_VNA. The MMS_VSA implements the BDI model incorporating learning that takes into account the power management aspects related to the mission.

The mappings of the various VERMILLION framework operational semantics functions to their NetLogo implementation is provided below:

genrteStrgBeliefs - (See sections V.H and VI.A) The upper part of Figure 19 shows the layout of the BDI procedures, while the middle portion shows the "Generate_Link_Beliefs_C" procedure. The lower most part of the figure shows the initialization of the link beliefs at Waypoint C. These indicate the link availability beliefs for the various maneuvering positions of the sUAS.

mapStrgBeliefsToLrngEsmEnts - (See sections V.H and VI.A) As an example, Figure 20 shows how the beliefs in "Link_vs_position_Beliefs_C" are mapped to "Q_Matrix_Link_C", the matrix of elements that provide the placeholders for the learning mechanism. The link beliefs at Waypoint C are binary valued representations of link availability. The matrix elements will hold relative scores for each of the positions as the learning accrues over the sUAS mission execution.

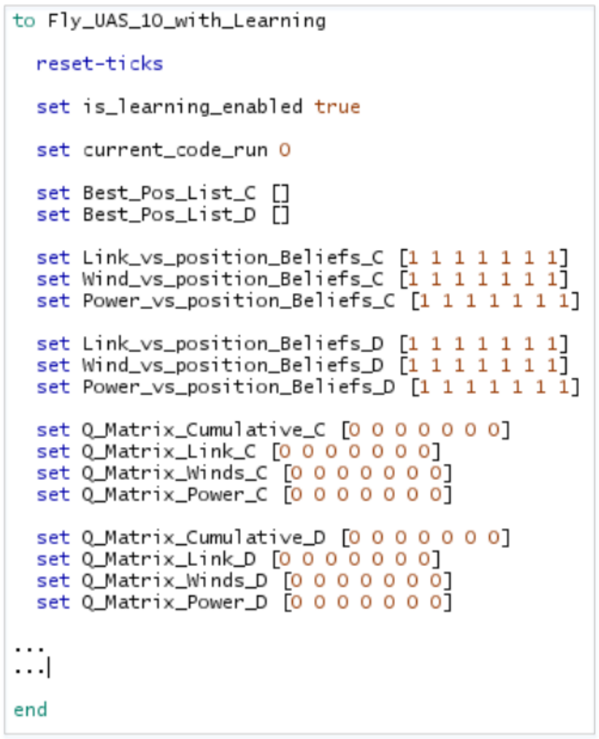


```
to CL_VSA_Impl ; CL Strategic Agent Implementation
|
  ask communicationLinkMgmtAgent 8 [

    set color green
    if (B_Sync_CL = false)[

      ; Checking for Strategic Beliefs
      if(B_Link_Loss = true)[

         ; If CL Agent sends link loss detected
         if (Current_Waypoint = 2)[
           set B_CL_Choose_Maneuver true

          ; Generate the maneuvers for which Link is good
          Generate_Link_Beliefs_C

          if (is_new_episode_CL_C = true)
          [
            Generate_Link_Regain_Desires_C
            Generate_Link_Regain_Intention_C
            set is_new_episode_CL_C false
          ]

  ...
  ...

end
```

```
to Generate_Link_Beliefs_C

  let Current_Best_Position item current_code_run Best_Pos_List_C

  let count2 0

  while [count2 < 7][

    set Link_vs_position_Beliefs_C replace-item count2 Link_vs_position_Beliefs_C random 2
    set count2 count2 + 1

  ]

  set Link_vs_position_Beliefs_C replace-item Current_Best_Position Link_vs_position_Beliefs_C 1

end
```

```
 to Fly_UAS_10_with_Learning

   reset-ticks
   set is_learning_enabled true
   ...
   ...

   set Link_vs_position_Beliefs_C [1 1 1 1 1 1 1]
   set Wind_vs_position_Beliefs_C [1 1 1 1 1 1 1]
   ...
   ...
 end
```

Fig. 19   NetLogo Implementation of Beliefs Generation (Communication Link VSAgent)



```
to Fly_UAS_10_with_Learning

  reset-ticks

  set is_learning_enabled true

  set current_code_run 0

  set Best_Pos_List_C []
  set Best_Pos_List_D []

  set Link_vs_position_Beliefs_C [1 1 1 1 1 1 1]
  set Wind_vs_position_Beliefs_C [1 1 1 1 1 1 1]
  set Power_vs_position_Beliefs_C [1 1 1 1 1 1 1]

  set Link_vs_position_Beliefs_D [1 1 1 1 1 1 1]
  set Wind_vs_position_Beliefs_D [1 1 1 1 1 1 1]
  set Power_vs_position_Beliefs_D [1 1 1 1 1 1 1]

  set Q_Matrix_Cumulative_C [0 0 0 0 0 0 0]
  set Q_Matrix_Link_C [0 0 0 0 0 0 0]
  set Q_Matrix_Winds_C [0 0 0 0 0 0 0]
  set Q_Matrix_Power_C [0 0 0 0 0 0 0]

  set Q_Matrix_Cumulative_D [0 0 0 0 0 0 0]
  set Q_Matrix_Link_D [0 0 0 0 0 0 0]
  set Q_Matrix_Winds_D [0 0 0 0 0 0 0]
  set Q_Matrix_Power_D [0 0 0 0 0 0 0]

...
...|

end
```

Fig. 20 NetLogo Implementation of mapping beliefs to learning elements (Communication Link VSAgent)

learnByMapngLrngEsmEntsToVal - (See sections V.H and VI.A)Figure 21 shows how the real valued elements of the "Q_Matrix_Link_C" are generated by the Q-Learning algorithm that learns about the best positions to regain the communication link. Each matrix element represents the Q value for the mapped position in the possible list of sUAS maneuver positions. Alpha and Gamma are the learning rate and discount factors respectively in the Q-Learning algorithm. max_Q_Link_C denotes the maximum Q value over all actions at the next state.

genrteStrgGoalsWithVal - (See sections V.H and VI.A) Figure 21 also shows how Goals are actually represented within the "Q_Matrix_Link_C". Since the element positions are also representative of the maneuvers, these positions are actually goals in terms of the maneuvers. Notice that in Figure 21, the first line of code assigns the reward values based on "Link_vs_position_Beliefs_C". Therefore the values of the elements are actually representative of the Goals with value.

mapAvsAppToIntentionStrategy, selectIntention - (See sections V.H and VI.A) Figure 22 provides the realization of the function of choosing a strategy for Intention determination by the Communication Link VSA Agent. The Intention strategy mechanism representation for mapAvsAppToIntentionStrategy in this case alters the highest value in the QValue Matrix by an amount epsilon if the difference between the highest and the second highest is less than a predefined delta value (say 5%). The "Generate_Link_Regain_Intention_C" procedure incorporates this mechanism and adjust the

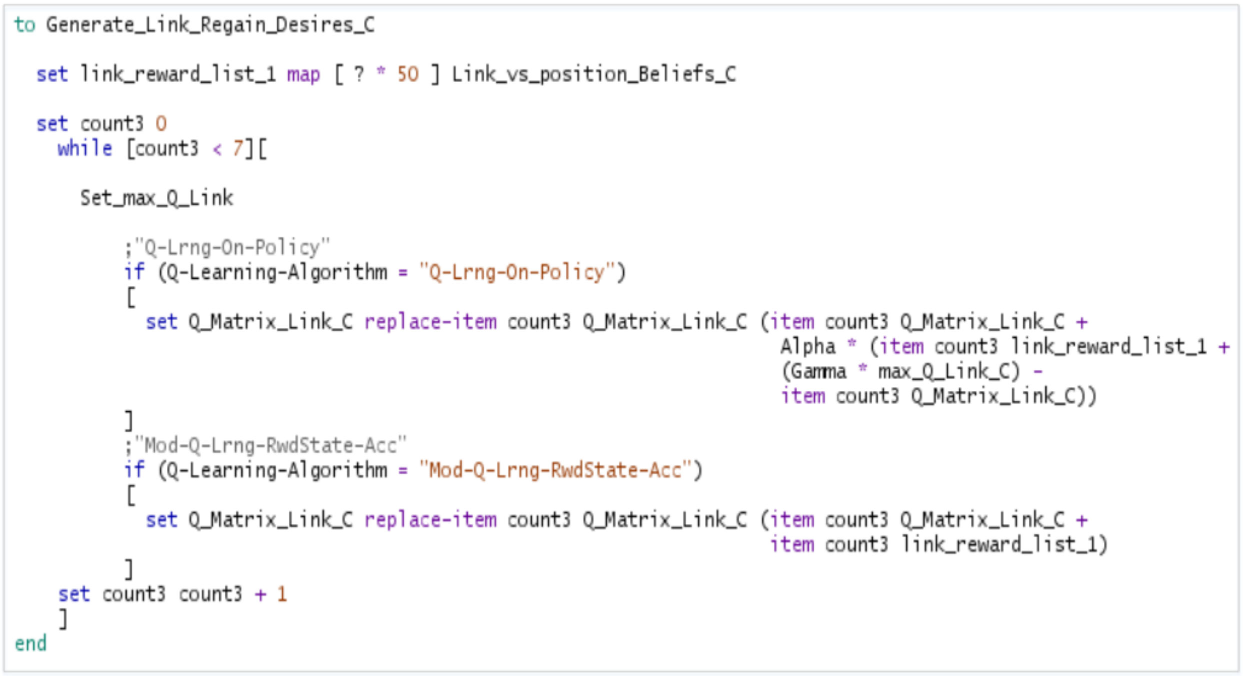

```
to Generate_Link_Regain_Desires_C

  set link_reward_list_1 map [ ? * 50 ] Link_vs_position_Beliefs_C

  set count3 0
    while [count3 < 7][

      Set_max_Q_Link

          ;"Q-Lrng-On-Policy"
          if (Q-Learning-Algorithm = "Q-Lrng-On-Policy")
          [
            set Q_Matrix_Link_C replace-item count3 Q_Matrix_Link_C (item count3 Q_Matrix_Link_C +
                                                                     Alpha * (item count3 link_reward_list_1 +
                                                                     (Gamma * max_Q_Link_C) -
                                                                     item count3 Q_Matrix_Link_C))
          ]
          ;"Mod-Q-Lrng-RwdState-Acc"
          if (Q-Learning-Algorithm = "Mod-Q-Lrng-RwdState-Acc")
          [
            set Q_Matrix_Link_C replace-item count3 Q_Matrix_Link_C (item count3 Q_Matrix_Link_C +
                                                                    item count3 link_reward_list_1)
          ]
    set count3 count3 + 1
    ]
end
```

Fig. 21 NetLogo Implementation of Desire Generation (Communication Link VSAgent)

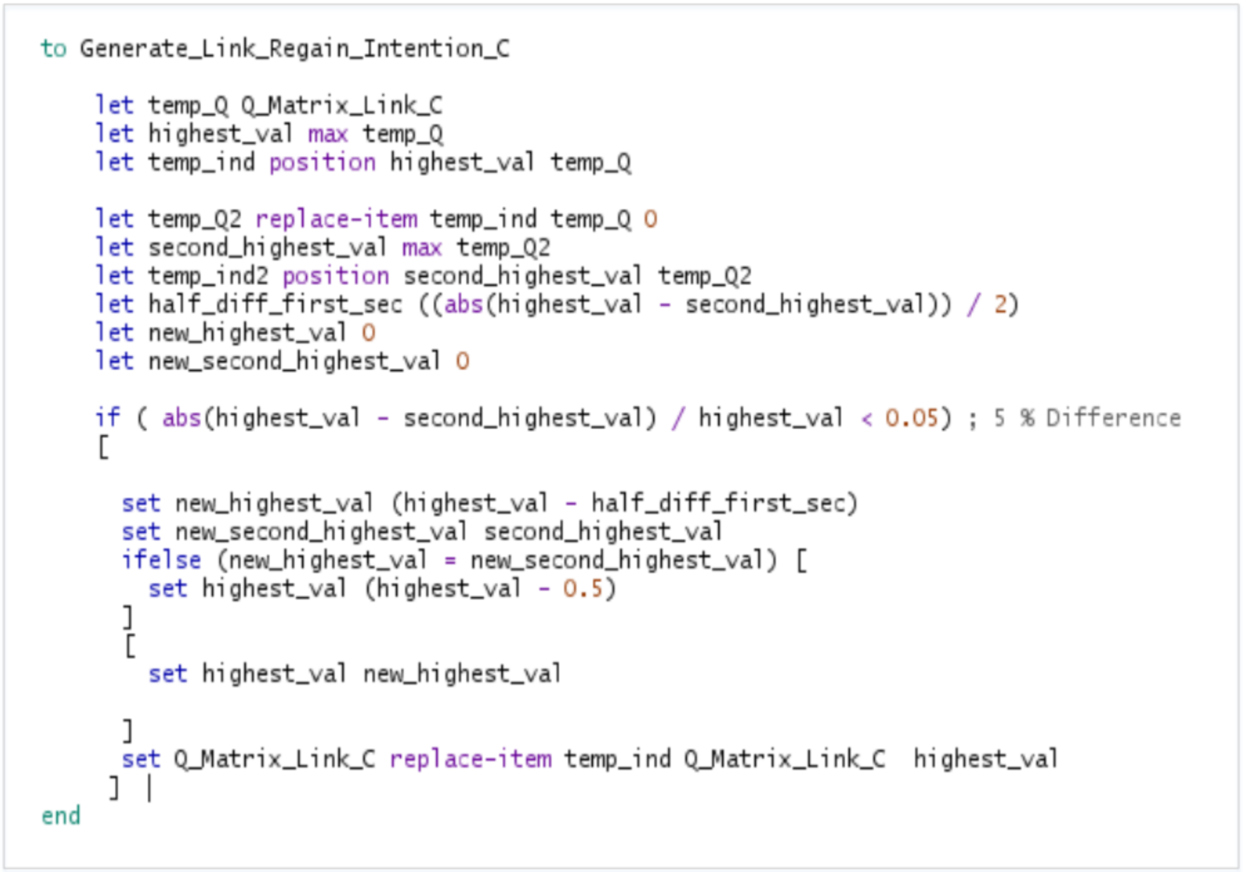
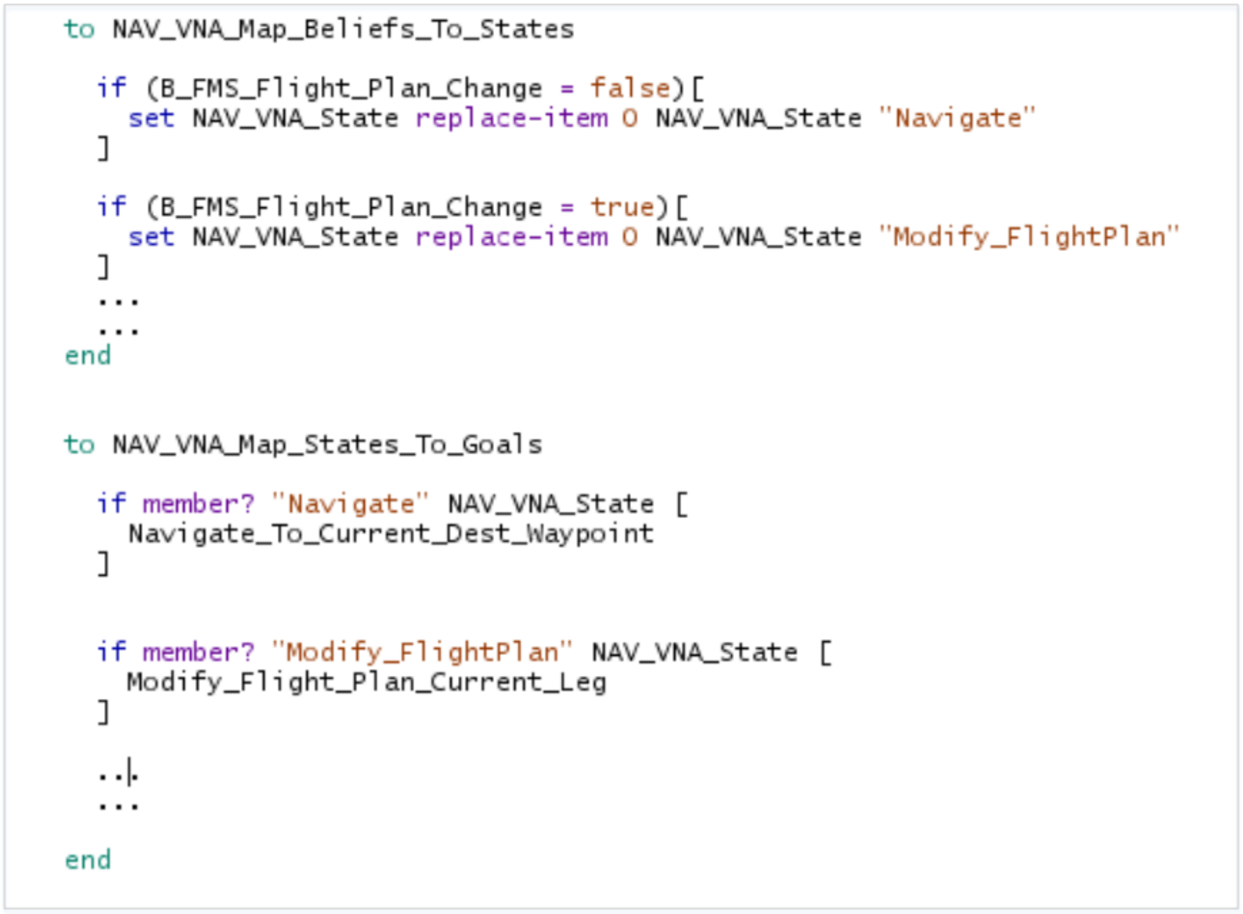

```
to Generate_Link_Regain_Intention_C

    let temp_Q Q_Matrix_Link_C
    let highest_val max temp_Q
    let temp_ind position highest_val temp_Q

    let temp_Q2 replace-item temp_ind temp_Q 0
    let second_highest_val max temp_Q2
    let temp_ind2 position second_highest_val temp_Q2
    let half_diff_first_sec ((abs(highest_val - second_highest_val)) / 2)
    let new_highest_val 0
    let new_second_highest_val 0

    if ( abs(highest_val - second_highest_val) / highest_val < 0.05) ; 5 % Difference
    [

      set new_highest_val (highest_val - half_diff_first_sec)
      set new_second_highest_val second_highest_val
      ifelse (new_highest_val = new_second_highest_val) [
        set highest_val (highest_val - 0.5)
      ]
      [
        set highest_val new_highest_val

      ]
      set Q_Matrix_Link_C replace-item temp_ind Q_Matrix_Link_C  highest_val
     ] |
end
```

Fig. 22 NetLogo Implementation of Intention Selection (Communication Link VSAgent)



```
to NAV_VNA_Map_Beliefs_To_States

  if (B_FMS_Flight_Plan_Change = false)[
    set NAV_VNA_State replace-item 0 NAV_VNA_State "Navigate"
  ]

  if (B_FMS_Flight_Plan_Change = true)[
    set NAV_VNA_State replace-item 0 NAV_VNA_State "Modify_FlightPlan"
  ]
  ...
  ...
end


to NAV_VNA_Map_States_To_Goals

  if member? "Navigate" NAV_VNA_State [
    Navigate_To_Current_Dest_Waypoint
  ]


  if member? "Modify_FlightPlan" NAV_VNA_State [
    Modify_Flight_Plan_Current_Leg
  ]

  ...
  ...

end
```

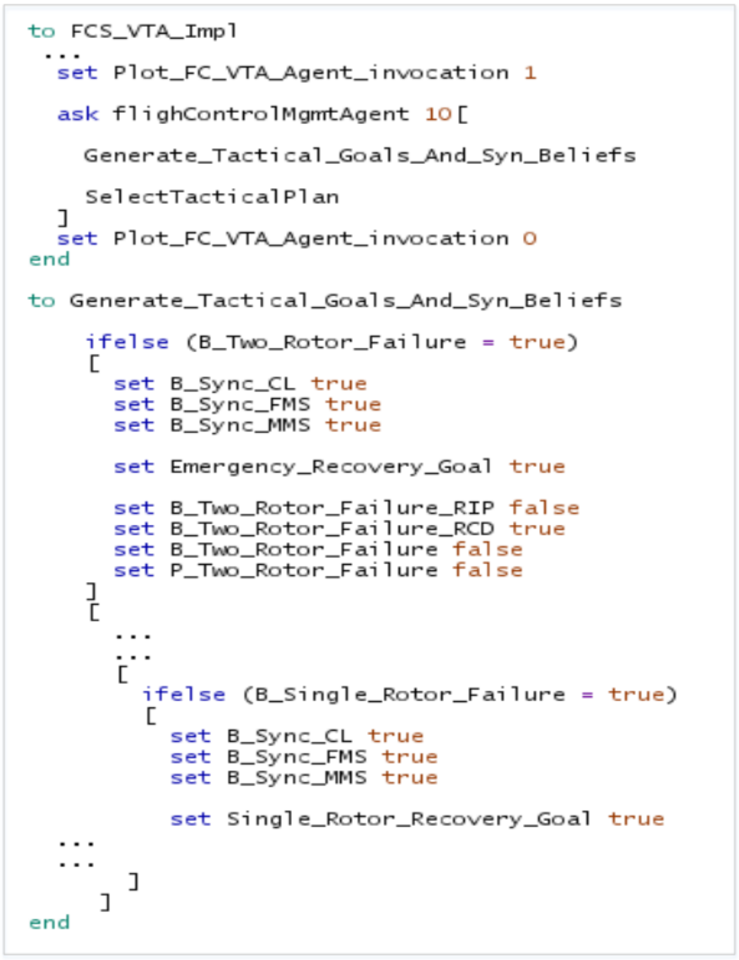
Fig. 23 NetLogo Implementation of Mapping Beliefs to States to Goals (Navigation VNAgent)

Q_Matrix_Link_C so that is contains the appropriate desire selected as the committed one at the end of the procedure execution. The FMS_VSA and MMS_VSA have similar implementations for the intention selection.

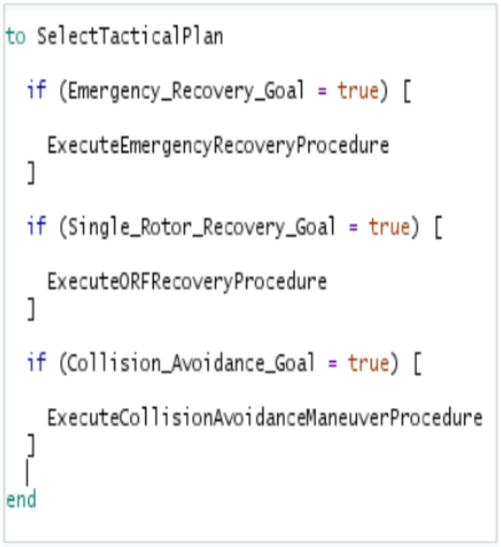
mapNormalBeliefsToStates, mapStatesToNormalGoals - (See sections V.I and VI.B) Figure 23 shows the realizations of these functions for the Navigation Normal Agent and in this case too, they are quite straight forward.

generateTacticalBeliefs, generateTactGoalsAndSynBeliefs, selectTacticalPlan - (See sections V.J and VI.C)Figure 24 shows how the FCS_VTA implements the prioritization of beliefs that require tactical responses. Notice the generation of synchronization beliefs which are used to coordinate responses from strategic agents. The prioritized goals are set in the 'Generate_Tactical_Goals_And_Syn_Beliefs 'procedure by way of processing the events in order. These goals are acted upon in the 'SelectTacticalPlan 'procedure.

generateModeratedOutputs - (See sections V.K and VI.D) Figure 25 shows how the FMS_VSA implements the process of moderating between the other two strategic agents: CL_VSA and MMS_VSA. 'B_CL_Send_Maneuver_to_FMS',

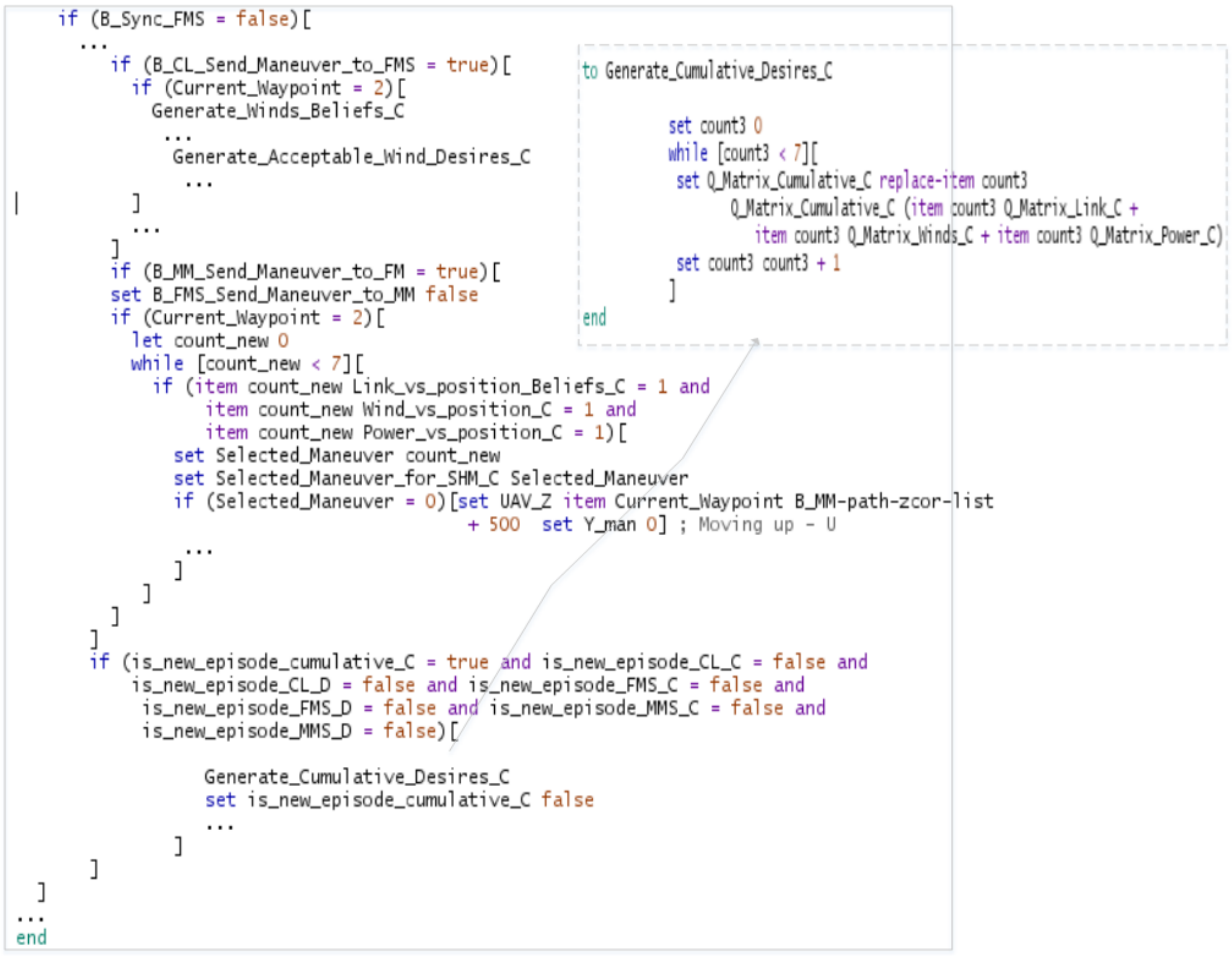


```
to FCS_VTA_Impl
 ...
  set Plot_FC_VTA_Agent_invocation 1

  ask flighControlMgmtAgent 10[

    Generate_Tactical_Goals_And_Syn_Beliefs

    SelectTacticalPlan
  ]
  set Plot_FC_VTA_Agent_invocation 0
end

to Generate_Tactical_Goals_And_Syn_Beliefs

    ifelse (B_Two_Rotor_Failure = true)
    [
      set B_Sync_CL true
      set B_Sync_FMS true
      set B_Sync_MMS true

      set Emergency_Recovery_Goal true

      set B_Two_Rotor_Failure_RIP false
      set B_Two_Rotor_Failure_RCD true
      set B_Two_Rotor_Failure false
      set P_Two_Rotor_Failure false
    ]
    [
      ...
      ...
      [
        ifelse (B_Single_Rotor_Failure = true)
        [
          set B_Sync_CL true
          set B_Sync_FMS true
          set B_Sync_MMS true

          set Single_Rotor_Recovery_Goal true
  ...
  ...
        ]
      ]
end
```



```
to SelectTacticalPlan

  if (Emergency_Recovery_Goal = true) [

    ExecuteEmergencyRecoveryProcedure
  ]

  if (Single_Rotor_Recovery_Goal = true) [

    ExecuteORFRecoveryProcedure
  ]

  if (Collision_Avoidance_Goal = true) [

    ExecuteCollisionAvoidanceManeuverProcedure
  ]
  |
end
```

Fig. 24 NetLogo Implementation of Tactical Goals generation and Tactical Plan Selection (FCS VTAgent)



```
    if (B_Sync_FMS = false)[
      ...
        if (B_CL_Send_Maneuver_to_FMS = true)[
          if (Current_Waypoint = 2)[
            Generate_Winds_Beliefs_C
             ...
             Generate_Acceptable_Wind_Desires_C
              ...
          ]
          ...
        ]
        if (B_MM_Send_Maneuver_to_FM = true)[
        set B_FMS_Send_Maneuver_to_MM false
        if (Current_Waypoint = 2)[
          let count_new 0
          while [count_new < 7][
            if (item count_new Link_vs_position_Beliefs_C = 1 and
                item count_new Wind_vs_position_C = 1 and
                item count_new Power_vs_position_C = 1)[
              set Selected_Maneuver count_new
              set Selected_Maneuver_for_SHM_C Selected_Maneuver
              if (Selected_Maneuver = 0)[set UAV_Z item Current_Waypoint B_MM-path-zcor-list
                                      + 500  set Y_man 0] ; Moving up - U
               ...
              ]
            ]
          ]
        ]
      if (is_new_episode_cumulative_C = true and is_new_episode_CL_C = false and
          is_new_episode_CL_D = false and is_new_episode_FMS_C = false and
           is_new_episode_FMS_D = false and is_new_episode_MMS_C = false and
           is_new_episode_MMS_D = false)[

              Generate_Cumulative_Desires_C
              set is_new_episode_cumulative_C false
              ...
           ]
      ]
  ]
...
end
```

```
to Generate_Cumulative_Desires_C

        set count3 0
        while [count3 < 7][
         set Q_Matrix_Cumulative_C replace-item count3
               Q_Matrix_Cumulative_C (item count3 Q_Matrix_Link_C +
                  item count3 Q_Matrix_Winds_C + item count3 Q_Matrix_Power_C)
         set count3 count3 + 1
        ]
end
```

Fig. 25 NetLogo Implementation of Moderation Mechansim (FMS VSAgent)

'B_MM_Send_Maneuver_to_FM', and 'B_FMS_Send_Maneuver_to_MM'are the variables to synchronize the responses. The joint learning that happens is captured in the 'Q_Matrix_Cumulative_C', shown in the right side of the figure, which accumulates the experiences from the 'Q_Matrix_Link_C', 'Q_Matrix_Winds_C', and 'Q_Matrix_Power_C'that provide the individual learnings of each of the agents.

3. sUAS Verification and Validation

Towards satisfying the DO178C process, for this case study, as a first step we have shown (a) how high-level software requirements are complied with system requirements (b) how low-level requirements/architecture comply with high level requirements. These details are provided in our paper
[5]. In the second step to show compliance of source code with low-level requirements, we provide an example wherein we abstract the implementation to arrive at a state transition model that is checked against CTL specifications laid out in section VII.B.1. The state transition model is predominantly described with boolean variables and is shown in Figure 26. Deterministic responses from agents are elicited by having a scheduler, shown in the top portion of the diagram.

Notice how events requiring real time response are handled by tactical agents transitioning to active states from inner loop state S2. In the context of a link loss event detected in state S12, agents CL_VSA and MMS_VSA provides their responses in state S18, which are eventually received by the FMS_VSA in state S11. FMS_VSA, acting as a moderator in state S21 provides the agreed upon manuever, which is a result of the joint learning that has been accumulated by all three agents. The model in Figure 26 with the application of a labeling function depicts a NuSMV model with atomic propositions appropriately distribtuted over all the states, a snippet of which is provided in Figure 27. The snippet shows a predominantly boolean abstraction of the learning mechanism of the NetLogo implementation which was discussed in section VII.B.2. The transitions of the FMS_VSA acting as a moderator is provided in the top portion of the figure as indicated by the variable 'FMS_VSA_Desire_Intention_Generated', while bottom portion indicates the evolution of the variable 'Agreement_CLMS_FMS_MMS_Reached'in state S17. Using the NuSMV model during the course of verification of the CTL specifications routinely, we came across one case where the NuSMV tool reported a counter example for the property cited in equation 8. The counter example is shown in Figure 28 and indicates a condition where even though there is enough fuel and all agents have a provided a mutually accepted manuevering position to regain the link, the number of attempts to regain the link has exceeded the limit (State:1.313 in Figure 28 from the tool output, equivalent to State S25 in Figure 26). This can potentially lead to a situation wherein the UAS can attempt to regain the link indefinitely at the same geographical location and not progress with the mission. The design flaw was corrected by introduction of recovery state SD2 in Figure 26. In our quest to rationalize our approach to validating non-functional requirements that we adopted in our first case study, we extended the same approach by proposing similar and additional measures based on self-* properties. For the sUAS case study, we have defined and measured the following: (a) Self Configuration Index, SCI_VS_i, a measure that provides insight into the system when a VSA agent or a group of VSA agents fail (b) Self-Healing Index, a measure that can provide insights into how and which faults can degrade a mission (SHI_s_VS) and also how components adapt to failures (SHI_c_VS) (c) Self-Optimization Index, SOI_VS, a measure providing an indication of the efficiency of adaptation via learning mechanisms (d) Self-Protection Indices, providing a measure that indicate safety in particular contexts (SPI_2_VS). Table 3 shows a representative sample of the measures from our simulation experiments with the NetLogo implementation of the AvsUAS application. More details about this work and a comprehensive treatment of the subject is available in [5].

C. Method of Eliciting Abstractions for Model Checking in Case studies 1 and 2

The verification of systems described in case studies 1 and 2 using VERMILLION relies on model checking. In order to arrive at the NuSMV model used in the model checking process, we need to elicit abstractions that will be used for the model. We now describe the methodology to derive the abstractions. We start with the base model in the VERMILLION framework and look at each of the agent Z schemas in the formal model. An observation is that each function of the Z schema changes the state of the agent by producing a data item. Also, since each of these functions are

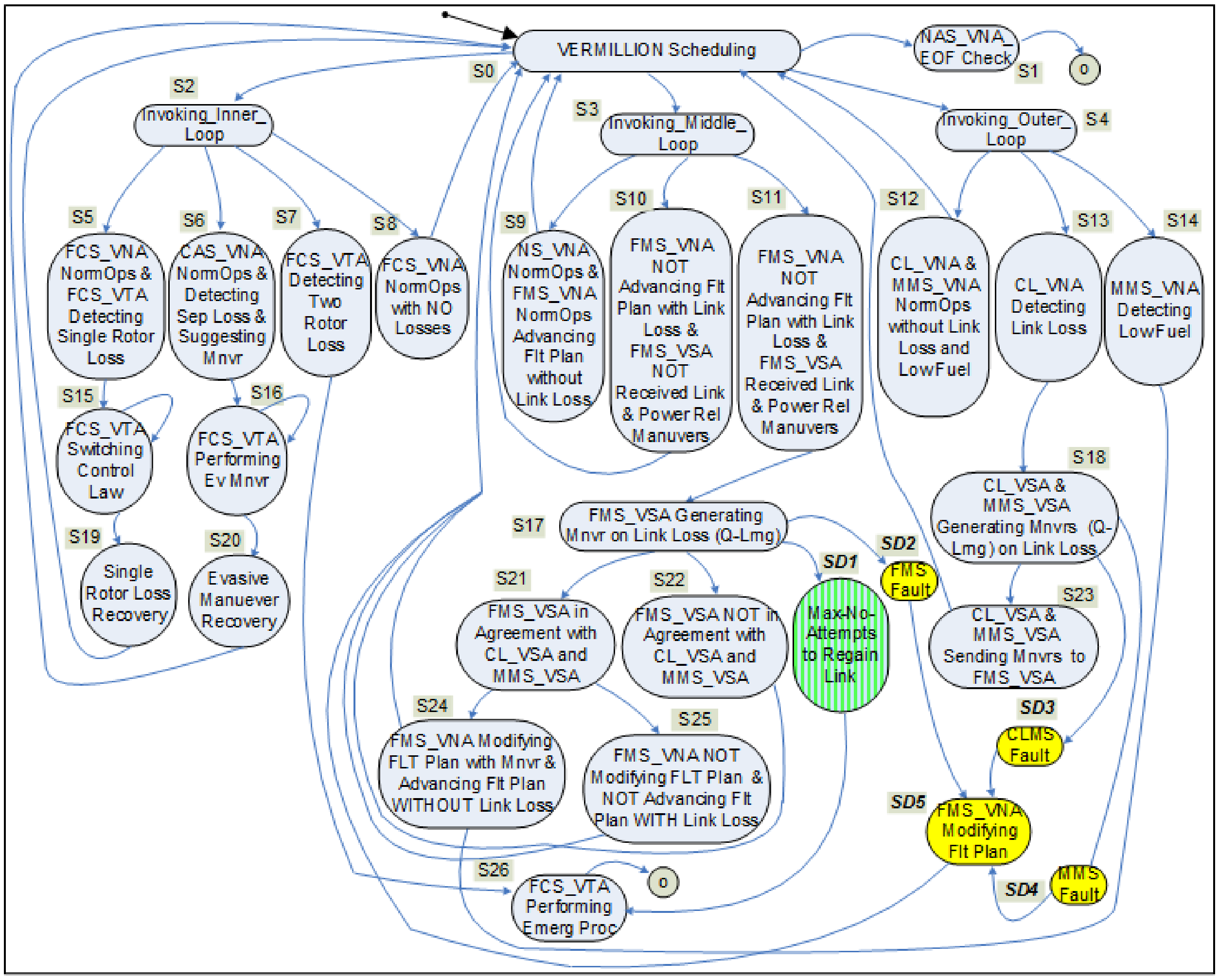
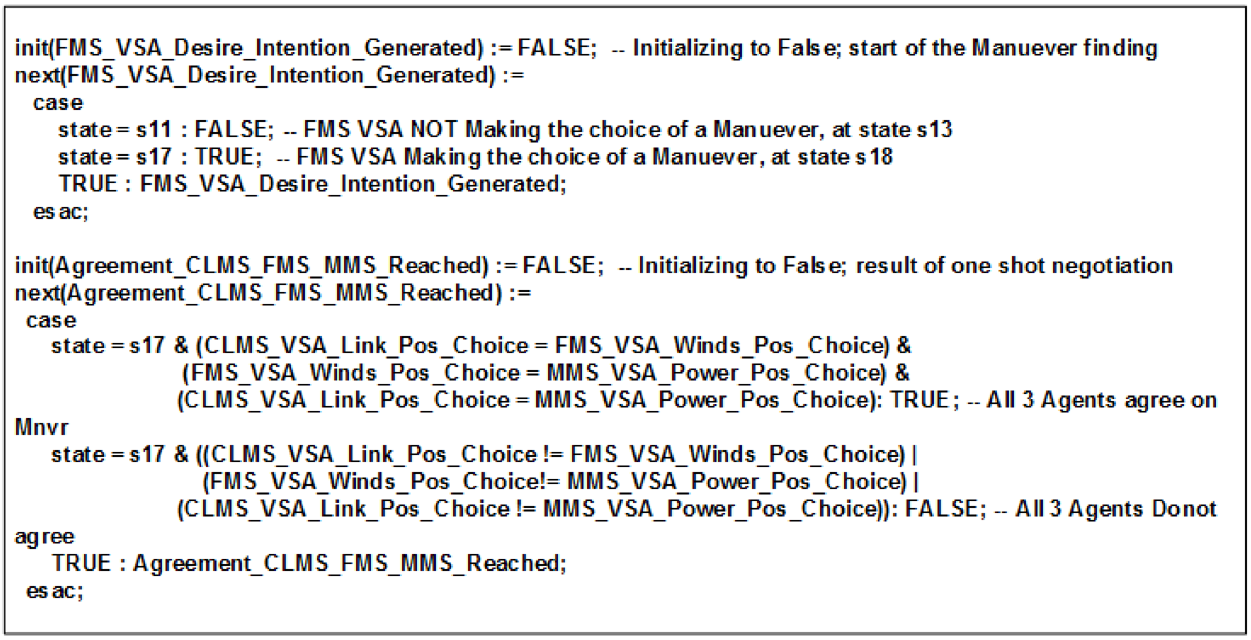

VERMILLION Scheduling
S0
NAS_VNA_ EOF Check
S1
0
S2
Invoking_Inner_ Loop
S3
Invoking_Middle_ Loop
S4
Invoking_Outer_ Loop
S5
FCS_VNA NormOps & FCS_VTA Detecting Single Rotor Loss
S6
CAS_VNA NormOps & Detecting Sep Loss & Suggesting Mnvr
S7
FCS_VTA Detecting Two Rotor Loss
S8
FCS_VNA NormOps with NO Losses
S9
NS_VNA NormOps & FMS_VNA NormOps Advancing Flt Plan without Link Loss
S10
FMS_VNA NOT Advancing Flt Plan with Link Loss & FMS_VSA NOT Received Link & Power Rel Manuvers
S11
FMS_VNA NOT Advancing Flt Plan with Link Loss & FMS_VSA Received Link & Power Rel Manuvers
S12
CL_VNA & MMS_VNA NormOps without Link Loss and LowFuel
S13
CL_VNA Detecting Link Loss
S14
MMS_VNA Detecting LowFuel
S15
FCS_VTA Switching Control Law
S16
FCS_VTA Performing Ev Mnvr
S17
FMS_VSA Generating Mnvr on Link Loss (Q-Lmg)
S18
CL_VSA & MMS_VSA Generating Mnvrs (Q-Lmg) on Link Loss
S19
Single Rotor Loss Recovery
S20
Evasive Manuever Recovery
S21
FMS_VSA in Agreement with CL_VSA and MMS_VSA
S22
FMS_VSA NOT in Agreement with CL_VSA and MMS_VSA
S23
CL_VSA & MMS_VSA Sending Mnvrs to FMS_VSA
S24
FMS_VNA Modifying FLT Plan with Mnvr & Advancing Flt Plan WITHOUT Link Loss
S25
FMS_VNA NOT Modifying FLT Plan & NOT Advancing Flt Plan WITH Link Loss
S26
FCS_VTA Performing Emerg Proc
0
SD1
Max-No-Attempts to Regain Link
SD2
FMS Fault
SD3
CLMS Fault
SD4
MMS Fault
SD5
FMS_VNA Modifying Flt Plan

Fig. 26 Abstract transition system for sUAS, case study 2

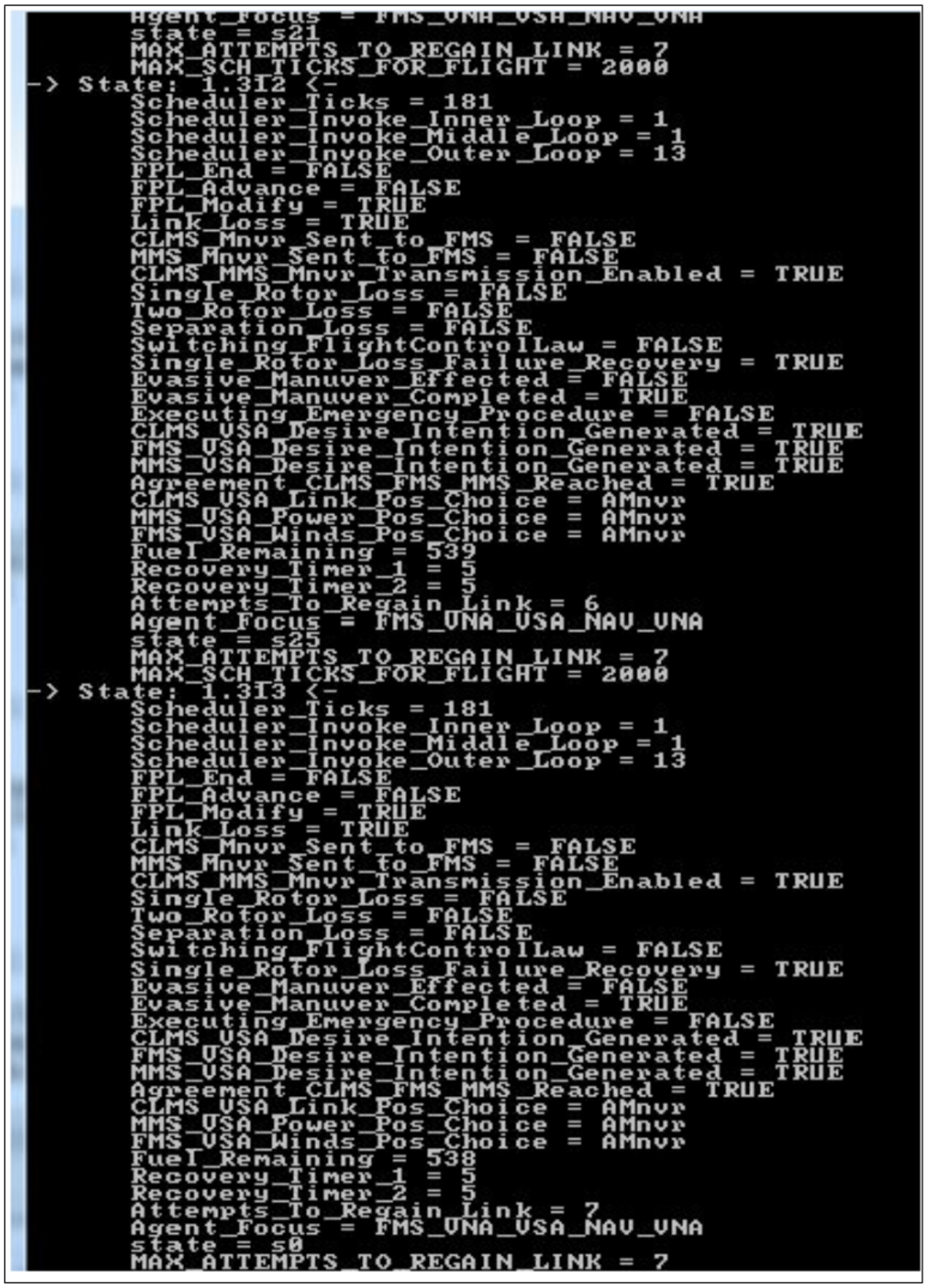


```
init(FMS_VSA_Desire_Intention_Generated) := FALSE;  -- Initializing to False; start of the Manuever finding
next(FMS_VSA_Desire_Intention_Generated) :=
  case
    state = s11 : FALSE; -- FMS VSA NOT Making the choice of a Manuever, at state s13
    state = s17 : TRUE;  -- FMS VSA Making the choice of a Manuever, at state s18
    TRUE : FMS_VSA_Desire_Intention_Generated;
  esac;

init(Agreement_CLMS_FMS_MMS_Reached) := FALSE;  -- Initializing to False; result of one shot negotiation
next(Agreement_CLMS_FMS_MMS_Reached) :=
 case
   state = s17 & (CLMS_VSA_Link_Pos_Choice = FMS_VSA_Winds_Pos_Choice) &
                  (FMS_VSA_Winds_Pos_Choice = MMS_VSA_Power_Pos_Choice) &
                 (CLMS_VSA_Link_Pos_Choice = MMS_VSA_Power_Pos_Choice): TRUE; -- All 3 Agents agree on
Mnvr
   state = s17 & ((CLMS_VSA_Link_Pos_Choice != FMS_VSA_Winds_Pos_Choice) |
                    (FMS_VSA_Winds_Pos_Choice!= MMS_VSA_Power_Pos_Choice) |
                 (CLMS_VSA_Link_Pos_Choice != MMS_VSA_Power_Pos_Choice)): FALSE; -- All 3 Agents Donot
agree
   TRUE : Agreement_CLMS_FMS_MMS_Reached;
 esac;
```

Fig. 27 NuSMV Model for Abstract transition system of sUAS, case study 2

Table 3 Measures enabling Validation of Non-Functional Requirements, sUAS

Measure	Factors affecting Measure	Measure value
SCI_VS_eff	Fail in Leg (1), CL_VSA Fail	0.5
SCI_VS_eff	Fail in Leg (1), FMS_VSA Fail	0.75
SCI_VS_eff	Fail in Leg (1), MM_VSA Fail	
	FMS_VSA Fail, CL_VSA Fail	0.375
SCI_VS_eff	Fail in Leg (6), MM_VSA Fail	
	FMS_VSA Fail, CL_VSA Fail	0.149
SHI_s_VS	CL_VSA Fault, FMS_VSA Fault	
	CL_VSA Fault, Sep_Loss	2.0
SHI_c_VS (CL_VSA)	Number of Episodes = (15) [20] {30}	(0.32) [0.32] {0.286}
SHI_c_VS (FMS_VSA)	Number of Episodes = (15) [20] {30}	(0.30) [0.32] {0.287}
SHI_c_VS (MM_VSA)	Number of Episodes = (15) [20] {30}	(0.32) [0.32] {0.328}
SOI_VS	Number of Episodes = (15) [20] {30}	(1.85) [0.32] {0.286}
SPI_2_VS	Flight Leg = (1) [3] {5}	(.019) [.006] {.005}
	Upper Bound for the measure 20 ms	



```
        Agent_Focus = FMS_VNA_VSA_NAV_VNA
        state = s21
        MAX_ATTEMPTS_TO_REGAIN_LINK = 7
        MAX_SCH_TICKS_FOR_FLIGHT = 2000
-> State: 1.312 <-
        Scheduler_Ticks = 181
        Scheduler_Invoke_Inner_Loop = 1
        Scheduler_Invoke_Middle_Loop = 1
        Scheduler_Invoke_Outer_Loop = 13
        FPL_End = FALSE
        FPL_Advance = FALSE
        FPL_Modify = TRUE
        Link_Loss = TRUE
        CLMS_Mnvr_Sent_to_FMS = FALSE
        MMS_Mnvr_Sent_to_FMS = FALSE
        CLMS_MMS_Mnvr_Transmission_Enabled = TRUE
        Single_Rotor_Loss = FALSE
        Two_Rotor_Loss = FALSE
        Separation_Loss = FALSE
        Switching_FlightControlLaw = FALSE
        Single_Rotor_Loss_Failure_Recovery = TRUE
        Evasive_Manuver_Effected = FALSE
        Evasive_Manuver_Completed = TRUE
        Executing_Emergency_Procedure = FALSE
        CLMS_VSA_Desire_Intention_Generated = TRUE
        FMS_VSA_Desire_Intention_Generated = TRUE
        MMS_VSA_Desire_Intention_Generated = TRUE
        Agreement_CLMS_FMS_MMS_Reached = TRUE
        CLMS_VSA_Link_Pos_Choice = AMnvr
        MMS_VSA_Power_Pos_Choice = AMnvr
        FMS_VSA_Winds_Pos_Choice = AMnvr
        Fuel_Remaining = 539
        Recovery_Timer_1 = 5
        Recovery_Timer_2 = 5
        Attempts_To_Regain_Link = 6
        Agent_Focus = FMS_VNA_VSA_NAV_VNA
        state = s25
        MAX_ATTEMPTS_TO_REGAIN_LINK = 7
        MAX_SCH_TICKS_FOR_FLIGHT = 2000
-> State: 1.313 <-
        Scheduler_Ticks = 181
        Scheduler_Invoke_Inner_Loop = 1
        Scheduler_Invoke_Middle_Loop = 1
        Scheduler_Invoke_Outer_Loop = 13
        FPL_End = FALSE
        FPL_Advance = FALSE
        FPL_Modify = TRUE
        Link_Loss = TRUE
        CLMS_Mnvr_Sent_to_FMS = FALSE
        MMS_Mnvr_Sent_to_FMS = FALSE
        CLMS_MMS_Mnvr_Transmission_Enabled = TRUE
        Single_Rotor_Loss = FALSE
        Two_Rotor_Loss = FALSE
        Separation_Loss = FALSE
        Switching_FlightControlLaw = FALSE
        Single_Rotor_Loss_Failure_Recovery = TRUE
        Evasive_Manuver_Effected = FALSE
        Evasive_Manuver_Completed = TRUE
        Executing_Emergency_Procedure = FALSE
        CLMS_VSA_Desire_Intention_Generated = TRUE
        FMS_VSA_Desire_Intention_Generated = TRUE
        MMS_VSA_Desire_Intention_Generated = TRUE
        Agreement_CLMS_FMS_MMS_Reached = TRUE
        CLMS_VSA_Link_Pos_Choice = AMnvr
        MMS_VSA_Power_Pos_Choice = AMnvr
        FMS_VSA_Winds_Pos_Choice = AMnvr
        Fuel_Remaining = 538
        Recovery_Timer_1 = 5
        Recovery_Timer_2 = 5
        Attempts_To_Regain_Link = 7
        Agent_Focus = FMS_VNA_VSA_NAV_VNA
        state = s0
        MAX_ATTEMPTS_TO_REGAIN_LINK = 7
```

Fig. 28 Counter example - property violation of sUAS specification, case study 2

also mapped to the NetLogo implementation, we arrange these side by side as shown in Tables 4 and 5. Simultaneously, we take each CTL specification associated with the agent and examine whether any part of the specification could be used for an abstraction that is relevant and goes along seamlessly for the function identified in the Z schema. With this initial arrangement of the functions from the Z schema, NetLogo code, and CTL predicates, we apply the abstraction techniques provided in section III, to elements identified in columns 2 and 3. The technique applied to elicit the abstraction is documented in column 5 and the resulting abstraction that is elicited is noted in the last column of the table. The abstractions are then checked whether they are sound - this involved manually verifying that they include all behaviors of the system. The counter examples produced during the model checking process could be spurious and they need to be validated by simulating the example in the NetLogo model.

Case Study 1 : AFPS Consider Table 4. As an example for VSA (first row item entry in the table), in arriving at the boolean variables Hd_Wx, Md_Wx, and Ld_Wx, all of the data items: stb, stg, lem, lee, leev, and stgv which evolved through their respective generation/modification functions listed in column 2, we have used data abstraction, since the goals with values which are finally generated can be abstracted as Highest Desire, Moderate Desire, and Lowest Desire Goals. Notice that the CTL specifications from related equations 3 and 5 contain the predicate stated in column 4 and hence the cone of influence technique is applied to record the Fuel_Remaining as one of the abstracted variables in column 6. The second row entry shows how predicate abstraction is used to arrive at the Commitment_Maintained boolean abstraction for VSA's intention selection. The next row shows how the Nego_Done_Navigate boolean abstraction is elicited considering the VMA.

Case Study 2 : AVsUAS Consider Table 5. The first row entry pertains to how the scheduling mechanism is abstracted into the model. The inner, middle and outer loops are abstracted as integers with a range value and use the modulus operator in the NuSMV model. The NetLogo procedure implementation counts the ticks in the scheduler and also uses a similar modulo counting mechanism. The second row shows a one to one mapping between the CTL specification predicate *TwTRtrLoss* (see equation 7) and the NuSMV abstraction. This is in agreement with the NetLogo procedure update_FCS_Beliefs which also uses the same variable. The third row shows an example of how the outputs from three VNA functions in column 2 are reconciled with the NetLogo procedure that implements the functionality specified in equation 7 wherein two separate aspects (collision avoidance manuever and its timeliness requirement) are captured. Observe that the timeliness requirements is abstracted as an integer with a range of values, tracking the state S16 in in Figure 26.

Brief Note on guidelines for constructing the state transition system : The basic BDI Interpreter cycle provided in section I is examined in conjunction with the various functions provided by each type of agent/entity of sections V.H, V.I, V.J, V.K, and V.L. The implementation code is also cross checked with these functions. A state would normally exist between each step of the BDI interpreter cycle. However, one would only explicitly capture interesting states and drop others. The CTL specifications can serve as a check to reconcile the various states in state transition system.

VIII. Related Work, Comparison, and Discussion

A. Requirements

Our work on Self-Adaptive Software encompasses many areas: Requirements, Architecture/Design, Verification of functional requirements and Validation of non-functional requirements. In order to gain insights at a very broad level, we looked at [25] to quickly assess the spectrum encompassed by adaptive-software. In the context of requirements and specification capture for avionics systems, we have looked at efforts involving usage of different formalisms. [80] provide a method using PVS and targets functional and safety requirements. The use of RSML for a Traffic Collision Avoidance System and RSML-e for a flight Guidance system is provided as an example in [81]. [82] provide usage of formal specifications captured as Java classes and the use of design by contract principles that formally checks the application for a flight management system. A model based approach to capturing specifications with a focus on model properties (relational extension of first order logic) is provided in [83]. [84] is a fairly recent report on the use of ACSL for formal specification of low level requirements. While all of the works cited earlier have their own strengths, our treatment of requirements closely examines them using aspects provided in equation 1 targeted for avionics domain, with a desire to have a pragmatic line of attack. Our exploration of literature on these aspects led us to works cited in [85–88] that address concerns that we have raised, but our approach holistically is quite different. A survey of existing literature provided in [89] provides several methods for the eliciting [RqFAd Part], the prominent ones being GORE (Goal-Oriented Requirements Engineering) including KAOS (Knowledge Acquistion in autOmated Specification) methodology and Goal Based Requirements Analysis Method, Non Functional Requirement (NFR) framework, i* framework, Tropos, RELAX, and GBRAM (Goal Based Requirements Analysis Method). Although these requirements processes and methodolgies exist, we are primarily interested in methods that explicitly deal with safety, real-timeliness, deterministic responses, and monitoring, using filters provided in equation 1. In this context, we have come across methodologies that have a flavor similar to ours in [90–92]. Although TCTL [93] could have been used, we found that it is much simpler and effective to use CTL: (a) we could sufficiently formulate specifications that capture problem domain requirements (b) ‘CTL model checkers’ were effective enough to ensure that all aspects of requirements that we were interested are verified. In this context, it is to be mentioned that our abstraction variables provide the necessary representations to capture the specifications accurately in the context of timeliness, safety, and deterministic responses.

Table 4 Extracting Abstractions for Model Checking - Case Study 1

Agent	Z schema function with [data Item] modified/produced	NetLogo Code Procedure	CTL Spec predicate	Abstraction Technique	Nusmv Model Abstraction Element
VSA	genrteStrgBeliefs [stb], genrteNofStrgGoals [stg], mapAvsAppToLrngEsmMethod [lem], mapStrgBeliefsToLrngEsmEnts [lee], learnByMapngLrngEsmEntsToVal [leev], genrteStrgGoalsWithVal [stgv]	Generate_Weather_Belief, Generate_Weather_Desire	$Fuel\_Remaining > Qty\_Reqd\_Reach\_LWPT$	Data Abstraction Cone of Infl.	Hd_Wx, Md_Wx, Ld_Wx, Fuel_Remaining
VSA	selectIntention [stgv]	Generate_Weather_Intention	None	Predicate Abstraction	Commitment_Maintained
VMA	generateModerationBeliefs [mb], selectModerationPlan [mopl], generateModerationGoalsWtSources [mog, ms], selectModerationPlan [mopl], generateModeratedOutputs [mop]	Negotiate_Stage_1, Negotiate_Stage_2	$(HDWx \lor MDWx \lor LDWx)$	Predicate Abstraction	Nego_Done_Navigate

Table 5 Extracting Abstractions for Model Checking - Case Study 2

Agent Entity	Z schema function with [data Item] modified/produced	NetLogo Code Procedure	CTL Spec predicate	Abstraction Technique	Nusmv Model Abstraction Element
VSch	VermillionInnerLoopProcSchedule, VermillionMiddleLoopProcSchedule, VermillionOuterLoopProcSchedule	Fly_UAS_10_with_Learning	None	Data Abstraction	Scheduler_Invoke_Inner_Loop, Scheduler_Invoke_Middle_Loop, Scheduler_Invoke_Outer_Loop
VTA	generateSafetyBeliefs [sb]	update_FCS_Beliefs	*TwTRtrLoss*	Data Abstraction	Two_Rotor_Loss
VNA	generateNormalBeliefs [nb], mapNormalBeliefsToStates [as], mapStatesToNormalGoals [ng]	FCS_VNA_Impl	*CSpLoss* *MvrRcvrTm*	Data Abstraction Predicate Abstraction	Separation_Loss Recovery_Timer_2

B. Architecture and Design

In the area of architecture and design we have looked at works addressing agent architectures for Avionics and related systems. Our earliest reference on this topic is [94] which provides directions on architecting systems, especially approaches to dynamism at architectural level, using two exemplar systems, a C-2 Style architecture for logistics and Weaves for a realtime stereo tracking system. While we do pickup principles of adaptation management from this work, VERMILLION is more formal and tailored towards avionics applications. In order to guide us through a structured and orderly approach on the subject, we used [22], which is a survey paper on engineering approaches to self-adaptive software, as an index to search through literature on the subject. We have looked at [95] which uses model driven approach, context awareness and separation of concerns while architecting VERMILLION. This work details the MUSIC framework which is intensely focussed on middleware adaptation; however, the ideas of separating business logic is similar to ours for avionics applications. [96] presents an adaptive architecture framework set in the domain of the automobile industry and has similar top level focus areas as ours: safety and criticality. However, SafeAdapt relies on a platform core that encapsulates the basic adaptation mechanisms which slightly differs from our approach. [97] uses the Prometheus methodology and presents an architecture with several levels of abstraction with a case study of an ATC simulator. Our approach is more rigorous. [98] provides an approach similar to ours, for the architecture of each agent from a top level perspective. [99] provides details on the Brahms style modeling and has an approach similar to our philosophy. However our approach to architecture and design is slightly different and there are other aspects in the areas related to validation that make VERMILLION different. [100] provides an interesting work based on the Gaia methodology, that uses a meta-model and liveness expressions for specifying the architecture. Our work provides abstractions at a higher level and subsumes entities like roles and protocols. [101] provides a good reference for the design of MAPE-K loops characteristic of adaptive systems. However, there are many areas that need to be viewed differently for the avionics domain. [102] is an extensive work and describes the AGENTFLY framework set in the context of UAVs.

C. BDI for Avionics

Early work in the area of using BDI in the area of aircraft related systems is dMARS, where it was used in NASA's space shuttle malfunction handling [103] and Australian Air Traffic Control [104]. Our BDI agents are rooted in a philosophy akin to dMARS and are targeted for similar usage environments. [105] provides excellent insights for a design of a framework of BDI agents for a domain similar to ours and also lays out strong arguments for using the BDI model of agency; we have implicitly absorbed the suggestion on decision making fidelity from this exercise. Yet another excellent source is [106] which details the development of TDF, a comprehensive framework addressing issues of tactical decision making by autonomous systems in dynamic environments. Our work though has similarities with this, differs in several areas: we capture requirements formally while also specifying the components of the framework formally in

Z. The usage of a distributed market mechanism amongst a set of agents using the BDI model is demonstrated for a flight conflict detection Algorithm for UAS in [107]. Our framework differs in several aspects: (a)Not all agents are co-ordinators (b) The aspect related to predetermined sequence of announcing agents is handled more deterministically and periodically in our system (c) The bid value calculation is analogous to our utility value generation but significantly differs in the approach, in that our system provides a plug and play placeholders to incorporate any of the mechansims. [108] is a recent work on using BDI agents for decision making in the context of collaborative strategic air traffic flow management, with experimental data from the model being made available. VERMILLION is projected as a generic framework for airborne systems and can easily be extended for similar domains (non-airborne) since many parts of the framework can be tailored.

D. Learning in Avionics

Artificial Intelligence techniques like machine learning are slowly gaining ground for avionics systems. [109, 110] report on early work on adaptive and intelligent systems for cockpit automation. NASA's experiments on an intelligent context aware system for improving the pilot's situational awareness is detailed in [111] and employs the Enhanced Algorithm for Reflex Learning (EARL). VERMILLION framework is extensible in this aspect by allowing for a library of learning algorithms via its strategic agents. A comprehensive NASA report on the usage of intelligent flight planning and guidance is provided in [112]. Our publication [6] details how VERMILLION is different from this work. [113] is a very recent work on using machine learning techniques for failure detection in avionics and so is [114] where learning by imitation of pilots approach using artificial neural networks is reported. [115, 116] are applications of reinforcement learning on onboard avionics systems. Since VERMILLION takes a broader solution framework approach, the techniques cited in all of these could be adopted for any avionics application. Related work not directly related to onboard avionics are : [117] providing how learning could be used to predict aircraft performance. [118] is a current report of how EuroControl is using deep learning with neural networks to enable Air traffic flow and capacity Management. [119] is an interesting and recent tutorial outlining areas where intelligence and learning can be embraced in aviation, possibly incorporated into the plethora of avionics systems and applications.

E. Modeling Notation

At the base level, we examined a number of agent methodologies for constructing agent systems like PASSI, Gaia, ADELFE, Prometheus, Tropos, and INGENIAS [22, 120]. While these methodologies provide a canvas to generate artifacts, we need a more precise language and notation that provides seamless transition to correct design and analysis of the system modeled. Formal languages fulfill the need for capturing the specifications unambiguously and hence we looked at algebraic specification languages like CASL, OBJ, Clear, Larch, ACT-ONE, and LOTOS [121]. However, we found that model based specifications are more expressive than the algebraic type specification [122]. In the declarative category, we looked at number of variants based on lambda-calculus. In the model based category we looked at Event-B, Z, VDM, Lustre, and SCADE (more of a modeling environment) [123]. VDM is a language in which the specification structures are not fixed but provides support for object orientation and concurrency. Z provides a number of advantages even though it does not directly provide support for timing aspects and concurrency - it is domain independent, provides tool support, helps capture model constructs precisely, and obtain refinements that are easily tractable. In terms of syntax and structure Z does not differ much from VDM and hence Z was chosen for all of the reasons mentioned herein.

F. Verification and Validation

We used two main strategies to research work on this topic. In the first strategy, we searched the literature on verification and validation aspects related to systems using the BDI model and formal methods like model checking. In the second strategy we looked for verification methods for adaptive systems in avionics and also verification related to systems using agents in avionics or similar systems where safety, timeliness, and determinism was addressed. In order to aid our second strategy, we used [22] to uncover important work that has settings similar to ours. In adopting the first strategy we came across early work at NASA [124] which provided some directions in the area of model checking for autonomous software. [125] presents Gwendolen, a prototype language used in developing an Agent Infrastructure language which enables model checking. [126, 127] are applications demonstrating the use of Gwendolen. The language provides what is known as MCAPL interface that allows one to inspect agents to deduce agent properties, enabling model checking. Our approach is quite different from this, as delineated in section VII.C. An approach which uses a BDI based MAS model along with Java implementation of its semantics is provided in [128]. Our work does not basically work from this viewpoint and directly uses the abstracted model culled from CTL requirements, implementation, and the framework. In [129], the authors use techniques that automatically translate AgentSpeak (BDI language) programs into LTL language which is then used for modelchecking. The CASP (Checking Agent Speak Programs) tools developed translate an AgentSpeak agent or multiagent system into both PROMELA and Java, which are then passed onto Spin and Java PathFinder model checking tools respectively. Our work, though parallels this in some areas, takes a different approach since we look at requirements, reference architecture, and implementation for arriving at the models. Also the abstraction method akin to program slicing used in their work is not utilized in VERMILLION.
We now present details about works that we uncovered using the second strategy. The work in [130] is a recent case study using the MCMAS tool and is set in the Air Traffic Management / Avionics domain, where we find some similarities with our work (model checking formulae). However, several aspects like real-timeliness and deterministic response are not available although safety aspects have been explicitly dealt with. [91, 131] provides details on using timed automata with Uppaal, a tool that aids in verifying flexibility and robustness properties set in the context of a self-adaptive system (traffic monitoring). They discuss model checking related to agent reorganization and self healing at subsystem level and report the increase in time required for checking when the number of agents involved increases. Part of our work

takes a similar approach but encompasses the entire framework. [132] uses a Restore Invariant approach to model and detect misbehavior at runtime, providing for an area of correct behavior in which the system shows the expected properties, using observer/controller architectures (adaptive production cell and autonomous virtual power plants). Our approach is significantly different from this. We also discovered that [133] is an article that broadly attempts to propose a combination of an architecture and logical framework to verify agent behaviors. [134], [14], and [135] provided us additional insights and guidelines for our approach. [136] and [137] provide examples from the industry and our work has a lot of similarities with the approach in [137], especially the case study on model checking (Chapter 4). [92] is set in the domain of air traffic control and uses LTL. We have used a similar approach. Work in [12] relates to an avionics display system and provides guidance on state space reduction techniques while using model checking. Our work uses similar principles. [138] discusses issues of learning convergence and stability which is important when learning algorithms are employed. These issues can be mitigated to some extent by using the two phase principles outlined for the STPA process in [139]. [140] provides usage of tools for deductive, abstract interpretation, and model checking techniques for formal verification, and suggests a judicious mix of testing and formal verification as the way forward for avionics products. [141] uses LTL model checking and has an approach similar to ours, but our approach covers formal aspects of scheduling and also covers aspects of validation. We explored the possibility of runtime verification as in [142] but consciously chose to avoid it considering the domain that we operate in. In adopting the third strategy we identified FORMS [143] which also uses Z specification for the models. However, our approach is different and considers certification and validation aspects. [144] brings forth aspects related to guarantees in the context of hierarchical control loops and quality of service analysis which is addressed by VERMILLION. We also looked at [145] which defines AdaCTL, a temporal logic for meeting resilience requirements, verifying dynamic software product lines, but found that for the domain that we operate in, this may not be easily fit in. [146] presents an interesting approach for hazard analysis considering timing properties and illustrated with an examplar case study from RailCab project.

G. Evaluation Approaches

[147] is a good reference to arrive at parameters that will evaluate our framework. [148] provides measures for evaluating a self adaptive system, especially the SAFU unit. VERMILLION framework makes available a set of measures that is similar but more appropriate for the avionics domain. Several measures for evaluating autonomic computing systems are proposed in [149] and we cover most of them in VERMILLION, of course with different viewpoints that provide more insights into the system. [150] provides insights into the Self-* properties which may be used to arrive at some meaningful measures.

IX. Conclusions and Future Work

The work presented in this paper is a summary of the work that we have been performing in the area of self-adaptive software specifically for the avionics domain. The field of activity encompasses several sub areas like requirements, architecture, design, verification, validation, and also evaluation of these systems. Each sub area poses several challenges and we have attempted to overcome many problems and the proposition of VERMILLION framework is testimony to this. Certification of systems in this domain is a central aspect that needs to be kept in focus during all cycles of product development. Our approach to requirements specification formally using CTL, focusing on key aspects of realtimeliness, determinism, and safety while at the same time keeping it simple enough for a pragmatic and scalable approach, to be used by the majority software development community is the hallmark of our methodology. We believe that agent technology is a good approach to realizing adaptive software and the BDI model of agency though simple, lends itself as an attractive proposition for several reasons outlined in previous sections of this paper. We have separated the tactical-strategic aspects and provided means to moderate evolving desires and monitor the entire system, addressing aspects of safety, determinism, and real-timeliness. The formal specifications captured in Z are comprehensive enough for further development and implementation of the system. Recognizing that several areas of opportunities seem to emerge for incorporating artificial intelligence on onboard avionics, VERMILLION is favorably positioned to accomodate them via appropriate placeholders in strategic BDI agents. In order to demonstrate the practicality and efficacy of applying this framework, two case studies positioned for onboard avionics systems are presented with implementations. We have conducted experiments in each of the case studies and collected data for some simple yet novel measures to evaluate the system.

References

[1] (ISTC), I. S. T. C., "ROADMAP FOR INTELLIGENT SYSTEMS IN AEROSPACE," AMERICAN INSTITUTE OF AERONAUTICS AND ASTRONAUTICS (AIAA), 2016, pp. 1–111.

[2] Philipp Helle, C. S., Wladimir Schamai, "Testing of Autonomous Systems - Challenges and Current State-of-the-Art," *26th Annual INCOSE International Symposium (IS 2016)*, Winter Simulation Conference, 2016, pp. 150–158. URL `https://doi.org/10.1002/j.2334-5837.2016.00179.x`.

[3] Goodloe, A., "Challenges in the Verification of Flight-Critical Systems," *CPS V&V I& F Workshop 2014- Talks*, National Science Foundation, 2014. URL `www.ls.cs.cmu.edu/CPSVVIF/slides/AlwynGoodloe.pptx`.

[4] Cheng, B. H., and et al, "Software Engineering for Self-Adaptive Systems," Springer-Verlag, Berlin, Heidelberg, 2009, Chaps. Software Engineering for Self-Adaptive Systems: A Research Roadmap, pp. 1–26. doi:10.1007/978-3\-642\-02161\-9_1, URL `http://dx.doi.org/10.1007/978\discretionary{-}{}{}3\discretionary{-}{}{}642\discretionary{-}{}{}02161\discretionary{-}{}{}9_1`.

[5] Kashi, R. N., D'Souza, M., and Kishore, K. R., "Incorporating Formal Methods and Measures Obtained through Analysis, Simulation Testing for Dependable Self-Adaptive Software in Avionics Systems," *Proceedings of the 10th ACM India Conference*, ACM, New York, NY, USA, 2017. doi:https://doi.org/10.1145/3140107.3140128.

[6] Kashi, R. N., D'Souza, M., Baghel, S. K., and Kulkarni, N., "Incorporating adaptivity using learning in avionics self adaptive software: A case study," *2016 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2016, Jaipur, India, September 21-24, 2016*, 2016, pp. 220–229. doi:10.1109/ICACCI.2016.7732051, URL `https://doi.org/10.1109/ICACCI.2016.7732051`.

[7] Kashi, R. N., D'Souza, M., Baghel, S. K., and Kulkarni, N., "Formal Verification of Avionics Self Adaptive Software: A Case Study," *Proceedings of the 9th India Software Engineering Conference*, ACM, New York, NY, USA, 2016, pp. 163–169. doi:10.1145/2856636.2856658, URL `http://doi.acm.org/10.1145/2856636.2856658`.

[8] Smith, T., Barhorst, J., and Urnes, J. M., *Design and Flight Test of an Intelligent Flight Control System*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 57–76. doi:10.1007/978\-3\-642\-10690\-3_4, URL `https://doi.org/10.1007/978\discretionary{-}{}{}3\discretionary{-}{}{}642\discretionary{-}{}{}10690\discretionary{-}{}{}3_4`.

[9] Marquardt, O., Riedlinger, M., Ahmadi, R., and Reichel, R., "Autonomous peripherals integration for an adaptive avionics platform," *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 2016, pp. 1–10. doi:10.1109/DASC.2016.7777953.

[10] Bosworth, J., and Williams-Hayes, P., "Flight test results from the NF-15B IFCS project with adaptation to a simulated stabilator failure," *Proceedings of AIAA Infotech@Aerospace Conference, AIAA-2007-2818*, 2007.

[11] Thompson, M. O., and Welsh, J. R., "Flight test experience with adaptive control systems," *Proceedings of Advanced Control System Concepts, AGARD*, 1970, pp. 141–147.

[12] Whalen, M., and et al, "ADGS-2100 Adaptive Display & Guidance System Window Manager Analysis," Tech. rep., NASA, 2014. URL `http://shemesh.larc.nasa.gov/fm/fm\discretionary{-}{}{}collins\discretionary{-}{}{}pubs.html.`

[13] Goodloe, A., "Challenges in High-Assurance Runtime Verification," *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I*, 2016, pp. 446–460.

[14] FAA, "Verification of Adaptive Systems," Tech. rep., Federal Aviation Administration, National Technical Information Services (NTIS), Springfield, Virginia 22161., USA, 2016.

[15] Farooq, U., and Grudin, J., "Human-computer Integration," *J. ACM Interactions, Vol. 23, No. 6*, 2016, pp. 27–32.

[16] Georgeff, M. P., and et al, "The Belief-Desire-Intention Model of Agency," *Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*, Springer-Verlag, London, UK, UK, 1999, pp. 1–10. URL `http://dl.acm.org/citation.cfm?id=648205.749450`.

[17] Adam, C., and Gaudou, B., "BDI agents in social simulations: a survey," *The Knowledge Engineering Review*, Vol. 31, No. 3, 2016, pp. 207–238. URL `https://www.cambridge.org/core/journals/knowledge-engineering-review/article/bdi-agents-in-social-simulations-a-survey/ADBACF4B23F625D42A7DB45E0D7556C4-http://oatao.univ-toulouse.fr/17127/`.

[18] Müller, J. P., and Fischer, K., "Application Impact of Multi-agent Systems and Technologies: A Survey," *Agent-Oriented Software Engineering*, Springer, 2014, pp. 27–53.

[19] Mascardi, V., Demergasso, D., and Ancona, D., "Languages for Programming BDI-style Agents: an Overview," *WOA*, Pitagora Editrice Bologna, 2005, pp. 9–15.

[20] Rao, D. M. S. S., and Jyothsna.A.N, "BDI: Applications and Architectures," *International Journal of Engineering Research & Technology (IJERT)*, Vol. 2, No. 2, 2013.

[21] Woodcock, J., and Davies, J., *Using Z: Specification, Refinement, and Proof*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[22] Krupitzer, C., and et al, "A Survey on Engineering Approaches for Self-adaptive Systems," *Pervasive Mob. Comput.*, Vol. 17, No. PB, 2015, pp. 184–206. doi:10.1016/j.pmcj.2014.09.009, URL `http://dx.doi.org/10.1016/j.pmcj.2014.09.009`.

[23] Girardi, R., and Leite, A., "A Survey on Software Agent Architectures." *IEEE Intelligent Informatics Bulletin*, Vol. 14, No. 1, 2013, pp. 8–20. URL `http://dblp.uni-trier.de/db/journals/cib/cib14.html#GirardiL13`.

[24] Müller, J. P., "Architectures and Applications of Intelligent Agents: A Survey," *Knowl. Eng. Rev.*, Vol. 13, No. 4, 1999, pp. 353–380. doi:10.1017/S0269888998004020, URL `http://dx.doi.org/10.1017/S0269888998004020`.

[25] Salehie, M., and Tahvildari, L., "Self-adaptive Software: Landscape and Research Challenges," *ACM Trans. Auton. Adapt. Syst.*, Vol. 4, No. 2, 2009, pp. 14:1–14:42. doi:10.1145/1516533.1516538, URL `http://doi.acm.org/10.1145/1516533.1516538`.

[26] Aerospace, S., "Guidelines for Development of Civil Aircraft and Systems," Tech. rep., SAE Aerospace, 2010.

[27] Kazman, R., and Cervantes, H., "ADD 3.0: Rethinking Drivers and Decisions in the Design Process," , 2015. URL `https://resources.sei.cmu.edu/asset_files/Presentation/2015_017_101_438648.pdf`.

[28] Huth, M., and Ryan, M., *Logic in Computer Science: Modelling and Reasoning About Systems*, Cambridge University Press, New York, NY, USA, 2004.

[29] Romanski, G., "The Challenges of Software Certification," *CrossTalk Journal of Defense Software Engineering*, Vol. 14, No. 9, 2001.

[30] Kevin, D., Brendan, H., Håkan, S., and Phil, Z., "Byzantine Fault Tolerance, from Theory to Reality," *Computer Safety, Reliability, and Security*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 235–248.

[31] Aerospace, S., "SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," Tech. rep., SAE Aerospace, 1996.

[32] RTCA, S.-., "DO-178B: Software Considerations in Airborne Systems and Equipment Certification," , 1982.

[33] RTCA, S.-., "DO-178C: Software Considerations in Airborne Systems and Equipment Certification," , 2011.

[34] RTCA, S.-., "DO-254: Design Assurance Guidance for Airborne Electronic Hardware," , 2000.

[35] FAA, "Avionics Evolution Impact on Requirements Issues and Validation and Verification," Tech. rep., Federal Aviation Administration, William J. Hughes Technical Center, Aviation Research Division, Atlantic City International Airport, New Jersey 08405 USA, 2015. URL `https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/ReqsDefinitionVVPhase2FinalDraft\discretionary{-}{}{}112315.pdf`.

[36] RTCA, S.-., "DO-333:Formal Methods Supplement to DO-178C and DO-278A," , 2013.

[37] Cofer, D., and Miller, S., "DO-333 Certification Case Studies," *Proceedings of the 6th International Symposium on NASA Formal Methods - Volume 8430*, Springer-Verlag New York, Inc., New York, NY, USA, 2014, pp. 1–15.

[38] Wilensky, U., "NetLogo multi-agent programmable modeling environment," , 2000. URL `https://ccl.northwestern.edu/netlogo/`.

[39] Scholand, A., "Model Checking and Abstraction," , 2006. URL `http://www2.cs.uni-paderborn.de/cs/kindler/Forschung/ComponentTools/ctseminar/ModelcheckingAbstractionPaper.pdf`.

[40] Pelanek, R., "Reduction and Abstraction Techniques for Model Checking," , 2006.

[41] Ball, T., Podelski, A., and Rajamani, S. K., "Boolean and Cartesian Abstraction for Model Checking C Programs," *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, 2001, pp. 268–283.

[42] Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H., "Counterexample-guided Abstraction Refinement for Symbolic Model Checking," *J. ACM*, Vol. 50, No. 5, 2003, pp. 752–794.

[43] Cimatti, A., Clarke, E. M., Giunchiglia, F., and Roveri, M., "NUSMV: A New Symbolic Model Verifier," *Proceedings of the 11th International Conference on Computer Aided Verification*, Springer-Verlag, London, UK, UK, 1999, pp. 495–499. URL `http://dl.acm.org/citation.cfm?id=647768.733923`.

[44] Baier, C., and Katoen, J.-P., *Principles of Model Checking*, The MIT Press, 2008.

[45] Levine, D., Gill, C., and Schmidt., D., "Dynamic Scheduling Strategies for Avionics Mission Computing," *17th IEEE/AIAA Digital Avionics Systems Conference, Seattle, Washington*, IEEE/AIAA, 1998, pp. pp. C15/1–8, volume 1.

[46] Dodd, R. B., Science, D., and (Australia)., T. O., *An analysis of task scheduling for a generic avionics mission computer [electronic resource] / R.B. Dodd*, DSTO Fishermens Bend, Vic, 2006. URL `http://pandora.nla.gov.au/apps/PandasDelivery/WebObjects/PandasDelivery.woa/wa/tep?pi=24764`.

[47] Craig, I. D., *Formal Specification of Advanced AI Architectures*, Ellis Horwood, Upper Saddle River, NJ, USA, 1991.

[48] Singh, S., Chana, I., and Singh, M., "Z language based an algorithm for event detection, analysis and classification in machine vision," *2013 International Conference on Human Computer Interactions (ICHCI)*, 2013, pp. 1–7. doi: 10.1109/ICHCI-IEEE.2013.6887803.

[49] Project, C. Z. T., "Standalone CZT IDE Version 1.6.0.201301310424," , 2013. URL `https://sourceforge.net/projects/czt/files/`.

[50] d'Inverno, M., Luck, M., Georgeff, M. P., Kinny, D., and Wooldridge, M., "The dMARS Architecture: A Specification of the Distributed Multi-Agent Reasoning System," *Autonomous Agents and Multi-Agent Systems*, Vol. 9, No. 1-2, 2004, pp. 5–53.

[51] FAA, "Artificial Intelligence With Applications for Aircraft," Tech. rep., Federal Aviation Administration, National Technical Information Services (NTIS), Springfield, Virginia 22161., USA, 1994.

[52] Du, K.-L., and Swamy, M. N. S., *Fundamentals of Machine Learning*, Springer-Verlag, London, 2014.

[53] Guerra-Hernández, A., El Fallah-Seghrouchni, A., and Soldano, H., "Learning in BDI Multi-agent Systems," *Proceedings of the 4th International Conference on Computational Logic in Multi-Agent Systems*, Springer-Verlag, Berlin, Heidelberg, 2004, pp. 218–233.

[54] Singh, D., Sardina, S., Padgham, L., and James, G., "Integrating Learning into a BDI Agent for Environments with Changing Dynamics," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, edited by C. K. Toby Walsh and C. Sierra, AAAI Press, Barcelona, Spain, 2011, pp. 2525–2530.

[55] Lamport, L., Shostak, R., and Pease, M., "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, Vol. 49, No. 3, 1982, pp. 382–401.

[56] Wikipedia, "Byzantine fault tolerance," , 2018. URL `https://en.wikipedia.org/wiki/Byzantine_fault_tolerance`.

[57] Wichman, K., Lindberg, L. G., Kilchert, L., and Bleeker, O. F., "Europes Emerging Trajectory-based Atm Environment," *Proceedings of the 11th International Conference on Computer Aided Verification*, 2003.

[58] Cummings, M. L., Bruni, S., Mercier, S., and Mitchell, P. J., "Automation Architecture for Single Operator, Multiple UAV Command and Control," *The International C2 Journal*, Vol. 1, No. 2, 2007, pp. 1–24.

[59] Dong, J. S., Hao, P., Qin, S., and Jun Sun, W. Y., "Formalising process scheduling requirements for an aircraft operational flight program," *Proceedings of the First IEEE International Conference on Formal Engineering Methods*, IEEE, 1997, pp. 218–233. doi:10.1109/ICFEM.1997.630423.

[60] Project, C. Z. T., "Classic CZT IDE Version 1.5.0," , 2009. URL `https://sourceforge.net/projects/czt/files/czt/`.

[61] Wilensky, U., and Rand, W., *An Introduction to agent-based modeling: Modeling natural, social and engineered complex systems with Netlogo*, MIT Press, 2015.

[62] Wikipedia, "Flight planning," , 2018. URL `https://en.wikipedia.org/wiki/Flight_planning`.

[63] ICAO, "Annex 2 to the Convention on International Civil Aviation, Rules of the Air," , 2005.

[64] ICAO, "PROCEDURES FOR AIR NAVIGATION SERVICES AIR TRAFFIC MANAGEMENT, Doc 4444-ATM/501," , 2007.

[65] Koelman, H., "Certification of Tactics and Strategies in Aviation," *Proceedings of the Human Factors Certification of Advanced Aviation Technologies Conference*, mbry-Riddle Aeronautical University Press 1994, Daytona Beach, FL 32114-3900, USA, 1993, pp. 53–75. URL `https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19950028343.pdf`.

[66] Altus, S., "Effective Flight Plans Can Help Airlines Economize," , 2009. URL `https://www.boeing.com/commercial/aeromagazine/articles`.

[67] HOEKSTRA, J. M., *Designing for Safety:the Free Flight Air Traffic Management Concept*, Delft University, 2001.

[68] Wikipedia, "Free flight (air traffic control)," , 2018. URL `https://en.wikipedia.org/wiki/Free_flight_(air_traffic_control)`.

[69] Eurocontrol, "Free route airspace (FRA)," , 2018. URL `https://www.eurocontrol.int/articles/free-route-airspace`.

[70] Midkiff, A. H., Hansman, R. J., and Reynolds, T. G., "Air Carrier Flight Operations," Tech. rep., MIT International Center for Air Transportation, Massachusetts Institute of Technology, Cambridge MA 02139 USA, 2004. URL `https://dspace.mit.edu/bitstream/handle/1721.1/35725/FlightOpsICATfinal2.pdf`.

[71] AAI, "RNAV-I (GNSS or DME/DME/IRU) SIDs and STARs," , 2009. URL `www.aai.aero/public_notices/29-2009.pdf`.

[72] FAA, "General Aviation Pilot's Guide to Preflight Weather Planning,Weather Self-Briefings, and Weather Decision Making," , 2005. URL `https://www.faasafety.gov/files/gslac/courses/content/25/185/GA%20Weather%20Decision-Making%20Dec05.pdf`.

[73] Sommerville, I., *Software Engineering*, 9$^{\text{th}}$ ed., Addison-Wesley Publishing Company, USA, 2010.

[74] FBK-irst, CMU, Univ. of Genova, and Univ. of Trento, "NuSMV: a new symbolic model checker," , 2015. URL http://nusmv.fbk.eu/.

[75] FAA, "FAA Aerospace Forecasts Fiscal Years 2018-2038," , 2018. URL https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/FY2018\discretionary{-}{}{}38_FAA_Aerospace_Forecast.pdf.

[76] Parimal Kopardekar, e. a., "Unmanned Aircraft System Traffic Management (UTM) Concept of Operations," *Proceedings of 16th AIAA Aviation Technology, Integration, and Operations Conference*, American Institute of Aeronautics and Astronautics, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807, 2016, pp. 92–95. doi:https://doi.org/10.2514/6.2016-3292.

[77] RTCA, S.-., "DO-304: Guidance Material and Considerations for Unmanned Aircraft Systems," , 2007.

[78] DOT, U., "Unmanned Aircraft System (UAS) Service Demand 2015-2035: Literature Review and Projections of Future Usage, Version 0.1," Tech. rep., U.S. Department of Transportation, 55 Broadway, Cambridge, MA 02142, USA, 2013.

[79] FAA, "Advisory Circular 107-2, Small Unmanned Aircraft Systems (sUAS)," , 2016. URL https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_107\discretionary{-}{}{}2.pdf.

[80] Dutertre, B., and Stavridou, V., "Formal requirements analysis of an avionics control system," *IEEE Transactions on Software Engineering*, Vol. 23, No. 5, 1997, pp. 267–278. doi:10.1109/32.588520.

[81] DOT, U., "FAA Requirements Engineering Management [REM] Handbook," , 2009. URL https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/AR\discretionary{-}{}{}08\discretionary{-}{}{}32.pdf.

[82] Schmitt, and et al, "A Case Study of Specification and Verification Using JML in an Avionics Application," *Proceedings of the 4th International Workshop on Java Technologies for Real-time and Embedded Systems*, ACM, New York, NY, USA, 2006, pp. 107–116. doi:10.1145/1167999.1168018, URL http://doi.acm.org/10.1145/1167999.1168018.

[83] Brunel, J., Rioux, L., Paul, S., Faucogney, A., and Vallée, F., "Formal safety and security assessment of an avionic architecture with alloy," *Proceedings Third International Workshop on Engineering Safety and Security Systems*, 2014, pp. 8–19. URL https://www.onera.fr/sites/default/files/328/main.pdf.

[84] Dordowsky, F., "An experimental Study using ACSL and Frama-C to formulate and verify Low-Level Requirements from a DO-178C compliant Avionics Project," *Proceedings Second International Workshop on Formal Integrated Development Environment, F-IDE 2015, Oslo, Norway, June 22, 2015.*, 2015, pp. 28–41. doi:10.4204/EPTCS.187.3, URL https://doi.org/10.4204/EPTCS.187.3.

[85] Shaha, T., and Patel, S., "A novel approach for specifying functional and non-functional requirements using RDS (Requirement Description Schema)," *Procedia Computer Science*, Vol. 79, 2016, pp. 852–860.

[86] Bharadwaj, A. K., and Nair, T. R. G., "Mapping General System Characteristics to Non-Functional Requirements," *2009 IEEE International Advance Computing Conference*, 2009, pp. 1634–1638. doi:10.1109/IADCC.2009.4809262.

[87] Paech, B., and et al, “Functional requirements, non-functional requirements and architecture specification cannot be separated,” *Proc. 8th Int’l Workshop on REFSQ*, Essener Informatik, Beitrage Band 7, Essen, Germany, 2002, pp. 102–107. URL `https://pdfs.semanticscholar.org/322f/5199bbf08f9b1b442b50ae961672277e52dd.pdf`.

[88] Rosa, N. S., Justo, G. R. R., and Cunha, P. R. F., “A Framework for Building Non-functional Software Architectures,” *Proceedings of the 2001 ACM Symposium on Applied Computing*, ACM, New York, NY, USA, 2001, pp. 141–147. doi:10.1145/372202.372299, URL `http://doi.acm.org/10.1145/372202.372299`.

[89] Sucipto, S., and Wahono, R. S., “A Systematic Literature Review of Requirements Engineering for Self-Adaptive Systems,” *Journal of Software Engineering*, Vol. 1, No. 1, 2015, pp. 17–27.

[90] Faulkner, S., and et al, “A Formal Description Language for Multi-Agent Architectures,” *Agent-Oriented Information Systems IV, Lecture Notes in Computer Science*, Vol. 4898, 2008, pp. 143–163. URL `http://dx.doi.org/10.1007/978\discretionary{-}{}{}3\discretionary{-}{}{}540\discretionary{-}{}{}77990\discretionary{-}{}{}2_9`.

[91] Iftikhar, M. U., and Weyns, D., “Formal Verification of Self-Adaptive Behaviors in Decentralized Systems with Uppaal: An Initial Study,” , 2012. URL `https://www.diva\discretionary{-}{}{}portal.org/smash/get/diva2:514688/FULLTEXT01.pdf`.

[92] Zhao, Y., and Rozier, K. Y., “Formal Specification and Verification of a Coordination Protocol for an Automated Air Traffic Control System,” *ECEASST*, Vol. 53, 2012.

[93] Lepri, D., Ábrahám, E., and Ölveczky, P. C., “A Timed CTL Model Checker for Real-Time Maude,” *CALCO*, Lecture Notes in Computer Science, Vol. 8089, Springer, 2013, pp. 334–339.

[94] Oreizy, and et al., “An Architecture-Based Approach to Self-Adaptive Software,” *IEEE Intelligent Systems*, Vol. 14, No. 3, 1999, pp. 54–62. doi:10.1109/5254.769885, URL `http://dx.doi.org/10.1109/5254.769885`.

[95] Hallsteinsen, S., and et al., “A Development Framework and Methodology for Self-adapting Applications in Ubiquitous Computing Environments,” *J. Syst. Softw.*, Vol. 85, No. 12, 2012, pp. 2840–2859. doi:10.1016/j.jss.2012.07.052, URL `http://dx.doi.org/10.1016/j.jss.2012.07.052`.

[96] Schleiss, P., Zeller, M., Weiss, G., and Eilers, D., “SafeAdapt - Safe Adaptive Software for Fully Electric Vehicles,” *Proc. of 3rd Conference on Future Automotive Technology (CoFAT) (2014)*, 2014.

[97] Canino, J. M., and et al, “A Multi-Agent Approach for Designing Next Generation of Air Traffic Systems,” , 2012. doi:10.5772/50284, URL `https://www.intechopen.com/books/advances\discretionary{-}{}{}in\discretionary{-}{}{}air\discretionary{-}{}{}navigation\discretionary{-}{}{}services/a\discretionary{-}{}{}multi\discretionary{-}{}{}agent\discretionary{-}{}{}approach\discretionary{-}{}{}for\discretionary{-}{}{}designing\discretionary{-}{}{}next\discretionary{-}{}{}generation\`

discretionary{-}{}{}of\discretionary{-}{}{}air\discretionary{-}{}{}traffic\discretionary{-}{}{}systems.

[98] Breil, R., and et al., "Multi-agent Systems for Air Traffic Conflicts Resolution by Local Speed Regulation," *7th International Conference on Research in Air Transportation (ICRAT 2016), (Philadelphie, PA, United States), Proceedings*, 2016.

[99] Hunter, J., Raimondi, F., Rungta, N., and Stocker, R., "A Synergistic and Extensible Framework for Multi-agent System Verification," *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2013, pp. 869–876. URL http://dl.acm.org/citation.cfm?id=2484920.2485058.

[100] Gorodetsky, V., and et al., "Multi-Agent Technology for Air Traffic Control and Incident Management in Airport Airspace," *Proceedings of the International Workshop Agents in Traffic and Transportation, Portugal*, 2008, pp. 119–125.

[101] Iglesia, D. G. D. L., and Weyns, D., "MAPE-K Formal Templates to Rigorously Design Behaviors for Self-Adaptive Systems," *ACM Trans. Auton. Adapt. Syst.*, Vol. 10, No. 3, 2015, pp. 15:1–15:31. doi:10.1145/2724719, URL http://doi.acm.org/10.1145/2724719.

[102] Pechoucek, M., and et al., "Intelligent Software Agent Control of Combined UAV Operations for Tactical Missions," Tech. rep., 2010.

[103] Georgeff, M., and Ingrand, F., "Monitoring and control of spacecraft systems using procedural reasoning," , 1989. URL https://pdfs.semanticscholar.org/5234/c99ab330d29614414118ea097bb8d11a54db.pdf.

[104] Ljungberg, M., and Lucas, A., "The OASIS Air Traffic Management System," , 1992.

[105] Wolfe, S. R., Sierhuis, M., and Jarvis, P. A., "To BDI, or Not to BDI: Design Choices in an Agent-Based Traffic Flow Management Simulation," *in SpringSim '08: Proceedings of the 2008 Spring simulation multiconference (ACM, New York, NY, USA,*, 2008, pp. 63–70. URL http://doi.acm.org/10.1145/1400549.1400558.

[106] Evertsz, R., Thangarajah, J., Yadav, N., and Ly, T., "A Framework for Modelling Tactical Decision-making in Autonomous Systems," *J. Syst. Softw.*, Vol. 110, No. C, 2015, pp. 222–238. doi:10.1016/j.jss.2015.08.046, URL https://doi.org/10.1016/j.jss.2015.08.046.

[107] Xia, Q., Wang, L., and Li, X., "Flight Conflict Detection Algorithm for UAV and MAV Under the Whole Airspace," *JOURNAL OF INFORMATION AND COMPUTATIONAL SCIENCE*, Vol. 11, No. 6, 2014, 2069. doi:10.12733/jics20103331, URL http://manu35.magtech.com.cn/Jwk_ics/EN/abstract/article_2292.shtml.

[108] Wu, X., Yang, H., Yang, B., Yu, J., and Wang, S., "Research on Collaborative Strategic Air Traffic Flow Management Based on BDI Agent," *ICESS*, IEEE Computer Society, 2016, pp. 103–107.

[109] Baron, S., and Feehrer, C., "An Analysis of the Application of AI to the Development of Intelligent Aids for Flight Crew Tasks," Tech. rep., NASA Langley Research Center, Hampton, VA., USA, 1985. URL `https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19860002745.pdf`.

[110] Abeloos, A. L. M., Mulder, M., and Paassen, M. M. V., "The Applicability of an Adaptive Human-Machine Interface in the Cockpit," *Proceedings of the 19th European Annual Conf. on Human Decision Making and Manual Control*, 2000, pp. 193–198. URL `https://pdfs.semanticscholar.org/049f/7d82f48b9bf1acd28a673817161d78002e46.pdf`.

[111] Spirkovska, L., and Lodha, S. K., "Context-Aware Intelligent Assistant Approach to Improving Pilot's Situational Awareness," Tech. rep., NASA Ames Research Centre, Mountain View, California, United States, 2004. URL `https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20040031829.pdf`.

[112] Tsiotras, P., and Johnson, E., "Advanced Methods For Intelligent Flight Guidance and Planning In Support Of Pilot Decision Making," Tech. rep., Georgia Institute of Technology, Georgia Institute of Technology, School of Aerospace Engineering,Atlanta, GA 30332-0150, 2012.

[113] Chen, S., Imai, S., Zhu, W., and Varela, C. A., "Towards Learning Spatio-Temporal Data Stream Relationships for Failure Detection in Avionics," *Dynamic Data-Driven Application Systems (DDDAS 2016)*, Hartford, CT, 2016. Appears in Cluster Computing Journal.

[114] Baomar, H., and Bentley, P. J., "An Intelligent Autopilot System that learns piloting skills from human pilots by imitation," *Proceedings of International Conference on Unmanned Aircraft Systems (ICUAS)*, Arlington, VA, 2016.

[115] Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y., "An application of reinforcement learning to aerobatic helicopter flight," *In Advances in Neural Information Processing Systems 19*, MIT Press, 2007, p. 2007.

[116] Clement, G., and Doina, P., "Smart Exploration in Reinforcement Learning Using Absolute Temporal Difference Errors," *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2013, pp. 1037–1044. URL `http://dl.acm.org/citation.cfm?id=2484920.2485084`.

[117] Hrastovec, M., and Solina, F., "Machine learning model for aircraft performances," *2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC)*, 2014, pp. 8C4–1–8C4–10. doi:10.1109/DASC.2014.6979541.

[118] Naessens, H., and et al, "Predicting flight routes with a Deep Neural Network in the operational Air Traffic Flow and Capacity Management system," , 2017. URL `https://www.eurocontrol.int/sites/default/files/publication/files/predicting\discretionary{-}{}{}flight\discretionary{-}{}{}routes\discretionary{-}{}{}with\discretionary{-}{}{}neura\T1\l-network\discretionary{-}{}{}tpi\discretionary{-}{}{}technical\discretionary{-}{}{}document.pdf`.

[119] Larranaga, P., "MACHINE LEARNING IN AVIATION," , Jul 2013. URL `cig.fi.upm.es/sites/default/files/com_phocadownload/container/Ml\discretionary{-}{}{}in\discretionary{-}{}{}aviation.pdf`.

[120] Low, G., Beydoun, G., Henderson-Sellers, B., and Gonzalez-Perez, C., "Towards method engineering for multi-agent systems: a validation of a generic MAS metamodel," *Agent Computing and Multi-Agent Systems, LNAI 5044.*, 2009, pp. 255–267. doi:10.1016/j.pmcj.2014.09.009, URL `http://dx.doi.org/10.1016/j.pmcj.2014.09.009`.

[121] Almeida, J. B., Frade, M. J., Pinto, J. S., and Sousa, S. M. d., *Rigorous Software Development: An Introduction to Program Verification*, 1st ed., Springer Publishing Company, Incorporated, 2011.

[122] Denney, R., "A Comparison of the Model-based &Amp; Algebraic Styles of Specification As a Basis for Test Specification," *SIGSOFT Softw. Eng. Notes*, Vol. 21, No. 5, 1996, pp. 60–64. doi:10.1145/235969.235988, URL `http://doi.acm.org/10.1145/235969.235988`.

[123] Pandey, T., and Srivastava., S., "Comparative Analysis of Formal Specification Languages Z, VDM and B," *International Journal of Current Engineering and Technology*, Vol. 5, No. 3, 2015, pp. 2086–2091. URL `http://inpressco.com/wpcontent/uploads/2015/06/Paper1082086\discretionary{-}{}{}2091.pdf`.

[124] Schumann, J., and Visser, W., "Autonomy Software: V&V Challenges and Characteristics," *Proceedings of IEEE Aerospace Conference*, 20006, pp. 1–6. doi:10.1109/AERO.2006.1656023.

[125] Dennis, L. A., and Farwer, B., "Gwendolen: A BDI Language for Verifiable Agents," *Logic and the Simulation of Interaction and Reasoning*, edited by B. Löwe, AISB, Aberdeen, 2008. AISB'08 Workshop.

[126] Fernandes, L. E. R., and et al, "A Rational Agent Controlling an Autonomous Vehicle: Implementation and Formal Verification," , Jul 2013. doi:10.4204/EPTCS.257.5, URL `http://livrepository.liverpool.ac.uk/id/eprint/3009370`.

[127] Kamali, M., and et al, "Formal verification of autonomous vehicle platooning," , Jul 2017. doi:10.1016/j.scico.2017.05.006, URL `http://livrepository.liverpool.ac.uk/id/eprint/3007886`.

[128] Hunter, J., Raimondi, F., Rungta, N., and Stocker, R., "A Synergistic and Extensible Framework for Multi-agent System Verification," *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2013, pp. 869–876. URL `http://dl.acm.org/citation.cfm?id=2484920.2485058`.

[129] Bordini, R. H., Fisher, M., Wooldridge, M., and Visser, W., "Model checking rational agents," *IEEE Intelligent Systems*, Vol. 19, No. 5, 2004, pp. 46–52. doi:10.1109/MIS.2004.47.

[130] Raimondi, F., "Case Study Description: Avionic scenario," *Dagstuhl Reports*, Vol. 3, 2013, pp. 180–184.

[131] Iftikhar, M. U., and Weyns, D., "A Case Study on Formal Verification of Self-Adaptive Behaviors in a Decentralized System," *FOCLASA*, EPTCS, Vol. 91, 2012, pp. 45–62.

[132] Nafz, F., Steghofer, J., Seebach, H., and Reif, W., "Formal modeling and verification of self-* systems based on observer/controller-architectures," *Assurances for Self-Adaptive Systems, ser. LNCS*, Vol. 7740,

2013. URL https://link.springer.com/chapter/10.1007/978\discretionary{-}{}{}3\discretionary{-}{}{}642\discretionary{-}{}{}36249\discretionary{-}{}{}1_4.

[133] Fisher, M., Dennis, L. A., and Webster, M. P., "Verifying autonomous systems," *Commun. ACM*, Vol. 56, No. 9, 2013, pp. 84–93.

[134] Moy, Y., Ledinot, E., Delseny, H., Wiels, V., and Monate, B., "Testing or Formal Verification: DO-178C Alternatives and Industrial Experience," *IEEE Softw.*, Vol. 30, No. 3, 2013, pp. 50–57. doi:10.1109/MS.2013.43, URL http://dx.doi.org/10.1109/MS.2013.43.

[135] Wiels, V., and et al, "Formal Verification of Critical Aerospace Software," , May 2012. URL http://www.aerospacelab\discretionary{-}{}{}journal.org/sites/www.aerospacelab-journal.org/files/AL04\discretionary{-}{}{}10_1.pdf.

[136] Bochot, T., Virelizier, P., Waeselynck, H., and Wiels, V., "Model checking flight control systems: the airbus experience," *In: ICSE 2009. 31st International Conference on Software Engineering, Companion Volume*, IEEE, 2009, pp. 18–27.

[137] Cofer, D., and Miller, S., "Formal methods case studies for DO-333," Tech. rep., NASA, Langley Research Center, Hampton, Virginia 23681-2199, 2014. URL https://shemesh.larc.nasa.gov/people/bld/ftp/NASA\discretionary{-}{}{}CR\discretionary{-}{}{}2014\discretionary{-}{}{}218244.pdf.

[138] Jacklin, S., "Closing the certification gaps in adaptive flight control software," *Proceedings of the AIAA Guidance Navigation and Control Conference 2008*, 2008.

[139] Colley, J., and Butler, M., "Safety and Security Considerations for Software and Digital Hardware Verification," , April 2017. URL http://www.testandverification.com/wp\discretionary{-}{}{}content/uploads/2017/Verification_Futures/John_Colley_University_of_Southampton.pdf.

[140] Souyris, J., Wiels, V., Delmas, D., and Delseny, H., "Formal Verification of Avionics Software Products," *Proceedings of the 2Nd World Congress on Formal Methods*, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 532–546. doi:10.1007/978\-3\-642\-05089\-3_34, URL http://dx.doi.org/10.1007/978\discretionary{-}{}{}3\discretionary{-}{}{}642\discretionary{-}{}{}05089\discretionary{-}{}{}3_34.

[141] Webster, M., Cameron, N., Fisher, M., and Jump, M., "Generating Certification Evidence for Autonomous Unmanned Aircraft Using Model Checking and Simulation," *Journal of Aerospace Information Systems*, Vol. 11, No. 5, 2014, pp. 258–279. doi:10.2514/1.I010096, URL https://doi.org/10.2514/1.I010096.

[142] Tamura, G., and et al, "Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems," *Software Engineering for Self-Adaptive Systems*, Lecture Notes in Computer Science, Vol. 7475, Springer, 2010, pp. 108–132.

[143] Weyns, D., Malek, S., and Andersson, J., "FORMS: A Formal Reference Model for Self-adaptation," *Proceedings of the 7th International Conference on Autonomic Computing*, ACM, New York, NY, USA, 2010, pp. 205–214. doi:10.1145/1809049.1809078, URL http://doi.acm.org/10.1145/1809049.1809078.

[144] de Lemos, and et al, “Software Engineering for Self-Adaptive Systems: A second Research Roadmap,” *Software Engineering for Self-Adaptive Systems II*, Lecture Notes in Computer Science (LNCS), Vol. 7475, edited by R. de Lemos, H. Giese, H. Müller, and M. Shaw, Springer, 2013, pp. 1–32. URL `http://dx.doi.org/10.1007/978\discretionary{-}{}{}3\discretionary{-}{}{}642\discretionary{-}{}{}35813\discretionary{-}{}{}5_1`.

[145] Cordy, M., Classen, A., Heymans, P., Legay, A., and Schobbens, P., “Model Checking Adaptive Software with Featured Transition Systems,” *Assurances for Self-Adaptive Systems*, Lecture Notes in Computer Science, Vol. 7740, Springer, 2013, pp. 1–29.

[146] Priesterjahn, C., Steenken, D., and Tichy, M., “Timed Hazard Analysis of Self-healing Systems,” *Assurances for Self-Adaptive Systems*, Lecture Notes in Computer Science, Vol. 7740, Springer, 2013, pp. 112–151.

[147] Villegas, N. M., Müller, H. A., Tamura, G., Duchien, L., and Casallas, R., “A Framework for Evaluating Quality-driven Self-adaptive Software Systems,” *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ACM, New York, NY, USA, 2011, pp. 80–89. doi:10.1145/1988008.1988020, URL `http://doi.acm.org/10.1145/1988008.1988020`.

[148] Cheng, S.-W., Garlan, D., and Schmerl, B., “Evaluating the Effectiveness of the Rainbow Self-adaptive System,” *Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, IEEE Computer Society, Washington, DC, USA, 2009, pp. 132–141. doi:10.1109/SEAMS.2009.5069082, URL `https://doi.org/10.1109/SEAMS.2009.5069082`.

[149] McCann, J. A., and Huebscher, M. C., “Evaluation Issues in Autonomic Computing,” *Grid and Cooperative Computing - GCC 2004 Workshops*, edited by H. Jin, Y. Pan, N. Xiao, and J. Sun, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 597–608.

[150] Berns, A., and Ghosh, S., “Dissecting Self-* Properties,” *SASO*, IEEE Computer Society, 2009, pp. 10–19.