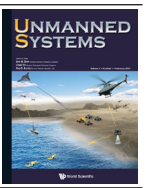
Unmanned Systems, Vol. 7, No. 4 (2019) 261–279

DOI: 10.1142/S2301385019500109

UNMANNED SYSTEMS
http://www.worldscientific.com/worldscinet/us



Failsafe Mechanism Design for Autonomous Aerial Refueling using State Tree Structures

Ke Dong*,§, Quan Quan†,¶, W. Murray Wonham‡,‖

*Institute for Aerospace Study, University of Toronto
Toronto, Ontario, M3H 5T6, Canada

†School of Automation Science and Electrical Engineering
Beihang University, Beijing 100191, P. R. China

‡Department of Electrical and Computer Engineering
University of Toronto, Toronto, Ontario, M5S 3G4, Canada

Autonomous Aerial Refueling (AAR) is vulnerable to various failures and involves cooperation among autonomous receivers, tankers and remote pilots. Dangerous flight maneuvers may be executed when unexpected failures or command conflicts happen. To solve this problem, a failsafe mechanism based on State Tree Structures (STS) is proposed. The failsafe mechanism is a control logic that guides what subsequent actions the autonomous receiver should take, by observing real-time information of internal low-level subsystems such as guidance and drogue&probe and external instructions from tankers and pilots. To generate such a controller using STS, the AAR procedure is decomposed into several modes, and safety issues related with seven low-level subsystems are summarized. Then common functional demands and safety requirements are textually described. On this basis, the AAR plants and specifications are modeled by STS, and a supervisor is synthesized to control the AAR model. To prove its feasibility and correctness, a simulation environment incorporating such a logic supervisor is built and tested. The design procedures presented in this paper can be used in decision-making strategies for similar flight tasks. Supporting materials can be downloaded in Github, [https://github.com/KevinDong0810/Failsafe-Design-for-AAR-using-STS] including related software, input documents and output files.

Keywords: Aerial refueling; STS; failsafe mechanism.

US

1. Introduction

Autonomous Aerial Refueling (AAR) is an effective approach to increase the range and endurance of Unmanned Aerial Vehicles (UAV) by refueling them in air. Since the first successful AAR concept demonstration by the Boeing company, several ambitious AAR projects have been carried out by NASA [1], the US Air Force and US Navy [2], which demonstrate the feasibility and reliability of AAR. During an AAR operation, as shown in Fig. 1, a UAV (the receiver) detaches from its formation and approaches the rear of a tanker for refueling. Once refueled and cleared, the receiver disengages from the tanker and rejoins the formation. The boom system or drogue&probe system is used for fuel transfer; this paper focuses on the drogue&probe system, as shown in Fig. 2, for the drogue&probe's advantages for autonomous systems. [1, 3].

The AAR is a semi-automated process, where the control of unmanned receivers relies on frequent interactions among the on-board receiver controller, external tanker controller and remote human pilots. Therefore, command conflicts may happen and jeopardize the refueling task. Besides, an AAR process is vulnerable to various system failures like probe damage and unsteady airflow. Therefore, a high-level logic controller that coordinates all these control components and produces the desired safe functionality in the presence of faults and failures is needed, namely a

Received 26 October 2018; Revised 19 June 2019; Accepted 19 June 2019; Published 3 September 2019. This paper was recommended for publication in its revised form by editorial board member, Hai Lin.
Email Addresses: §ke.dong@mail.utoronto.ca, ¶qq_buaa@buaa.edu.cn, ‖wonham@ece.utoronto.ca

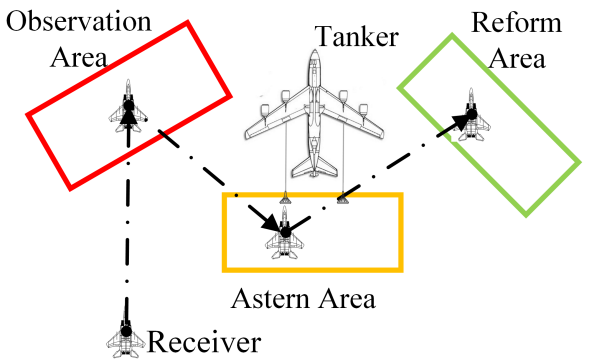


Fig. 1. Typical AAR refueling procedures.

failsafe mechanism. A wealth of researches have been done in this domain. Reference [4] formalized the general framework for formal verification of human-automation interface, where both automation and human controller share the authority over the system. References [5] and [6] investigated the safe path planning in AAR to avoid collision with consideration of receiver-pilot interaction. In the Research Flight Control System developed by [1] in NASA, human–machine interface is put into practice. But none of them has taken system failures into consideration for logic design, under which dangerous maneuvers may happen. In contrast, fault-related literature is mainly focused on low-level controllers, such as fault detection techniques [7] and fault-tolerant control algorithms [8, 9]. As far as the authors know, no high-level logic controller has been studied.

At present, the failsafe mechanism design for AAR is seldom investigated. Related applications like rotorcrafts mainly count on engineering experience and manual calculation to design the control logic, like DJI autopilot and ArduPilot autopilot. Man-made mistakes, logical bugs or an incomplete treatment are hard to avoid in this approach. And when it comes to complex systems, this approach is very time-consuming. Therefore, this paper aims to use a model-based method, namely supervisory control theory (SCT) of state tree structures (STS), to design a comprehensive failsafe mechanism.

Fig. 2. The drogue&probe system.

SCT is a formal method for synthesizing supervisors of discrete-event systems. On observing event strings generated by the plant, at each state SCT would disable a suitable subset of controllable events to satisfy the given control specifications [10]. First proposed in [11], SCT has a sound theoretical foundation [12, 13], and numerous industrial applications such as testbed assembly process [14], telephone directory assistance call center [15] and magnetic resonance image (MRI) scanner [16]. Later, to solve the problem of exponential computational effort faced by finite state machine (FSM) used in original SCT, an adaption of *state charts* [17] called state-tree structures [18] is adopted; this new version of SCT is called STS. The STS uses the AND (Cartesian product) and OR (disjoint union) operation to present system hierarchies and uses *binary decision diagrams* (BDD) [19] to manage the computational complexity.

Compared with current failsafe mechanisms based on engineering experience, the STS approach has three main advantages: (1) *Correct by design*: The restricted behaviors under the supervision of the STS ensure that requirements are satisfied. Therefore, the control logic is free from incomplete treatment and logical bugs in the design phase. (2) *Compact*: The resulting supervisor is presented in the simplest way, which means that when making a decision, only necessary information instead of the whole system state would be required. Therefore, a minimal disabling action is made for the system to stay within safe states. (3) *Flexibility*: The STS provides a modular modeling method and ensures the system design can be readily evolved. Different modules (subsystems, requirements, etc.) can be added easily when taking different control hierarchies (from high-level to low-level) into consideration. This has been well presented in the application of theme park vehicles [20].

The main contributions of this paper are the following:

- This paper investigates the failsafe logic design of autonomous receivers using a sound formal verification theory. A supervisor is synthesized to cover common system failures and interaction among receivers, tankers and pilots. This design framework can be easily adopted for different aerial refueling procedures and other similar control tasks.
- Tools for synthesizing, implementing and simulating supervisors gained through STS for aerial refueling have been developed and posted in Github. This may stimulate a similar failsafe mechanism design in related domains.

The remainder of this paper is organized as follows. Section 2 summarizes the basics of STS. Section 3 presents the discretization of AAR procedures and a summary of

common safety issues. In Sec. 4, functional demands and safety requirements are provided, as well as the event definitions. On this basis, Sec. 5 presents the STS design of AAR and the generated supervisor using related software. In Sec. 6, the supervisor is implemented in a simulation environment for AAR and tested for correctness. Section 7 concludes this paper.

2. Preliminaries of STS

This section presents the basics of STS theory, while details can be found in textbooks [12, 13, 18].

2.1. *State tree structure*

The State Tree Structure is an extension of the finite state machine in SCT, which introduces natural hierarchical structure into the system model. It mainly consists of two parts: *state tree* and *holons*. These two parts depict the same system but focus on different aspects. State tree, say **ST**, organizes the state space of an STS, while the holons, say **H**, organize the transitions of an STS. The nodes on a state tree are called states. Every state tree has a unique root state. A state on a state tree is called an OR (respectively AND) *superstate* if it can be represented as the disjoint union $\dot{\cup}$ (respectively Cartesian product X) of its children. Each child state is called an OR (respectively AND) superstate. The lowest level states are called *simple states*, which are not decomposable. In the following text, the states of STS including simple states and superstates, are written in italics and start with capitals like *Standby*. In most application scenarios, OR superstates work sequentially while AND superstates work concurrently.

Each holon organizes the state transitions associated with an OR state of the STS; it can be regarded as an automaton with hierarchies. It depicts the boundary and local state transitions of the state tree. A holon **H** is a 5-tuple $\mathbf{H} := \{X, \Sigma, \delta, X_o, X_m\}$, where X is a finite set consisting of states in **ST**; Σ is the finite event set (also called an *alphabet*); δ is the (partial) transition function on the state set: $\delta : X \times \Sigma \to X$; $X_o \subseteq X$ is a subset of X consisting of *initial states*; $X_m \subseteq X$ is another subset of X consisting of *marker states* that the whole system desires to reach.

In SCT, the alphabet Σ is partitioned as $\Sigma = \Sigma_c \cup \Sigma_u$, where Σ_c is the subset of *controllable events* that can be disabled at will while Σ_u is the subset of *uncontrollable events* that cannot be disabled by the controller. In engineering practice, Σ_c usually represents relevant discrete commands to the low-level controller, while Σ_u usually represents messages sent to the supervisor such as system health information and sensor signals like the waiting time exceeding a pre-defined threshold.

2.2. *Plant, specification and supervisor*

SCT aims to synthesize the supremal nonblocking language (event sequences) of *Plant*, while satisfying the requirements of *Specification*. The generated language is called *Supervisor*; their detailed descriptions are given as follows:

- **Plant.** Plants are models of behaviors of physical systems and processes, which contain all the possible states and transitions.
- **Specification.** Specifications are models of control requirements, which point out the unwanted states and transitions in correspondence to forbidden states. It can be given as illegal state sets [21] or illegal event sequences [20]. The latter form is used in this paper.
- **Supervisor.** The supervisor is the minimally restrictive event sequence of plants under the restriction of specifications, and is implemented by state feedback control functions [18] for events. In every state of the plant, the supervisor will provide a subset of controllable events among which the plant can choose one to execute. By doing so, the specifications are always satisfied.

In the synchronization algorithm, namely *Supcon*, for supervisors [18], all uncontrollable event sequences leading to *blocking states*, namely states that cannot reach marker states by any event transitions, are deleted iteratively. Therefore, to model the specifications, any undesired event sequences should be constructed to lead to blocking states.[a]

3. Mode Discretization and Safety Issues

This section describes the plant for aerial refueling. Related AAR procedures are discretized and common safety issues are summarized, according to which the STS plant will be modeled.

3.1. *Area definition*

The procedure discretization is based on air space decomposition. The air space is divided into different areas according to their locations relative to the tanker.

As shown in Figs. 3 and 4, the air space is divided into two parts: *Task Space* and *Withdrawal Space*. The task space is the space where AAR tasks are carried out. It is assumed to be a box space centered at the tanker with 300 m height (downward), 3704 m length and an appropriate width.[b] The task space contains three areas: observation area, astern

[a] Readers can find a similar example in Exercise 3.10.3 of Chap. 3 in the DES textbook [13] for a better understanding.
[b] These data are obtained according to [22, p. 54].

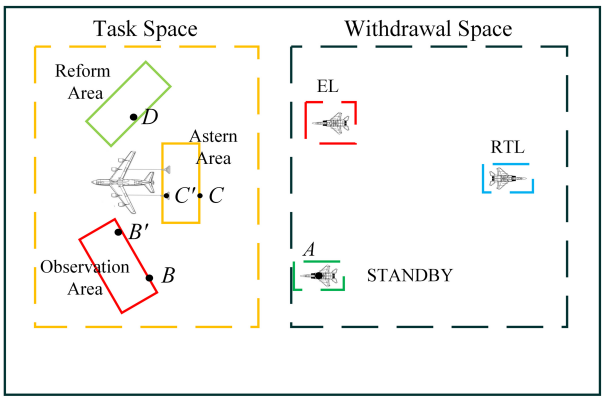


Fig. 3. Top view of the air space division.

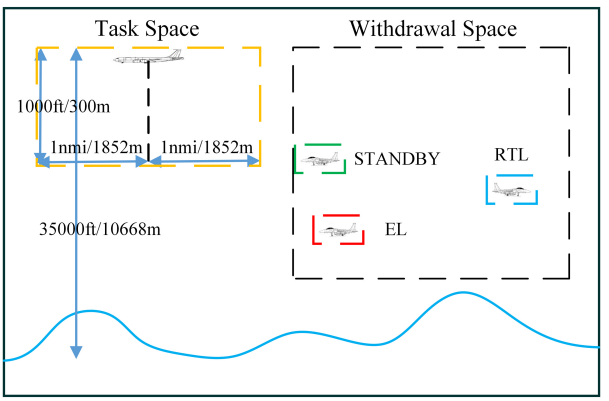


Fig. 4. Side view of the air space division.

area and reform area. The observation area at the left-hand side of the tanker is allocated for joining receivers. The astern area is the stabilized formation position behind the AAR equipment, approximately 100 ft-aft of the drogue directly [22, p. 32]. The reform area is the area at the right and level or slightly above the tanker formation.

The withdrawal space is the airspace outside of the task space used for emergency and AAR preparation. When AAR is achieved or failed, the receiver will enter into the withdrawal space from the task space. On the other hand, when the receiver's health conditions allow a possible AAR, the receiver will enter the task space from the withdrawal space. The withdrawal space has three maneuver modes including STANDBY, RTL and EL, whose definitions are presented in Sec. 3.2.1.

3.2. *Mode decomposition*

Based on the air-to-air refueling demonstration of [1] and standard procedures for manned aircraft proposed by [22], the AAR work cycle is decomposed into the following distinct phases and modes. In the following text, the AAR mode will be presented in capitals like STANDBY MODE.[c]

3.2.1. *Withdrawal phase*

The withdrawal phase is in correspondence to the withdrawal space. As mentioned before, there are three modes in this phase or space, namely STANDBY, RTL and EL.

(a) **STANDBY MODE:** The receiver is waiting at point A (as shown in Fig. 3) for the next instruction. In this mode, the receiver has the ability to carry on AAR tasks.
(b) **Return to Land (RTL) MODE:** The receiver returns to the nearest airbase to land, which may result from subsystem failures or pilot instructions. In this mode, the receiver may lose the ability to carry on AAR tasks but still keep the ability to return to base.
(c) **Emergency Landing (EL) MODE:** The receiver makes an emergency landing, which may result from subsystem failures or pilot instructions. In this mode, the receiver may lose the ability to return to base and can only make an emergency landing.

3.2.2. *Task phase*

The task phase contains all the AAR task procedures, mainly the joining, refueling and reforming sub-phases.

(a) **Joining Sub-Phase:** Joining refers to the sub-phase where the receiver maneuvers from STANDBY MODE to the observation area. It has two modes:
  (1) **JOINING-INIT MODE:** As shown in Fig. 5(a), the receiver flies to the observation area. The control objective is to control the receiver flying from STANDBY MODE to point B.
  (2) **JOINING-WAIT MODE:** As shown in Fig. 5(b), the receiver stays in the observation area and waits for the *connection clearance* from the tanker, namely the command given by the tanker to allow the receiver to fly to the astern area and connect its probe with the drogue. The control objective is to control the receiver flying to point B' and then keep the position.
(b) **Refueling Sub-Phase:** Refueling efers to the sub-phase where the receiver connects to the tanker and transfers the fuel. It has three modes.
  (1) **REFUELING-INIT MODE:** As shown in Fig. 5(c), once the receiver is cleared for the connection it flies

[c]Standby can be a mode like STANDBY MODE or a simple state like *Standby*. There are two different concepts in this paper, but refer to the same thing in the physical world.

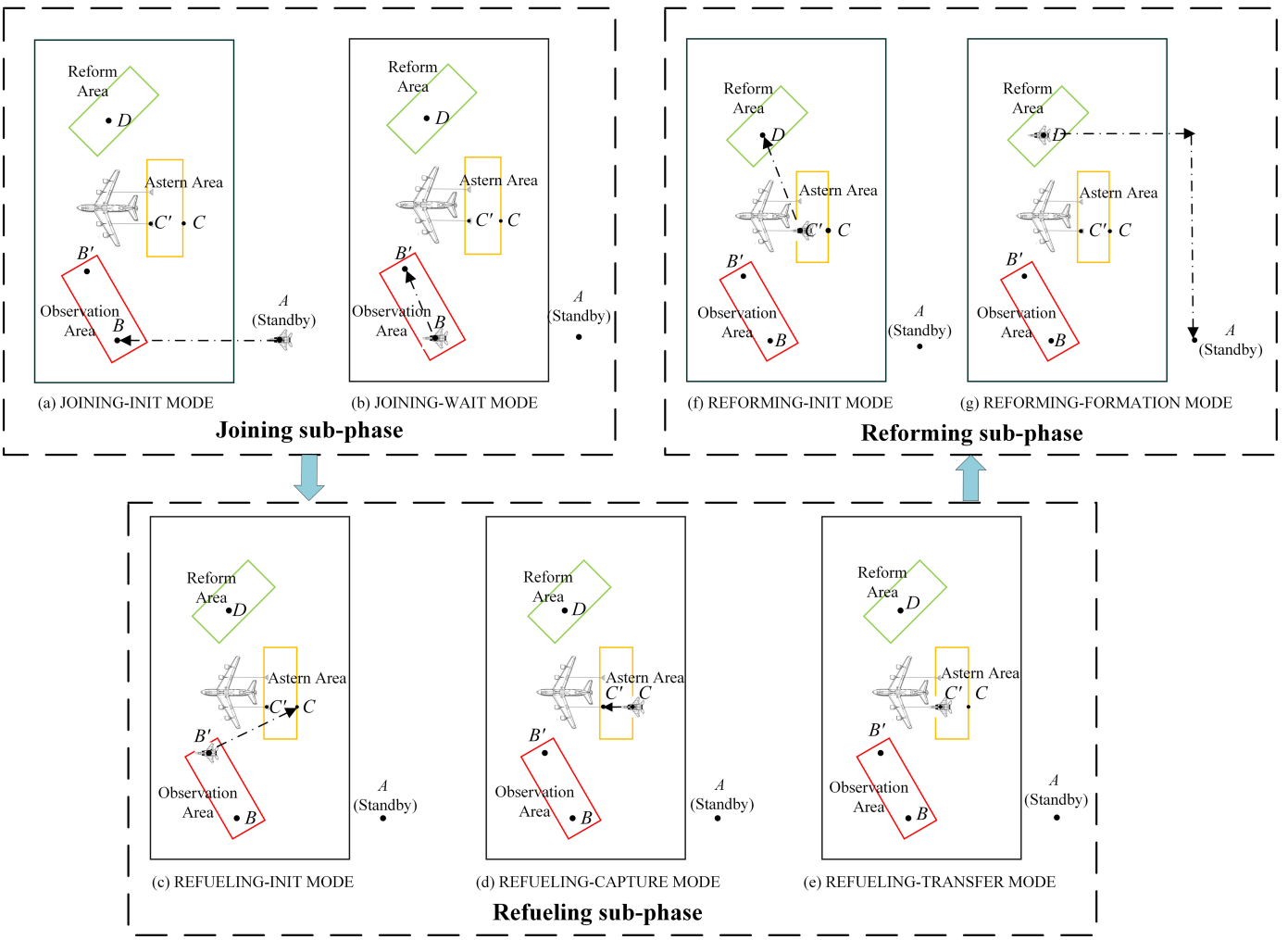


Fig. 5. Side view of the air space division.

to the astern area. The control objective is to control the receiver flight to point C and then maintain the position.

(2) **REFUELING-CAPTURE MODE:** The receiver initiates the connection between the probe and the tanker's drogue. As shown in Fig. 5(d), the control objective is to control the receiver flying to point C', and then keep the position for connecting the probe with the drogue.

(3) **REFUELING-TRANSFER MODE:** After a successful capture, the receiver and the tanker keep relatively stationary to transfer the fuel. When the fuel transfer is finished, the receiver should still keep connected with the tanker and wait for the *disconnection clearance*, namely the command given by the tanker to allow the receiver to disconnect from the drogue. As shown in Fig. 5(e), the control objective is to keep the receiver at the point C' and transfer the fuel.

(c) **Reforming Sub-Phase:** Reforming refers to the sub-phase where the receiver disconnects with the tanker, flies to the reform area and then leaves the formation. It has two modes.

(1) **REFORMING-INIT MODE:** The receiver is cleared for disconnection and then flies to the reform area. As shown in Fig. 5(f), the control objective is to control the receiver flying to point D and then keep the position.

(2) **REFORMING-FORMATION MODE:** The receiver rejoins the flight formation and then leaves the task space, which can be regarded as returning to the withdrawal space to form a circle in the model. As shown in Fig. 5(g), the control objective is to

control the receiver to maneuver from point D to the withdrawal space.

3.3. *Safety issues*

This subsection summarizes the common failure behaviors in aerial refueling. They are the health information that should be taken into consideration when making decisions. These failures mainly happen in those subsystems shown in Table 1.

Most of their failure behaviors can be depicted as a set of holons shown in Fig. 6. According to concrete requirements, a subsystem can have more states to describe the health, but only three different health states of each subsystem are considered here, namely *normal*, *minor damage* and *critical damage*. "Normal" state represents that the system satisfies the basic requirements with all components being healthy. "Minor damage" state represents that although there are some failures in the system, the system can still satisfy the basic requirements. "Critical damage" state represents that the system can no longer satisfy the basic requirements.

Table 1. Subsystem information.

Category	Name	Basic requirement
Receiver	Navigation	Provide data about the relative position and velocity among receivers and tankers to facilitate docking.
	Control	Keep the receiver's position and velocity within an allowable range according to its current mode, to avoid collision with other aircraft and achieve successful docking.
	Fuel	Provide the necessary fuel for flight.
	Engine	Provide the necessary power and thrust for flight
Connection	Drogue&probe	Establish robust contact between the drogue and probe, and then transfer fuel from the tanker to the receiver.
	Datalink	Exchange data for communication and high-accuracy computation of relative locations.
Tanker	Tankersafety[1]	Facilitate the stable connection and transfer the fuel.

Note: [1]To avoid redundancy, the tanker is modeled as a whole instead of with different subsystems like those of the receiver.

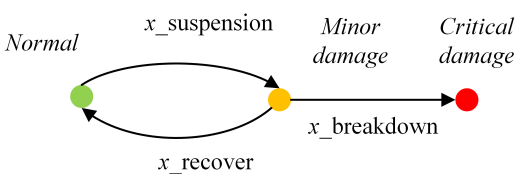


Fig. 6. The holons of the general subsystems.

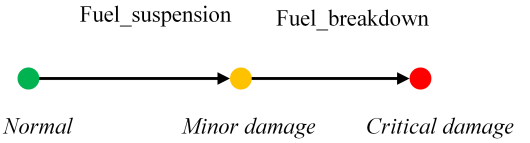


Fig. 7. The holons of Fuel subsystem. To be consistent with other subsystems, the exhaustion of fuel is called *Fuel_breakdown* here.

Transitions among different states, namely x_suspension, x_recover and x_break-down are detected by low-level modules, where x refers to an exact subsystem name as shown in Table 1, e.g., Navigation. They can be defined according to the specific designers' purpose. For example, the transitions of the Navigation subsystem are given as follows:

- **Navigation_suspension:** The precision of navigation data has degraded to be lower than a specified threshold, but it can still satisfy the basic requirements of the navigation subsystem. Causes of degradation are multiple, such as bad weather for cameras, radars and lasers, multi-path effects, hostile jamming and spoofing for GPS [2].
- **Navigation_recover:** Precision quality exceeds a specified threshold, and the Navigation subsystem is healthy again.
- **Navigation_breakdown:** The *minor damage* state has lasted more than a given threshold time or the data provided do not fulfill the basic requirements of the navigation subsystem.

Owing to the special characteristics of the Fuel subsystem, there is no x_recover transition in this subsystem. Its holons are shown in Fig. 7.

4. User Requirements and Event Definitions

This section textually describes the requirements of aerial refueling. Functional demands and safety requirements are summarized from common user requirements of the logic control system; they will guide the design of plants and specifications, respectively. Event definitions are also provided here.

4.1. *User requirements*

In engineering practice, aerial refueling systems have to satisfy multiple functional demands and safety requirements, as summarized in Tables 2 and 3. Functional demands deal

Table 2. Functional demands.

Name	Definition
FD1	The receiver, which switches from the task phase to the withdrawal phase, should first try entering STANDBY. If not satisfying the safety requirement for STANDBY MODE, the receiver should try entering RTL. If still not satisfying the safety requirement for RTL MODE, the receiver should try entering EL.
FD2	In EL MODE, if health conditions satisfy the safety requirements, the receiver can switch to STANDBY MODE or RTL MODE. Similarly, the receiver can switch to STANDBY from RTL.
FD3	If the receiver cannot carry on AAR tasks further, it should break away and switch to the withdrawal phase as soon as possible.
FD4	The receiver has to wait for the clearance commands from the tanker to connect or disconnect with the tanker.
FD5	Pilots can manually switch the receiver to the STANDBY, RTL or EL MODE from any state. When health conditions allow, such instructions should be executed as soon as possible and override any other automatic mode progression.
FD6	Pilots can force the enablement of the connection initiation and fuel transfer initiation, even though such maneuvers are forbidden by the autopilot due to severe health conditions.
FD7	Considering pilots may not know the real-time health information of receivers in timely fashion, when pilots ask the receiver to return to STANDBY MODE, the receiver can go to STANDBY, RTL and EL MODE. When asked to return to RTL MODE, the receiver can go to RTL and EL MODE. When asked to go to EL MODE, it can only switch to EL MODE.

with what functions should exist in the system, like "the receiver can be forced by the pilot to try connecting with the tanker even when the health conditions do not allow so". They are used to guide the design of plants in STS. Safety requirements deal with what behaviors are illegal, e.g., "when control subsystem breaks down, the receiver cannot continue the current AAR task". They are used to design the specifications in STS.

Table 3. Safety requirements in withdrawal phase.

Name	Description
SR1	Entering and staying in STANDBY MODE are allowed when Navigation, Control, Engine, Drogue&probe, Datalink and Tankersafety subsystems are at *minor damage* or *normal*, and Fuel subsystem is at *normal*. If this requirement is not satisfied, transitions to STANDBY MODE are forbidden and transitions to RTL should be made.
SR2	Entering and staying in RTL MODE are allowed when Navigation, Fuel and Engine are at *minor damage* or *normal*. If not, transitions to RTL MODE are forbidden and transitions to EL MODE should be made.
SR3	The transition from STANDBY MODE to JOINING-INIT MODE is allowed when all the subsystems are at *normal*. If not, transitions to JOINING-INIT MODE are forbidden and the receiver should wait in STANDBY MODE.
SR4	Staying at JOINING-WAIT MODE is allowed when Fuel subsystem is at *normal* and other subsystems are at *minor damage* or *normal*. If not, transitions to JOINING-WAIT MODE are forbidden and transitions to the withdrawal phase should be made.
SR5	If the waiting time at JOINING-WAIT exceeds a specified threshold, then the receiver should make a transition to STANDBY MODE.
SR6	Entering REFUELING-INIT MODE is allowed when Navigation, Engine, Datalink and Tankersafety subsystems are at *minor damage* or *normal*, while Control, Fuel and Drogue&probe subsystems are at *normal*. If not, the receiver, even if cleared for connection, should be kept waiting at JOINING-WAIT MODE or make transitions to the withdrawal phase.
SR7	Staying at REFUELING-CAPTURE MODE or REFUELING-TRANSFER MODE (when the fuel transfer is not finished) is allowed when all the subsystems are at *normal*. If this requirement is not satisfied, the receiver is not allowed to stay at this mode. Under this situation, if Navigation, Engine, Datalink and Tankersafety subsystems are at *minor damage* or *normal*, while Control, Fuel and Drogue&probe subsystems are at *normal*, the receiver should retreat to REFUELING-INIT MODE. Otherwise, the receiver should make transitions to the withdrawal phase.
SR8	When the fuel transfer is finished, staying at REFUELING-TRANSFER MODE is allowed when Navigation, Fuel, Engine, Datalink and Tankersafety are at *minor damage* or *normal*, while Control and Drogue&probe are at *normal*. If not, the receiver should make transitions to the withdrawal phase.
SR9	When the receiver is waiting for disconnection clearance at REFUELING-TRANSFER MODE after a successful fuel transfer, the receiver should disconnect from the drogue and switch to REFUELING-INIT MODE, if the waiting time exceeds a specified threshold.
SR10	Entering and Staying at REFORMING-INIT MODE or REFORMING-FORMATION MODE are allowed when Navigation, Control, Fuel and Engine subsystems are at *minor damage* or *normal*. If not, transitions to the withdrawal phase should be made.

4.2. *Event definitions*

Based on event characteristics of AAR tasks, four types of events are defined here, namely Mode Control Events (MCEs), Mode Input Events (MIEs), Automatic Triggered Events (ATEs) and Subsystem Failure Events (SFEs). MCEs and MIEs are controllable events, while ATEs and SFEs are uncontrollable events. Their detailed descriptions are given as follows:

(1) **MCEs:** Commands generated by the autopilot to proceed automatically.
(2) **MIEs:** Instructions sent from pilots to change the automatic AAR procedures.
(3) **ATEs:** Detection results of AAR maneuvers. In most cases, an MCE has two possible ATEs, which mean success and failure. For example, the event MCE04 is to control the receiver to fly from the STANDBY position to the observation area, thus this command has two possible results: "arrived" ("ATE01: Join-Init-Succ") and "not arrived" ("ATE02: Join-Init-Fail").
(4) **SFEs:** Failure related behaviors of subsystems. These events correspond to transitions including *x*_suspension, *x*_recover and *x*_breakdown, which are presented in Sec. 3.3.

The exact definition of every MCE is shown in Table 4. As for the MIEs, ATEs and SFEs, their meanings can be easily interpreted from their full names as shown in Figs. 10–19. Their detailed definitions are presented in Appendix B.

Table 4. Mode control event definitions.

Name	Description
MCE01	Control the receiver to stay in EL MODE.
MCE02	Control the receiver to stay in RTL MODE.
MCE03	Control the receiver to stay in STANDBY MODE.
MCE04	Control the receiver flying from STANDBY position to the observation area.
MCE05	Control the receiver to wait in the observation area while avoiding collision with other aircraft.
MCE06	Control the receiver flying to the astern area.
MCE07	Control the receiver to abandon the current connection initiation or fuel transfer initiation and fly to the astern area.
MCE08	Control the receiver to connect its probe with the tanker's drogue.
MCE09	The forced version of MCE08. It can only be activated by pilots, and cannot be forbidden by autopilots.
MCE10	Control the receiver to keep relatively stationary to the tanker, and open the valve to receive fuel.
MCE11	The forced version of MCE10. It can only be activated by pilots, and cannot be forbidden by autopilots.
MCE12	Control the receiver to wait in the astern area while keeping connected with the tanker.
MCE13	Control the receiver flying to the reform area.
MCE14	Control the receiver to rejoin the formation.

5. State Tree Structure Design

Based on the preparation in Secs. 3 and 4, this section shows the AAR failsafe design in the form of state tree structures, whose overall diagram is illustrated in Fig. 8.

5.1. *Plant design*

In this subsection, the plant design is presented, which includes all the possible behaviors of the aerial refueling tasks.

5.1.1. (*Receiver*) *autopilot*

Autopilot is the AND component of *Receiver*, which is responsible for recording the task procedures of AAR. It is an OR superstate with three simple states (*Standby*, *RTL* and *EL*) and three OR superstates (*Joining*, *Refueling* and *Reforming*), as shown in Fig. 9.

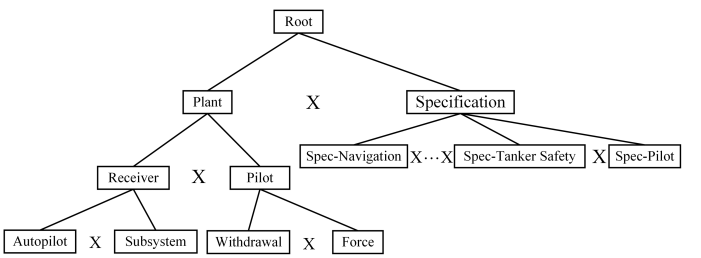


Fig. 8. The state tree of AAR with only AND superstates and their components displayed.

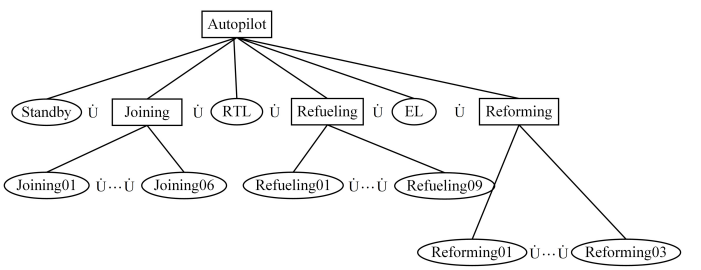


Fig. 9. The state tree of *Autopilot*. *Joining*01, . . . , *Joining*06 are children (OR components) of superstate *Joining*, whose detailed structure is shown in Fig. 11. It is similar to *Refueling* and *Reforming*, whose structure is shown in Figs. 12 and 13, respectively.

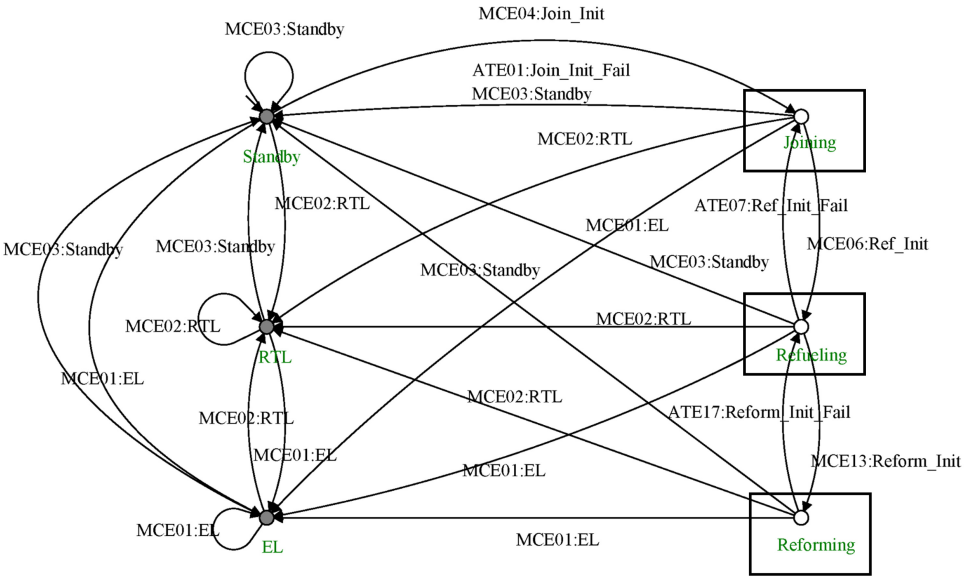


Fig. 10. The holon of *Autopilot*. Figures are plotted through SUPREMICA [23]. Note that circles with arrow pointing to are initial states like *Standby*, and gray circles are marker states like *RTL*. The black squares indicate superstates.

Figure 10 shows the inner transitions of *Autopilot* (Detailed information on the three superstates is introduced later). In the normal work cycle, the *Autopilot* starts from *Standby* state, goes through the *Joining* superstate (MCE04), the *Refueling* superstate (MCE06), the *Reforming* superstate (MCE13) and finally returns to the *Standby* state (MCE03). In detail, taking event MCE04 as an example, it leads the receiver from *Standby* state to *Joining* superstate. When this maneuver fails, namely "ATE01:Join-Init-Fail" happens, the receiver retreats to *Standby* state. Otherwise, "ATE02:Join-Init-Succ" happens, which leads the *Autopilot* to *Joining* superstate (shown in Fig. 11).

When failures occur, the *Autopilot* can retreat to *Standby*, *RTL* and *EL* states from *Joining*, *Refueling* and *Reforming* superstates through events MCE03, MCE02 and MCE01, respectively. Meanwhile, according to the real-time health

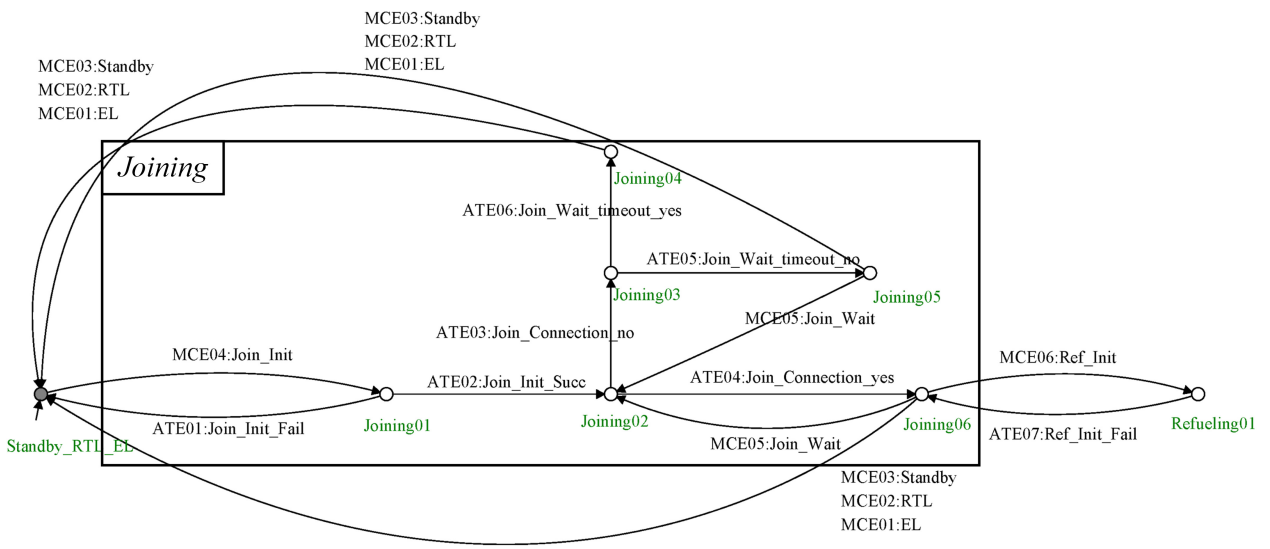


Fig. 11. The holon of *Joining*. The simple states and transitions inside superstate *Joining* are enclosed by the black square. To avoid messy transition lines, this diagram has been simplified by using node *Standby_RTL_EL* to represent three separate nodes including *Standby*, *RTL* and *EL*, which have the same transition lines from node *Joining*04, *Joining*05, and *Joining*06. The following two diagrams have been simplified as well.

conditions, the receiver can make transitions among states in the withdrawal phase like from *RTL* to *Standby*.

(a) *Joining* superstate: This is an OR superstate as shown in Fig. 11. It corresponds to the joining sub-phase presented in Sec. 3.2, and contains six simple states.

When MCE04 happens, the *Autopilot* enters *Joining* superstate from *Standby*, and stays in *Joining*01 state (belonging to JOINING-INIT MODE). When this maneuver succeeds ("ATE02:Join-Init-Succ"), the receiver enters *Joining*02 state (belonging to JOINING-WAIT MODE), where it has to wait for the connection clearance from the tanker (FD4, see Table 2). When the receiver is not cleared ("ATE03:Join-Connection-no"), it has to check whether the waiting time exceeds the specified threshold (SR5, see Table 3). If so, the receiver has to abandon the current AAR task and return to the withdrawal phase. Otherwise, the receiver can keep waiting (MCE05) or retreat to the withdrawal phase according to its health conditions. Once the receiver is cleared for connection ("ATE04:Join-Connection-yes"), it has to check whether it satisfies the safety requirement for MCE06 (SR6, see Table 3). If satisfied, the receiver can execute the event MCE06. Otherwise, it has to wait or withdraw.

(b) *Refueling* superstate: This is an OR superstate as shown in Fig. 12. It corresponds to the Refueling sub-phase, and contains nine simple states.

Event MCE06 brings the *Autopilot* from *Joining* to *Refueling* superstate. In the *Refueling*02 state (belonging to REFUELING-CAPTURE MODE), according to the safety requirement (SR7), the receiver can choose to initiate a connection (MCE08), back to *Refueling*01 state (MCE07) or back to the withdrawal phase. MCE09 is the substitute for MCE08, so that the pilot can use it to force the connection action when MCE08 is forbidden by the generated supervisor (FD6). The connection action has two results: "ATE09:Ref-Cap-Fail" and "ATE10:Ref-Cap-Succ". Failure brings the receiver back to the *Refueling*02 state while success leads the receiver to the *Refueling*04 state (belonging to REFUELING-TRANSFER MODE), whose transitions are similar to those of *Refueling*02 state.

Success of fuel transfer leads the receiver to *Refueling*06 state, where it has to wait for the disconnection clearance (FD4). If not cleared, the receiver has to check the waiting time (SR9), similar to *Joining*02 state. If time runs out, the receiver will directly disconnect with the tanker instead of returning to the withdrawal phase. If cleared, the receiver can initiate the event MCE13 or return to the withdrawal phase.

(c) *Reforming* superstate: This is an OR superstate as shown in Fig. 13. It corresponds to the Reforming sub-phase, and contains three simple states. MCE13 will bring the receiver from *Refueling* to the *Reforming* superstate. Its success will lead to the *Reforming*02 state, where the receiver can return to the withdrawal phase due to health conditions or form the formation (MCE14). Event "ATE20: Reform-Formation-Succ" implies the completion of a work cycle of AAR, and then the receiver should return to the withdrawal phase.

5.1.2. (*Receiver*) *Subsystem*

Subsystem is another AND component of Receiver, working in parallel with *Autopilot*. This superstate is responsible for recording the health information of different subsystems, namely Navigation, Control, Fuel, Engine, Drogue&probe, Datalink and Tanker safety subsystems. Their state trees are shown in Fig. 14, whose holon structures are similar to Figs. 6 and 7 and omitted here for brevity.

5.1.3. (*Pilot*) *Withdrawal*

Withdrawal is an AND component of *Pilot*, which is responsible for implementing the FD7 in Table 2. Its holons are shown in Fig. 15. As shown in this figure, when "MIE02: RTL" happens, i.e., the pilot requires the receiver retreat to

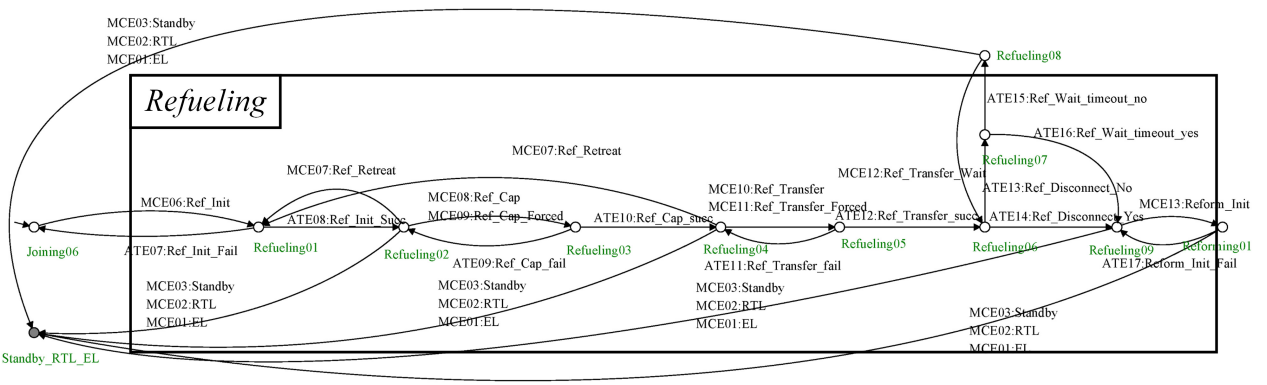


Fig. 12. The holon of *Refueling*.

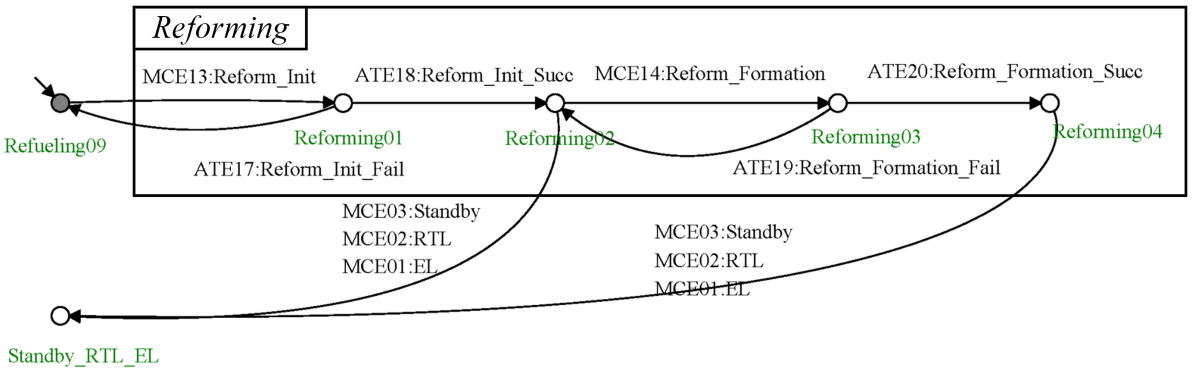


Fig. 13. The holon of *Reforming*.

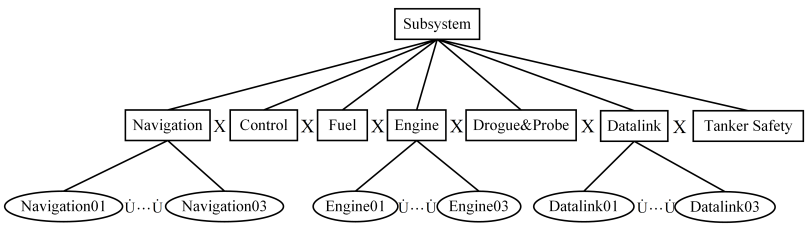


Fig. 14. The state tree of *Subsystem*. Only part of the simple states are presented here owing to limited space. These simple states represent different health conditions. For example, *Navigation*01 means *normal*, *Navigation*02 means *minor damage*, and *Navigation*03 means *critical damage*.

RTL MODE, both MCE02 and MCE01 are allowed to happen. This means the receiver can also switch to EL MODE besides RTL MODE in case the receiver is unhealthy and cannot return to base.

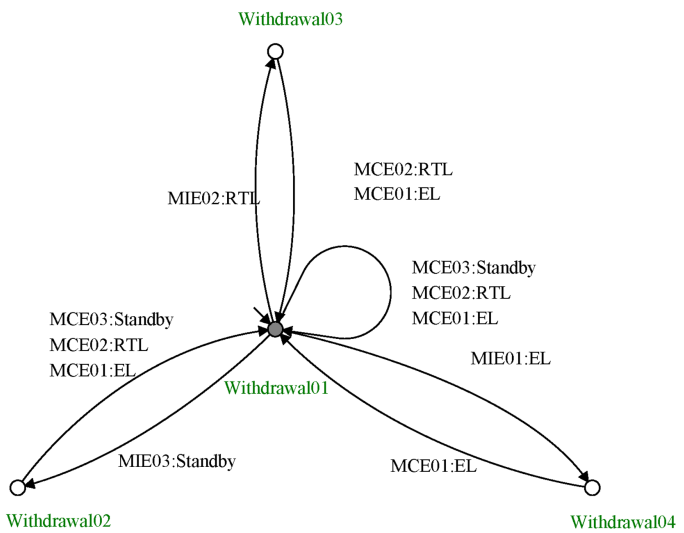


Fig. 15. The holon of *Withdrawal* superstate.

5.1.4. (*Pilot*) *Force*

Force is another AND component of *Pilot* and implements the FD6 (in Table 2). As shown in Fig. 16, MCE09, a substitute of MCE08, can only be enabled when "MIE04:Force-Ref-Cap" happens, i.e., the pilot gives the corresponding command. After MCE09 or MCE08, this subsystem will return to the original state *Force*01. MCE03 is similar to MCE09.

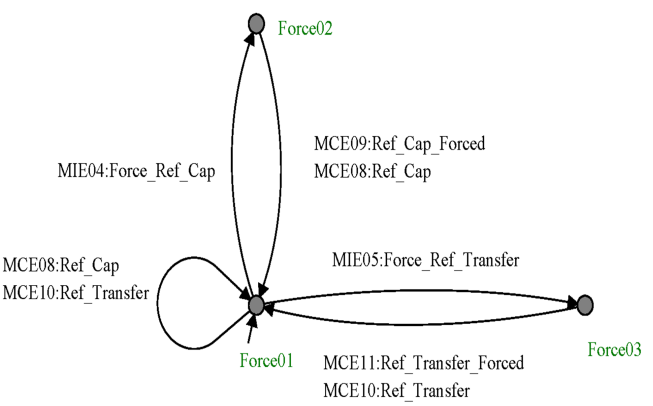


Fig. 16. The holon of *Force* superstate.

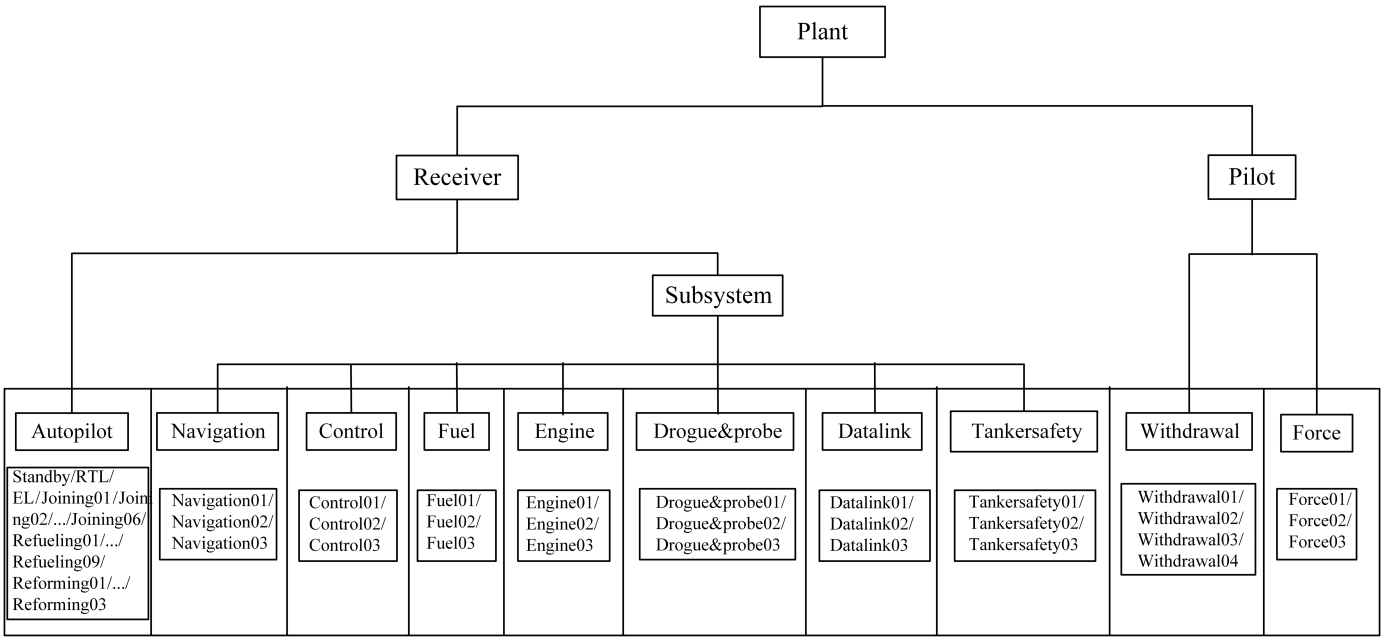


Fig. 17. Plant overview.

5.1.5. *Plant summary*

So far, all the superstates and simple states of the *Plant* have been introduced, as shown in Fig. 17. Recall that the state of superstate depends on states of its children. Therefore, the state of *Plant*, namely *Plant State*, can be represented as a 10-tuple

$$\mathrm{PS} = (x_1, x_2, \ldots, x_{10}),$$

where x_i is the state of the 10 superstates in the sequence of *Autopilot*, *Navigation*, *Control*, *Fuel*, *Engine*, *Drogue&probe*, *Datalink*, *Tankersafety*, *Withdrawal* and *Force*. An example of plant state is $PS = (RTL, Navigation01, Control01, Fuel01, Engine01, Drogue\&probe01, Datalink01, Tankersafety02, Withdrawal03, Force03)$, where the meaning of $Control01, Fuel01, \ldots, Tankersafety02$ is displayed in Fig. 14.

5.2. *Specification design*

Specifications are mainly used to implement the safety requirements, restricting the undesired and unsafe behaviors of the system. As stated in Sec. 2.2, these parallel specifications eliminate, in a minimally restrictive fashion, all behaviors leading to blocking states. They can be divided into two categories, including subsystem-related and pilot-related, as introduced in the following.

5.2.1. *Subsystem-related specifications*

These specifications, including *Spec-Navigation*, *Spec-Control*, *Spec-Fuel*, *Spec-Engine*, *Spec-Drogue&probe*, *Spec-Datalink* and *Spec-Tankersafety* are used to restrict system behaviors when certain subsystems are damaged or break down according to the requirements presented in Table 3.

Take the *Spec-Navigation* as example. According to SR3 and SR7, when "SFE01: Navigation-suspension" happens, MCE04, MCE08 and MCE10 should be disabled. Therefore, as shown in Fig. 18, event sequences including (SFE01, MCE04), (SFE01, MCE08) and (SFE01, MCE10) will lead *Spec-Navigation* into blocking state *SpecNavi99*. And according to all safety requirements, when "SFE03:Navigation-breakdown" happens, all MCEs except for MCE01 should be forbidden. This means that when the Navigation subsystem breaks down, the receiver should make an emergency landing.

5.2.2. *Pilot-related specifications*

This part refers to the specification *Spec-pilot*, which is used to implement the priority of pilot commands (MIEs) over the autopilot commands (MCEs), as stated in FD5. That is to say, when MIEs happen, except for MCE01, MCE02 and MCE03, other MCEs should be forbidden. As shown in the Fig. 19, when "MIE02: RTL" happens, events such as MCE04 and MCE05 are forbidden. Under this situation, the receiver has to perform corresponding mode transitions MCE02 or MCE01.

5.3. *Supervisor*

The *Plant* provides the plant state, a 10-tuple representing the state of the whole system, while the *Specification* points

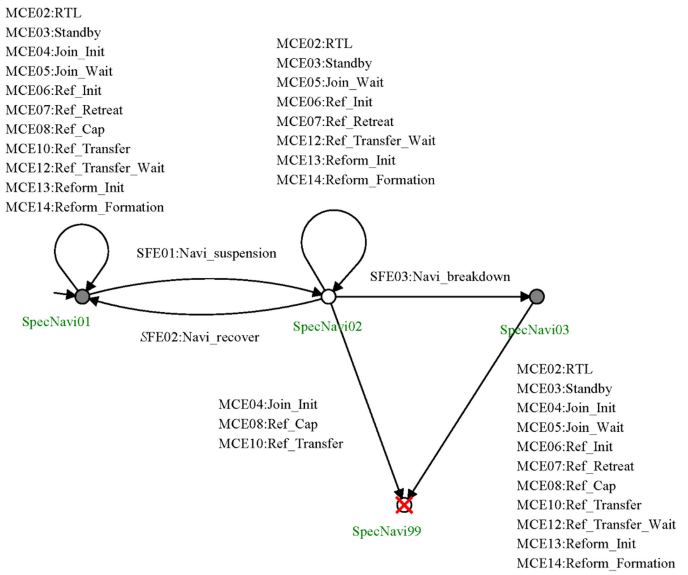


Fig. 18. The holon of *Spec-Navigation*.

out the forbidden plant states. The supervisor is responsible for disabling certain controllable events (or enabling the rest) at certain plant states to guarantee that the *Plant* never reaches those forbidden plant states. The STSLib developed by [24] is used to compute the supervisor of AAR, which finishes within 2 s on a personal laptop with 3.2 GHZ I5 CPU and 8GB RAM. The supervisor is implemented in binary decision diagrams (BDD) for every MCE, and is converted into a lookup table including all plant states where the MCEs are allowed to happen. The converting program can be found in the supporting material.

Take the resulting supervisor for MCE02 as an example. Its supervisor is shown in Table 5, which shows eight different plant states where MCE02 is allowed to happen. That

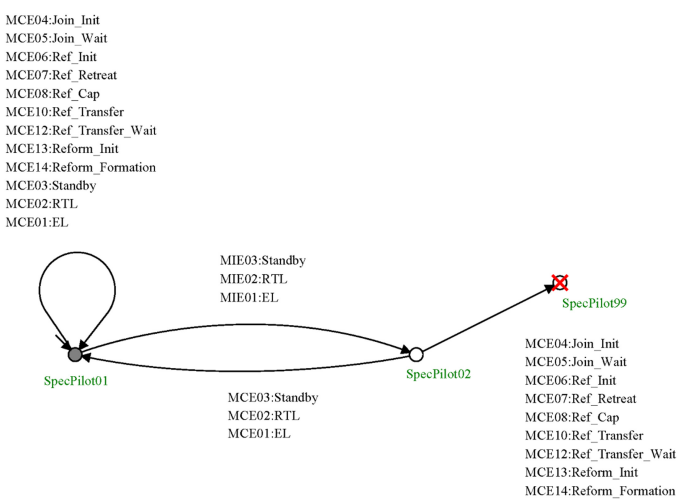


Fig. 19. The holon of *Spec-pilot*.

Table 5. Supervisor for MCE02 in the form of a lookup table.

Scenarios	Navigation	Fuel	Engine
1	1	1	1
2	1	1	2
3	1	2	1
4	1	2	2
5	2	1	1
6	2	1	2
7	2	2	1
8	2	2	2

only *Navigation*, *Fuel* and *Engine* are presented means that other superstates in the 10-tuple (such as *Control, Datalink*) have no influence on the enablement of MCE02 and no need for checking as well. For example, plant state (*Standby*, *Navigation*01, *Control*01, *Fuel*01, *Engine*02, *Drogue&probe*01, *Datalink*01, *Tankersafety*01, *Withdrawal*01, *Force*01) satisfies the second scenario in Table 5 and MCE02 is allowed to happen at this state.

Putting all these different lookup tables together and taking the union of related superstates, the supervisor for the whole system can be presented in Table 6. Although the whole system has 413343 states in total, this table has only 235 rows, which shows the power of STS. Note that only 8 superstates need to be checked in the supervisor, fewer than the 10 superstates of plant states. The *Autopilot* and *Force* are ignored since there are no specific requirements about them in the specifications.

6. Implementation and Simulation

With the well-designed *Plant* and obtained supervisor, this section presents the implementation architecture of applying such a logic controller to the autonomous receiver control system. Based on this, a simulation platform of AAR is built using MATLAB2016b to test the logical correctness.

6.1. *Implementation architecture*

As shown in Fig. 20, the logic controller should be put between low-level controllers and information sources including *Sensor management*, *Prognostics and Health Management* (PHM) [25] and *Communication management*. The sensor management module is used to estimate the state of receivers such as location, velocity and altitude, determine what mode the receiver is currently in and then output the ATEs. For example, if this module confirms that the receiver has not arrived at the astern area (point *C* in Fig. 3), then "ATE07:Ref-Init-Fail" is generated and output.

Table 6. Supervisor in the form of a lookup table. Only part of the table is shown.[1]

Event	Navigation	Control[2]	Fuel	Engine	Drogue probe	Datalink	Tanker safety	Withdrawal
MCE02	1	1–3	1	1	1–3	1–3	1–3	1–4
MCE02	1	1–3	1	3	1–3	1–3	1–3	1–4
MCE07	1	1	1	1	1	1	1	1
MCE07	1	1	1	1	1	1	3	1
MCE07	1	1	1	1	1	3	1	1
MCE07	1	1	1	1	1	3	3	1
MCE12	3	1	1	1	1	1	1	1
MCE12	3	1	1	1	1	1	3	1
MCE12	3	1	1	1	1	3	1	1
MCE12	3	1	1	1	1	3	3	1
MCE14	3	3	1	3	1–3	1–3	1–3	1
MCE14	3	3	3	1	1–3	1–3	1–3	1
MCE14	3	3	3	3	1–3	1–3	1–3	1

Notes: [1]In the implementation, states are represented by binary bits instead of decimal digits to improve program searching performance.
[2]Here, 1–3 means that *Control* could be at *Control*01, *Control*02 or *Control*03. It is similar for other superstates.

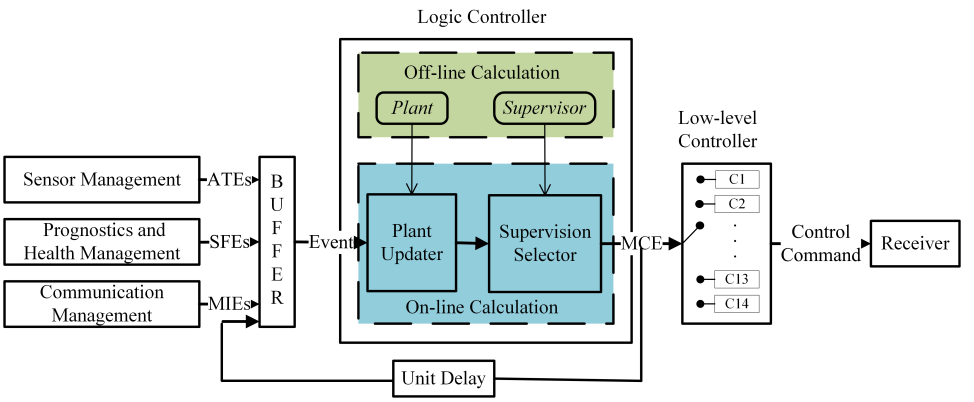


Fig. 20. The implementation architecture.

The PHM module is used to detect subsystem failures and generate SFEs. The communication management is responsible for receiving pilot instructions and generating MIEs, e.g., when the pilot requires the receiver to return to land then "MIE02:RTL" is generated. The low-level controller is used to generate the direct control command for the receiver's velocity, acceleration, angular velocity, etc. There are 14 low-level controllers in correspondence to those 14 target modes such as STANDBY, JOINING-WAIT, and REFUELING-CAPTURE. At run time, the logic controller will choose one and only one low-level controller at the same time, i.e., only one MCE would be enabled.

The logic controller is divided into two parts: offline and online. The *Plant* (a set of holons or automaton with hierarchy structures) and *Supervisor* (a loop-up table) can be computed offline in advance and used as the input of online calculation. As for the online part, the *Plant Updater* module receives events including ATEs, SFEs and MIEs from the corresponding source and then updates the current plant state according to the pre-defined *Plant* holons. Note that the high-level decision-making is relatively slow in practice, but the low-level detection is fast. For example, the event ATEs, SFEs may be generated every 0.01 s, but the decision period may be 1 s. Thus, a buffer is added to store those events according their occurrence sequence, and then to feed them all into the updater in every decision period. The *Supervision Selector* would enable certain MCEs according to the current plant state and the pre-defined

Table 7. Pseudo-code for online logic controller.

Online logic controller
Input: The holons of *Plant*, the lookup table *Supervisor*, initial *Plant State* $\mathbf{S} = \mathbf{S_0}$, Δ is a positive integer representing a mode decision period, and a time counter $k \in N$ and starts at 0, namely $k = 0$.
Step 1: $k = k + 1$.
Step 2: The sensor management, PHM and communication management detect the occurrence of ATEs, SFEs and MIEs. if k mod $\Delta = 0$, go to Step 3; Otherwise, go to Step 1.
Step 3: Collect events occurring in the mode decision period Δ.
Step 4: Update the plant state $\mathbf{S}$ to $\mathbf{S_1}$ with events according to their occurrence sequence one by one.
Step 5: Enable certain MCEs according to *Supervisor* and current state $\mathbf{S_1}$. Compare the priority of enabled MCEs and select one MCE with the highest priority.
Step 6: Update the plant state $\mathbf{S_1}$ to $\mathbf{S_2}$ with the selected MCE.
Step 7: $\mathbf{S} = \mathbf{S_2}$, go to Step 1.

Supervisor. The pseudo-code for the procedure is shown in Table 7 (The priority comparison presented in this algorithm is explained in Sec. 6.2).

6.2. *MCE priority*

As mentioned before, the supervisor enforces the minimally restrictive behavior (set of generated event sequence) of plants under the restriction of specifications. Therefore, it will happen that several MCEs may be enabled at the same time and the same plant state. However, the aircraft can only be at one state, i.e., only one MCE can be picked and executed. Thus, the priority comparison among different MCEs is necessary for engineering practice. In this paper, the priority of MCEs is given according to their nodal distance to the success of AAR as defined in the flowchart shown in Fig. 21. This priority is reflected by their indexes. That is to say, MCE01 possesses the lowest priority while MCE14 enjoys the highest.

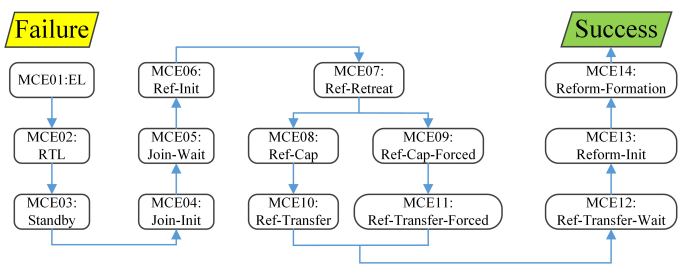


Fig. 21. The flowchart of all MCEs in the AAR task. The distance of MCE is the nodal distance between the MCE node and the "Success" node along the line.

For example, when the *Plant* is at state (*Refueling*02, *Navigation*01, *Control*01, *Fuel*01, *Engine*01, *Drogue& probe*01, *Datalink*01, *Tankersafety*01, *Withdrawal*01, *Force*01), MCE01, MCE02, MCE03, and MCE08 are all enabled. But since REFUELING-CAPTURE MODE is proceeding to the success of AAR but STANDBY, RTL and EL are proceeding to the failure of AAR. Therefore, the event transiting to REFUELING-CAPTURE MODE is assigned with a higher priority than events transiting to STANDBY, RTL and EL, and would be chosen in this state.

6.3. *Simulation*

The simulation platform is built within MATLAB2016b according to the implementation architecture shown in Fig. 20. This platform mainly consists of three parts: a Graphical User Interface (GUI), processing programs and a 3D visualization environment, as shown in Figs. 22 and 23. The GUI enables the user to start, pause and terminate the simulation, input pilot commands (MIEs) and subsystem failures (SFEs),[d] and check the plant state and current MCE. The processing programs are the MATLAB functions used to implement the logic controller and low-level controller illustrated in Fig. 20. In detail, the embedded dynamic models of Fighter-16 and Boeing-727 in MATLAB2016b are used to work as the receiver and the tanker.[e] Each low-level controller (in correspondence to each MCE) would give commands such as engine thrust, pitch angle, yaw angle to control the movement of the receiver. The 3D visualization environment presents the

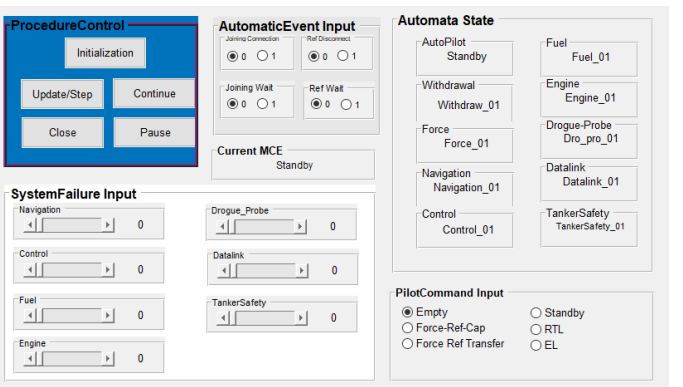


Fig. 22. Graphical user interface.

[d] The ATEs can be detected and generated automatically by the processing programs.

[e] For more detailed information about dynamic models and controller designs, please refer to https://www.youtube.com/watch?v=spuXvSr31D8&feature=youtu.be.

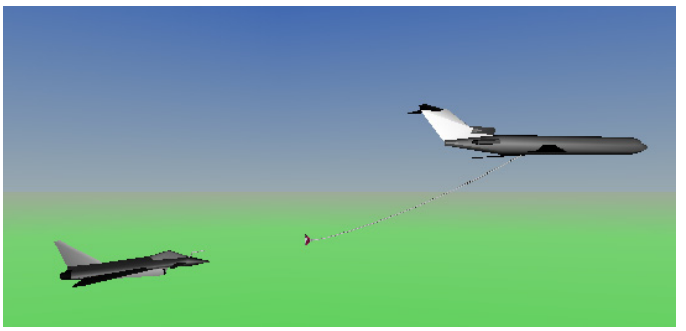


Fig. 23. 3D simulation environment.

real-time state of the receiver, tanker, and the drogue-probe for a more direct observation.

Within this simulation platform, the user can add subsystem failures and pilot commands at any time. The failsafe mechanism or online logic controller will select out the best MCE according to the receiver's current health conditions and pre-defined safety requirements. The corresponding low-level controller will then control the movement of the receiver, which will be shown in the 3D simulation environment. This can enable the user to test the receiver's behavior under different possible situations. Three different testing scenarios are presented as follows, which cover the typical functional and safety requirements of AAR. Their textual descriptions are in Appendix A, while videos can be found in Youku and Youtube.[f]

6.4. *Comparison with document-based method*

Compared with traditional document-based methods, the most important advantage of our methodology is that it is "correct by design": The logic controller is generated automatically by the STS program, and there will be no logic bugs in the final supervisor when all necessary requirements are provided by the user. If the final supervisor's behavior at some scenarios is not expected by the user, all we need to do is to check the corresponding requirement plants.

However, traditional document-based methods usually rely on developers to manually merge different requirements from different documents through complex and lengthy nested control-of-flow statements (e.g., "while", "if...else..."). This is time-consuming and error-prone. When the final logic supervisor's behavior is not satisfactory, we need to check both the logic implementations in lengthy code and the user requirements.

For example, when implementing FD 6 in Table 2 and SR 6 in Table 3, a typical pseudo-code implementation is

[f]For Youtube, https://youtu.be/R4-eMR9zqr0. For Youku, https://v.youku.com/v_show/id_XMzgwMTgwODU0OA==.html.

```
.....

Enable = False

if (Naviagtion is minor damage) or (Navigation is normal):
   if (Engine is minor damage) or (Engine is normal):
            if (Datalink is minor damage) or (Datalink is normal):
                     if (Tankersafety is minor damage) or (Tankersafety is normal):
                              if (Control is normal):
                                       if (Fuel is normal):
                                                if (Drogue&probe is normal):
                                                         Enable = True
                                                end
                                       end
                              end
                     end
            end
   end
end

If (Pilot_Enforce == True):
   Enable = True
end

.....
```

Fig. 24. A pseudo-code implementation example of SR 6 and FD 6.

shown in Fig. 24. SR 6 specifies the safety requirement of entering REFUELING-INIT MODE, while FD 6 requires that even such safety requirements are not satisfied, the pilot can still force the airplane to enter REFUELING-INIT MODE. If the developer forgets to add the priority of pilot commands or misses one of the safety requirements (highlighted in red boxes in Fig. 24), it is hard to locate them in lengthy code. Furthermore, a modification in one place may result in unexpected bugs in another; this is a common situation in the nested control-of-flow statements. But in our method, all we need to do is to check the relevant plant or specification graphs (e.g., Fig. 16) and then regenerate the logic controller automatically. Checking a missing event or node in a visualized graph is far easier than checking logic bugs in nested control-of-flow statements and lengthy code.

7. Conclusion

This paper presents a new way to synthesize the failsafe mechanism of autonomous aerial refueling, namely the State Tree Structure. This formal method helps the system designer clarify all aspects of a client's requirements at the early stage of product development. It can precisely define every detail of the logic control system, and thus achieve less ambiguity and more consistency compared with traditional document-based methods.

To apply this method to AAR, the continuous AAR procedure is first decomposed into 10 discrete flight modes such as STANDBY, JOINING-INIT and REFUELING-CAPTURE MODE, and common safety issues of seven important

subsystems like Navigation and Drogue &Probe are collected. These describe the unrestricted behaviors of AAR, and thus define the plants of STS. Then, common user requirements including seven functional demands and 10 safety requirements are summarized. The functional demands emphasize what behaviors must be included in the system, so they are used to guide the design of plants. Since the safety requirements point out what behaviors are illegal, they define the specifications of STS. Finally, the software package STSLib is used to compute the supervisor. This final logic controller only has 235 rows, but controls a system with 413343 states.

Based on this, an implementation scheme is proposed and tested in a simulation environment built within MATLAB2016b. Three test cases are presented, recorded and proven to be error-free. Related tools and programs have been uploaded onto Github. The compact form of supervisor and the success of implementation reveals the power and significant potential of supervisory control theory in the field of AAR.

Acknowledgments

This work was partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada, Grant #DG_480599, and by the National Natural Science Foundation of China under Grant 61473012.

Appendix A. Testing Cases for the Simulation Platform

Case A: Transitions in Withdrawal Phase: According to the FD1 and FD5 shown in Table 2, the pilot can force the receiver to switch from the task phase to the withdrawal phase, and the receiver would choose the best mode according to its health conditions (related with SR1 and SR2 shown in Table 3). When the autopilot is at JOINING-WAIT MODE, namely *Joining*05 state. When "MIE03:EL" happens, according to FD5 (implemented in *Spec-pilot* shown in Fig. 19), MCE05 is forbidden, and the *Autopilot* has to choose one routine from MCE01–MCE03. Since its health conditions satisfy SR1, MCE03 is enabled and the *Autopilot* enters *Standby* state. But then if the Control subsystem is in *critical damage*, SR1 is not satisfied anymore. According to FD1 (implemented in *Spec-Control*), MCE03 is forbidden and the *Autopilot* executes the MCE02 instead.

Case B: Transitions in Joining-Refueling Phases: According to SR1, SR2 and SR6 shown in Table 3, the transition to REFUELING-INIT MODE requires certain subsystems to be healthy. Otherwise, the receiver has to retreat to the joining phase or withdrawal phase. Say that the receiver is at *Joining*06 state and REFUELING-INIT MODE, and flies from point *B* to point *C* as shown in Fig. 3. Then the Navigation subsystem is in *minor damage*, according to SR6, the receiver can still remain in REFUELING-INIT MODE. But then the Drogue&probe subsystem is in *minor damage*. According to SR6, the receiver has to retreat to JOINING-WAIT MODE. Otherwise, if the Navigation subsystem is in *critical damage*, then according to SR1 and SR2, the receiver should change to EL MODE right away.

Case C: Transitions in Refueling Phase: According to SR7 shown in Table 3 and FD6 in Table 2, in the REFUELING-CAPTURE MODE, if certain subsystems are in *minor damage*, the receiver abandons the current capture and retreats to the REFUELING-INIT MODE. But then the pilot can force the receiver to initiate the refueling capture even if the health conditions do not allow this decision. When the *Autopilot* is at *Refueling*02 state and the Datalink subsystem is in *minor damage*, then MCE08 is forbidden (implemented in *Spec-Datalink*). Therefore, MCE07 is executed. Since the receiver does not satisfy SR7, it has to wait at the REFUELING-INIT MODE. But when "MIE04:Force-Ref-Cap" is given, the MCE09 is enabled (implemented in *Force* shown in Fig. 16). Thus the receiver is forced to initiate the refueling capture.

Appendix B. Event Definitions

This appendix presents the complete definitions of MIEs, SFEs and ATEs in Tables B.1–B.3. The definitions of MCEs have already been given in Table 5.

Table B.1. Mode input event definitions.

Name	Description
MIE01	Require the receiver to return to STANDBY MODE as soon as possible.
MIE02	Require the receiver to return to RTL MODE as soon as possible.
MIE03	Require the receiver to return to EL MODE as soon as possible.
MIE04	Activate MCE09
MIE05	Activate MCE11

Table B.2. Failure related event definitions.

Name	Description
SFE01: Navigation-suspension SFE02: Navigation-recover SFE03: Navigation-breakdown	The failure related behaviors of Navigation subsystem.
SFE04: Control-suspension SFE05: Control-recover SFE06: Control-breakdown	The failure related behaviors of Control subsystem.
SFE07: Fuel-suspension SFE08: Fuel-breakdown	The failure related behaviors of Fuel subsystem.
SFE09: Engine-suspension SFE10: Engine-recover SFE11: Engine-breakdown	The failure related behaviors of Engine subsystem.
SFE12: dro-pro-suspension SFE13: dro-pro-recover SFE14: dro-pro-breakdown	The failure related behaviors of Drogue&probe subsystem.
SFE15: Datalink-suspension SFE16: Datalink-recover SFE17: Datalink-breakdown	The failure related behaviors of Datalink subsystem.
SFE18: Tanker-suspension SFE19: Tanker-recover SFE20: Tanker-breakdown	The failure related behaviors of Tankersafety subsystem.

Table B.3. Automatic triggered event definitions.

Name	Description
ATE01	The event MCE04 fails, i.e., the receiver does not arrive at point B[1].
ATE02	The event MCE04 succeeds, i.e., the receiver has arrived at point B.
ATE03	The receiver is not cleared for connection.
ATE04	The receiver has been cleared for the connection.
ATE05	The waiting time at the observation does not exceed the specified threshold.
ATE06	The waiting time at the observation area exceeds the specified threshold.
ATE07	The event MCE06 fails, i.e., the receiver has not arrived at point C.
ATE08	The event MCE06 succeeds, i.e., the receiver has arrived at point C.
ATE09	The event MCE08 fails, i.e., the receiver fails to connect its probe with the tanker's drogue.
ATE10	The event MCE08 succeeds, i.e., the receiver successfully connects its probe with the tanker's drogue.
ATE11	The event MCE10 fails, i.e., the tanker fails to transfer the fuel to the receiver.
ATE12	The event MCE10 succeeds, i.e., the tanker successfully transfers the fuel to the receiver.
ATE13	The receiver has not been cleared for disconnection.
ATE14	The receiver has been cleared for disconnection.
ATE15	The waiting time at the astern area does not exceed the specified threshold.
ATE16	The waiting time at the astern area exceeds the specified threshold.
ATE17	The event MCE13 fails, i.e., the receiver has not arrived at point D.
ATE18	The event MCE13 succeeds, i.e., the receiver arrives at point D.
ATE19	The event MCE14 fails, i.e., the receiver has not rejoined the receiver formation.
ATE20	The event MCE14 succeeds, i.e., the receiver has successfully rejoined the receiver formation.

[1]"has", "does", "has not", "does not" phases used here mean the estimated corresponding state is larger or smaller than a given threshold at the sample time of sensors. For example, ATE01 means that the estimated distance between the receiver and point *B* is larger than a given threshold.

References

[1] R. Dibley, M. Allen and N. Nabaa, Autonomous airborne refueling demonstration: Phase i flight-test results, in *Guidance, Navigation, and Control and Co-located Conf.* (South California, USA, 2007).

[2] P. R. Thomas, U. Bhandari, S. Bullock, T. S. Richardson and J. L. du Bois, Advances in air to air refuelling, *Progr. Aerosp. Sci.* **71**(Supplement C) (2014) 14–35.

[3] C. Bolkcom, Air force aerial refueling methods: Flying boom versus hose-and-drogue, tech. rep., Library of Congress Washington DC Congressional Research Service (2006).

[4] A. Degani and M. Heymann, Formal verification of human-automation interaction, *Human Factors* **44**(1) (2002) 28–43.

[5] I. M. Mitchell, A. M. Bayen and C. J. Tomlin, A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games, *IEEE Trans. Automat. Contl.* **50**(7) (2005) 947–957.

[6] J. Ding, J. Sprinkle, C. J. Tomlin, S. S. Sastry and J. L. Paunicka, Reachability calculations for vehicle safety during manned/unmanned vehicle interaction, *J. Guid. Contl. Dyn.* **35**(1) (2012) 138–152.

[7] N. Meskin, K. Khorasani and C. A. Rabbath, A hybrid fault detection and isolation strategy for a network of unmanned vehicles in presence of large environmental disturbances, *IEEE Trans. Contl. Syst. Technol.* **18**(6) (2010) 1422–1429.

[8] Y. Zhang and J. Jiang, Issues on integration of fault diagnosis and reconfigurable control in active fault-tolerant control systems, in *Fault Detection, Supervision and Safety of Technical Processes 2006* (Elsevier, 2007), pp. 1437–1448.

[9] G. P. Falconí and F. Holzapfel, Adaptive fault tolerant control allocation for a hexacopter system, in *American Control Conf. (ACC), 2016* (IEEE, Boston, USA, 2016), pp. 6760–6766.

[10] W. Wonham, K. Cai and K. Rudie, Supervisory control of discrete-event systems: A brief history-1980–2015, *IFAC-PapersOnLine* **50**(1) (2017) 1791–1797.

[11] P. J. G. Ramadge and W. M. Wonham, The control of discrete event systems, *Proc. IEEE* **77** (1989) 81–98.

[12] K. Cai and W. M. Wonham, Supervisor localization: A top-down approach to distributed control of discrete-event systems, *IEEE Trans. Automatic Contl.* **55**(3) (2010) 605–618.

[13] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems* (Springer, New York, 2018).

[14] B. A. Brandin and F. E. Charbonnier, The supervisory control of the automated manufacturing system of the aip, in *Proc. Fourth Int. Conf. Computer Integrated Manufacturing and Automation Technology* (New York, USA, October, 1994), pp. 319–324.

[15] M. Seidl, Systematic controller design to drive high-load call centers, *IEEE Trans. Contl. Syst. Technol.* **14**(2) (2006) 216–223.

[16] R. J. M. Theunissen, M. Petreczky, R. R. H. Schiffelers, D. A. van Beek and J. E. Rooda, Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner, *IEEE Trans. Automat. Sci. Eng.* **11**(1) (2014) 20–32.

[17] D. Harel, Statecharts: A visual formalism for complex systems, *Sci. Comput. Program.* **8**(3) (1987) 231–274.

[18] C. Ma and W. M. Wonham, Nonblocking supervisory control of state tree structures, *IEEE Trans. Automat. Contl.* **51**(5) (2006) 782–793.

[19] R. E. Bryant, Graph-based algorithms for boolean function manipulation, *IEEE Trans. Comput.* **C-35** (1986) 677–691.

[20] S. T. J. Forschelen, J. M. V. D. Mortel-Fronczak, R. Su and J. E. Rooda, Application of supervisory control theory to theme park vehicles, *Discrete Event Dyn. Syst.* **22** (2012) 511–540.

[21] J. Markovski, K. Jacobs, D. van Beek, L. Somers and J. Rooda, Coordination of resources using generalized state-based requirements, in *IFAC Proc. Volumes 10th IFAC Workshop on Discrete Event Systems*, Vol. 43(12) (Berlin, Germany, 2010), pp. 287–292.

[22] N. S. Agency, Air to air refuelling, ATP-56(B), Tech. Rep., North Atlantic Treaty Organization (2010).

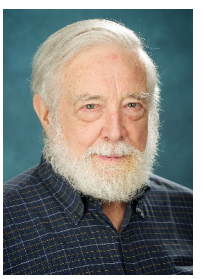
[23] R. Malik, K. kesson, H. Flordal and M. Fabian, Supremica-an efficient tool for large-scale discrete event systems, *IFAC-PapersOnLine* **50**(1) (2017) 5794–5799.

[24] C. Ma and W. M. Wonham, Stslib and its application to two benchmarks, in *9th Int. Workshop on Discrete Event Systems 2008* (Göteborg, Sweden, 2008), pp. 119–124.

[25] P. W. Kalgren, C. S. Byington, M. J. Roemer and M. J. Watson, Defining phm, a lexical evolution of maintenance and logistics, in *2006 IEEE Autotestcon* (Anaheim, USA, September, 2006), pp. 353–358.

Ke Dong is currently an MA Sc student in Dynamics System Lab at the University of Toronto Institute for Aerospace Studies, Toronto, Canada. He received his BEng degree in Aerospace Engineering from Beihang University, Beijing, China in 2014. His research interests are in the areas of mobile robotics and machine learning.

Quan Quan has been an Associate Professor with Beihang University since 2013. He received his BS and PhD degrees in Control Science and Engineering from Beihang University, Beijing, China, in 2004 and 2010, respectively. His current research interests include reliable flight control and vision-based navigation.



W. Murray Wonham received his BEng degree in Engineering Physics from McGill University, USA, in 1956, and the PhD in Control Engineering from the University of Cambridge, UK in 1961. From 1961 to 1969 he was associated with several US research groups in control, including the Research Institute of Advanced Studies (RIAS), Brown University's Department of Applied Mathematics, and NASA's Electronics Research Center. Since 1970 he has been a faculty member in Systems Control, with the Department of Electrical and Computer Engineering of the University of Toronto. In 1996 he was appointed University Professor and in 2000 University Professor Emeritus. Wonham's research interests have included stochastic control and filtering, geometric multivariable control, and discrete-event systems.