On Modeling and Verifying of Application Protocols of TTCAN in Flight-Control System with UPPAAL

Xiao WU, Heng LING, Yunwei DONG
College of Computer Science, Northwestern Polytechnic University, Xi’an 710072, China

Abstract—TTCAN is the most potential protocol used to construct the communication layer of flight control system of unmanned aircraft vehicle (UAV). In this paper, we propose a novel UAV flight control system design which is based on TTCAN. We not only design the model of the system but also verify its non-functional properties such as reliability, security, schedulability and fault-tolerance ability with the model checker UPPAAL. In addition, we design 58 timed automata to verify UAV flight control system. The result of the experiment is also demonstrated to illustrate that the system is feasible and reliable.

Keywords—Application Protocols of TTCAN; UPPAAL; Model Verification

I. INTRODUCTION

Unmanned aircraft vehicle (UAV) flight control system is one of the most important parts in all airborne equipments. With the requirements of high dependability, designing the UAV flight control system with CAN (Controller Area Network) [1] bus instead of the traditional point-to-point communication is becoming more and more popular, for CAN has many advantages like low spending, strong anti-jamming to electromagnetism, long data transfer distance, high speed, fault detecting etc. However, CAN is event-trigger mechanism based and adopts non-destructive bit-wise arbitration mechanism in Medium Access Control layer, when bus is busy extremely, the nodes with low priority can hardly send messages or send them with a serious delay, which affects the data transmission adversely. TTCAN (time-triggered CAN) [2] is a master-slave time-triggered protocol based on TDMA (Time Division Multiple Access), and messages are sent in settled time slices, which can eliminate bus disputing. The efficiency of the bus will be improved and the data transmission will be more reliable with TTCAN.

The fault-tolerant redundancy [3] is also a good way to improve the dependability of the UVA flight control system. We present a triple redundant UVA flight-control system based on TTCAN in this paper and adopt the model checker UPPAAL [4] to analyze the requirements and design the system model and verify the validity and performance of the system. UPPAAL has been used in various fields [5] [6] [7] [8], such as arithmetic analysis and multimedia application. In recent years, UPPAAL has been used in the study of industry control bus protocol [9] [10]. In our project, TTCAN is adopted to construct communication interface in flight control system of unmanned aircraft vehicle in order to deduce I/O system.

The rest of the paper is organized as follows: the architecture of the UAV flight-control system is introduced in section 2, the design of the system model is described in section 3, the result of the verification is given in section 4, and section 5 concludes.

II. THE ARCHITECTURE OF THE UAV FLIGHT CONTROL SYSTEM

In this paper, the UAV flight-control system mainly consists of host controllers, rudders, and TTCAN bus. The architecture is depicted in Figure 1.

Host controller is the central part of the whole flight-control system, which controls the flight by handling the collected message and sending the control instructions. Rudders adjust the flight according to the control instructions from host controller and send back a response. TTCAN bus is the communication carrier for message transmission. The failures of the system are divided into three categories: 1) unrecoverable failure of host controller; 2) unrecoverable failure of TTCAN controller; 3) the physical line would be broken, short-circuit or else.

Using single host controller may bring tremendous risks. Once the host controller entering an unrecoverable failure, the lack of redundancy in system design may lead to a harmful result. However, this problem can be solved by using backup host controllers. More redundancy will cause more costs. Therefore, we use three host controllers for the system. A hot backup mechanism is used between host controllers and the priority of the host controllers is predefined in the leaving state.

Analogously, single TTCAN controller or single bus can’t guarantee the communication of the whole system well. Therefore, we use dual redundant mechanism in this part.

III. THE SYSTEM MODEL

A. Premises and assumptions

The purpose of this paper is to reflect the validity and essential performance of this Unmanned Aerial Vehicle flight-control System with UPPAAL. Describing the whole system elaborately will of course lead to a gigantic work of
modeling and state space explosion of verification. Therefore, it is necessary to neglect some unimportant details. We design the system model based on the premises and assumptions as follows:

- At least one TTCAN controller is in good state at any time for single host controller;
- All the messages in the design are assumed to be periodic;
- The concrete format and content of messages are not taken into consideration;
- Error messages are not considered;
- All clocks are assumed to proceed at the same rate;
- Only essential services of the system are modeled.

B. Model designing

We divide the system into application layer and communication layer, as shown in Figure 1 and Figure 2. We define two work modes for host controllers: current mode and backup mode. At a time, only one host controller works in the current mode while others in backup mode. In the current mode, it receives messages and sends or/and handles them; Meanwhile, host controllers in backup mode backup messages as the same as the current one do, and they monitor the heartbeat messages from the current host controller to predict whether the current host controller is active or not. In particular, host controllers in backup mode are never allowed to send messages unless they have changed to the current mode.

The communication layer plays an important role in reliable message transmission among host controllers and rudders. TTCAN controllers would transfer messages between application and physical layer according to TTCAN matrix [1], which is another outstanding characteristic of TTCAN bus.

C. Application layer model

Application layer contains three processors and an application layer transceiver unit, as shown in Figure 2. Send Message Processor generates messages periodically and puts them to Send Cache and each message is assigned with a number num. Receive Message Processor handles non-heartbeat messages. Heartbeat Message Processor, including a counter watchdog and a constant Time_max, monitors the heartbeat message of current host controller. When a heartbeat message reaches, the watchdog will be reset to zero. When watchdog reaches Time_max and there is still no heartbeat message, the Heartbeat Message Processor will send Mastererror[node]! to App Ctrl, which indicates that backup host controllers have known the current host controller broken, and then App Ctrl will switch its mode from backup to current mode. App Ctrl includes two registers named Send Cache and Recv Cache which are used to send and receive messages. App Ctrl will send translate[node]! to Forward Ctrl when a message needs to be sent. Meanwhile, App Ctrl monitors receive[node]?, which indicates that a message has reached. Specifically, to avoid the repeated messages from dual TTCAN controllers to be handled, once the message reaches, its num will be compared with the num of the message received last time. If they are identical, the receiving message will be discarded, otherwise the message will be sent to message processors.

Application layer transceiver App Ctrl can be defined as the following 6-tuple:

\[
\text{App}_\text{Ctrl} = (L_i, l_i, A_i, G(C_i, V_i), I_i, E_i)
\]

\[
L_i = \{S_{0}, \sim S_{1}, \sim S_{2}, \sim S_{3}\}, I_o = \{S_{0}\} \in L_i
\]

\[
A_i = \{\text{Master\_error}[\text{node}], \text{translate}[\text{node}],
\]

\[
\text{Be\_Master}[\text{node}], \text{receive}[\text{node}],
\]

\[
\text{run\_out}[\text{node}], \text{beat}, \text{req\_1}, \text{req\_2}, \text{req\_3}.
\]

\[
G_i(C_i, V_i) = \{\text{watchdog} \in \mathbb{C}, \text{master}, \text{verify\_clock}[j], \text{current\_number}, \text{recv}[\text{node}]\} \in V_i
\]

\[
E_i \subseteq \{I_4 \times A_i \times G(C_i, V_i) \times 2^{L_i} \times l_i\}
\]

The App Ctrl described as time automation model with UPPAAL is shown in Figure 3.

D. Communication layer model

Communication layer contains four units: Forward Ctrl, Transceiver Ctrl, Send Scheduler and Receive Scheduler. As shown in Figure 2, Forward Ctrl is a control unit which applies to exchange messages between communication layer and application layer. Forward Ctrl transfers message from Send Cache to Tx_Cache-1-4 when it is informed by translate[node]!. Spontaneously, it monitors \text{rx\_type}[\text{bus}][\text{node}][\text{type}]? sent by Receive Scheduler and transfers a certain message from Tx_Cache-1-4 to Recv Cache, at the same time sends receive[node]! to App Ctrl. The Forward Ctrl can be described as the following 6-tuple:

\[
\text{Forward}_\text{Ctrl} = (L_i, l_i, A_i, G(C_i, V_i), I_i, E_i)
\]

\[
L_i = \{S_{0}, \sim S_{1}, \sim S_{2}, \sim S_{3}\}, I_o = \{S_{0}\} \in L_i
\]
A₁ = {translate[node], rxtype[bus][node][tp], receive[node]},
G₁(Cᵥ, Vᵥ) = \{send[node]&255 ≤ Vᵥ\},
E₁ ⊆ [L₁ \times A₁ \times G(Cᵥ, Vᵥ) \times 2^{55} L₁].

The Forward Ctrl described as time automata model with UPPAAL shown in Figure 4.

Transceiver Ctrl is a control unit in charge of exchanging messages with Physical layer. Transceiver Ctrl listens in physical line by sendbus[bus].. Once a message arrives, it will be put into Recv Reg and then the message's type will be checked, if the type matching is OK, the message will be transferred to Rx Cache1-4. Meanwhile, it monitors txtype[bus][node][type]? from Send Scheduler. Transceiver Ctrl can be described with a 6-tuple:
Transceiver_Ctrl₁ = {L₁, L₀, A₁, G(Cᵥ, Vᵥ), I₁, E₁}
L₁ = \{S₁₀, S₁₁, S₁₂, S₁₃\}, L₀ = \{S₁₀\} ⊆ L₀,
A₁ = [sendbus[bus], txtype[bus][node][type]],
G₁(Cᵥ, Vᵥ) = ((RecvReg[bus][node]&255) ∈ Vᵥ),
E₁ ⊆ [L₁ \times A₁ \times G(Cᵥ, Vᵥ) \times 2^{55} L₁].

Send Scheduler is a scheduler for sending messages. The information of TTCAN matrix is loaded in Send Matrix when the system starts up. We use TMX[node][bc][i][k] to present the information of the matrix. When the message reaches its sending time, Send Scheduler notifies Transceiver Ctrl to send messages through txtype[bus][node][type].
Send Scheduler can be described with a 6-tuple:
Send_Scheduler₁ = {L₁, L₀, A₁, G(Cᵥ, Vᵥ), I₁, E₁}
L₁ = \{S₁₀, S₁₄\}, L₀ = \{S₁₀\} ⊆ L₀,
A₁ = [BeMaster[node], runout[node], ref[bus], txe[bus][node],
type[bus][node][type]},
G₁(Cᵥ, Vᵥ) = \{x ∈ Cᵥ, ((master||Slave), i) ∈ Vᵥ\},
E₁ ⊆ [L₁ \times A₁ \times G(Cᵥ, Vᵥ) \times 2^{55} L₁].

The time automation of Send Scheduler described with UPPAAL is shown in Figure 5.

Receiver Scheduler is a scheduler for receiving messages, and its operation is analogous to Send Scheduler. Thus we do not describe it elaborately here.

IV. FORMAL VERIFICATION OF THE SYSTEM
We design a simple system with three host controllers, three rudders, twelve TTCAN controllers and two physical lines to verify the validity and performance of the system with UPPAAL.

Seven types of messages will exchange among the six application layer nodes, named REQ1, REQ2, REQ3, ACK1, ACK2, and ACK3. REQi are messages transmitted to rudders by host controllers; ACKi are messages transmitted to host controllers by rudders; BEAT represents the heartbeat message sent to backup host controllers by the current host controller.

Entity in Column III in Table 1 and a simple TTCAN matrix is designed according to Table 1 and it is given in Figure 6, the basic cycle time is designed 50 time units, and each TC is designed 10 time units.

We design 58 timed automata for the system, and they are described in Table 2. The properties of the system are described with TCTL (Time Computation Tree Logic) [11] in UPPAAL, and we divided the properties into four groups as follows.
A. The verification of reliability

The reliability of the system lies in two aspects: 1) whether backup host controller can communicate with the current host controller steadily or not; and 2) whether the current host controller communicates with rudders well or not. Additionally, when a failure happens, we need to verify whether the communication can still be in good state or not.

We have verified eight reliability properties as follows:

1. [E<>Rudder1.S6]
2. [E<>Rudder2.S4]
3. [E<>Rudder3.S2] //rudders receives the msg named REQ1, REQ2, and REQ3 rightly;
4. [E<>Master1.S12]
5. [E<>Master1.S13]
6. [E<>Master1.S14] //current host controller1 can receive the msg named ACK1, ACK2, and ACK3 well;
7. [E<>(Master2.S11 imply Master1.master==true)]
8. [E<>(Master3.S11 imply (Master1.master == true || Master2.master==true))] //backup host controllers can receive the heart message.

Experimental data of 8 properties are satisfied, Prop1-6 prove that the communication between rudders and the current host controller is in good state; prop7-8 prove that when current host controller works in good state, backup host controllers could receive the heart messages successfully.

B. The verification of security

The security of the system means that some bad behaviors will never happen, for example, system would never get into dead lock state and each rudder would never be allowed to receive the messages not belonging to it.

Security is verified with properties 9-15 as follows,

9. [A] not deadlock //the system will never be in dead lock;
10. [A] not Rudder 1.S4
11. [A] not Rudder 1.S2 //Rudder1 can never receive the msg named REQ2, REQ3;
12. [A] not Rudder2.S2
13. [A] not Rudder2.S6 //Rudder2 can never receive the msg named REQ1, REQ3;
Experimental data of 10-15 properties are satisfied, which shows that rudders can never receive messages not belonging to them. Unfortunately, Property 9 was not verified successfully, the verification of pro.9 leads to system crashed, for the model of the system is so huge that the property cannot be verified easily.

C. The verification of faulty tolerance

According to the requirements, the system can still work well when current host controller is broken or single TTCAN fails. We verified the fault tolerance with Properties 16-19 as follows:

- [16] E<> ErrorState.good1
- [17] E<> ErrorState.good2
- [18] E<> (Master2.master==true imply Master1.broken )
- [19] E<> (Master3.master==true imply Master2.broken)

//when current host controller is broken; the backups can take over its work in a certain time;

Experimental data of 16-19 properties are satisfied which shows that the system has good capability of faulty tolerance.

D. The verification of schedulability

According to the requirements of the system, the sending/receiving time of the messages is constrained; In addition, when a failure happens, the time spent on switching backup host controller to current host controller can’t be too long. Schedulability is verified with properties 20-26 as follows,

- [20] E<> (ErrorState.good1 imply ErrorState.x <=100) //switching time is not beyond double BC;
- [21] E<> Master1.S18 //ACK1 sent within one BC;
- [22] E<> Master1.S17 //ACK2, 3 would be sent within double BC;
- [23] E<> Rudder1.S8 //REQ1 would be sent within one BC;
- [24] E<> Rudder2.S9 //REQ2,3 would be sent within double BC;
- [26] E<> Rudder3.S9

//REQ2,3 would be sent within double BC

Experimental data of 20-26 properties are satisfied which shows that each message would be sent according to the schedule and the time spending would not beyond the setting time window. When a failure happens, the time for switching would not exceed double BC, which means the system can recover quickly.

TABLE I. THE LIST OF DATA

<table>
<thead>
<tr>
<th>Type</th>
<th>Resource</th>
<th>Period (time units)</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>Master_1/2/3</td>
<td>55</td>
</tr>
<tr>
<td>REQ2</td>
<td>Master_1/2/3</td>
<td>55</td>
</tr>
<tr>
<td>REQ3</td>
<td>Master_1/2/3</td>
<td>110</td>
</tr>
<tr>
<td>ACK1</td>
<td>Rudder_1</td>
<td>55</td>
</tr>
<tr>
<td>ACK2</td>
<td>Rudder_2</td>
<td>55</td>
</tr>
<tr>
<td>ACK3</td>
<td>Rudder_3</td>
<td>110</td>
</tr>
<tr>
<td>BEAT</td>
<td>Current host controller</td>
<td>110</td>
</tr>
</tbody>
</table>

TABLE II. THE LIST OF AUTOMATA

<table>
<thead>
<tr>
<th>The name of the automation</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master[k],k∈[1,3]</td>
<td>The application layer of the host controller k</td>
</tr>
<tr>
<td>Rudder[p],p∈[1,3]</td>
<td>The application layer of the rudders p</td>
</tr>
<tr>
<td>Bus[i] Node[j] Sch TX,i∈[1,2],j∈[0,5]</td>
<td>sending scheduler of TTCAN controller j connected the line i</td>
</tr>
<tr>
<td>Bus[i] Node[j] Sch RX ,i∈[1,2],j∈[0,5]</td>
<td>receiving scheduler of TTCAN controller j connected the line i</td>
</tr>
<tr>
<td>Bus[i] Node[j] ForwardCtrl, i∈[1,2],j∈[0,5]</td>
<td>Forward Ctrl units of TTCAN controller j connected the line i</td>
</tr>
<tr>
<td>Bus[i] Node[j] Transceiver, i∈[1,2],j∈[0,5]</td>
<td>transceiver of TTCAN controller j connected the line i</td>
</tr>
<tr>
<td>Appclock</td>
<td>time sequence of message in application layer</td>
</tr>
<tr>
<td>TTCANclock</td>
<td>clock in communication layer, controlling the time of BC</td>
</tr>
<tr>
<td>ErrorState</td>
<td>Injecting faulty</td>
</tr>
<tr>
<td>RunState</td>
<td>Reflecting the current state of the system</td>
</tr>
</tbody>
</table>
V. CONCLUSIONS

The flight control system of unmanned aircraft vehicle is a highly dependable embedded software system. Many non-functional properties are needed to be verified to meet the requirements of the system during the design phase, instead of testing them after coding. It is a key project to model TTCAN communication protocols in application layer and the models need to be verified to meet the requirements of flight-control system. Triple Faulty tolerance model based on TTCAN is designed to ensure insurance reliable communicating in the flight control system of UAV. We adopt UPPAAL to analyze the system model and to verify the key properties of the system. The system model, which includes 58 timed automata, is a real abstract and accurately expresses the system’s interactive relationship between modules. The experimental result indicates that the system is dependable and feasible.

ACKNOWLEDGMENT

This paper is granted by NSFC program “Designing and Verifying on High Reliable Component-based Embedded System Development Environment (No. 60736017)” and national 863 plan program “High Confidence Embedded software product lines (No. 2007AA010304)” in China.

REFERENCES


