International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:6, No:11, 2012

# Studying on ARINC653 Partition Run-time Scheduling and Simulation

Dongliang Wang, Jun Han, Dianfu Ma, and Xianqi Zhao

***Abstract*—**Avionics software is safe-critical embedded software and its architecture is evolving from traditional federated architectures to Integrated Modular Avionics (IMA) to improve resource usability. ARINC 653 (Avionics Application Standard Software Interface) is a software specification for space and time partitioning in Safety-critical avionics Real-time operating systems. Arinc653 uses two-level scheduling strategies, but current modeling tools only apply to simple problems of Arinc653 two-level scheduling, which only contain time property. In avionics industry, we are always manually allocating tasks and calculating the timing table of a real-time system to ensure it's running as we design. In this paper we represent an automatically generating strategy which applies to the two scheduling problems with dependent constraints in Arinc653 partition run-time environment. It provides the functionality of automatic generation from the task and partition models to scheduling policy through allocating the tasks to the partitions while following the constraints, and then we design a simulating mechanism to check whether our policy is schedulable or not.

***Keywords*—**Arinc653, scheduling, task allocation, simulation.

## I. Introduction

NOWADAYS, the development of Avionics software system are facing huge challenges, including increasing verification cost, rising safety requirement and shortening time to market demands, caused by sharply increasing software scale and complexity [1]. Therefore, highly reliable design methodologies are strongly required for the development and verification of avionics software. Currently, model-driven correct-by-construct methodology has become an important development method in designing safe-critical embedded system [2].

AADL (Architecture Analysis & Design Language) released by the SAE in November 2004, is dedicated to embedded real time systems in safe-critical domains [3]. It provides an industry-standard, textual and graphic notation with precise semantics to allows early function or quality properties verification at the model level and automatically code generation respecting the requirement specification from the models, which greatly speeding the development of the system and assuring high reliability of such system.

And with the rapid development of aviation technology, computer technology and microelectronics technology, modern avionics architecture has gradually evolved from the traditional federated architecture to the integrated modular avionics system structure (IMA)[4].

IMA integrated all the subsystems on a common computer platform, so as to greatly reuse various hardware resources. But this potentially reduces the error control of functions. In order to introduce the error isolation as which in the federated architecture into IMA, the concept of partition is introduced in [5]. This can be achieved through a partitioning of resources with respect to available time and memory capacities, also called special partitioning and temporal partitioning respectively. We can avoid the errors spreading from a function to another by means of the partition mechanism, thus making the function software running on the same IMA is like running on a single and safe computer system.

Several standards for avionics software have been proposed for IMA, like ARINC653 standard. Considering avionics application engineers from different application domain mainly focus on the respective avionics application function logic design and usually do not care about the execution architecture the applications will be in. Therefore, we can extract some properties and focus on these, by which we can model this issue and analyze it on logic level.

According to the above requirements for the avionics software, we have proposed an AADL-based model-driven mechanism to facilitate the development of the avionics software constructed on ARINC653 OS more easily, rapidly and reliably. This paper mainly focuses on the scheduling parts in which we can automatically generate the scheduling policy through analyze the models of tasks and partitions and then allocate the tasks to the partitions while follow the constraints. In this paper we represent an automatically generating strategy and then we give a simulating mechanism to check whether our policy is schedulable or not.

The rest of this paper is organized as follows. Section II gives a brief introduction to Arinc653 and two-level scheduling. Section III discusses the modeling method for avionics tasks and the automatically generating strategy with a case study. In Section IV, an simulating mechanism is discussed. Finally, Section V ends this paper with a conclusion and future work.

Dongliang Wang is with the Institude of Advanced Computer Technology, School of Computer Science and Engineering, Beihang University, Beijing China (e-mail:wangdl@act.buaa.edu.cn).

Jun Han, is with the Institude of Advanced Computer Technology, School of Computer Science and Engineering, Beihang University, Beijing China (e-mail: Jun_han@buaa.edu.cn).

Dianfu Ma is with the National Lab of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing, China (e-mail: dfma@buaa.edu.cn).

Xianqi Zhao is with the Institude of Advanced Computer Technology, School of Computer Science and Engineering, Beihang University, Beijing China (e-mail:zhaoxq @act.buaa.edu.cn).

## II. Overview of ARINC653

### A. ARINC653

ARINC653 standard is aimed to define a general-purpose interface for avionics application software in the partition execution environment and widely employed by manufacturers in the avionics industry [6]. It supports time and space partitioning in accordance with the IMA philosophy. Fig. 1 is an overview of the ARINC653 execution architecture which contains application software layer, core software layer and APEX Interface.

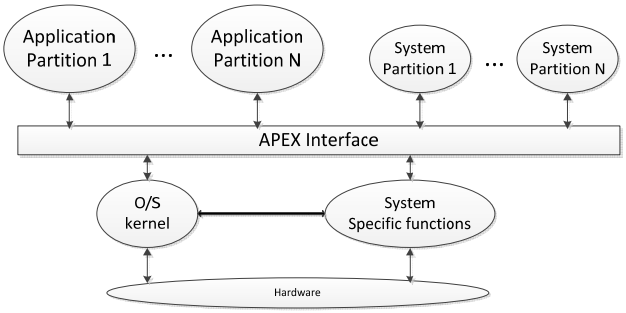


Fig. 1 ARINC653 Execution Architecture

In the application software layer, application partition is the software part of avionics software applications supported by the core module. The ARINC653 application partition should have reliable space and time partitions, which limit calling system services through ARINC653 APEX service interfaces. The system partition is optional specification of the core software layer, which needs to call interface beyond APEX services. It should also have reliable space and time partitions. These partitions may handle some function such as communication management of hardware and error management mode. The operating system kernel in core software layer provides the API and behaviors defined in the specification, and it's a common and standard environment supporting the implementation of the application software. The APEX interface is intended to provide a general-purpose interface between application software and operating systems in the IMA.

### B. Partition

The partition is a core concept in ARINC653 specification, which is a functional partitioning in avionics applications. The partition is a core concept in ARINC653 specification, which is a functional partitioning in avionics applications. Potential architecture of partition is similar to the multi-tasking applications of a common computer [5]. Each partition consists of one or more Concurrent processes, sharing processor resources. These processes are uniquely identified, and they contain properties which affect scheduling, synchronization, and the entire execution.

According to the above, Arinc653 uses the two-level scheduling, which is the partition-level scheduling and the process-level scheduling. The main features of partition-level scheduling are:

- Use partition as scheduling unit
- Partition has no priority
- Scheduling algorithm is pre-defined, and repeated periodically (configured by integrator only).

In each cycle, the system must allocate at least one time window to each partition. And the process-level scheduling uses priority-based preemptive scheduling.

## III. Allocation Policy

In the avionics industry, we are always calculating the timing table of a real-time system to ensure it's running as we design. This can cause a significant time consuming. And when we are allocating tasks to partitions manually, we cannot guarantee the correctness either, which may cause excessive consumption of manual and computing resources. In this section, we represent an automatically generating strategy which applies to the two scheduling problems with dependent constraints in Arinc653 partition run-time environment. It provides the functionality of automatic generation from the task and partition models to scheduling policy through allocating the tasks to the partitions while following the constraints.

### A. Dependency Constraint

In the airborne avionics systems, the main objective is to use the control signals from the joystick, pedals to control the rudder and the engine, so as to control the aircraft's attitude changes. Onboard computer system can also calculate the corresponding control commands according to flight plan formulated in advance and the current state parameters of aircraft, so as to realize the autopilot. In this scenario, we are aiming to allocating some tasks to subsystems (here we use partition).

The interaction between avionics system and environment is the existence of an objective, independent of the cognitive behavior of designers. Interaction between avionics system and environment can be as follows: interaction with atmospheric data, interaction with navigation equipment, interaction with aircraft attitude control, interaction with the engine control, interaction with data communication and interaction with database and so on.

We can represent the interaction between the system and the environment as a set: $\{X, S, Y\}$

Wherein the set of system input is the collection of data of all the input:

$$X = \{AirData, IRS, RNS, PitchControl,$$

$$FlightMode, DataLinkin, DataBasein\}$$

The set of system output is all output data collection:

$$Y = \{Elevator, Ailerons, Rudder, Engine,$$

$$DataLinkout, DataBaseout\}$$

There is a sequence relationship between the input and the output, i.e. when one or several of the input data arrives or changes, some output data may be consequently generated or changed. Therefore, we use a duality to describe such a sequence relationship.

In airborne systems, each output data must be calculated from the input data and the intermediate data obtained by a series input data. So there is a series of set

$$\{x_i, x_{i+1}, \ldots, x_m, r\}\{x_{m+1}, \ldots, x_j, r, s\}\{s, t\}\{t, y_i\},$$

which is able to describe how the system calculated output data by the input data and intermediate data.

We may take a simplified case system S as an example, the tasks and their sequence relationship are as follows:

- Task AD: $\{X_{AD}, S_{AD}, Y_{AD}\}$

  $X_{AD} = \{AirSensor\}$

  $Y_{AD} = \{AirData\}$

  $R_{AD} = (AirSensor, AirData)$

- Task NP: $\{X_{NP}, S_{NP}, Y_{NP}\}$

  $X_{NP} = \{GPS, AirData\}$

  $Y_{NP} = \{Attitude\}$

  $R_{NP} = (IRS, AirData, Attitude)$

- Task NA: $\{X_{NA}, S_{NA}, Y_{NA}\}$

  $X_{NA} = \{GPS\}$

  $Y_{NA} = \{Position\}$

  $R_{NA} = (IRS, Position)$

- Task FM: $\{X_{FM}, S_{FM}, Y_{FM}\}$

  $X_{FM} = \{AirData, Attitude\}$

  $Y_{FM} = \{PitchData\}$

  $R_{FM} = (AirData, Attitude, PitchData)$

- Task FCS: $\{X_{FC}, S_{FC}, Y_{FC}\}$

  $X_{FC} = \{PitchControl, PitchData\}$

  $Y_{FC} = \{Elevator\}$

  $R_{FC} = (PitchControl, PitchData)$

Here are four subsystems and their input data, output data and their sequence relationship. Obviously, there is a sequence relationship among them, as we can see in Fig. 2:

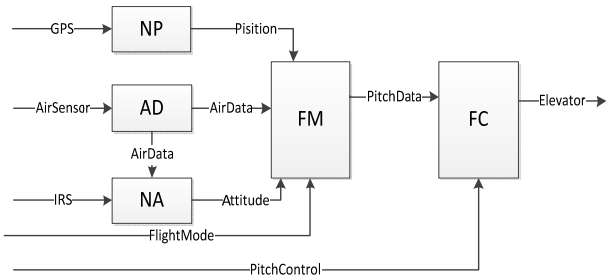


Fig. 2 The structure

In system S, we can easily get an execution sequence, which is AD, NP, NA, FM, FC. But if there is another system T same as system S, and the elevator is controlled by both of them. That is the elevator is put out only when both systems are finished. Then we can make a execution sequence same as System S.

According to the above, dependency constraint exist among tasks in avionics systems. Through observation, we found that there are some tasks that have similar function, and some tasks have the same or partly same input. And there may also be some tasks the preorder task of which is the same. All these dependency constraints limit our allocating of tasks, but at the same time, they will help us remove some allocation schemes that do not satisfy these constraints. So this can help reduce the size of the problem.

Observing the structure and data path figure above, we can get that AD must execute before NP, NA and NS share the same input data. In the general case, there will also be these kinds of result. So we can group the tasks according to the dependency constraint. We define the grouping rule as follows:

- Tasks with the same pre-order task will be in the same group.
- Tasks needing the same data (like IRS in the case) will be in the same group.
- Tasks with sequence relationships must be the different groups (like ADC and NS)

Through these rules, we can get a series of groups:

$$\{\{AD\}, \{NP, NA\}, \{FM\}, \{FC\}\}$$

*B. Allocating Policy*

In ARINC653 two-level scheduling, the operating system periodically schedules the partition assignment processor according to the main time frame with a fixed length of time. In this section, we will give an allocation strategy which automatically generates one or a group of partition scheduling window for a given set of task as well as partition model. (See Fig. 3):

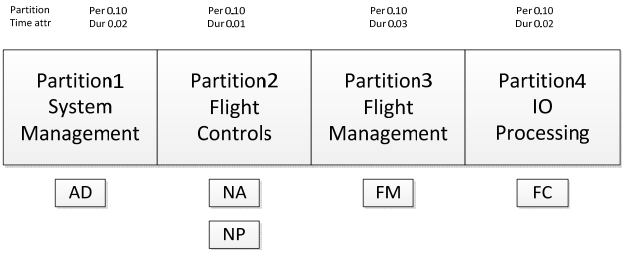


Fig. 3 Allocation of tasks to partition

In this paper, we focus on the problem with dependency constraint and timing constraint. We can define the task and partition model as follows:

Definition 1:

Assume that the system have partition $P = \{P_1, P_2, \ldots, P_N \mid N \geq 1\}$, each partition have parameters tuples (TL, Dur), where Dur is

the partition duration, and TL is the tasks set allocated to the partition, initially empty.

Definition 2:

Assume that the system has a task set $T = \{T_1, T_2, ..., T_M \mid M \geq 1\}$, each task with the time parameters Triples <c, p, d>, where c is the task execution time, p is the period of the task, d is the deadline of the task. Each task has a dependency constraint $\{i \mid 1 \leq i \leq M\}$, where i represent the pre-order task, denoted by Con. Thus the parameters of task $T_i$ can be denoted as $< c_i, p_i, d_i, Con_i >$.

Definition 3:

The main time frame $W = \{P, ..., P_j \mid 1 \leq i \leq N, 1 \leq j \leq N\}$, where each partition has at least one time window in the frame, and may have more than one.

Given these models, there are always feasible means to find a rational allocation scheme, which satisfy the timing and constraints. The easiest way is the brute force method. But this can waste a lot of time and resources. Taking the dependency and timing constraint into consideration, according to the above talking about the dependency constraint, all these dependency constraints limit our allocating of tasks, but at the same time, they will help us remove some allocation schemes that do not satisfy these constraints. The grouping method is as bellow:

TABLE I
THE TASK GROUPING METHOD

```
TaskGrouping(T)
1   Input: T
2   Output: TG
3   Begin
4     While( TaskRemains>0)
5     Begin
6       T=SelectRemain();//select a task from the remain set, remove it
                          and take it to a new group G
7       for  Ti in Remains
8       Begin
9         if CheckInput(T, Ti) then
10          InGroup(G, Ti) //combine Ti to group G
11        Endif
12      End
13    End
14    While (GroupRemina >0)
15    Begin
16      for Gi in GroupSet
17      Begin
18        if ZeroIndegree(Gi) then //if  the indegree of tasks in Gi is
                                    zero
19        Combine(); //combine it to current group
20        IndegreeDecrease(); // the indegree of the remain group
                              decrease by 1
21        Endif
20      End
21    End
22  End
```

In the above grouping method, we focus on the indegree and input of tasks. We can define the indegree of a task as below:

Definition 4:

- If there is no pre-order in front of the task, then the indegree of the task is 0;
- If there is a pre-order in front of the task, then the indegree of the task is the indegree of pre-order increase by 1;
- If there are more than one indegree in a task, we choose the larger one as its indegree.

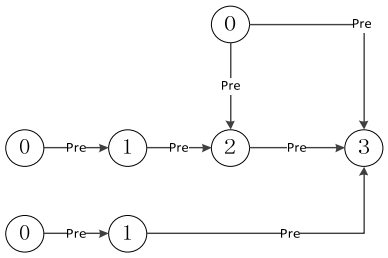


Fig. 4 Calculation of Indegree

We can see in Fig. 4 that there are 4 groups distinguished by the indegree.

TABLE II
THE TASK ALLOCATING METHOD

```
TaskAllocating ( G_i , P_j )
1     Input:  G_i , P_j
2     Output: W
3     Begin
4      if ( i=0 ) then
5        for k=1 to NG do
6        begin
7          PartitionWindow( G_{i+1} , P_k )
8        end
9     endif
10    if ( validateTTP( G_i , P_k ) < 0) then
11      if ( i=M) then
12        Notes();
13        return
14      else then
15        for k=1 to M do
16        begin
17          TaskToPart( G_{i+1} , P_k )
18        end
19      endif
20    else then
21      return
22    endif
23    end
```

After the grouping, we get a set of sequenced groups the element of which is a set of tasks. And then we will allocate these groups to the partitions. In order to simplify the allocation procedure, we must combine some groups to make the number

of groups be the same as that of the partitions. So during the allocation, we just need to allocate one single group to one partition. As for the case in which some partitions are empty, it is obviously wasting CPU resources. So we will skip these cases. The allocating method is as below:

After calling the three methods, we get a allocated main time frame in which tasks are allocated to the partitions and partitions are arranged properly.

TABLE III
THE GROUP COMBINE METHOD

| GroupCombine (G,P) | |
|---|---|
| 1 | Input: G, P |
| 2 | Output: Gnew |
| 3 | if NG<NP then |
| 4 | SplitGroups(G,NP) |
| 5 | endif |
| 6 | else if NG>NP |
| 7 | CombineGP(G,NP)//combine the ones that have partly same input, otherwise combine the ones that have smaller capacity endif |
| 8 | else |
| 9 | TaskAllocating(G,P) |
| 10 | endif |

## IV. SIMULATING MECHANISM

In this paper, we will verify whether current scheduling window is schedulable by means of Simulation execution for a period of time, the length of which is decided by users. We calculate the priorities of tasks in every time unit, and then choose the task with the highest priority to execute. Therefor we get a processor allocation table and scheduling events table (processor preemption, partition switching event). After this, we can do some computational analysis. And the simulating tool we give here is web-based, and is aimed to deal with Arinc653 two-level scheduling issues.

### *A. Simulation Process*

The simulation process is divided into three stages [8], the priority stage, queuing stage, election stage, as shown below in Fig. 5:

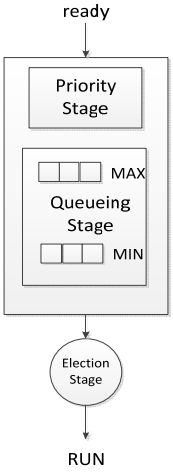


Fig. 5 Simulation process

Because in the ARINC653 two-level scheduling, task scheduling uses priority based preemptive strategy, then task scheduling simulation is for priority calculation. We can determine whether the task scheduling is a dynamic priority or static priority scheduling through the scheduling algorithm inside the partitions.

In queuing stage and election stage, Tasks queue according to priority in partition, task with high priority takes up high position in the queue. The system will select the task with the highest priority in queue to execute. After this, it will record the current events, that is, whether preemption or partitions switch occurs.

### *B. Event Analysis*

In this paper, we have two ways to perform schedulability analysis that is simulation and feasibility test. In the previous section, we calculate and record two tables, the processor table and the event table, which contains basic information about scheduling. We can do some data analysis based on the event table, calculated for example, the worst, best, average response time, the worst, best, average blocking time , turnaround time, and whether there is a task exceeds its deadline and so on.

## V. CONCLUSION

In this paper we provide a utility for Arinc653 two-level study. We can automatically generate the scheduling policy through analyze the models of tasks and partitions and then allocate the tasks to the partitions while follow the constraints. And then we give a simulating mechanism to check whether our policy is schedulable or not. All of these implementations may form a useful tool for research.

Future work will focus on other constraint such as communication, and the performance of the allocating policy.

## REFERENCES

[1] Jean-Louis Camus, "The Airbome Software Development Challenge," White Paper, ESTEREL TECHNOLOGIES, March 2010.

[2] Sandeep K. Shukla, "Model-Driven Engineering and Safety-Critical Embedded Software," *Computer*, vol. 42, no. 9, pp. 93-95, Sept. 2009, doi:10.1109/MC.2009.294

[3] SAE Aerospace. SAE AS5506: Architecture Analysis and Design Language (AADL), Version 2.0, 2009.

[4] C.B.Watkins and R.Walter, "Transitioning from federated avionics architectures to Integrated Modular Avionics," In Proceedings of the IEEE/AIAA 26th Digital Avionics Systems Conference (DASC '07), October 2007.

[5] John Rushby,Partitioning in Avionics Architectures:Requirements, Mechanisms, and Assurance, Langley Research Center Prepared for Langley Research Center Hampton, Virginia 23681-2199

[6] Airlines electronic engineering committee (AEEC), avionics application software standard interface - ARINC specification 653 - part 1 (REQUIRED SERVICES), December 2005, ARINC, Inc.

[7] ARINC CHARACTERISTIC 702A-3, December 15, 2006, ARINC, Inc.

[8] F Singhoff, J Legrand, L Nana, L Marce Cheddar : a Flexible Real Time Scheduling Framework in ACM SIGADA 2004 International conference Proceedings (2004)