Architecture-Level Schedulability Analysis with IO Constraint using AADL

Yunwei Dong  
Northwestern Polytechnical University  
Xi’an, China  
yunweidong@nwpu.edu.cn

Qianwen Gou  
Northwestern Polytechnical University  
Xi’an, China  
gqwen@mail.nwpu.edu.cn

Pei Tang  
Northwestern Polytechnical University  
Xi’an, China  
tangpei@mail.nwpu.edu.cn

Abstract—The rapid development of network communication technology and the integration of several independent systems together led to the emergence of large-scale complex software systems, called system of systems (SoS). With the increasing complexity of the interaction between system and resource allocation, it is necessary to analyze the schedulability of resource-constrained systems from architecture level. System resources would affect the schedulability of the system, and if system resources are not configured correctly, the task set may not be scheduled due to competing resources. This paper proposes an architecture-level schedulability analysis method based on IO constrained AADL model. At first, we extended and created the IO Resource Model Annex (IOMA) to describe the use of IO resources in the system to make up for the lack of AADL description capability. Then a schedulability analysis method of the AADL architecture model under IO resource constraints is proposed. The core idea of this method is to transform the AADL architecture model containing IO resource model into a timed automaton. The constraints of IO resources and scheduling-related attributes in the architecture model are expressed as the transition conditions and time constraints of the timed automaton. A case study on a civil airborne system has been employed to demonstrate the feasibility of our proposed approach.

Keywords: AADL, Schedulability analysis, IO constraint, IO resource model annex, Embedded system architecture

I. INTRODUCTION

System of Systems (SoS) is a task-oriented collection of systems that shares the resources and performance of each subsystem in order to achieve a new, more complex, more efficient meta-system that will exceed the sum of its individual systems, have been widely used in avionics, automotive control and other safety-critical areas [1]. That is to say, software architecture is the cornerstone of the success of any software system [2]. Considering the complexity of the system architecture, its development needs to pay special attention to its system architecture. For example, as the scale and complexity of avionics systems have increased, avionics systems have evolved from the traditional federated avionics system to the form of Integrated Modular Avionics (IMA) [3]. During the process of designing IMA system, the architecture model should be analyzed and verified during the architecture modeling phase to ensure that the confidence attributes of the system are met [4]. When multiple processes or threads competing for IO resources, failing to consider IO resource constraints in the schedulability analysis is equivalent to ignoring the dependencies between tasks, which may result in systems failures or even a serious accident. Contention for IO resources by different threads may lead to thread blocking, which can change the original execution order of threads and possibly make the system unschedulable, with serious consequences. When the task in the system accesses the same resource, it may cause mutual restriction among tasks, which inevitably have an impact on the system schedulability [5]. Therefore, the schedulability analysis of the architecture model is a key index to measure the performance of the resource-constrained system, especially in the aerospace, military, and other critical technology areas.

The Architecture Analysis and Design Language (AADL) is a modeling language that supports the analysis of non-functional attributes of embedded real-time systems in the early stages of development. To analyze system schedulability, memory and processor components are used to model the storage and computing resources of the system, and processes and threads are used to model tasks in the system. The AADL property set provides properties that describe the scheduling information of threads, and schedulability analysis based on the AADL architecture model is mainly for the analysis of thread components in the model. In above schedulability analysis method, it is assumed that each thread is independent of each other. However, in actual systems, threads are mutually constrained by accessing resources, and the thread may be limited by resource access, which affects the scheduling of threads, and thus affects the schedulability of the system. Therefore, it is necessary to analyze schedulability of resource-constrained system.

Although the AADL core language standard describes the relationships between thread components in connection relationships, it doesn’t provide a complete modeling approach for describing IO resource information in the system. To analyze the schedulability of AADL architecture model under IO resource constraints, the paper first defines the IO resources in AADL, and then expands the AADL IO resource model annex (IOMA) to make up for the lack of AADL modeling ability. By annotating an architecture model with the IO resource model, a schedulability analysis model with IO resource constraints can

This work was supported by the National Science Foundation of China (Grant No. 61772423)
be built. Finally, the schedulability verification approach based on IO constrained AADL model is proposed to verify the time constraints of the system. The method contains the model transformation from an AADL model to a timed automaton for calculating the time constraints of each thread. In summary, this paper makes the following contributions:

- An IO resource model annex based on AADL is proposed to enhance the AADL description ability to model IO resource information in system architecture.
- A schedulability analysis method is also proposed based on the AADL IO resource model, which can consider the competition of threads for IO resources. It gives an IO resource constraint scheduling algorithm based on timed automata to obtain the schedulability of the IO resource-constrained AADL architecture model.

The paper is organized as follows: Section 2 presents the IOMA including IO resource model library (IOMLib) and IO resource model subclause (IOMSub). The details of the schedulability analysis method are shown in Section 3. Section 4 illustrates the feasibility of our method through a case study. In section 5, we introduce some related work. Finally, section 6 gives our conclusions and discusses future work.

II. IO RESOURCE MODEL ANNEXE

A. Semantics of IO resource

IO generally refers to an interface for data exchange between computing components and other outside components, such as printer and monitor, etc. In AADL, IO interface can be modeled with the features of the component or component, such as data components, subprogram components, access and port in component features. Data components and subprogram components in AADL can be used as interfaces for data exchange among processes or threads. Ports and accesses as component features can provide data exchange capability for connected components. This paper defines IO in AADL model from the perspective of resource competition.

Definition 1 (IO resource): An IO resource in AADL is a model element that can be excluded or shared by multiple components. Such resources can be expressed in the form of data components, subprogram components, accesses and ports in component features.

![Fig. 1. IO resource types](image)
that a process or thread in an IO object collection must access exclusively when accessing an IO subject, and the priority of that thread change to the highest priority among the threads blocked by it during the access.

In the AADL architecture model, the IO resource model is defined in the implementation of a component declaration. If the IO subject is a component, the IO resource model is defined on its parent component. If the IO subject is some features of component (such as port, a data access or a subprogram access which is defined in component type declaration), the IO resource model is defined on the component itself. The IO resource model of AADL components can be described as follows.

**Definition 5 (IO resource model):** The IO resource model of system is defined as a four-tuple \( M_{io} = (S, O, L, P) \).

1) \( S \) is a set of all IO subjects in the system, and \( S = \{s_1, s_2, \ldots, s_n\} \).
2) \( O \) is a set of all IO objects in the system, and \( O = \{o_1, o_2, \ldots, o_m\} \).
3) \( L \) is a set of all IO access rules, and \( L = \{\text{share, exclusive, PCP, PIP}\} \).
4) \( P \) represents the IO protocol in the system, \( P : S \times O \rightarrow L \). An IO protocol \( p_i \) defined as follows: \( \forall (s_i, O_i) \), where \( s_i \in S, O_i \in O, (i = 1, 2, \ldots, n) \), \( \exists l_i \in L, (i = 1, 2, 3, 4) \), \( p_i: (s_i, O_i) \rightarrow l_i \), Which means that the IO objects \( O_i \) accesses the IO subject \( s_i \) follows the real-time lock protocol defined by \( l_i \).

### Syntax of IO Resource model annex

AADL provides an annex extension mechanism to support researchers in architectural modeling and non-functional attribute analysis. The AADL IO resource model annex (IOMA) is designed to support IO resource modeling of embedded system, and can be described with the extended Bacchus paradigm. Firstly, analyze the situation that multiple components access an IO resource, and analyze the type of IO resources in the system and the IO rules that need to be used. Then define the IO subject, IO object in the implementation of the component. Finally, the IO rules are referenced, and analyze the type of IO resources in the system and the IO rules that need to be used.

**B. Syntax of IO Resource model annex**

The IO resource model library is defined via IOResourceType and IOResourceRule. The IOResourceType indicates the type to which the IO subject belongs. Elements of architecture that can be accessed by multiple components in AADL are the transport ports and access ports in the component features, the data components and the subprogram components. Therefore, the IOResourceType is defined as three types: dataResource, portResource, and subprogramResource, which are used to describe the types of IO resources for modeling IO resources of the system. The IOResourceRule are defined in the

#### 1) IO resource model library:

The IO resource model library contains a set of reusable declarations, such as IO resource types and IO rules, that enable them to be referenced in the IOMSub. The grammar rules for the syntax of IOMLib are shown in Fig. 4.

**Fig. 3. IO resource model annex**

**Fig. 4. IO resource declaration library**

The IO resource model library is defined via IOResourceType and IOResourceRule. The IOResourceType indicates the type to which the IO subject belongs. Elements of architecture that can be accessed by multiple components in AADL are the transport ports and access ports in the component features, the data components and the subprogram components. Therefore, the IOResourceType is defined as three types: dataResource, portResource, and subprogramResource, which are used to describe the types of IO resources for modeling IO resources of the system. The IOResourceRule are defined in the

---

An example is shown in Fig. 2 to illustrate the AADL-based IO resource model, the system contains two processes, Process1 and Process2. Process1 contains two threads, Thread1-1 and Thread1-2, which both request access to the data component Data1. Process2 contains three threads, Thread2-2 and Thread2-3 access the event data port of Thread2-1 at the same time. According to the above definition, the IO resource model of the system can be expressed as \( M_{io} = (S, O, L, P) \), where,

1) \( S = \{\text{Data1, Thread2-1.outport}\} \).
2) \( O = \{\text{Thread1-1, Thread1-2, Thread2-2, Thread2-3}\} \).
3) \( L = \{\text{share, exclusive, PCP, PIP}\} \).
4) \( P = \{p_1, p_2\} \).
5) \( p_1: (\text{Data1, \{Thread1-1, Thread1-2\}}) \rightarrow \text{share} \).
6) \( p_2: (\text{Thread2-1.outport, \{Thread2-2, Thread2-3\}}) \rightarrow \text{PCP} \).
IOMLib to indicate the rules that the IO object should follow to access the IO subject. By analyzing the real-time lock protocol when the resource is accessed, the IOResourceRule is defined as four types: SHARE, EXCLUSIVE, PCP, and PIP, which are used to describe the access rules needed to model the IO resources of the system.

2) IO resource model subclause: The IOMSub describes IO resource usage by referencing the type and rule in the IOMLib. It allows component implementations to be annotated with the IOMSubs. The grammar rules for the syntax of IOMSub are shown in Fig. 5.

![Fig. 5. IO resource model subclause](image)

If multiple subcomponents compete for access to the same IO resource, the component can be annotated with the IOMSub. The IOMSub is added to the implementation declaration of the component to describe the IO resource usage. The IO resource type and IO rule defined in the IOMLib need to be referenced. First, all the IO subjects and their types in the component are defined in the iom subjects section, and the IO object collections corresponding to each IO subject are defined in the iom objects section of the IOMSub declared by the component implementation. Then refer to the IO rules in the iom protocol section, define IO rules for each pair of IO subjects and IO objects, and represent the IO protocol in the form of triples. Finally, all the IO protocols are obtained to form the IO resource model on the component.

### III. SCHEDULABILITY ANALYSIS WITH IO CONSTRAINT

The competing access to IO resources by different threads may cause thread blocking, which will change the original execution sequence of the threads, thereby affecting system schedulability. In Fig. 6, it describes how the schedulability analysis will be conducted with an explanation of five steps.

#### A. Building architecture model

The architecture model is the basis for schedulability analysis. In terms of software requirement specification, software architects can design the architecture model of the embedded real-time system including software components, execution platform components, composite components and their interactions using AADL language. The storage and computing resources of system can be modeled via hardware components, such as memory and processor, and tasks in the system are described via software components processes and threads.

![Fig. 6. The process of schedulability analysis method](image)

Fig. 6. The process of schedulability analysis method

The AADL property set provides properties for describing the scheduling information of the system. It needs to add properties such as deadline, period, execution time, etc. to the thread, and binds processes and threads to the processor, and specifies the scheduling policy on the processor.

#### B. Building IO resource model

In the schedulability analysis of AADL architecture model under IO resource constraints, it is very important to analyze the competition relationship between processes and threads accessing IO resources in the system. To analyze the case where multiple processes or threads in a system access an IO resource, define the IO subject and IO object collection and determine the IO rules based on the requirements by using the AADL IO resource model annex. The scheduling analysis model with IO resource constraints can be constructed by annotating the architecture model with the IOMA. The IO resource model template is shown in Fig. 7.

![Fig. 7. The IO resource model template](image)

Fig. 7. The IO resource model template

In Fig. 7, Firstly, the IO resource type and IO rule need to be referenced, and then, all the IO subjects and their types in the component are defined in the iom subjects section, and the IO object collections corresponding to each IO subject are defined in the iom objects section. Finally, it refers to the IO
rules in the iom protocol section, and define IO rules for each pair of IO subject and IO object.

C. Instantiate the AADL model

A system instance model represents an operational physical system that is generated from a fully specified declarative model [6]. In this paper, the instantiation function of ESMEAT tool is used to generate the system instance model. This is accomplished by first instantiating the system component implementation of top-level system and recursively instantiating its subcomponents. The generated system instance model contains the runtime attributes of each component in the system and can represent a runnable complete system. Researchers can use these instance models to perform analysis, simulation, and automatic generation of embedded software systems, including their runtime executors.

The AADL architecture model which contains IO resource is instantiated. In addition to the original binding, connection, runtime attribute and other information, the component in the generated system instantiation model also includes the information of IO resource being accessed competitively. The generated model is called IO resource constraint scheduling analysis model. The schedulability analysis of this paper is based on this scheduling analysis model.

D. Schedulability analysis for IO resource constrained AADL model

The timed automata model was proposed by Alur et al in the 1990s [7]. It is essentially a finite state machine that extends with clock variables. The clock variable is used to describe the constraint of the system time, and the formal description and calculation of the real-time system behavior are effectively realized. In timed automata, the behaviors of the system are constrained by the clock variables, the states of the system are represented by the location nodes, and the directed edges between the location nodes represent the migration of the system states. At the beginning of the system operation, the system clock is initialized to 0. During a run of timed automaton, clock variables increase all with the same speed. Along with the transitions of the automaton, clock variables can be compared to integers. These comparisons form guards that may enable or disable transitions and by doing so constrain the possible behaviors of automaton.

In order to describe the dynamic operation of tasks in the system AADL model, the thread components need to be described in a timed automaton. The transition of each thread state of the system at runtime is represented by different state transitions of the timed automata, while the time properties such as execution time or deadline in the AADL architecture model are represented as time constraints in the state transition. The system schedulability analysis result can be obtained by analyzing the thread timed automata state.

In the AADL architecture model, threads are the basic scheduling unit, and each thread in the system has a timed automaton, as shown in Fig. 8. The thread timed automaton model describes the state of the thread component in the AADL in the scheduling. When performing the schedulability analysis, the Deadline and Compute Execution_Time properties of the thread component need to be extracted as the time constraint of the timed automaton. The thread timed automaton has four states: Ready, AwaitingResource, Running, and Finished. The clock variable \( r \) is the response time clock variable, \( c \) is the execution time clock variable, and \( c' \) is used to store the time that the thread has executed. The clock \( ET = \text{Compute Execution Time} \) indicates the execution time of the thread, \( DT = \text{Deadline} \), indicating the deadline of the thread. The state transition trigger event \( \text{activate} \) indicates that the thread is activated, \( \text{preempt} \) indicates that the thread is preempted, \( \text{require IO fail} \) indicates that the thread failed to request IO resources at runtime, and \( \text{IO release} \) indicates that the IO resource requested by the thread is released. The thread timed automata in the AADL architecture model is represented as a six-tuple \( A = (S, s_0, \Sigma, X, I, E) \) where

1. \( S \) represents a set of finite states, \( S = \{ \text{Ready, Running, AwaitingResource, Finished} \} \).
2. \( s_0 \) represents the initial state, that is, \( s_0 = \text{Ready} \).
3. \( \Sigma \) indicates the set of state transition trigger events, \( \Sigma = \{ \text{active, preempt, require IO fail, IO release} \} \).
4. \( X \) represents a finite set of clock variables, \( X = \{ r, c \} \).
5. \( I \) represents a mapping of \( S \rightarrow F(x) \), which is a mapping of a state set to a clock set, indicating that a clock constraint is added to the state.
6. \( E \) is a set of state transition conditions, with \( e \) representing the elements, \( e = (s, a, \phi, \lambda, s') \) is a tuple indicating that the state \( s \) shifts to the state \( s' \) when the condition occurs. \( \phi \) is a clock constraint that must be satisfied when the state transitions, and \( \lambda \) is the set of clocks to be reset at the time of the transition.

The variables \( r, c', ET \) and \( DT \) are initialized after a thread is dispatched. Initially, the state is \( \text{Ready} \), the time clock is 0, and the thread is waiting for input event \( \text{activate} \). When the input event \( \text{activate} \) is received, the thread updates the state to \( \text{Running} \) and sets the execution time clock to 0, which is equal to running state of the thread. If the thread
requests blocking IO resources during Running state, the state changes to WaitingResource, and stores the time that the thread has been executed in the variable \( c' \). After the thread receives the IO_release event, it returns to the Ready state and waits for rescheduling. When the execution clock \( c \) is less than the thread execution time, and the higher priority thread deprives the processor resource of the current running thread, the current thread transitions from the Running state to the Ready state. When the execution time clock \( c \) is equal to the thread execution time \( ET \) and the response time clock \( r \) is less than the thread deadline \( DT \), the thread is normally executed. If the response time clock \( r \) is overstepping the thread deadline \( DT \), the thread is unschedulable.

When the system is running, all threads’ states are initialized to Ready. The priority of a thread is calculated according to the scheduling algorithm specified by the architecture model. Find the thread with the highest priority, name its thread_cur and set the state of it to Running. When a new task is released or a partition switch occurs, the priority of all threads whose current state is Ready is calculated again, and the highest priority thread named thread new is selected to be compared with thread_cur. The state of the thread with the highest priority is set to Running, and the state of the thread with the lower priority is set to Ready. When the thread_cur in the Running state is transferred to the Finished state, all threads whose state is WaitingResource are viewed, find threads that have IO resource constraint relationship with thread_cur and set their states to Ready. This indicates that the IO resources they are waiting for are released.

The competition among threads for IO resources affects the scheduling sequence of threads, and it affects system schedulability. To analyze whether the system can be scheduled, it is necessary to analyze whether the thread blocked by the IO resource can be executed before its deadline. According to the thread scheduling timed automata, the IO resource constraint scheduling algorithm is designed. The input of the algorithm is the IO constrained AADL instance model, and the output of algorithm is a task scheduling sequence under IO resource constraints.

According to Algorithm 1, a scheduling sequence is obtained, where \( T_i^j \) represents the \( j \)th instance of thread \( T_i \). Assume that the scheduling sequence contains \( n \) threads represented by \( T \), where \( T = \{T_1, T_2, ..., T_n\} \). Now let’s assume \( T_i \in T \), there are \( m \) instances of thread \( T_i \) in a supercycle, among which the collection of instances is \( \{T_i^1, T_i^2, T_i^3, ..., T_i^m\} \). Now consider the execution of the \( j \)th instance \( T_i^j \) of thread \( T_i \). Suppose there are some threads competing with \( T_i^j \) for IO resources during the execution of \( T_i^j \), and the threads that compete with \( T_i^j \) have higher priority than \( T_i^j \). So the response time of \( T_i^j \) can be calculated according to Formula (1), where \( p \) represents the the number of thread \( T_i^j \) breakpoint where thread \( T_i^j \) is interrupted by other thread with a higher preemption, and \( q_u \) represents the number of thread which are preemptive to interrupt thread \( T_i^j \) at \( u \)th breakpoint, where \( T_u^s \) represents threads competing with \( T_i^j \) for IO resources at \( u \)th breakpoint, denotes as \( T_u^s = \{T_u^s, T_u^s, T_u^s, ..., T_u^s\} \).

\[
RT_{T_i^j} = \sum_{u=1}^{p} \sum_{v=1}^{q_u} ET_{T_u^s} + ET_{T_i^j}
\]  

Therefore, the worst response time of thread \( T_i \) is the maximum response time of each instance of thread \( T_i \), which can be calculated according to Formula (2), where \( m \) represents the number of thread instance \( T_i^j \) executes in a supercycle.

\[
WCR T_i = \max \left\{ RT_{T_i^1}, RT_{T_i^2}, RT_{T_i^3}, ..., RT_{T_i^m} \right\}
\]

It just compares each task WCRT with its DT respectively. It is said, \( \forall T_i \in T \), if \( WCRT_{T_i} \leq DT_{T_i} \), \( i \in N \), and the AADL architecture is schedulable; Otherwise, \( \exists k, T_k \in T \), if \( WCRT_{T_k} > DT_{T_k} \), and the AADL architecture is unschedulable.
E. Computation and simulation

The schedulability analysis method of this paper uses the timed automata tool, named Cheddar, to calculate the time constraint of the task to get the schedulability analysis result. In order to calculate AADL model scheduling with Cheddar, the AADL scheduling model must be transformed to a timed model as an input model of the Cheddar tool which is defined using the Cheddar language.

1) Transforming AADL model to timed automata with Cheddar: In order to use the Cheddar tool to perform schedulability analysis on the AADL architecture model, the AADL architecture model needs to be analyzed and converted into an input model of the Cheddar tool. Specifically, the IO resources, processors, processes, threads and other components in the AADL model are analyzed, converted into components with corresponding functions in the Cheddar tool. The processor component in the AADL model is a scheduling unit in the system for scheduling and execution of tasks. The process components act as protected address spaces. The thread component is the task unit in the AADL model and is the basic unit of processor scheduling. In the AADL architecture model schedulability analysis, it is mainly to analyze whether the processor component can correctly schedule all threads, that is, to determine whether the thread completes execution within its deadline.

Some mapping rules between the AADL architecture scheduling model with IO resource constraints and the Cheddar input model are shown in Table I, and mapping rules are implemented the conversion tool from the AADL architecture model to the Cheddar input model in the form of a plug-in, and provides input files for the simulation calculation of the calling tool.

2) Define scheduling strategy: The Cheddar provides an Ada-like language that allows you to design user-defined scheduling policies. Based on the scalability of the Cheddar scheduling policy, it can be used to analyze the schedulability of the AADL architecture model under IO resource constraints.

Timed automata are used to represent the running and switching of tasks, and time constraints are used to control the state transition process. Cheddar uses a timed automaton to model the thread scheduling process and run the above-mentioned Ada-like language block to read and write simulation data. The code snippet of a timed automaton defined in Cheddar is shown in Fig. 9.

![Fig. 9. The timed automata snippet](image)

### Table I

<table>
<thead>
<tr>
<th>AADL elements and property</th>
<th>Cheddar elements and property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Processor</td>
</tr>
<tr>
<td>Priority</td>
<td>Scheduler</td>
</tr>
<tr>
<td>Dispatch-Protocol</td>
<td>Address space</td>
</tr>
<tr>
<td>Resource protocol</td>
<td>Protocol</td>
</tr>
<tr>
<td>IO subject</td>
<td>Resource</td>
</tr>
<tr>
<td>Name of thread</td>
<td>task-name</td>
</tr>
<tr>
<td>IO object (thread)</td>
<td>Compute-Execution-Time of thread</td>
</tr>
<tr>
<td>Type</td>
<td>task-type</td>
</tr>
<tr>
<td>name of processor</td>
<td>task</td>
</tr>
<tr>
<td>period</td>
<td>task</td>
</tr>
<tr>
<td>Dispatch-time</td>
<td>task</td>
</tr>
<tr>
<td>Compute-Execution-Time</td>
<td>task</td>
</tr>
<tr>
<td>deadline</td>
<td>task</td>
</tr>
<tr>
<td>Priority</td>
<td>task</td>
</tr>
</tbody>
</table>

IV. CASE STUDY

In this section, we have selected an IMA-based civil airborne system to illustrate the feasibility of the proposed method. First, we introduced the schedulability analysis tool we implemented. Second, we build the architecture model and the IO resource model for the system. Third, we extract the information in the instantiated model file and construct the timed automata. Finally, we generate the Cheddar input model file and the scheduling policy file, calculate by the Cheddar tool and get the schedulability analysis result.

A. Towards a schedulability analysis tool

We implemented a plug-in in the Eclipse integrated development environment to perform a schedulability analysis of the AADL architecture model containing the IO resource model. Firstly, to establish the modeling environment of IO resource model, we use Xtext to implement IOMA as a modeling tool. Secondly, we implement a plug-in named S for model transformation and generation of scheduling models. The plug-in converts the AADL model to a Cheddar input model. Meanwhile, timed automata files and scheduling policy files are generated based on the AADL model content. Thirdly, to obtain the schedulability analysis results calculated by Cheddar, we implement a plug-in named IO, to call Cheddar to simulate the generated scheduling simulation model file.

B. Building architecture model

An architecture platform based on IMA for civil airborne systems, which consists of 4 GPM(General purpose processing module), 8 AFDX switches, 4 RDC(Remote data concentrator), avionics(Integrated navigation system, smart sensor, atmospheric computer, etc.), and electromechanical...
equipment (landing gear, hydraulics, fuel, environmental control, etc.). It is shown in Fig. 10. The system uses double redundancy design, the general processing module GPM-A and GPM-B form double redundancy, responsible for avionics tasks, GPM-C and GPM-D form double redundancy, responsible for electromechanical tasks. The ARINC653 Avionics Application Interface Standard provides a guideline for the development of avionics software based on the IMA architecture. It proposes the concept of a partitioned operating system, the core concept of which is spatial and temporal partition isolation [8]. The AADL ARINC653 Annex is an appendix developed by AADL to support the modeling of the ARINC653 partition system. It provides a guide to the AADL architecture modeling and the properties required in ARINC653, giving a general approach to representing the ARINC653 architectural model using AADL standardized components [8]. The time-space partition modeling using AADL is achieved by the scheduling strategy of time slice rotation and the division of memory.

This case study just focuses on the avionics module GPM-A. In GMP-A, there are 3 partitions, which are display control partition, integrated processing partition and flight management partition. The partition situation and description are shown in Table II. According to the above, this paper divides the tasks of the display control partition, integrated processing partition and central maintenance partition in GPM-A. The task division is shown in Table III.

With a process bound to a virtual processor representing a partition, the system requires 3 virtual processors and 3 processes to model 3 partitions separately. Using the thread in the process to represent the task in the partition. The time properties of a task are represented by the relevant properties of the thread component. The system architecture model of the GPM-A module is established based on AADL, as shown in Fig. 11.

C. Building IO resource model

According to the system architecture model, we need to analyze the system IO resource information and establish a system IO resource model. In the three partitions of the system, the data collection tasks in the display control partition and the integrated processing partition need to collect data from the sensor, and the data output task needs to write the data into a shared memory and then read by the central maintenance partition. There are shared data in the display partition and the integrated processing partition for the task to use.

D. Schedulability analysis for IO resource-constrained AADL model

Instantiate the AADL architecture model that includes the IO resource model. According to Algorithm 1, extracts task, IO constraint and partition information from the instantiated model file, and constructs the timed automata.

There are three partitions in the AADL scheduling analysis model of this case, so there are four states in the partition timed automata. When the partition scheduling is running, it is necessary to extract the scheduling strategy of the partition and the property such as the time slice T, and the time slice T is used as the time constraint of the timed automata. The timed automaton of the partition is in the restart state. When the

<table>
<thead>
<tr>
<th>Partiton</th>
<th>Tasks</th>
<th>Period</th>
<th>Execution time</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display control (RMS) (T=4)</td>
<td>Data collection</td>
<td>40</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Data output</td>
<td>80</td>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>display</td>
<td>100</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Command</td>
<td>100</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>Integrated processing (RMS) (T=6)</td>
<td>Data collection</td>
<td>40</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Data output</td>
<td>80</td>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Process</td>
<td>40</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Record</td>
<td>80</td>
<td>4</td>
<td>80</td>
</tr>
<tr>
<td>Central maintenance (RMS) (T=4)</td>
<td>Monitor</td>
<td>80</td>
<td>5</td>
<td>80</td>
</tr>
</tbody>
</table>

Fig. 10. The architecture of a civil airborne system
When the partition running time is equal to the time slice $T$ of the partition, the timed automata state is switched from the current partition to the next partition. When all partitions have run their respective time slices, the timed automata state goes to the restart state and sets the partition clock to 0, waiting for the processor to schedule again.

For each partition, when the processor scheduling event is received, the partition enters the running state. In the running state, the partition calculates the priority of the thread in the partition according to the scheduling algorithm of the partition and finds the thread with the highest priority to run.

According to the analysis framework proposed in this paper, the simulation operation is performed using the Cheddar tool, and the result is shown in Fig. 12. The upper part of Fig. 12 is a simulated Gantt chart without considering IO resources, while the lower part of Fig. 12 is a Gantt chart with IO resource constraints. The red line represents the sum of the time slices for all partitions. All tasks are schedulable in this case. At 14 ms, task T12 is blocked by task T13, causing task T13 to execute first. Because each task’s WCRT is smaller than its deadline. In a hypercycle, each task of the system is schedulable, and the system is schedulable.

V. RELATED WORK

The AADL with property supports system architecture modeling and scheduling information description in the early phases of system development. Some literature has also proposed many extended models through annex. Wei Xiaomin et al. [9] proposed a hazard model annex to describe the possible hazards in the system. Through the defined hazard sources, the hazard model was associated with the error model provided by AADL to form the architecture system safety model. This paper [10] extends the time attribute based on AADL to describe burst error duration and random error duration. However, these extended annexes cannot model the IO resource constraints in the system, and cannot describe the impact of IO constraints on system scheduling.

In terms of schedulability analysis, Zhexu Liu et al. proposed a modeling method of the IMA partitioning based on AADL [11]. Based on the discussion of mapping rules, an IMA partition model is established from two aspects: architecture modeling and scheduling policy modeling. The architecture model describes the framework of the IMA partition, and the scheduling policy model describes the actual scheduling operations. Finally, the schedulability of IMA partition is verified based on the model. Mourad Dridi et al. [12] proposed a new NoC communication model called Exact Communication Time Model (ECTM) in order to analyze the scheduling of periodic tasks exchanging messages over a NoC(Network-on-Chips). Then performed scheduling analysis with a list scheduling algorithm called HLFET. Jian Sun et al. [13] proposed a formal test method using timed automata, designs a transformation method from AADL scheduling model to timed automata model, simulates and verifies timed automata model in UPPAAL tool, and verifies the schedulability of the original model equivalently by using relevant verification statement. None of the above studies have considered modeling the IO
resources of the partition system. When analyzing the schedulability of the system, the constraints of the IO resources on the system tasks are not considered. This paper extends the AADL core language standard, establishes an IO resource model for the system, and proposes a schedulability analysis method for the AADL architecture model under IO resource constraints.

In terms of the extension of the AADL core language standard, Procter et al. [14] presented the EMV2 error library, as well as its underlying concepts, inspirations, and impetuses, and provided a guidance on using ontologies with modern hazard analyses. Jinhiao Xu et al. [15] proposed a hierarchical extension of the AADL behavioral annex which is named Hierarchical Behavior Annex (HBA). Firstly, the formal syntax of HBA is given, and then the semantics of HBA is formally defined. The metamodel of HBA is proposed, and the text and graphic editors of HBA are implemented in OSATE environment. Finally, an industrial example is given to verify the effectiveness of the proposed method. Ehsan M et al. [16] proposed an AADL-DEVs framework, a DEVs behavioral annex targeted for the DEVs-Suite simulator is developed and introduced to OSATE which supports AADL. For the event port and event data port in the AADL standard, several method sare proposed to evaluate the memory requirements of the AADL architecture model. We refer to the real-time scheduling theory information, integrate the information needed for scheduling analysis from the perspective of IO resources, abstract the AADL IO resource model annex, and enable AADL to describe more kinds of IO resources.

VI. CONCLUSION AND FUTURE WORKS

In a real-time embedded system, some hardware or software code may be in conflict during the process of being accessed as a resource. The competition of components in the system may affect the performance of the system. The impact of resource configuration on the non-functional attributes of the system is critical. If a problem occurs, the system may not respond within the specified time, causing serious consequences. Although some of the parts in the system can be represented using some of the components in the AADL core language standard, such modeling results in a lack of clarity on the description of system resource information. In the actual development process, after the system architecture modeling is completed, non-functional attribute analysis is also performed. In this process, the resource information in the system needs to be extracted and analyzed. The AADL IO resource model annex proposed in this paper makes up for the lack of AADL description ability, and gives the syntax and semantics of the IO resource model annex. The IO resource model can be built for the system by using the IO resource model annex. This paper uses the theory of timed automata to transform the system’s IO resource constraints scheduling analysis model into a timed automaton. Finally, through the model transformation and custom scheduling strategy, using tools for calculation and simulation, the results of schedulability analysis are obtained. The schedulability analysis of the IO resource-constrained AADL architecture model is implemented.

This paper specifies the schedulability analysis method of IO resource-constrained AADL architecture model through an industrial case. Include the usage of the IO resource model annex in the system modeling, and the application of the IO resource model in schedulability analysis. Based on the proposed resource model to analyze the non-functional attributes of the system. This paper only considers the schedulability of the system without analyzing the security and reliability of the system, and the analysis of schedulability only considers the schedulability of a single node system. The subsequent work will analyze the impact of system resources from the aspects of safety and reliability, and extend the study of resource-constrained schedulability to a multi-node distributed system, and continue to expand the IO resource model annex to support the analysis of system safety and reliability.

REFERENCES