Research on MagicGrid-Based Requirements Development Process of Flight Control System

Hangqi Liu[1], Yong Chen[1](✉), Meng Zhao[1], and Zhuo Yin[2]

[1] School of Aeronautics and Astronautics, Shanghai Jiao Tong University, Shanghai, China
{liuhangqi,aerocy,magicmeng}@sjtu.edu.cn
[2] Shanghai Institute of Satellite Engineering, Shanghai, China

Abstract. With the development of aviation technology, the complexity of aircraft systems has been increasing. The development of requirements for aircraft systems has become increasingly important. However, traditional requirements development methods often overly rely on text and manual operations, resulting in unclear requirements, difficulties in changes, and low efficiency. They are no longer able to meet the needs of designers and developers. As an emerging requirements development method, the MagicGrid method fully considers the complexity and scalability of the system, and through the decomposition and organization of system-level requirements, designers can understand the system requirements more clearly, and is being continuously applied to aircraft system design.However, in current engineering practice, the MagicGrid method has not yet been fully implemented due to its high complexity and insufficient tool support. Therefore, this paper describes the process of requirements development using the MagicGrid method in detail, and demonstrates the specific process through a practical case of requirements development for the flight control system.Finally, the advantages and disadvantages of the method were further analyzed, and some feasible solutions were proposed, providing new practice and experience for the application of the MagicGrid method in aircraft system design.

Keywords: MagicGrid · Civil Aircraft System · Requirements development

1 Introduction

As a typical complex product system, aircraft have the characteristics of high research and development costs, high degree of technological intensity, long development cycle, and strong interdisciplinary nature [1]. It requires comprehensive multi-disciplinary technologies to optimize the overall performance of the aircraft. Requirements development is the beginning of civil aircraft development, the basis of civil aircraft development, and runs through the entire life cycle of aircraft development [2]. It transmits the customer's requirements layer by layer, so that the final produced aircraft meets the customer's requirements and the requirements of operational safety.

The traditional requirements development methods can often be termed as document-based systems engineering, with documents as the core in the requirements development process. However, in the face of highly dynamic and interactive activities, the information expression with the document-based systems engineering approach was often incomplete, making it difficult to ensure the integrity and consistency of engineering requirements, and ambiguity is easy to occur [3]. Additionally, the lack of testing and verification methods makes the requirements development process difficult, and it is easy to bring early requirements problems into the later design process, resulting in substantial changes to the system design process. Therefore, the traditonal document-based systems engineering approach cannot effectively support the development of complex systems of today [4].

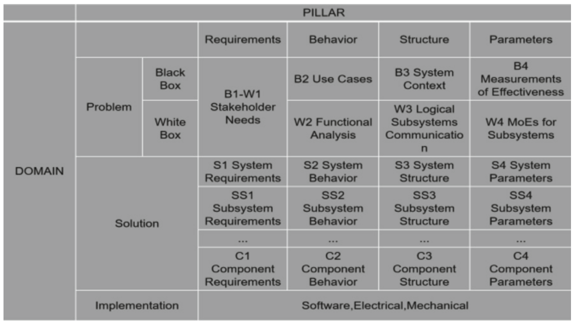
Therefore, there should be some new methods for developing the requirements for complex engineering systems. This research examines the MagicGrid approach to see whether it can be applied to the requirements development of complex systems,especially for the development of aircraft systems. MagicGrid is an approach based on system engineering thought, which aim to solve the shortcomings of traditional requirements development methods. Compared with the traditional requirements development method, MagicGrid is expected to provide several advantages. First, it is a model-based systems engineering method [5], which has the advantages of visualization and automation. This allows systems engineers and other stakeholders to better understand the system, reducing manual errors, and improving the efficiency and quality of system development. Second, the MagicGrid method emphasizes the traceability of requirements, which can help designers better manage the requirements development process. Finally, the MagicGrid method has better scalability and can be flexibly adjusted and optimized according to actual requirements. Therefore, the MagicGrid method can well solve the existing problems in the field of requirements development and improve the quality and efficiency of requirements development.

However, there are still some challenges in the practical application of the MagicGrid method. Firstly, the MagicGrid method is relatively complex and requires developers to have high capabilities in order to understand the required technology. In addition, the MagicGrid method requires a lot of experience accumulation to identify and handle problems that arise. Secondly, the types and quantities of aircraft system requirements are enormous, requiring an efficient method for requirement collection and management. Finally, the MagicGrid method needs the support from corresponding tools, and the current tools still have some shortcomings. Therefore, in order to better apply it to aircraft system design, further research and exploration are needed.

The purpose of this article is to investigate the applicability of the MagicGrid method in aircraft system requirements development, and to exploit how to use this method to solve the existing problems of traditional requirements development methods. Through the in-depth analysis of the MagicGrid method and the actual case studies of the development of flight control system requirements, the feasibility of the MagicGrid method in aircraft system requirement development is further verified, providing reference for developers and designers. Finally, some existing deficiencies are identified, and suggestions for improvement are proposed to better apply the MagicGrid method to solve problems in the requirement development process.

2 Overview of the MagicGrid Method

The MagicGrid modeling method, initially developed by No Magic and later integrated into the Dassault Systems 3DE platform, is a relatively recent addition. The MagicGrid approach is based on a framework, which can be represented as a Zachman-style matrix [6]. Its main process is shown in the Fig. 1.



			PILLAR			
DOMAIN			Requirements	Behavior	Structure	Parameters
	Problem	Black Box	B1-W1 Stakeholder Needs	B2 Use Cases	B3 System Context	B4 Measurements of Effectiveness
		White Box		W2 Functional Analysis	W3 Logical Subsystems Communicatio n	W4 MoEs for Subsystems
	Solution		S1 System Requirements	S2 System Behavior	S3 System Structure	S4 System Parameters
			SS1 Subsystem Requirements	SS2 Subsystem Behavior	SS3 Subsystem Structure	SS4 Subsystem Parameters
			...	...	...	...
			C1 Component Requirements	C2 Component Behavior	C3 Component Structure	C4 Component Parameters
	Implementation		Software,Electrical,Mechanical			

Fig. 1. The Structure of the MagicGrid

From Fig. 1, we can know that the MagicGrid method mainly includes three domains: the problem domain, the solution domain and the implementation domain [7]. Each domain considers and captures four aspects of the system, and these four aspects correspond to the SysML pillars as requirements, behavior, structure, and parameters [8].

The problem domain modeling is the starting point of the MagicGrid method. Its main task is to analyze the requirements of stakeholders [9], and refine them using SysML model elements. The problem domain is divided into two levels, black box and white box [10]. As shown in Fig. 2, the black box mainly focuses on how it is related to the outside world. After completing the black-box modeling of the problem domain, it enters the white-box modeling stage, which involves expanding the black-box model and analyzing it from the inside. In essence, white-box modeling relies on the black-box model, and the output at this stage will be more detailed and complete in describing the system [11].

The main modeling process of the solution domain is shown in Fig. 3. Through modeling in the solution domain, it can help developers understand the problem domain, better discover and understand user requirements [12], reduce the risk and cost of system design.

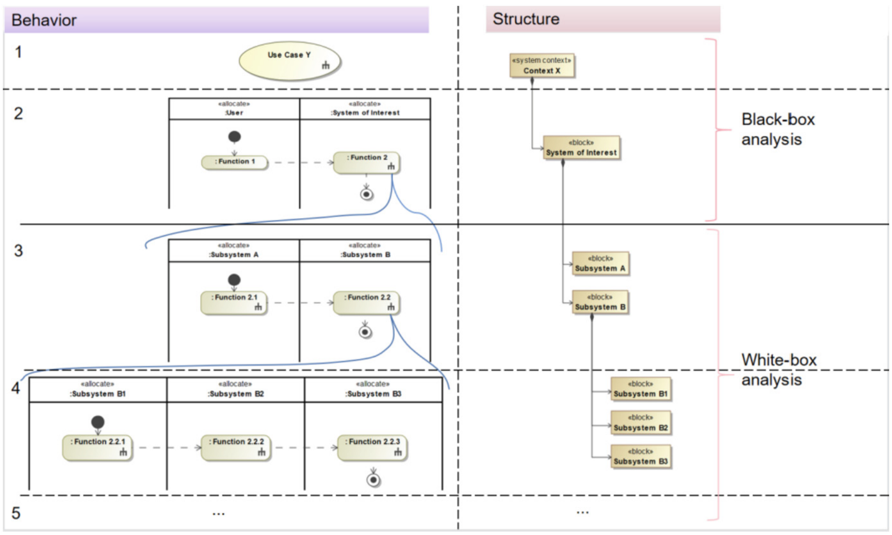


Fig. 2. The Process of the Problem Domain

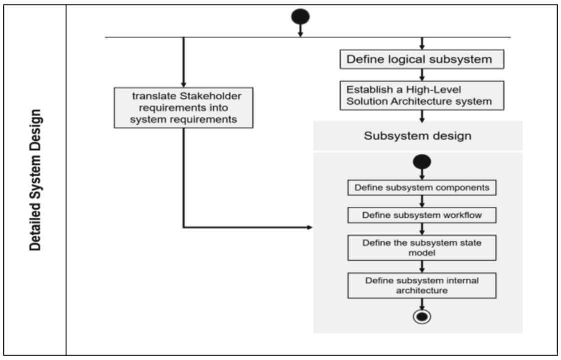

Detailed System Design
translate Stakeholder requirements into system requirements
Define logical subsystem
Establish a High-Level Solution Architecture system
Subsystem design
Define subsystem components
Define subsystem workflow
Define the subsystem state model
Define subsystem internal architecture

Fig. 3. The process of the Solution Domain

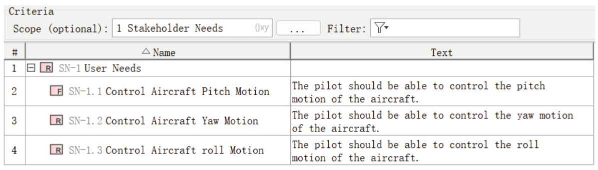
So far, the two core levels of the MagicGrid method have been fully implemented. Although the MagicGrid method defines the physical requirement specification of the implemented system, its detailed design is not part of MBSE, so it will not be analyzed. The following chapters 3 and 4 will describe the development process of the problem domain and solution domain through examples, providing a clearer understanding of the process of requirements development based on the MagicGrid method.

3 System-Level Requirements Development

3.1 Black Box Modeling

In the black box modeling stage of the problem domain, the system is considered as a black box, which means that we only need to analyze the input and output of the system, as well as the interaction logic between the system and external participants, without specifying its dynamic behavior and internal structure. According to the MagicGrid method, black-box modeling mainly includes three parts: stakeholder requirements, system use cases, and system environment. The following model diagrams will detail the modeling processes for each part.

Initial Stakeholder Needs. The MagicGrid method centers around establishing executable system models. Modeling is driven by requirements, so identifying system stakeholders, capturing stakeholder needs, and establishing SysML demand models are prerequisites for problem domain modeling. Stakeholder needs are those directly or indirectly related to the product or project, which should be captured, analyzed. But these needs obtained are often very rough, and, therefore, require detailed analysis and refinement [13]. Because this article mainly focuses on the development of functional requirements, as shown in the Fig. 4, three pieces of stakeholder needs related to flight control functions are listed as examples.


Criteria
Scope (optional): 1 Stakeholder Needs ... Filter:

#	Name	Text
1	SN-1 User Needs	
2	SN-1.1 Control Aircraft Pitch Motion	The pilot should be able to control the pitch motion of the aircraft.
3	SN-1.2 Control Aircraft Yaw Motion	The pilot should be able to control the yaw motion of the aircraft.
4	SN-1.3 Control Aircraft roll Motion	The pilot should be able to control the roll motion of the aircraft.

Fig. 4. Initial Stakeholder Needs

Definition and Analysis of Use Case. Use case is a model element derived from the abstraction of the functional requirements in the requirements model, which represents those functional activities that can be directly triggered or participated by external actors. The use case is the core element of the use case diagram and the basis for subsequent modeling and analysis. It will run through the entire project life cycle. In the MagicGrid method, the use cases mainly exist in the problem domain modeling stage, and the use cases are extracted by the top-level functions of the attitude control module. The use case diagram shown in Fig. 5 is established.

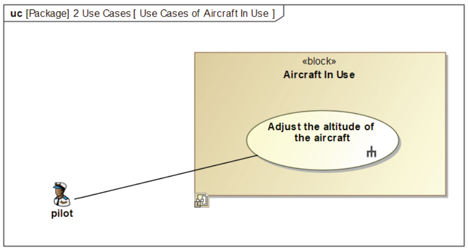

uc [Package] 2 Use Cases [Use Cases of Aircraft In Use]
«block»
Aircraft In Use
Adjust the altitude of the aircraft
pilot

Fig. 5. The Use Case Digram

The use case analysis is mainly to describe the use cases defined in the above process from the perspective of behavior, that is, to establish the dynamic behavior model of the aircraft system. Because we are currently in the black box stage, the established SysML behavior model mainly focuses on the interaction logic with external participants. The Fig. 6 shows a black-box activity diagram of the use case, displaying the system workflow from an external perspective. There are mainly two swim lanes, representing the pilot and the aircraft respectively, where the pilot represents the external participant in the context of the use case.

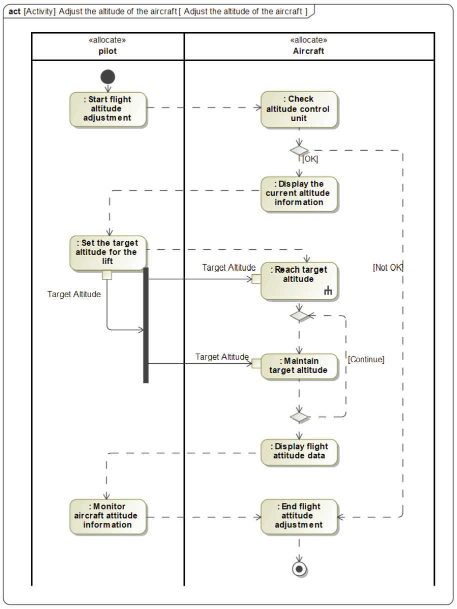

act [Activity] Adjust the altitude of the aircraft [Adjust the altitude of the aircraft]
«allocate»
pilot
«allocate»
Aircraft
: Start flight altitude adjustment
: Check altitude control unit
[OK]
: Display the current altitude information
: Set the target altitude for the lift
Target Altitude
Target Altitude
: Reach target altitude
[Not OK]
Target Altitude
: Maintain target altitude
[Continue]
: Display flight attitude data
: Monitor aircraft attitude information
: End flight attitude adjustment

Fig. 6. The activity diagram of adjusting the altitude of the aircraft

System Context. The system environment or system context mainly describes the external environment of the system, that is, creates a structural view of different use cases from an external perspective. In addition to the system itself, elements in a particular system's environment should include the external systems or users that interact with the system in that environment.

Through the analysis and definition of the system environment, developers can better understand the system requirements and ensure that system entities can interact well with the outside world. As shown in Fig. 7, the system environment of the aircraft, the pilot is an external participant, providing control to it.

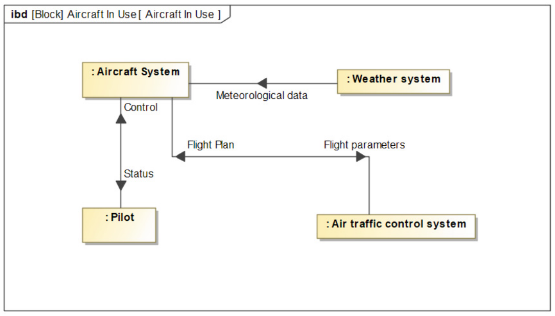


Fig. 7. The System Context for the aircraft system

3.2 White Box Modeling

After the completion of the black-box perspective modeling, we can enter the second stage of problem domain modeling: white-box modeling. White-box means the breakdown of black-box to gain deeper understanding of how the system operates. Through in-depth analysis of the system's functions, functional modules and associated logical subsystems are identified. In essence, the white box modeling stage is to refine and decompose the black box model, and assign more detailed requirements, activities and structures to system function modules.

System-Level Function Analysis. At this stage, establish a more detailed activity diagram model, and assigns behavioral activities to specific subsystems based on certain control logic, that is, which subsystems should satisfy which system behaviors.

Because this paper mainly studies the flight control system, it mainly analyzes the systems related to flight control.In the specific workflow, after receiving the target height command, it is necessary to carry out comprehensive calculation and processing of the height command, and then output the operation commands to other activities through the object flow.As shown in the Fig. 8, the thrust command, pitch command and trim Command are respectively output to other activities that change the thrust, control aircraft pitch, and perform trim operations. The four swim lanes represent the defined logical subsystems, and the corresponding swim lane implement corresponding activity.

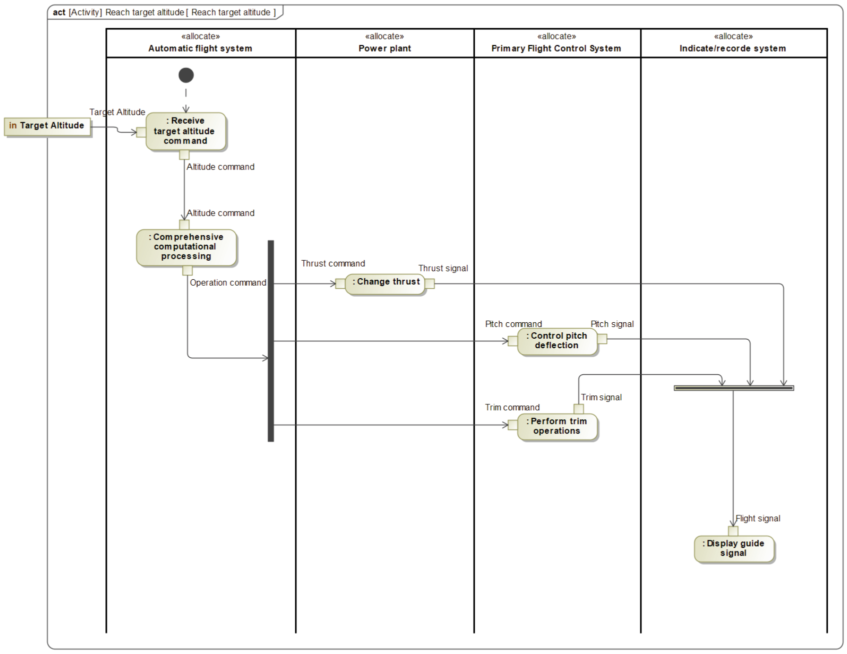

act [Activity] Reach target altitude [Reach target altitude]
«allocate» Automatic flight system
«allocate» Power plant
«allocate» Primary Flight Control System
«allocate» Indicate/recorde system
in Target Altitude
Target Altitude
: Receive target altitude command
Altitude command
Altitude command
: Comprehensive computational processing
Operation command
Thrust command
Thrust signal
: Change thrust
Pitch command
Pitch signal
: Control pitch deflection
Trim signal
Trim command
: Perform trim operations
Flight signal
: Display guide signal

Fig. 8. The SwimLane diagram of reaching target altitude

System-Level Architecture. With the completion of dynamic behavior modeling in the problem domain,the model describes the external and internal behavior activities of the aircraft system in detail from two perspectives of system use cases and functional analysis. However, we still need to further analyze the internal composition structure according to the results of functional analysis.

As shown in the Fig. 9, after further analysis, the related systems of the aircraft are decomposed, the internal IBD diagram of the system is established, the internal functional architecture model is described, and the port transmission of the internal system is established. And according to the events or data information transmitted by the port, the internal systems are connected to each other or to external ports.

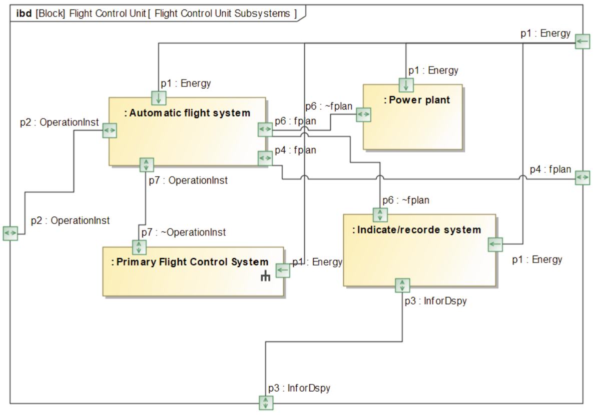
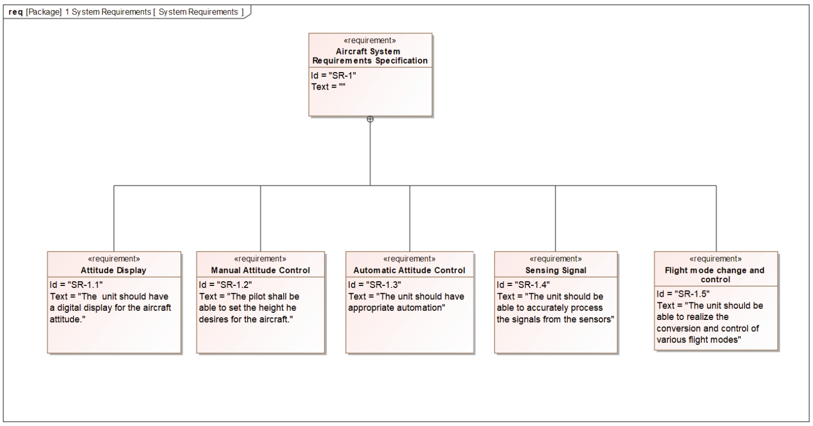


Fig. 9. The internal architecture for the Flight Control Unit

System-Level Requirements and Traceability

During the above problem domain modeling process, a stakeholder requirement model is established to meet the problem domain design task. In order to meet the precise system architecture design requirements, it is necessary to establish a system requirements specification to guide the domain modeling work. System requirements are derived from stakeholder requirements [12], and are refined and enriched according to the model of the problem domain.

System requirements are the premise of system design. Different from stakeholder needs, system requirements are observable, system-specific, and precise expressions. This article mainly discusses the development of the functional requirements of the flight control system, so we only list the functional requirements. As shown in the Fig. 10, five functional requirements for the aircraft system are listed for examples.


req [Package] 1 System Requirements [System Requirements]
«requirement»
Aircraft System Requirements Specification
«requirement»
Attitude Display
«requirement»
Manual Attitude Control
«requirement»
Automatic Attitude Control
«requirement»
Sensing Signal
«requirement»
Flight mode change and control

Id = "SR-1"
Text = ""

Id = "SR-1.1"
Text = "The unit should have a digital display for the aircraft attitude."

Id = "SR-1.2"
Text = "The pilot shall be able to set the height he desires for the aircraft."

Id = "SR-1.3"
Text = "The unit should have appropriate automation"

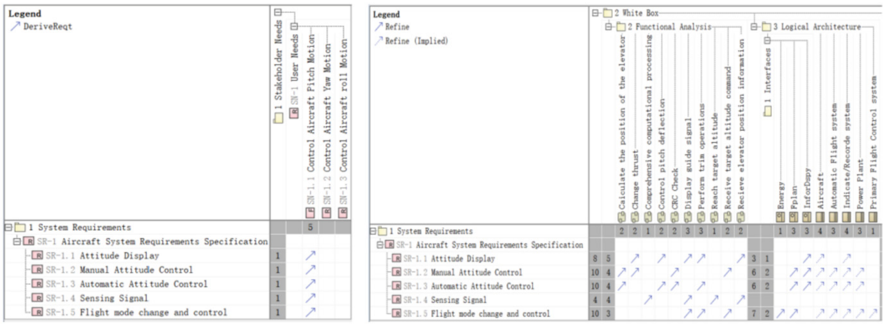
Id = "SR-1.4"
Text = "The unit should be able to accurately process the signals from the sensors"

Id = "SR-1.5"
Text = "The unit should be able to realize the conversion and control of various flight modes"

Fig. 10. The functional requirements for the aircraft system

In addition, it is necessary to establish traceability from system requirements to stakeholder requirements, that is, to determine which system requirements are derived from which stakeholder needs.At the same time,traceability relationships between system requirements and problem domain model elements need to be established to determine which system requirements are refined from which elements.

As shown in the Fig. 11, the left part of the picture is the traceability relationship between system requirements and stakeholder requirements. The horizontal row of the correlation matrix represents the system requirements, and the vertical row represents the stakeholder requirements. The arrow indicates that the system requirements of this row point to the stakeholder requirements of this column. The right part of the picture is the traceability relationship between system requirement and problem domain model elements. Similarly, the horizontal rows of the matrix represent the system requirements, and the vertical rows represent the model elements of the problem domain.



Legend
↗ DeriveReqt

	1 Stakeholder Needs	SN-1 User Needs	SN-1.1 Control Aircraft Pitch Motion	SN-1.2 Control Aircraft Yaw Motion	SN-1.3 Control Aircraft roll Motion
1 System Requirements			5		
SR-1 Aircraft System Requirements Specification					
SR-1.1 Attitude Display	1		↗		
SR-1.2 Manual Attitude Control	1		↗		
SR-1.3 Automatic Attitude Control	1		↗		
SR-1.4 Sensing Signal	1		↗		
SR-1.5 Flight mode change and control	1		↗		

Legend
↗ Refine
↗ Refine (Implied)

	2 White Box	2 Functional Analysis	Calculate the position of the elevator	Change thrust	Comprehensive computational processing	Control pitch deflection	CRC Check	Display guide signal	Perform trim operations	Reach target altitude	Receive target altitude command	Recieve elevator position information	3 Logical Architecture	1 Interfaces	Energy	Fplan	InforDspy	Aircraft	Automatic Flight system	Indicate/Recorde system	Power Plant	Primary Flight Control system
1 System Requirements			2	2	1	2	2	3	3	1	2	2			1	3	3	4	3	4	3	1
SR-1 Aircraft System Requirements Specification																						
SR-1.1 Attitude Display	8	5		↗		↗		↗	↗			↗	3	1			↗	↗		↗		
SR-1.2 Manual Attitude Control	10	4	↗	↗			↗				↗		6	2		↗	↗	↗	↗	↗	↗	
SR-1.3 Automatic Attitude Control	10	4	↗			↗	↗		↗				6	2		↗	↗	↗	↗	↗	↗	
SR-1.4 Sensing Signal	4	4			↗			↗		↗		↗										
SR-1.5 Flight mode change and control	10	3						↗	↗		↗		7	2	↗	↗		↗	↗	↗	↗	↗

Fig. 11. The traceability for the System-level requirements and elements

4 Component-Level Requirements Development

For the aircraft system, since the overall architecture system is relatively mature and the system composition of the control flight attitude module is relatively clear, the development of system-level requirements can be realized in the field of white-box modeling without re-analysis in the solution domain. Next, the component-level requirements will be further analyzed based on MagicGrid's solution-domain method.

4.1 System-Level Modeling

Defining Subsystems. After the completion of system requirements, the high-level solution structure of the aircraft needs to be established. The HLSA model specifies all logical subsystems in a single hierarchy.Since this article only focuses on the development of the flight control system, only the relevant systems are analyzed. Furthermore, as the aircraft system is a relatively well-structured product system, its system composition is relatively clear, so its HLSA model composition is the same as the problem domain. As shown in Fig. 12, not only the logical subsystem is specified, but also the subsystem ports and the data flow transmitted by the ports are specified.

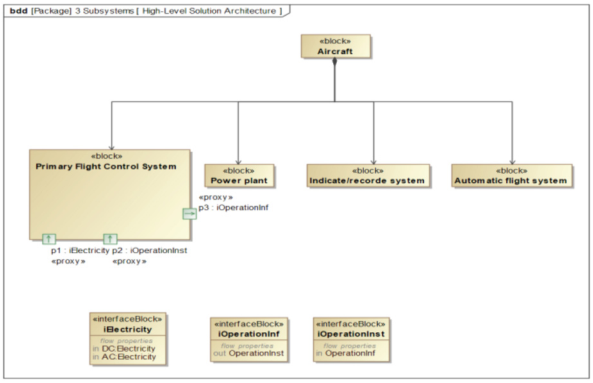

bdd [Package] 3 Subsystems [High-Level Solution Architecture]
«block» Aircraft
«block» Primary Flight Control System
«block» Power plant
«block» Indicate/recorde system
«block» Automatic flight system
«proxy» p3 : iOperationInf
p1 : iElectricity «proxy»
p2 : iOperationInst «proxy»
«interfaceBlock» iElectricity
flow properties
in DC Electricity
in AC Electricity
«interfaceBlock» iOperationInf
flow properties
out OperationInst
«interfaceBlock» iOperationInst
flow properties
in OperationInf

Fig. 12. The High-level Solution Architecture for the aircraft system

Structural Modeling of Primary Flight Control System. This paper selects the primary flight control system as the research object, and based on the required functions,six components of the main flight control system are mainly defined, including the elevator position sensor, remote electronic control unit, elevator aileron computer, actuator,main flight control computer and the actuator control electronic device. The connection relationships between each component and their respective port information are also defined, as shown in the Fig. 13. Three interaction ports between the main flight control system and the external system, the first is the pitch command transmitted by the automatic flight system; the second is the energy supply provided by the power unit; and the third is the pitch signal that needs to be sent to the indication/recording system.

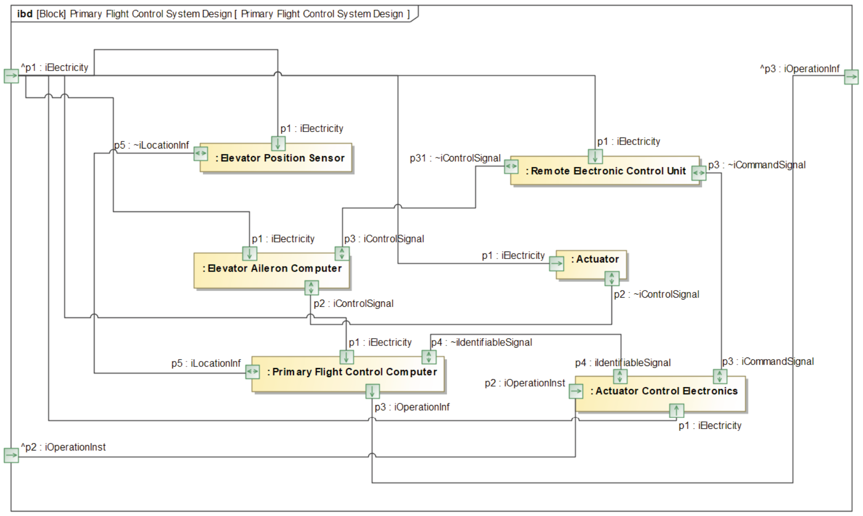


Fig. 13. The internal architecture of the Primary Flight Control System

Behavioral Modeling of Primary Flight Control System. After clarifying the subsystem structure model, it is also necessary to implement the subsystem behavior modeling, to ensure the integrity of the requirements-structure-behavior system design. At this stage, it is necessary to establish an independent behavior model for each subsystem. These models can be relatively abstract and not too detailed, but must completely describe the behavior characteristics and functions of each subsystem. After the behavior models of each subsystem are established, they need to be integrated to ensure the coordination between the various subsystems and the consistency of the system as a whole.

The following Fig. 14 is the activity diagram for the subsystem design, which assigns the activity elements of the main flight control system to their respective components, clarifying which system behaviors should be performed by which components. First, the actuator control electronic device receives the pitch command and converts it into a recognizable signal, which is sent to the main flight control computer for preliminary calculation. Finally, the elevator aileron computer drives the actuator to work according to the control signal. When the pitch adjustment is completed, data is transmitted back to the main flight control computer through the elevator position sensor, and the completed pitch signal is transmitted to the external system.

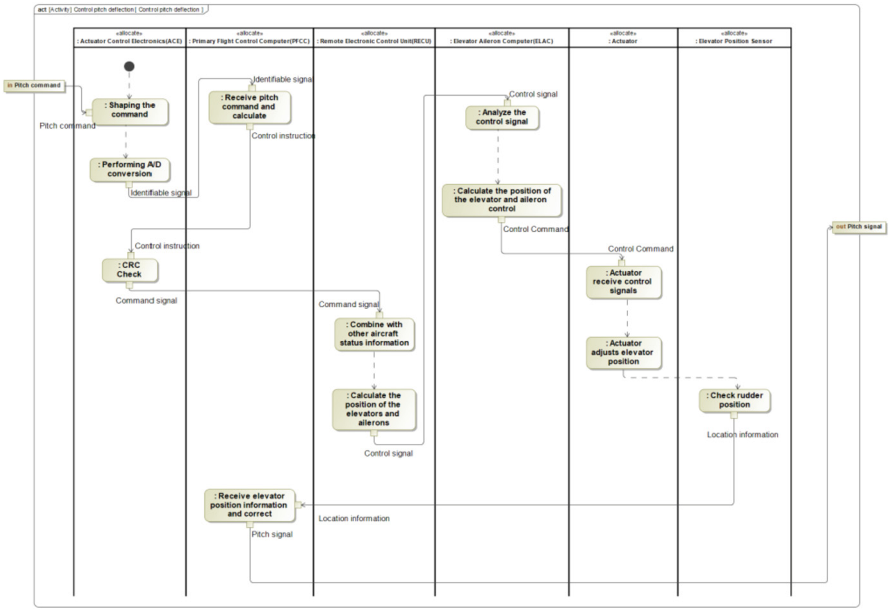

act [Activity] Control pitch deflection [Control pitch deflection]
«allocate» : Actuator Control Electronics(ACE)
«allocate» : Primary Flight Control Computer(PFCC)
«allocate» : Remote Electronic Control Unit(RECU)
«allocate» : Elevator Aileron Computer(ELAC)
«allocate» : Actuator
«allocate» : Elevator Position Sensor
in Pitch command
Pitch command
: Shaping the command
: Performing A/D conversion
Identifiable signal
: Receive pitch command and calculate
Control instruction
: CRC Check
Command signal
: Combine with other aircraft status information
: Calculate the position of the elevators and ailerons
Control signal
: Analyze the control signal
: Calculate the position of the elevator and aileron control
Control Command
: Actuator receive control signals
: Actuator adjusts elevator position
: Check rudder position
Location information
: Receive elevator position information and correct
Pitch signal
out Pitch signal

Fig. 14. The swimlane diagram of controlling pitch deflection

Based on the above subsystem activity diagram, it is possible to capture the state of the subsystem and the transition relationships between these states, in order to establish a SysML state machine that describes the changes in the subsystem's operational states. As shown in Fig. 15, the state model of the primary flight control system. Initially, when the system receives the pitch command, it will enter the conversion state. Through the conversion, the pitch command becomes an identifiable signal. It then proceeds through the calculation state and execution state. When the execution action is completed, the system enters the check state to determine whether the altitude requirement has been met. If the required altitude is not reached, the system will continue executing until the it is satisfied. Finally, when the pitch command is completed, the system enters the maintenance state to maintain the current flight attitude.

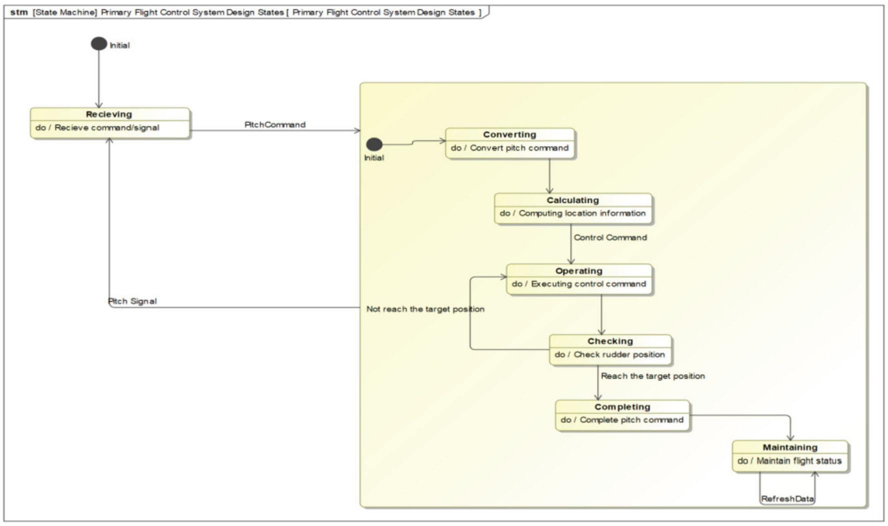

stm [State Machine] Primary Flight Control System Design States [Primary Flight Control System Design States]
Initial
Recieving
do / Recieve command/signal
PitchCommand
Initial
Converting
do / Convert pitch command
Calculating
do / Computing location information
Control Command
Operating
do / Executing control command
Pitch Signal
Not reach the target position
Checking
do / Check rudder position
Reach the target position
Completing
do / Complete pitch command
Maintaining
do / Maintain flight status
RefreshData

Fig. 15. The state diagram of thePrimary Flight Control System

Subsystem-Level Requirements and Traceability. As the refinement and concretization of system requirements, subsystem-level requirements defines the requirements that the subsystem must meet in order to satisfy the system requirements. They serve as the basis for realizing system requirements and the foundation for component requirements at the next level. As shown in the Fig. 16, there are 5 obtained functional requirements about the primary flight control system.

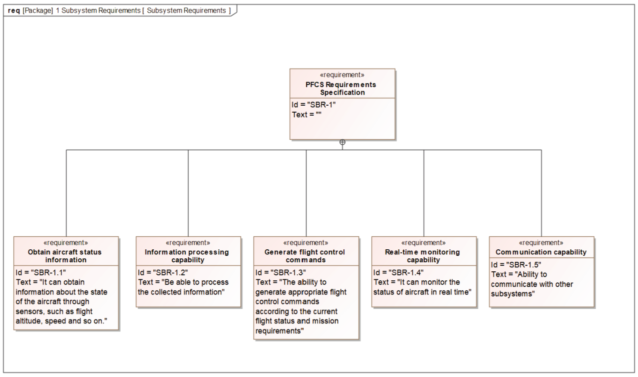
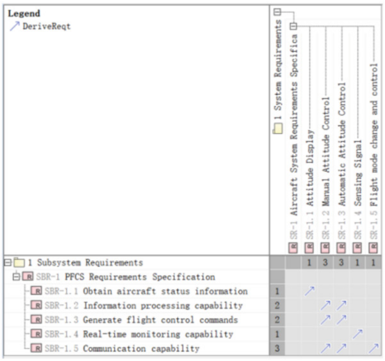
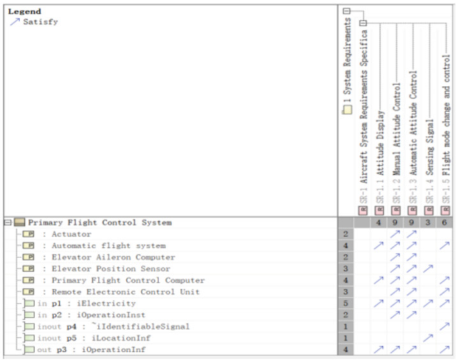

req [Package] 1 Subsystem Requirements [Subsystem Requirements]
«requirement» PFCS Requirements Specification
Id = "SBR-1"
Text = ""
«requirement» Obtain aircraft status information
Id = "SBR-1.1"
Text = "It can obtain information about the state of the aircraft through sensors, such as flight altitude, speed and so on."
«requirement» Information processing capability
Id = "SBR-1.2"
Text = "Be able to process the collected information"
«requirement» Generate flight control commands
Id = "SBR-1.3"
Text = "The ability to generate appropriate flight control commands according to the current flight status and mission requirements"
«requirement» Real-time monitoring capability
Id = "SBR-1.4"
Text = "It can monitor the status of aircraft in real time"
«requirement» Communication capability
Id = "SBR-1.5"
Text = "Ability to communicate with other subsystems"

Fig. 16. The funtional requirements for the Primary Flight Control System

In addition, the traceability from subsystem requirements to system requirements can be established, that is, to determine which subsystem requirements are derived from which system requirements, as well as the traceability between subsystem requirements and solution prototype elements. As shown in Fig. 17, the left part of the picture shows the traceability relationship between subsystem requirements and system requirements where the horizontal rows represents subsystem requirements and the vertical columns represents system requirements. Similarly, the right part of the picture shows the traceability relationship between system requirements and subsystem elements. In this case, the horizontal rows represent the model elements of the subsystem, and the vertical columns represent the system requirements.


Legend
↗ DeriveReqt

1 System Requirements		SR-1 Aircraft System Requirements Specifica	SR-1.1 Attitude Display	SR-1.2 Manual Attitude Control	SR-1.3 Automatic Attitude Control	SR-1.4 Sensing Signal	SR-1.5 Flight mode change and control
1 Subsystem Requirements			1	3	3	1	1
SBR-1 PFCS Requirements Specification							
SBR-1.1 Obtain aircraft status information	1		↗				
SBR-1.2 Information processing capability	2			↗	↗		
SBR-1.3 Generate flight control commands	2			↗	↗		
SBR-1.4 Real-time monitoring capability	1					↗	
SBR-1.5 Communication capability	3			↗	↗		↗


Legend
↗ Satisfy

1 System Requirements		SR-1 Aircraft System Requirements Specifica	SR-1.1 Attitude Display	SR-1.2 Manual Attitude Control	SR-1.3 Automatic Attitude Control	SR-1.4 Sensing Signal	SR-1.5 Flight mode change and control
Primary Flight Control System			4	9	9	3	6
: Actuator	2			↗	↗		
: Automatic flight system	4		↗	↗	↗		↗
: Elevator Aileron Computer	2			↗	↗		
: Elevator Position Sensor	3			↗	↗	↗	
: Primary Flight Control Computer	4		↗	↗	↗		↗
: Remote Electronic Control Unit	3			↗	↗		↗
in **p1** : iElectricity	5		↗	↗	↗	↗	↗
in **p2** : iOperationInst	2			↗	↗		
inout **p4** : ~iIdentifiableSignal	1						↗
inout **p5** : iLocationInf	1					↗	
out **p3** : iOperationInf	4		↗	↗	↗		↗

Fig. 17. The traceability for the Subsystem-level requirements and elements

4.2 Component-Level Modeling

Definition of components. After the requirements at the subsystem level are completed, enter the component layer. First, we need to establish the structure model of the main flight control system. Similar to the flight control system model, all components are specified in a single hierarchical structure. As shown in Fig. 18.

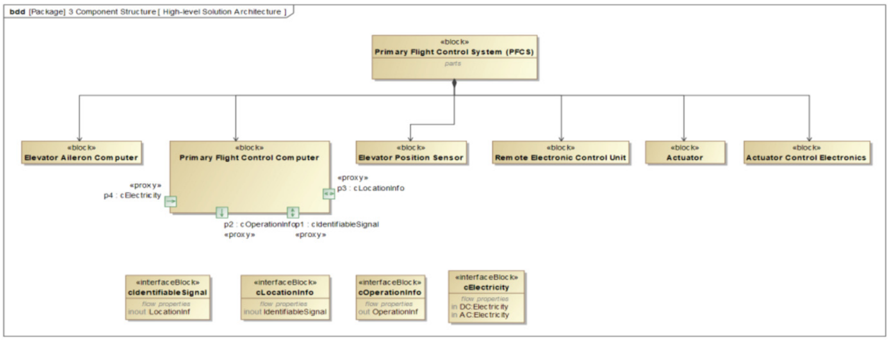

bdd [Package] 3 Component Structure [High-level Solution Architecture]
«block» Primary Flight Control System (PFCS)
parts
«block» Elevator Aileron Computer
«block» Primary Flight Control Computer
«block» Elevator Position Sensor
«block» Remote Electronic Control Unit
«block» Actuator
«block» Actuator Control Electronics
«proxy» p4 : cElectricity
«proxy» p3 : cLocationInfo
p2 : cOperationInfo «proxy»
p1 : cIdentifiableSignal «proxy»
«interfaceBlock» cIdentifiableSignal
flow properties
inout LocationInf
«interfaceBlock» cLocationInfo
flow properties
inout IdentifiableSignal
«interfaceBlock» cOperationInfo
flow properties
out OperationInf
«interfaceBlock» cElectricity
flow properties
in DC:Electricity
in AC:Electricity

Fig. 18. The High-level Solution Architecture of the Primary Flight Control System

Component structure modeling. When the function allocation is completed within the primary flight control system, its internal solution structure can be studied. This paper selects the main flight control computer of the main flight control system as the research object. According to the required functions, five components were mainly defined, including the input/output interface, central processing unit (CPU), controller memory and sensor data filter. The connection relationships between the components and the port information of each component were also defined, as shown in the Fig. 19. Four interaction ports between the main flight control computer and external components were

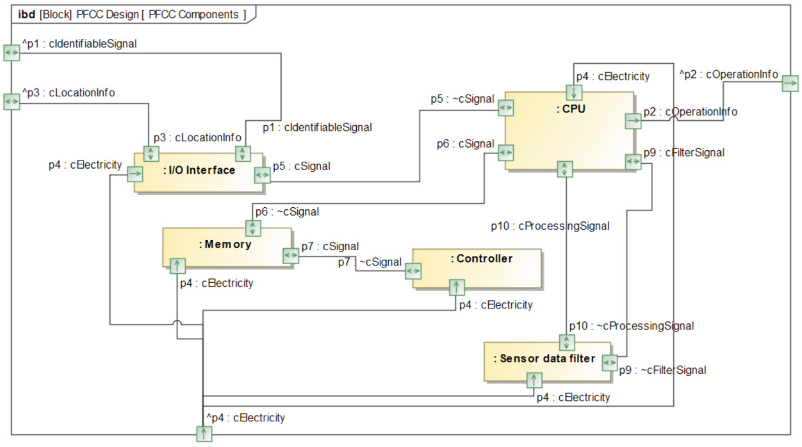


Fig. 19. The internal architecture for the Primary Flight Control Computer

defined, including the identifiable signal received by the main flight control computer; the position information received by the main flight control computer; the power supply provided by the power unit,and the control signal that needs to be sent to the remote electronic control device when the main flight control computer completes execution.

Component Behavior Modeling

After the structure model of the main flight control computer is clarified, behavioral modeling should be carried out. Figure 20 shows the activity diagram. Firstly, the input/output interface receives the identifiable signal and passes it to the central processing unit (CPU). The CPU reads the signal and stores it in the Memory. The Controller monitors the data in the Memory, and when new pitch signal data is detected in Memory, the Controller will send a signal to notify the CPU. The CPU reads the latest pitch signal data from Memory according to the signal from Controller, and inputs it into the Sensor Data Filter for filtering. The signal data processed by the Sensor Data Filter is then passed back to the CPU for calculation, and finally generate corresponding control command.

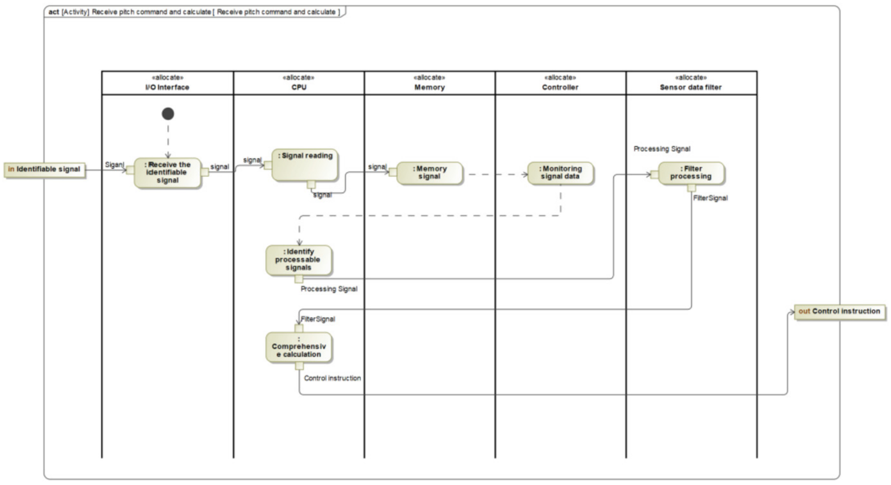
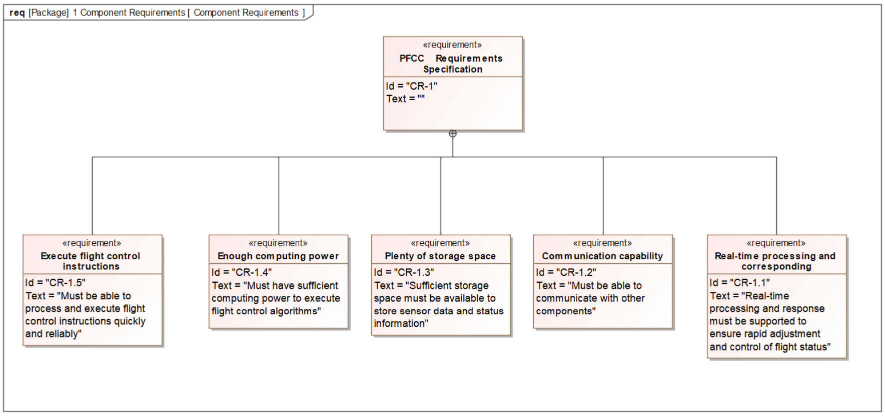


Fig. 20. The Swimlane diagram of recieving pitch command

Component-Level Requirements and Traceability. Component requirements, as the specificization of subsystem requirements at the component level, can enable developers to better understand and implement the system. According to the analysis of the main flight control computer structure and behavior model, component-level requirements can be obtained, as shown in Fig. 21.


req [Package] 1 Component Requirements [Component Requirements]

«requirement»
PFCC Requirements Specification
Id = "CR-1"
Text = ""

«requirement»
Execute flight control instructions
Id = "CR-1.5"
Text = "Must be able to process and execute flight control instructions quickly and reliably"

«requirement»
Enough computing power
Id = "CR-1.4"
Text = "Must have sufficient computing power to execute flight control algorithms"

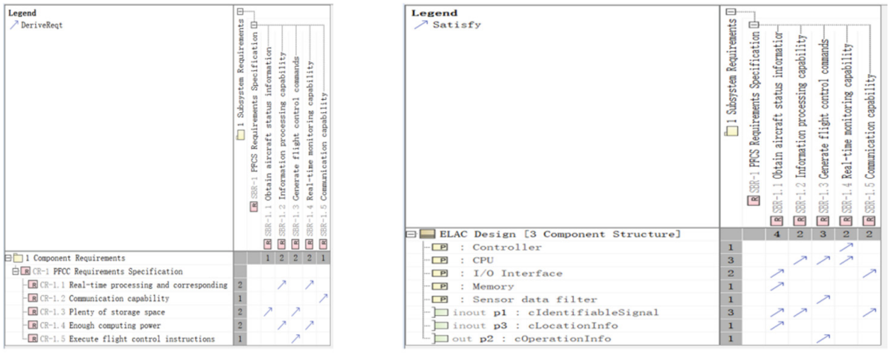
«requirement»
Plenty of storage space
Id = "CR-1.3"
Text = "Sufficient storage space must be available to store sensor data and status information"

«requirement»
Communication capability
Id = "CR-1.2"
Text = "Must be able to communicate with other components"

«requirement»
Real-time processing and corresponding
Id = "CR-1.1"
Text = "Real-time processing and response must be supported to ensure rapid adjustment and control of flight status"

Fig. 21. The functional requirements for the Primary Flight Control Computer

Similarly, the traceability relationship between component requirements and subsystem requirements and the traceability relationship between subsystem requirements and component elements can be established, as shown in Fig. 22.



Legend
↗ DeriveReqt

		SBR-1 PFCS Requirements Specification	SBR-1.1 Obtain aircraft status information	SBR-1.2 Information processing capability	SBR-1.3 Generate flight control commands	SBR-1.4 Real-time monitoring capability	SBR-1.5 Communication capability
1 Component Requirements			1	2	2	2	1
CR-1 PFCC Requirements Specification							
CR-1.1 Real-time processing and corresponding	2			↗		↗	
CR-1.2 Communication capability	1						↗
CR-1.3 Plenty of storage space	2		↗		↗		
CR-1.4 Enough computing power	2			↗		↗	
CR-1.5 Execute flight control instructions	1				↗		

Legend
↗ Satisfy

		SBR-1 PFCS Requirements Specification	SBR-1.1 Obtain aircraft status informatior	SBR-1.2 Information processing capability	SBR-1.3 Generate flight control commands	SBR-1.4 Real-time monitoring capability	SBR-1.5 Communication capability
ELAC Design [3 Component Structure]			4	2	3	2	2
: Controller	1					↗	
: CPU	3			↗	↗	↗	
: I/O Interface	2		↗				↗
: Memory	1		↗				
: Sensor data filter	1				↗		
inout p1 : cIdentifiableSignal	3		↗	↗			↗
inout p3 : cLocationInfo	1		↗				
out p2 : cOperationInfo	1				↗		

Fig. 22. The traceability for the Component-level requirements and elements

Through the solution domain process of MagicGrid, the high-level system requirements have been successfully transformed into implementable component-level requirements, and traceability relationship has been established between each layer of requirements to ensure that the source of each requirement can be traced, ensuring the accuracy of requirements, and providing clear guidance and assurance for the subsequent developers.

5 Discussion

Through the description of the MagicGrid method and its application in the actual requirements development process, it can be found that the MagicGrid method can produce good results in the requirements development process of complex systems; it can deeply understand and analyze the needs of stakeholders, and it is clear Requirements representation can also help team members better understand and collaborate, track and manage requirements more accurately; they can also have a clearer understanding of the development system for the layered thinking of the problem domain and solution domain raised by it, greatly improve development efficiency.

Although the MagicGrid method has advantages over traditional requirement development methods in some aspects, there are still some drawbacks. One of the main problems is consistency, because the MagicGrid method is based on text and may suffer from semantic ambiguity and inconsistency. For example, different people may use different vocabulary to describe the same requirement, which may result in duplicated or unclear requirements in the model.

To address this issue, a synonym dictionary can be defined to map different expressions to the same concept, thus avoiding inconsistency caused by different expressions. For example, when processing requirements for controlling an plane's height, different expressions such as "controlling the plane attitude angle", "controlling the plane pitch angle", and "controlling the plane climb and descent angle" can be mapped to the same concept, such as "controlling the pitch angle". This can eliminate semantic ambiguity and inconsistency and better understand and process requirements. In building a synonym dictionary, techniques such as corpus and word vectors can be used to match and map words or phrases with similar semantics (Fig. 23).

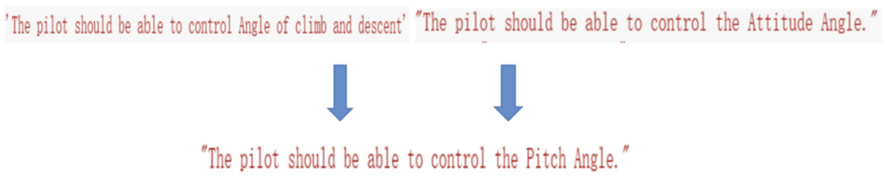
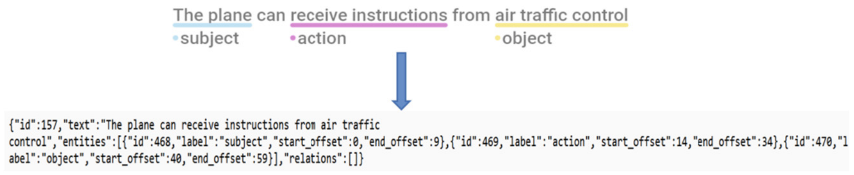

'The pilot should be able to control Angle of climb and descent'
"The pilot should be able to control the Attitude Angle."
"The pilot should be able to control the Pitch Angle."

Fig. 23. Synonym replacement

Another approach is to use named entity recognition (NER) to identify professional terms, adopt a standardized terminology, and maintain it jointly by users and developers to ensure consistency of understanding. This can effectively eliminate the problem of duplication or ambiguity of requirements caused by different people using different words, and improve the accuracy and reliability of the model. For example, for the requirement "the aircraft transmits tower instructions to the pilot," NER technology can identify the subject (plane), object (pilot), and action (transmit tower instructions) in the requirement, and form a structured description to express the content and object of the requirement more clearly.

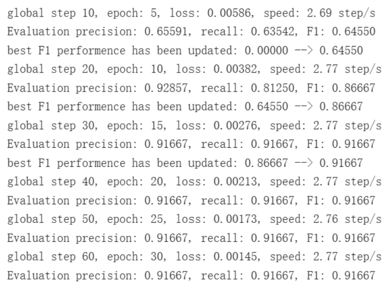
First, a certain amount of requirements data needs to be sorted out and labeled to establish a training set. Labeling requirements can use methods such as markup language to mark different parts of the requirements (such as subjects, objects, actions, etc.) separately (Fig. 24).



```
{"id":157,"text":"The plane can receive instructions from air traffic
control","entities":[{"id":468,"label":"subject","start_offset":0,"end_offset":9},{"id":469,"label":"action","start_offset":14,"end_offset":34},{"id":470,"l
abel":"object","start_offset":40,"end_offset":59}],"relations":[]}
```

Fig. 24. Label the data

Next, named entity recognition is used to train on the labeled data to learn how to recognize different entity types and their positions in sentences. During the training process, the accuracy of the model can be continuously optimized by adjusting algorithm parameters and increasing training data (Fig. 25).



```
global step 10, epoch: 5, loss: 0.00586, speed: 2.69 step/s
Evaluation precision: 0.65591, recall: 0.63542, F1: 0.64550
best F1 performence has been updated: 0.00000 --> 0.64550
global step 20, epoch: 10, loss: 0.00382, speed: 2.77 step/s
Evaluation precision: 0.92857, recall: 0.81250, F1: 0.86667
best F1 performence has been updated: 0.64550 --> 0.86667
global step 30, epoch: 15, loss: 0.00276, speed: 2.77 step/s
Evaluation precision: 0.91667, recall: 0.91667, F1: 0.91667
best F1 performence has been updated: 0.86667 --> 0.91667
global step 40, epoch: 20, loss: 0.00213, speed: 2.77 step/s
Evaluation precision: 0.91667, recall: 0.91667, F1: 0.91667
global step 50, epoch: 25, loss: 0.00173, speed: 2.76 step/s
Evaluation precision: 0.91667, recall: 0.91667, F1: 0.91667
global step 60, epoch: 30, loss: 0.00145, speed: 2.77 step/s
Evaluation precision: 0.91667, recall: 0.91667, F1: 0.91667
```

Fig. 25. The Process of Training Labeled Data

After the training is completed, entity information can be extracted from the requirement text by referencing the optimal model. For example, the requirement "The plane transmits tower instructions to the pilot" can extract entities "The palne", "the pilot" and "transmit tower instructions" and form a structured description, such as "the plane transmits tower instructions to the pilot". This structured description can express the content and object of the requirement more clearly, avoid unnecessary ambiguity and inconsistency, and improve the accuracy and reliability of the model (Fig. 26).

[{"subject": [{"text": "The plane", "probability": 0.9978847537644242}], "object": [{"text": "the pilot", "probability": 0.9992364275358696}], "action": [{"text": "transmit tower instructions", "probability": 0.9757586922001451}]}]

Fig. 26. The result of Named Entity Recognition

6 Conclusion

Requirements development, as the foundation of the aircraft development, runs through the entire life cycle of aircraft development. However, traditional requirements development method is based on text, which has problems such as incomplete information expression and ambiguity. Therefore, this paper proposes to introduce the requirements development method based on MagicGrid into the aircraft system design process. This paper demonstrates the specific process of using MagicGrid for requirements development in the flight control system example, successfully obtaining system-level, subsystem-level and component-level requirements, and establishing traceability to achieve requirement traceability. However, this method also has problems of some redundant and ambiguous requirements. Therefore, this paper proposes some improvement ideas, but it still needs further improvement and integration with the MagicGrid method in the future work, in order to better apply the MagicGrid method to solve the problrms in the aircraft system requirement development process and improve the quality of requirement development.

References

1. Chen, Y., Tian, B., Liu, Z., Xie, Y.: Philosophical thoughts on some basic concepts of civil aircraft development system engineering. Civil Aircraft Des. Res. **03**, 35–41 (2017)
2. Deng, Z.: Overview of the application of requirements engineering in civil aircraft life cycle project management. Technol. Vision **26**, 255 (2015)
3. Wu, Y., Liu, J., Zheng, D.: Analysis of model-based systems engineering technology. Aviat. Sci. Technol. **9**, 69–73 (2015)
4. Zhang, B., Qi, F., Xing, T., Liu, Y., Wang, W.: Research and practice of model-based manned spacecraft development method. Aeronaut. J. **41**(7), 72–80 (2020)
5. Estefan, J.A.: Survey of model-based systems engineering (MBSE) methodologies. Incose MBSE Focus Group **25**(8), 1–12 (2007)
6. Morkevicius, A., Aleksandraviciene, A., Armonas, A., Fanmuy, G.: Towards a common systems engineering methodology to cover a complete system development process. In: INCOSE International Symposium, vol. 30, no. 1, pp. 138–152 (2020)
7. Wang, L., Zhan, C.: Modeling and analysis of civil aircraft top level RNP system architecture. In: Journal of Physics: Conference Series, vol. 1827, no. 1, p. 012118. IOP Publishing (2021)
8. Lu Z, Liu X, Mao Y, Fan H, Zhao Y.: Application of model-based system engineering method in satellite overall design. Spacecraft Eng. (03), 7–16 (2018)
9. Weihao, L., Yuqiang, G., Qipeng, C., Hui, Z.: Model-based system engineering adoption for trade-off analysis of civil helicopter fuel supply system solutions. In: Krob, D., Li, L., Yao, J., Zhang, H., Zhang, X. (eds.) Complex Systems Design & Management, pp. 311–323. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-73539-5_24
10. Morkevicius, A., Aleksandraviciene, A., Krisciuniene, G.: From UAF to SysML: transitioning from system of systems to systems architecture. In: INCOSE International Symposium, vol. 31, no. 1, pp. 585–598 (2021)
11. Cui, Z., et al.: MBSE for civil aircraft scaled demonstrator requirement analysis and architecting. IEEE Access **10**, 43112–43128 (2022)
12. Herfman, H.: Model-Based System Engineering Best Practices. Aviation Industry Press, Beijing (2014)
13. Yang, M., Qian, X., Wang, J.: Research on capture method of civil aircraft operational requirements based on DoDAF modeling. J. Armament Eng. (06), 81–85 (2018)