ICAS 2021 SHANG HAI
International Council of the Aeronautical Sciences
32nd Congress of the International Council of the Aeronautical Sciences
September 6-10, 2021
Pudong Shangri-La, Shanghai, China

AN AGILE PRACTICE OF AVIONICS SYSTEM DEVELOPMENT

Lei Dong[1], Wen-Ming Song[1]

[1]China Aeronautical Radio Electronics Research Institute

Abstract

Model based systems engineering is the critical approach of future system engineering for coping with complexity. Key characteristics of high integration of physical domains, complex functional domains, and deep coupling of logical fields make the complexity of avionics systems continuously increasing. As the core of various aircraft platforms, avionics systems must adapt to the evolutionary acquisition and incremental development business mode. An Agile method focused on Incremental and parallel design, the balance between iteration and schedule, continuous integration and verification is applied to cope with system complexity and improve the quality of systems design in this paper. An interaction complexity management method is introduced to enhance system complexity management. A distributed integration and validation strategy and simulation environment suitable for adapting to the agile pattern are presented. The practice of an avionics system is presented to illustrate the agile method, and customized steps are applied based on the project's specific needs.

Keywords: MBSE; Avionics System; Agile; Collaborative Design; Distributed Simulation;

1. Introduction

Traditionally, a document-based approach was performed to the systems engineering. Information of systems such as requirements and specifications was often conveyed and transmitted through documents as text or graphics in the system lifecycle [1]. The completeness, consistency, and relationships between requirements, design, analysis, and verification are challenging to assess, making traceability and change impact assessments difficult.

The complexity of systems has increased significantly in recent years, and the increase of complexity will continue dramatically in the future [2]. The complexity is reflected directly in the number of requirements, interactions, functions, components, and the mutual dependencies between these system elements. Complex systems are introduced in system engineering with the complexity increasing. The nature of complex systems is quite different from traditional non-complex systems. System complexity which poses a continuous challenge to systems engineering, extends both in numbers and dependency of all kinds of system elements. The traditional approach is no longer sufficient to manage complexity.

Model based systems engineering is the critical approach of future system engineering for coping with complexity. The difference between model based systems engineering and traditional systems engineering is that models are the prime artifacts of systems engineering [3, 4]. The ever-growing system complexity is address with the system models, which are based on different levels of abstraction of systems. Virtual verification and validation are supported by multi-disciplinary analysis with continuously refined models. Early detection of defects could be realized, and design quality can be improved. Consistency among documents could be guaranteed based on a model-generated approach.

To adapt to the highly changing operational environments and threats [5], the informatization, networking, and systematization levels are significantly increasing with the development of avionics systems. These trends make system complexity increasing as any other complex system. As the critical system for various aircraft platforms to accomplish mission objectives, high integration of physical domains, complex functional domains, and deep coupling of logical fields are key characteristics of advanced avionics systems. These key characteristics make the complexity of the avionics system, which in terms of the continuous increase of requirements, system functions, interactions and number of components, and internal/external interactions is accelerating. With the

acceleration of technology updating, especially the rapid growth in information technology, the system's complexity and comprehensiveness are significantly improved. This acceleration leads to the increase of system engineering activities and the intensification of difficulties. Meanwhile, avionics systems need to adapt to the evolutionary acquisition and incremental development business mode. It is urgent to develop an agile approach to accelerate innovation and improve the quality and efficiency of avionics system design.

An agile and collaborative MBSE method is illustrated and applied to the development of an avionics system according to the needs of supporting virtual verification and validation, rapid selection of solutions, and iteration of system design. Complexity management and incremental development/agile iteration are also introduced in the new method in this paper.

2. Collaborative Agile MBSE Method

2.1 Overview of the Method

An Agile method is applied to cope with system complexity and improve the quality of systems design. Unlike the agile pattern in software development, the agile method in avionics systems mainly provides a flexible iteration approach that focuses on incremental and parallel design, a balance between iteration and schedule, continuous integration and verification in the concept and development stage of the system lifecycle. Iterations are executed inside each process, rather than just a single process. The agile method allows for a more flexible working approach that is more realistic than traditional V models. It is possible that the downstream design work is feedbacked to the requirements process and contributes to the formal requirements.

The Harmony aMBSE [6] and SYSMOD [7] are combined as the framework of the method. Additional activities and customization, such as effectively incorporating existing knowledge and interaction complexity management, are introduced in this paper. The method covers most technical processes in the concept and development stage and parts of the technical management processes of ISO15288 [8]. The agile method consists of planning for the project, stakeholder requirements analysis, system requirements analysis, architectural analysis, architectural design, integrate and validate the model, interaction complexity management, model quality management, and handover, as shown in Figure 1. SysML diagrams are created flexibly based on the analysis need. There are no strict rules on what kind and order of diagrams should be used for performing the given analysis.

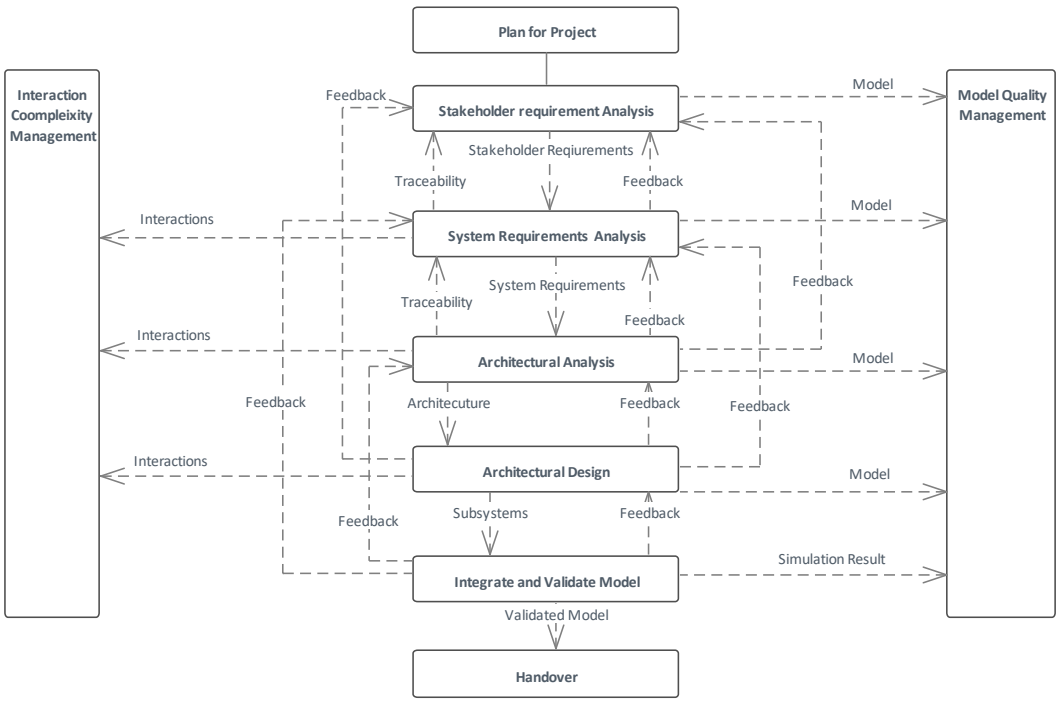

Figure 1 – Overview of the agile method

2.2 Planning for Project

The project should be well planned to provide a good foundation for collaborative and agile development. Conflict and omission may emerge due to the lack of planning. Project planning includes:

Set up and maintain the modeling environment: This process provides a suitable modeling environment to perform the practice. The modeling tool should be configured and embed into the project for the implementation of MBSE.

Set up modeling standards and guidelines: models are commonly collaboratively developed by multiply engineers. A minimal set of rules for using SysML constructs and how to organize the model should be established. All model builders should comply with the same specification to ensure that the final models are consistent and meet the requirements of the project and document generation.

Provide MBSE training: requisite training that including MBSE methodology, SysML, modeling tool tutorial, and modeling standards is indispensable to ensure that assistance for applying the MBSE methodology is provided in the project.

Tailor the MBSE method: tailor the given MBSE method to the specific need of the project based on constraints and modeling purposes. The MBSE activities must accomplish the purpose and meet the cost and schedule constraints of the project.

Define glossary: model elements such as actors, use cases, system architecture, and domain-specific terminology are commonly shared in the project. These terminologies should be maintained in a glossary for consistency in the whole project.

Set up model: considering many different engineering aspects may end up in a very confusing way because of an unclear organization of the model elements. Therefore, properly organizing the model is crucial for the understanding and usability of the model. A well-organized model project could make the model more normative and much easier for management and collaboration. Model structures such as packages with different purposes, model libraries, profiles should be well defined in the model.

2.3 Stakeholder Requirements Analysis

Stakeholder requirements are captured and elicited by use cases [9]. Use cases are conceptualized within the general object-oriented paradigm. Use case modeling starts from identifying actors that interact with the system and the goals or objectives the system must fulfill from an outside perspective. These goals or objectives are captured with the guidance of life cycle concepts, including acquisition, deployment, operational, support, and retirement concepts. System contexts are created with BDDs and IBDs to illustrate the system environments, stakeholders, and their connections. Actors are identified from system environments and related stakeholders.

Functional boundaries of the system are well-defined by use cases with the observable interactions of the system. Use Cases describe the interaction between the actor and the system and related to a particular goal of the actors. Sequence diagrams are introduced to capture these interactions for the elicitation of stakeholder requirements. The interactions focus on the stakeholder's goals and the quality and constraints of these goals, and all interactions serve the purposes of the stakeholders. Include relationship (in the situation that different use cases invoke the same system functions) and extend relationships are introduced to organize the structure of use cases.

Associations between use cases should be avoided to keep the independent of use cases, although non-associations are unachievable due to the system complexity. A further mechanism is discussed latterly in complexity management to eliminate the ambiguity between use cases in these exceptions.

Existing requirements and actors may be imported as sources of the given information and related to use cases. Completeness and consistency analysis should be applied due to there may be incomplete, inconsistent between these requirements.

2.4 System Requirements Analysis

System requirements analysis continues based on the stakeholder requirements analysis. System requirements analysis focuses on the reaction of the system of the use cases while stakeholder requirements focus on the external actors. The main goal of system requirements analysis with the

use cases is to identify and define the system functions and the external functional interfaces.

Notice that the use case is a specification of behavior. The implementation of the behavior is modeled using behavior diagrams with different aspects such as activity-based, interaction-based and state-based diagrams in functional analysis. Blocks with the stereotype of "Use Case" are applied to represent use cases and for further analysis.

Top-level functional requirements are identified from observable interactions of the use cases. Activities that decompose the top-level functions are commonly directly used to describe the detailed behaviors of use cases. Alternatively, sequence diagrams, state machines are used to describe use case detailed behaviors. Interactions are modeled as asynchronous events; thus, functional interfaces could be identified and characterized based on these interactions. IBDs are used to formalize the interfaces from the consistency between structure and behaviors, and ports are used to elicit the interactions points. Contracts of the ports are captured as "provided" and "required" services defined by SysML element "Interface" or "Interface Block."

Functional and interface requirements that both from existing requirements libraries and newly identified will be grouped into use cases and traced back to stakeholder requirements. System functions are modeled with stereotyped block "functional" to illustrate the hierarchy and functional interfaces. These elements are presented in BDDs and IBDs to perform a further functional architectural analysis and optimization. Black boxed view state machines are developed by considering events from the external environments that cause state change and execute functions described in the activity diagram to the external environments. The completeness, accuracy, correctness, and consistency of the Functional and interface requirements are analyzed and verified by the execution of the state machines to perform an early validation of stakeholder requirements. Parametric diagrams are used for the nonfunctional analysis, while not all the requirements could be related to use cases and could be analyzed with the state machine.

2.5 Architectural Analysis

Several reasonable candidate architecture solutions could meet the requirements in the design space. Architectural choices are made based on the architectural analysis. A trade study is used to compare all the solutions. Trade study could be multiple aspects or single aspect. A set of assessment criteria will be defined to perform the architectural trade study. The assessment criteria include many aspects that the system could optimize. Typical criteria might consist of cost, development time, power, the cohesiveness of functions, performance (such as latency), size, reliability, safety, reusability, etc.

Criteria computation follows after the criteria definition. Here take coupling Matrices as a case example: functional allocation between functions and subsystems for each candidate solution is performed to generate the N^2 diagram, coupling values are then calculated.

Since not all criteria are equally important, weights are assigned to the criteria for the trade study. An objective function is defined based on MOEs. MOEs are then assigned to candidate solutions to computing the objective function. The selected solution is then the candidate solution which resulted in the highest MOE score among the evaluated candidates. BDDs with constraint blocks and parametric diagrams are used to create an analysis context and perform the evaluation.

2.6 Architectural Design

The architectural design focuses on the subsystem, their functionalities, interfaces, and how subsystems accomplish system functionalities by collaboration. Subsystems are identified from the base architecture selected in the architectural analysis. System functions that represent system behaviors should be clearly allocated to subsystems. Interfaces between subsystems should be identified based on the allocation. External interactions of the system should be allocated to subsystems, and interfaces between system and actors should be updated as interfaces between subsystems and actors. System data in terms of block properties also should be allocated to subsystems. All these features must be decomposed into subsystem-level features that trace back to the system-level source feature. New features are derived when the direct allocation cannot be performed.

Block definition diagrams will be used to show the subsystem's types of ports interfaces and properties. A common approach to performing function allocation is to refine the activity diagrams from use case analysis into activity diagrams with swim lanes representing the different subsystems. For the interface specification, interfaces between subsystems should be added when control flows cross between different swim lanes. Another approach is to perform the refinement of sequence diagrams with precise interactions between subsystems. Interfaces and connections of subsystems then could be created from these interactions and presented with IBDs. Subsystem state machines are synchronized to provide a global view of all the functionalities of the subsystems after the allocation and interface definition of the subsystems.

2.7 Integrate and Validate Model

System models are integrated and validated after the models are built to support the validation of the system requirements via simulation. A distributed integration and validation strategy is considered to adapt to agility. Several factors influence the decision of strategy.

1. Performance: the scale of the complex system models often become huge; when a model is considerably large and involves many objects, the single model would take a long time to simulate and limit the amount of experimentation performed in the project, simulation delay would arise with an interactive simulation of state machines.
2. Cooperation: the design of complex systems requires the cooperation of several teams from different domains across the whole process. Models are distributed naturally, which complies with the "divide and conquer" principle for complexity management.
3. Continuous Integration and Validation: A single integrated model means the project has to wait for the completion of the model building or the finish of integration. It is difficult for incremental development and continuous verification due to the lack of flexibility. The system model will be repeatedly integrated or modified with the iteration of system design for supporting changes.

A distributed simulation that could accelerate the delivery of integrated capabilities of the system models is particularly suitable for the models with asynchronous parallelism. Models could be linked together as parts of larger models and interact directly over a communications network via middleware. Advantages such as building independent, interoperable simulations could be facilitated.

2.8 Interaction Complexity Management

Managing complexity is the main challenge of complex systems development. Abstraction has long been used as a means to cope with complexity by encapsulating complexity with the idea of "divide and conquer". Interaction truly defines the behavior of an object from its boundaries. Well-organized interaction management would significantly enhance system complexity management. Interaction complexity is managed based on the consistency of the model are considered. The SysML elements for addressing interaction are signals and flow. Since interactions are mostly modeled with signals, interactions complexity management is illustrated with the signal base interaction in the following chapters. Interactions that are based on flow are processed in the same way.

The well-defined boundaries of objects would enhance a distinct interaction identification and better interaction complexity management. A system consists of a set of elements that interact with one another. To be compatible with this thought, the object in the system domain is a generalized concept. Objects could be functional objects (such as use cases, activities, functions, and signals), logical components or physical components, and so on.

As shown in Figure 2, interactions are classified into four categories based on their origination: interactions from system boundary, interactions from use cases boundary, interaction from activities boundary, and interactions from subsystems allocation. Consistency is kept by a vertical traceability approach from the requirements analysis process to the architectural design process. It is vital to notice that these categories are not based on the final status of the interactions.

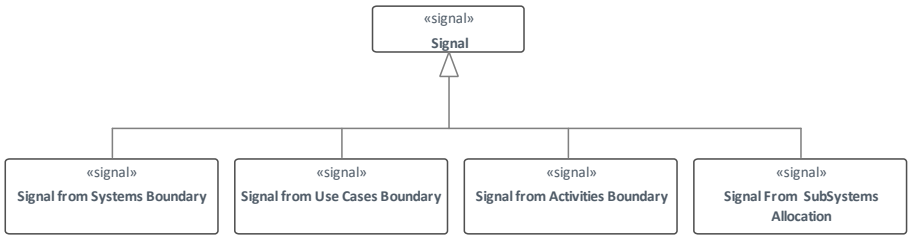

Figure 2 – Interaction categories with signals

Interactions from systems boundary and use cases boundary: interaction complexity issue arises from the ever beginning of use case identification. Use cases are used to capture system functionality. In other words, use cases represent functional characteristics of parts of systems. Two kinds of interactions exist in use cases: external interactions between system and actors and internal interaction of system between use cases. External interactions are characterized from system boundary with a common understanding, while the existence and identification of interactions between use cases need a further explanation.

Good use cases are independent from other use cases to allows independent analysis. What should be noticed is that the ideal complete independence of use cases may not be achieved due to the system complexity. The ideal complete independence means that the system functionalities are combined rather than integrated organically. A combination of functionalities implies that the system could be divided into unrelated systems to serve users' goals, which is noncompliance with the characteristic of complex systems. Association is inevitable between use cases when the actors get their value with the execution of the use cases. Boundaries between use cases are crystallized by the functional requirements related to use cases.

During the requirement analysis process, interactions between use cases are identified to eliminate ambiguity between use cases and ensure the coordination and consistency of use cases. A decoupling mechanism based on data definition and usage is introduced to illustrate the boundary of use cases and ensure consistency between use cases. Data between use cases are identified as the SysML elements of signals and flow schema. Traceability of these signals and flows are created in the lately functional analysis process (trace to activities) and architectural design process (trace to subsystems). Activities for keeping traceability of interactions between use cases are shown in Figure 3.

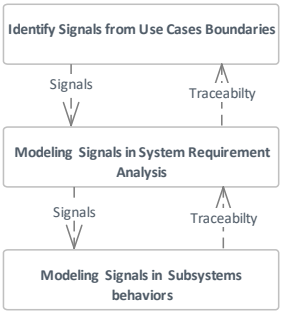

Figure 3 – Interaction traceability illustrated with interactions from use cases boundaries

Interactions from Activities boundary: activities as functional objects are relatively equal with the use case from the object-oriented perspective. Just as interactions between use cases, interactions between activities also exist within the same use case. These interactions would be identified with the functional analysis proceeding. The same tracing method is applied to these interactions.

Interactions From subsystems Allocation: interactions between subsystems emerge with the allocation of actions to subsystems in the architectural design process. Interactions between subsystems will be characterized by allocating activity partitions that can be used to perform the behavioral allocation.

Interactions from the system boundary are either providers or consumers, while the other categories of interactions have both provider and consumer roles inside the system. Four kinds of interactions

align their final states at the subsystem level in the architectural design process, as shown in Figure 4. Interactions from system boundaries eventually become interactions between subsystems and actors. Interactions from subsystems allocation eventually become interactions between subsystem; interactions from use cases boundary and activities boundary may become internal interaction of subsystem which means the provider and consumer of the signals is the same subsystem. Relationships between different classification methods of the signals are shown in Figure 5.

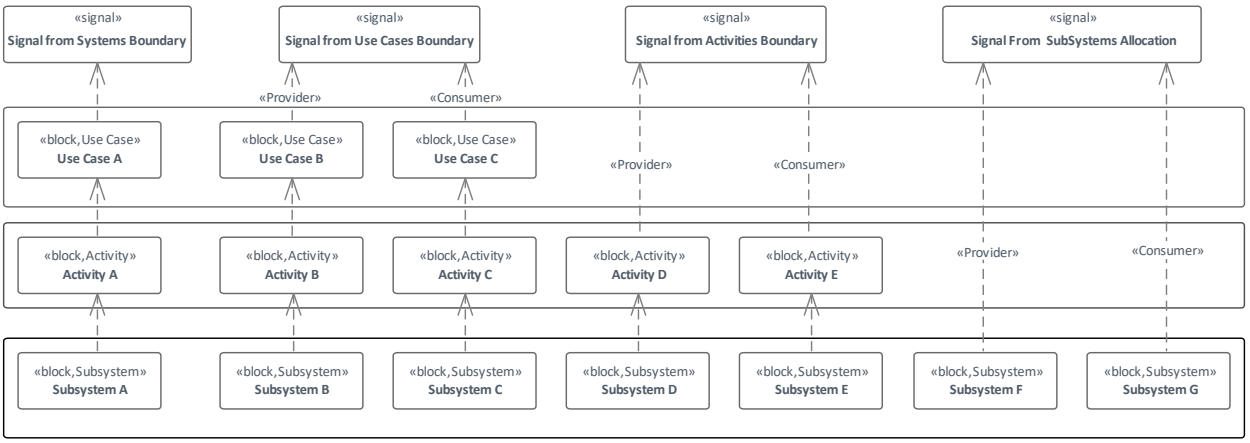

Figure 4 – Interaction traceability management

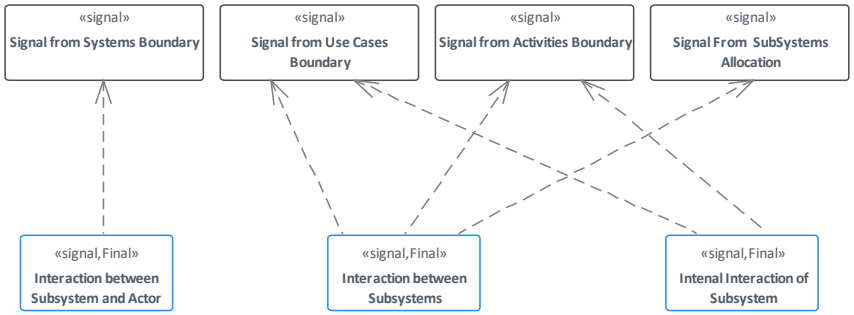

Figure 5 – Relationships between different classification methods

2.9 Model Quality Management

The review was the primary method to assure the design in traditional document-based system engineering, while model quality is the assurance of model-based design. The quality of models is affected by the knowledge and experience (both in the MBSE and avionics system domain) of modelers and the quality assurance techniques applied. Model quality criteria should be established to assess the quality of the models [10]. There are several criteria of model quality, such as correctness, completeness, consistency, complexity, changeability, etc. Correctness, completeness, and consistency are mainly considered in our method. Model quality is evaluated both by review and simulation and analysis of the model.

Correctness: the correctness includes grammar correctness and content correctness. Grammar correctness means that correct SysML elements and relations between these elements are created correctly without violating rules and conventions. Content correctness means that the model accurately represents the reality of the system.

Completeness: formal completeness and content completeness are considered. Formal completeness means that all the necessary model elements are built, such as properties, ports, connectors, interfaces, events, reception, states, transitions, actions, traceability. Content completeness means all the necessary information and logic of the system have been described without missing and with enough detail.

Consistency: some rules are built into SysML to ensure model consistency. For example, type consistency between different interfaces, the consistency between behaviors and structures,

consistency between different behavior diagrams.

Size and complexity metrics are introduced for the workload management of the IPT. Metrics targeting models are characterized by the number of associations, messages, state transitions, etc. The complexity of the describing model is measured by the state transition matrix in the state machine.

2.10 Handover

Once the subsystem specifications are ready, they must be handed off for the following engineering activities. Two kinds of handovers should be considered: from upstreaming design to the down steaming design, from the high-level system design to low-level system design. From the requirements view, the method should be requirements in and requirements out. For handover between upstream and downstream: all model elements as kinds of requirements are handed off to downstream for refinement, such as refining a logical interface to a physical interface. For handover between different levels of systems, high-level requirements are identified, analyzed, and validated, sub-requirements are generated based on the allocation, subsystem models together with the requirements should be handed over to next-level system designing.

3. The practice of avionics system development

The method proposed in the above chapter is applied to an avionics system practice in this part. An IPT is built for the collaborative and agile development of the avionics system and facilitating communication between team members. Customized design knowledge is introduced in the avionics system development based on the aircraft system design (such as requirements that hands over from aircraft design) and the existing avionics system design library.

3.1 Planning for Project Practice

The following activities are performed for the planning of the avionics system practice.

- Standards including naming of modeling elements style, grammar and diagram layout are established for the guidance of teamwork, as shown in Figure 6.
- Functional architecture analysis described in system requirement analysis is tailored due to the schedule constraint in practice.
- The glossary that both covers MBSE knowledge and avionics systems knowledge are created for the whole project.
- Model frame with the organization of packages is set up for a unified model structure.

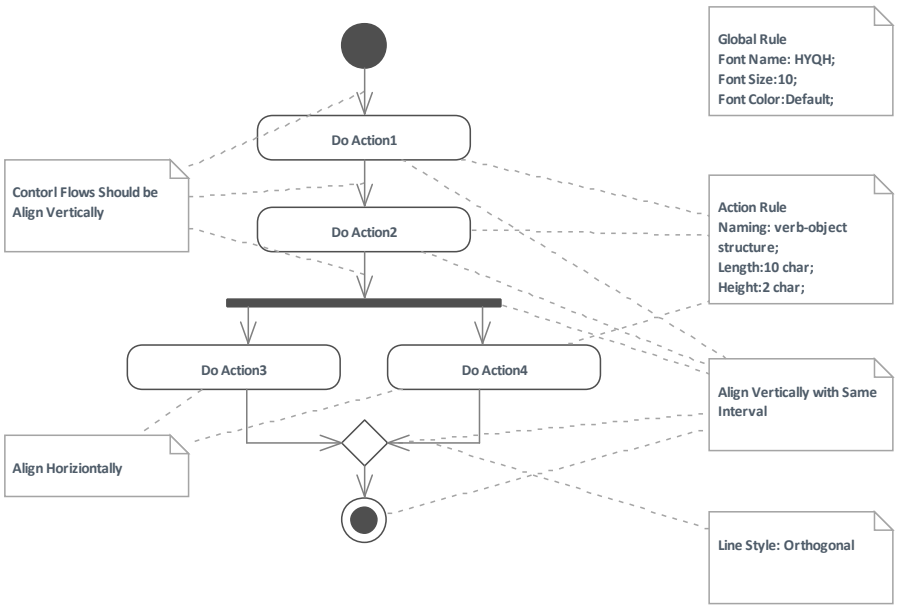

Figure 6 – Modeling standards

3.2 Stakeholder Requirement Analysis Practice

The avionics system is the core of the aircraft, which means that the avionics system would not solely finish the users' goals. The stakeholders and their requirements are different from the aircraft system. The primary stakeholder requirements should emphasize fulfilling what the aircraft want from the avionics system and necessarily keeping agreement with users' goal. Theoretically, Actors of the high-level system and architecture constitute a superset of actors of the low-level systems. Actors of the aircraft and the architecture are introduced to identify the actors of the avionics system. Moreover, new actors identified from the avionics system context are feedback to aircraft for the iteration of the aircraft system.

Use cases of the avionics system mainly focus on the operational concept and support concept (mainly focus on maintenance support) in our practice. Existing stakeholder requirements are used to aid the identification of the use case. All stakeholder requirements from existing and newly identified are allocated to use cases to keep traceability and validation purposes. The actors of the avionics system range from the physical environment, other connected systems, and humans, as shown in Figure 7.

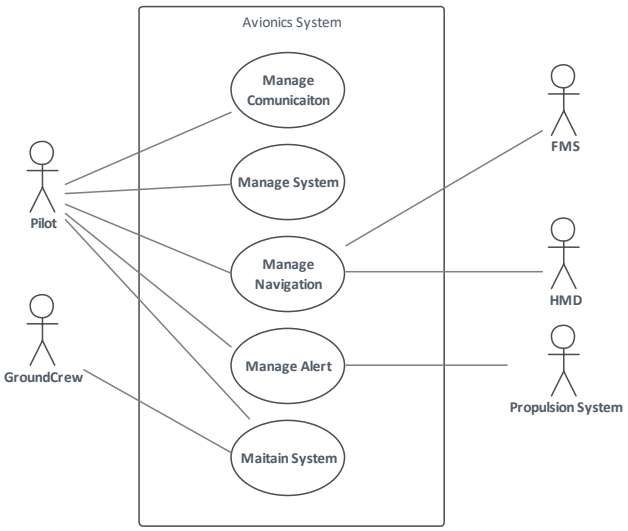

Figure 7 – Use cases and actors of avionics system

3.3 System Requirements Analysis Practice

System requirements that are decomposed and allocated from a high-level system (base on the work of aircraft requirements analysis) and a reusable avionics requirements library are imported as the initial requirements of the avionics system. Comprehension agreement for the avionics system requirements between aircraft and avionics developers is confirmed at the beginning of the analysis. Activities which as shown in Figure 8, are used to describe the detailed behaviors of use cases. External interfaces of the avionics systems are identified, as shown in Figure 9.

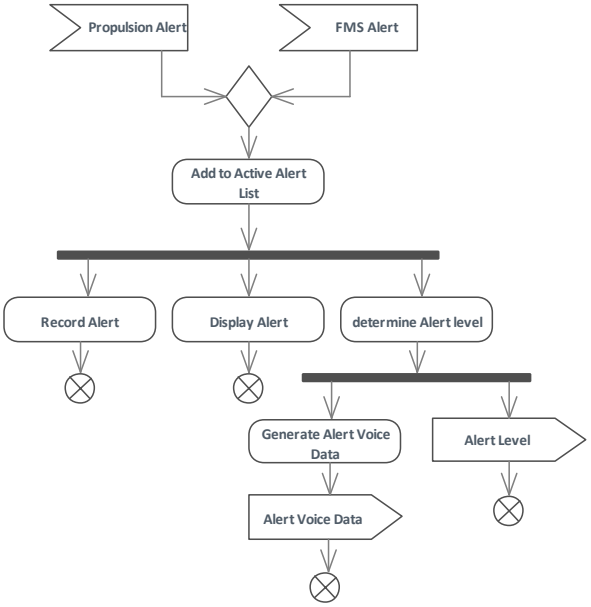

Figure 8 – System requirement analysis with activity diagram

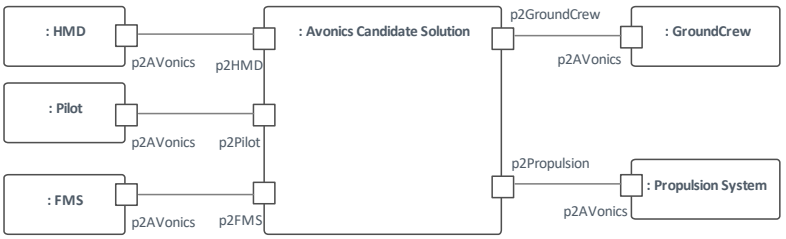

Figure 9 – External interfaces of avionics system

3.4 Architectural Analysis Practice

System functions, data, signals, and interfaces identified and defined from different use cases during functional analysis end up in the system architectural analysis. These features are copied and grouped to a new project with a block named “candidate solution” in the analysis context for the trade study without merging the existing use cases. Conflicts that exist in system functions between different use cases are solved after the grouping operation. Functionality integration, latency, weight, size, reliability, safety, supply reliability criteria are the main criteria in this avionics system practice.

A considerable number of functions are moved into software to provide an integrated avionics system. Function clustering that based on function types and layer pattern is performed as a pre-treatment. These function sets are considered as a unity for cohesiveness criteria calculation. The trade study context is illustrated in Figure 10.

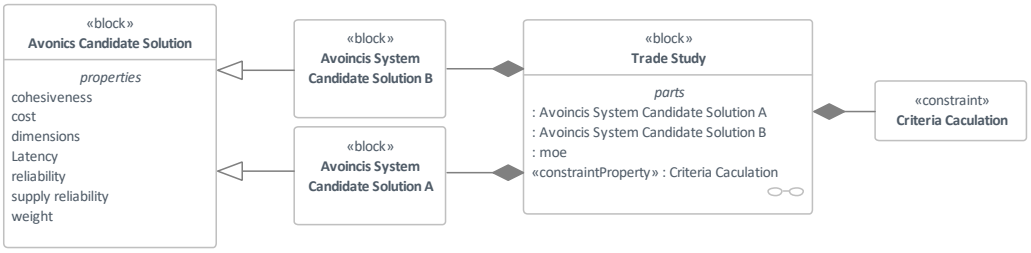

Figure 10 – Architectural analysis context

3.5 Architectural Design Practice

Subsystems are modeled by blocks with the “Subsystem” stereotype. Black box activities created

during system requirement analysis are copied and refined into white box activities with activity partitions for the avionics system function allocation, as shown in Figure 11. Necessary subsystem interaction messages are added in sequence diagrams to coordinate with the white box activities. All interactions are updated as between actors and subsystems or among subsystems. Subsystem interfaces and connections are created based on the interactions. From the use case view, each use case is accomplished by the collaboration of different subsystems. From the subsystems, their functions are distributed in different use cases. Finally, the subsystem state machine is established and integrated from different use cases, as shown in Figure 12.

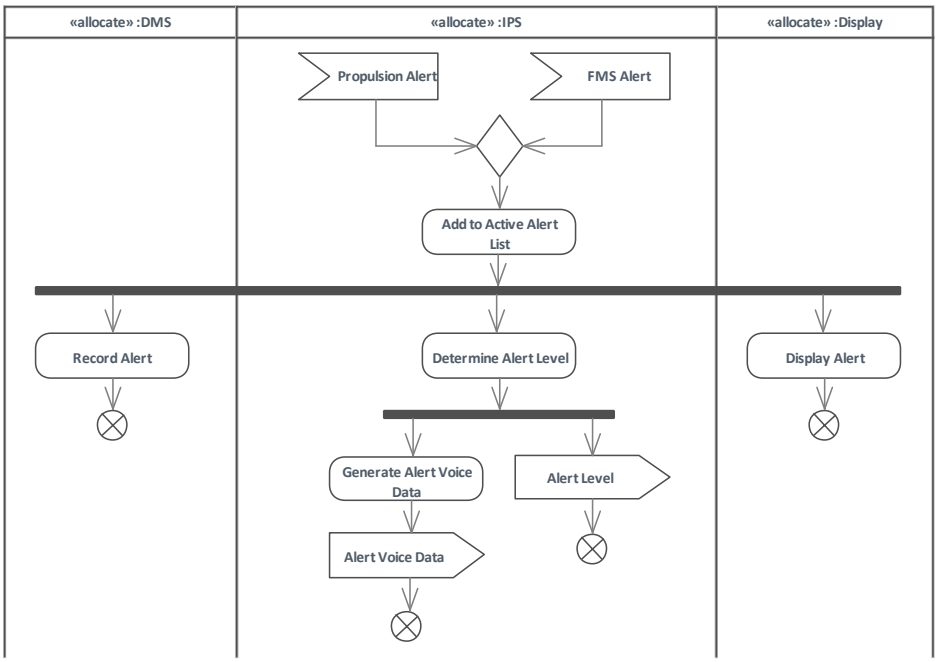

Figure 11 – Refinement of activity with partitions

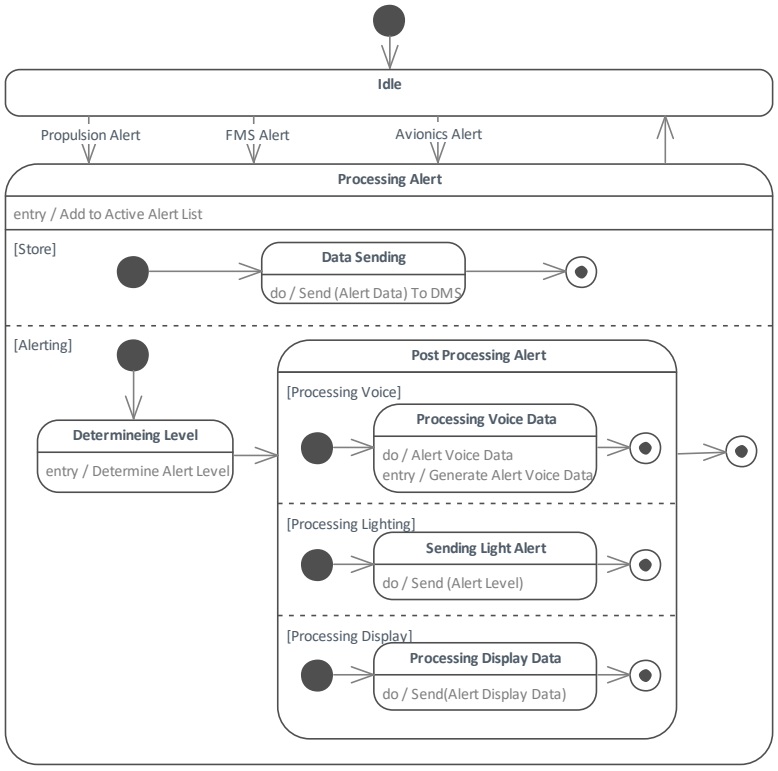

Figure 12 – Partial view of subsystem state machine

3.6 Integrate and validate Model Practice

A model-based distributed simulation environment is developed and sustained in practice for the integration and validation of the models as shown in Figure 13. The simulation environment architecture based on a co-simulation bus is developed to support to simulate models across the local network and provides a high degree of flexibility. A DDS data manager controls data synchronization between the different nodes throughout the co-simulation process.

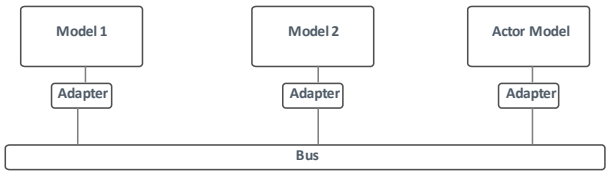

Figure 13 – Distributed simulation environment

Models are distributed to several different machines to overcome the difficulty in simulating large-scale systems. Load balancing and better simulation performance are guaranteed in this way. Models can be mixed and integrated with matched interfaces. Integration and iteration of different models could be carried out in an independent pattern. Incremental and continuous integration is achieved by this environment.

Additionally, Individual engineers can focus on their familiar models, and model errors would be isolated in a partial area in case of leading to a global impact on the whole project. Each model could be modified and integrated with an independent pattern without interference. Separate models for each subsystem are created from the use cases models. Both use case models and subsystem models are integrated and validated with the distributed simulation environment to achieve multiply views of validation purpose.

3.7 Interaction Complexity Management Practice

The interaction complexity management method is introduced in the above chapter. An example is illustrated for a detailed explanation. “Manage mission” and “manage alert” are two independent goals for the avionics system. Thus they are identified as two use cases. Manage mission would accomplish mission planning, monitoring based on the aircraft missions and various influences from the external and internal of the avionics system. A source of influence is internal of the avionics system that belongs to the “manage alert” use case. “Manage alert” just focuses on identifying, classifying, and sending alerts in various forms (display voice and lighting). These alerts may influence the execution of the missions which should be analyzed by the “manage mission” use case. An alert signal event generated from the manage alert use case and sent to the manage mission use case is modeled for this interaction in the system requirement analysis. This interaction is identified as an internal interaction of the same subsystem in the subsequence architectural design process.

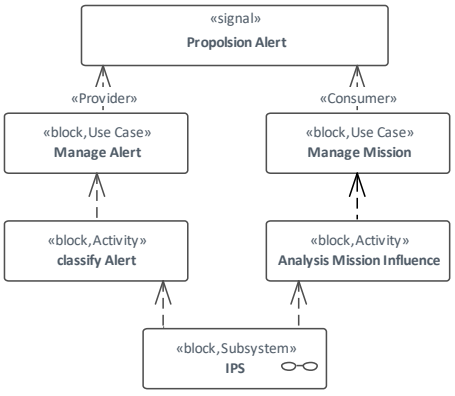

Figure 14 – Illustration of interaction complexity management

3.8 Model Quality Management Practice

A four-layer model for system model quality management that covers correctness, completeness, and consistency is created in our practice, as shown in Figure 15.

The first layer focus on the correctness of grammar and standard compliance. Gramma correctness is checked by checking rules that are supported by the modeling tool, while standard compliance is checked by reviews.

The second layer focus on the consistency of the model. The consistency that including type consistency, consistency between behaviors and structures, consistency between different behavior diagrams is checked by checking rules with the support of the modeling tool. Elements defined without usage are deleted by this check method, such as a defined signal reception without a corresponding trigger in the state machine.

The third layer focus on the content correctness of the model. Models are validated by simulations and reviews to ensure that the models provide the correct modeling of reality.

The fourth layer focuses on formal completeness and content completeness. Coverage analysis checks the formal completeness so that all the requirements could trace to subsystem solutions with all required model elements. The content completeness is checked by the same method with content correctness which focuses on the completeness of the model.

State transition matrices are used to the complexity of the model. The state transition is an index that reflects the logic complexity of the avionics system. With the inheritance between simple state and composition state， the elements of the transition matrix are calculated with the creation of the state transition matrix only with simple states as rows and columns.

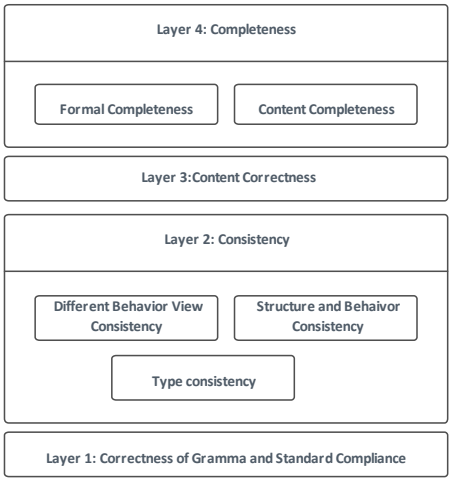

Figure 15 –Model for system model quality management

3.9 Handover Practice

The subsystem models are separated from the system models. The contents include subsystem blocks with properties, ports, operations, receptions, state machines, interfaces used in ports, actors that interact with subsystems. Subsystem requirements are derived from subsystems allocation at last.

4. Conclusion

An agile method for avionics system development is applied to cope with the complexity challenge in this paper. The method is formed to adapt to the agile and collaborative pattern of avionics system development. An Interaction complexity management method is introduced, and the effeteness is proved in practice. A suitable distributed integration and validation method and practice are introduced to support the avionics system's incremental development and continuous integration. A quality model which consists of correctness, completeness, consistency, complexity for the system model is initially constructed. More kinds of complexity should be managed, and further improvement for the quality model is required in future research.

References

[1] Friedenthal, Sanford, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. 3st edition, Morgan Kaufmann, 2014.

[2] Holt, Jon, and Simon Perry. *SysML for systems engineering*. 2st edition, IET, 2013.

[3] Borky, John M., and Thomas H. Bradley. *Effective model-based systems engineering*. 1st edition, Springer, 2019.

[4] Hart, Laura E. Introduction to model-based system engineering (MBSE) and SysML. *Delaware Valley INCOSE Chapter Meeting*. Mount Laurel, New Jersey: Ramblewood Country Club, 2015.

[5] Gaska, Thomas, Chris Watkin, and Yu Chen. Integrated modular avionics-past, present, and future. IEEE *Aerospace and Electronic Systems Magazine* Vol. 30, No. 9, pp 12-23, 2015.

[6] Douglass, Bruce Powel. *Agile systems engineering*. 1st edition, Morgan Kaufmann, 2015.

[7] Weilkiens, Tim. *SYSMOD-The systems modeling toolbox-pragmatic MBSE with SysML*. 3st edition, Lulu. com, 2016.

[8] Haskins, Cecilia, Kevin Forsberg, Michael Krueger, D. Walden, and D. Hamelin. *Systems engineering handbook*. 4st edition, John Wiley & Sons, Inc, 2006.

[9] Gomaa, Hassan, and E. Olimpiew. The role of use cases in requirements and analysis modeling. *Workshop on Use Cases in Model-Driven Software Engineering*. Montego Bay, Jamaica. 2005.

[10] Mohagheghi, Parastoo, Vegard Dehlen, and Tor Neple. Definitions and approaches to model quality in model-based software development–A review of literature. *Information and software technology*, Vol. 51, No. 12, pp 1646-1669, 2009.

5. Contact Author Email Address

Lei Dong: 1409881551@qq.com

6. Copyright Statement