



Article

X-RMTV: An Integrated Approach for Requirement Modeling, Traceability Management, and Verification in MBSE

Pengfei Gu ^{1,2,3}, Yuteng Zhang ^{1,2,3}, Zhen Chen ^{1,2,3}, Chun Zhao ⁴ , Kunyu Xie ^{1,2,3}, Zhuoyi Wu ^{1,2,3}
and Lin Zhang ^{1,2,3,*} 

- ¹ Hangzhou International Innovation Institute, Beihang University, 166 Shuanghongqiao Street, Pingyao Town, Yuhang District, Hangzhou 311115, China; by2003151@buaa.edu.cn (P.G.); zytbh@buaa.edu.cn (Y.Z.); czhen@buaa.edu.cn (Z.C.); kyxie@buaa.edu.cn (K.X.); 20373373@buaa.edu.cn (Z.W.)
 - ² State Key Laboratory of Intelligent Manufacturing Systems Technology, Yongding Road No. 51, Haidian, Beijing 100854, China
 - ³ School of Automation Science and Electrical Engineering, Beihang University, Xueyuan Road No. 37, Haidian, Beijing 100191, China
 - ⁴ School of Computer, Beijing Information Science and Technology University (BISTU), Taihang Rd 55, Changpin District, Beijing 102206, China; zhaochun@bistu.edu.cn
- * Correspondence: zhanglin@buaa.edu.cn

Abstract: Formal requirements modeling and traceability management are essential for effectively implementing Model-Based Systems Engineering (MBSE). However, few studies have explored the integration of requirement modeling, traceability management, and verification within MBSE-based systems engineering methodologies. Moreover, the predominant modeling language for MBSE, SysML, lacks sufficient capabilities for requirement description and traceability management and for depicting physical attributes and executable capabilities, making it challenging to verify functional and non-functional requirements collaboratively. This paper proposes an integrated approach for requirement modeling, traceability management, and verification, building on the previously proposed integrated modeling and the simulation language called X language. Our contributions primarily include defining the ReqXL specification for MBSE-oriented requirement modeling based on X language, proposing an algorithm for automatically generating requirement traces, and an integrated framework for requirements modeling, traceability management, and verification was developed by combining the X language with ReqXL. These functionalities were customized on the self-developed integrated modeling and simulation platform, XLab, which is specifically tailored for the X language. Furthermore, we showcase the efficacy and promise of our approach through a case study involving the design of an aircraft electrical system.

Keywords: requirement modeling; traceability management; model-based systems engineering; verification; X language



Citation: Gu, P.; Zhang, Y.; Chen, Z.; Zhao, C.; Xie, K.; Wu, Z.; Zhang, L. X-RMTV: An Integrated Approach for Requirement Modeling, Traceability Management, and Verification in MBSE. *Systems* **2024**, *12*, 443. <https://doi.org/10.3390/systems12100443>

Academic Editor: Jérôme Morio

Received: 28 August 2024

Revised: 21 September 2024

Accepted: 10 October 2024

Published: 20 October 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The successful development of complex multidisciplinary products such as airplanes, aircraft, and automobiles relies on rigorous systems engineering design methodologies. At the same time, it also requires robust support from effective requirements engineering practices [1]. During the development of complex systems, stakeholders' needs must evolve into design artifacts through the systems engineering design process. However, managing the requirements for such complex systems is fraught with challenges [2]. A report published by the Standish Group suggests that a significant reason for the failure of many projects is incomplete requirements and insufficient stakeholder involvement [3]. Therefore, in the system design process, it is crucial first to define correct and unambiguous requirements and ensure deep stakeholder involvement in the system design for the successful development of the system [4]. Secondly, different departments or organizations

develop various design models and data based on the requirements. This can obscure multiple relationships between elements at different levels, thereby raising higher demands for the traceability management of requirements [5]. Model-Based Systems Engineering (MBSE) is a mainstream approach supporting the design and development of complex systems throughout their entire lifecycle. Its core concept is to support requirements analysis, functional analysis, system design, and verification by developing a structured and standardized model [6,7]. SysML is a systems modeling language proposed by the Object Management Group (OMG) specifically for MBSE [8]. The requirements diagrams provided by SysML can formally describe requirements and establish relationships between requirements and system modeling elements. However, SysML requirements diagrams only define two attributes: the requirement ID and description, and do not provide any constraints. They lack detailed definitions of the relationships between stakeholders, stakeholder needs, system design requirements, and the system models generated during the system design process, making it difficult to achieve traceability management of requirements from the source effectively.

Recently, the standardization and formalization of requirements have been extensively researched and have made significant progress [9–11]. However, most requirements expressions remain independent of requirements traceability. In other words, the descriptions of requirements do not effectively assist in requirements traceability. Currently, there are several commercial requirements management tools, such as DOORS and ReqTify, that support requirements management and traceability. However, they need to be integrated with MBSE design tools to function effectively [12,13]. This multi-platform traceability management can lead to gaps in the top-down design process based on MBSE, making it difficult to ensure consistency between requirements and the models defined in MBSE. Additionally, there is related research that explores methods for formalizing and managing requirements traceability by extending SysML or defining domain-specific modeling languages [14,15]. However, while these methods can achieve requirements traceability management, they are not effectively integrated with MBSE methods. In other words, when requirements change, although it is possible to trace the affected upstream and downstream requirements and design artifacts, how this impact is linked through the MBSE-based system design process cannot be determined. Additionally, these methods almost always rely on or incorporate SysML for traceability management. However, SysML's lack of description for physical characteristics and its non-executable nature may require specific domain models to be implemented in conjunction with other simulation languages and tools, leading to significant challenges in requirements traceability management and verification [16]. Currently, there are also related efforts aiming to establish links between requirements and MBSE-based functional architectures by defining domain-specific modeling languages to achieve requirements traceability and simulation verification [17]. However, this verification is only focused on functional requirements.

In our previous research, we developed an integrated intelligent modeling and simulation language called X language, and its corresponding MBSE methodology, X-SEM, based on SysML, Modelica, and Discrete Event Systems Specification (DEVS) [18–20]. X language and its methodology effectively support object-oriented modeling, facilitating requirements analysis, functional analysis, system logical architecture design, and physical architecture design. Furthermore, X language is a simulatable modeling language supporting collaborative verification of functional and non-functional requirements.

Based on this foundation, this paper first extends X language and establishes a specification for requirement modeling in two forms—graphical and textual—oriented towards MBSE, termed ReqXL. Enriching the syntax and semantics of requirement description and traceability, enables the construction of correct, unambiguous requirements and facilitates the establishment of a complete traceability chain from requirement origins to requirement realization, encompassing design components, and their related parameters or attributes. Furthermore, the two modeling forms of requirement description bring benefits to modeling personnel in achieving traceability management: the graphical modeling form offers an

intuitive understanding of associations among various requirements and system design models, while the standardized textual modeling form defines unambiguous requirement models and facilitates automated requirement traceability.

Secondly, building upon ReqXL, this paper proposes an algorithm for automated requirement traceability, supporting the automatic capture of implicit requirement relationships among different hierarchical model elements. This ensures rapid identification of the impact of requirement changes on the system design process and facilitates prompt adjustments to system design solutions.

Lastly, leveraging ReqXL and X language, an integrated approach is proposed for MBSE encompassing requirement modeling, traceability management, and verification. This integrated approach effectively establishes associations between requirement models and hierarchical models generated during the system design process, swiftly clarifying how requirements influence the design process and effectively linking them with system design models. Consequently, when system requirements change, rapid identification of their impact on design allows for expedited iterative system design. Additionally, the integrated simulation capability of X language for both functional and non-functional requirements ensures the establishment of simulation models based on a unified language. This not only ensures consistency among different hierarchical models, but also avoids cross-platform traceability management issues. Such an integrated approach to requirement modeling, traceability management, and verification significantly enhances the efficiency of system development.

The remainder of this paper is structured as follows. Section 2 introduces the current state of research on requirements modeling, traceability management, and verification, as well as the relevant background on the X language family. In Section 3, we provide a detailed introduction to the proposed requirements modeling specification, ReqXL, including the defined requirements metamodel and requirements syntax. Section 4 presents an integrated framework for requirements modeling, traceability management, and verification based on X language and ReqXL. This section sequentially introduces the implementation methods and benefits of requirements modeling, traceability management, and verification. Finally, we describe the integrated platform developed based on the aforementioned methods. In Section 5, we demonstrate the effectiveness of our approach using a case study of an aircraft electrical system. The paper concludes with a summary and outlook on future work.

2. Related Works

In this section, we summarize the state-of-the-art requirement modeling, traceability management, and verification methods in MBSE and analyze the challenges currently faced.

2.1. Requirements Modeling Methods

Requirements modeling methods can mainly be divided into text-based and model-based methods in MBSE. Text-based methods generally use natural language to write requirements. The advantage of natural language is that it can describe any type of requirement. However, its main problem is imprecision and the potential for ambiguity. As an improvement to these issues, structured natural language has emerged [21]. Many scholars and official organizations have provided clear guidelines for the structured description of requirements [4,10,22]. These methods ensure, to some extent, that requirements are unambiguous and verifiable. Unfortunately, such structured text-based requirements make it difficult to establish relationships between requirements and between requirements and system design models.

Model-based methods provide a new perspective. SysML forms the foundation of model-based requirement modeling methods. SysML requirement diagrams allow requirements to be represented as model elements, and they can explicitly show various relationships between different requirements and between requirements and other system design models [23]. However, the description of requirements in SysML requirement dia-

grams is still based on natural language without any structured constraints. This can lead to ambiguity and misunderstanding. Additionally, the SysML requirements metamodel does not offer rich semantic support to effectively link requirements with previous and subsequent system design models or stakeholders. To address this, some innovative methods have been proposed that use SysML elements for requirements modeling [24,25]. Moreover, many scholars have explored the definition of domain-specific modeling languages or the combination of domain-specific modeling languages with SysML to achieve requirements modeling and effectively link requirements with functional architectures or system design models [15,17,26].

2.2. Requirements Traceability Methods

Requirement traceability methods are primarily model-based and ontology-based in MBSE. Model-based methods for establishing requirement traceability relationships primarily focus on constructing models, including requirements models and traceability models. Currently, common model-based methods are mainly divided into two categories: creating SysML profiles oriented towards requirements traceability using SysML extension mechanisms, and defining domain-specific modeling languages for requirements traceability. The literature [14] proposes a SysML-based method to enhance the traceability of requirements in SysML diagrams, including enriched SysML profiles for requirements definition and traceability, and algorithms for generating traceability models. These algorithms link requirements with design elements, generating traceability links at different levels of granularity. Deng et al. extended SysML models to model safety requirements and design artifacts, capturing traceability information [27]. The literature [17] defines five domain-specific languages based on the MBSE approach to achieve close association between requirements and functional architecture, laying the foundation for requirements traceability. The literature [28] defines a domain-specific modeling language oriented towards requirements and combines it with SysML to achieve bidirectional traceability between requirements and upstream/downstream system designers and components. Taromirad et al. propose an agile modeling method for managing requirements traceability in safety-critical systems engineering, using a domain-specific modeling language to describe traceability models [29]. However, although these two categories of methods establish associations between requirements and system design models to some extent, they rarely consider the system design process. Moreover, the non-executability of SysML models or models based on domain-specific languages poses difficulties for requirements verification.

Ontology-based methods are another approach to achieving requirements traceability. In the traceability management of MBSE, ontology refers to the types, attributes, and relationships of different objects such as requirements and system design models [30]. Therefore, a formal definition of such ontologies can effectively realize requirements traceability. Adithya et al. proposed an ontology called OntoReq, which uses machine learning, knowledge engineering, and the sunflower optimization algorithm to create high-quality requirements traceability matrices, effectively enhancing the stability and reliability of requirements traceability [31]. Murtazina et al. developed an ontology that includes theoretical knowledge of requirements engineering in an agile environment. This ontology accumulates knowledge of requirement types and requirement components, enabling effective traceability between them [32]. However, these methods seldom consider the system development process. Recently, Wu et al. defined a cognitive domain-oriented design ontology, which achieves traceability management among stakeholders, models, and the system development process within a developed MBSE toolchain [33,34]. Nevertheless, the different levels of models established during the system development process are developed using different languages and tools, posing challenges for requirements traceability and verification.

2.3. Requirement Verification Methods

The goal of requirement verification is to ensure that the designed system meets the system requirements in MBSE. Currently, there are two mainstream methods: formal verification methods and simulation-based verification methods. Formal verification typically involves transforming system design models into formal models and then using theorem provers or model checkers to verify system requirements. For example, Wang et al. proposed a method to transform SysML block definition diagrams and state machine diagrams into NuSMV models to verify safety properties [35]. Staskal et al. transformed SysML activity diagrams into NuXmv modules and verified the system using user-defined Linear Temporal Logic (LTL) formulas [36]. In another study, requirements modeled in RSML-e were converted into NuSMV models to verify a flight guidance subsystem [37].

Simulation-based methods are also a primary means of requirements verification. Current SysML-oriented software tools, such as Rhapsody and Cameo Systems Modeler, have dynamic model execution capabilities that can simulate system behavior models (state machine diagrams, activity diagrams, etc.) to verify requirements. However, both formal model-checking tools and executable platforms for SysML can only achieve verification and analysis of functional requirements.

The design and verification of complex products require not only the verification of functional requirements but also the verification of non-functional requirements (such as performance requirements, environmental requirements, etc.). Currently, to integrate system design and system simulation effectively, the main approaches are through model transformation or cosimulation [38]. Model transformation methods are primarily based on SysML extension mechanisms, constructing SysML metamodel extension configuration files for specific simulation languages (such as Modelica, Matlab/Simulink, etc.) to achieve automatic conversion from system design to simulation, thereby achieving collaborative verification of both functional and non-functional requirements. In 2012, OMG proposed a SysML and Modelica mapping method based on Query/View/Transformation (QVT), and defined the SysML4Modelica extension package for metamodel transformation, providing standard conversion methods between SysML and Modelica [39]. Zhou, Li, and others have further improved and expanded upon this foundation [40]. However, although Modelica can support unified modeling of complex systems across multiple domains, equation-based languages are not friendly for discrete event modeling and simulation [41]. Inconsistencies between support for discrete and continuous models make mapping and conversion between SysML and Modelica difficult, hindering bidirectional information exchange. In addition to model transformation, collaborative simulation is also a commonly used integrated system design and simulation method. Currently, it mainly utilizes the Functional Mock-up Interface (FMI) and its Functional Mock-up Units (FMU) to achieve collaborative simulation between SysML logical design models with discrete characteristics and physical models with continuous characteristics. For example, ref. [42] uses FMI to compile SysML models into .fmu files and integrates them with Simulink and Modelica, thereby achieving collaborative verification of both functional and non-functional requirements. However, both model transformation and collaborative simulation methods require cross-language, cross-platform adaptability, posing significant challenges for requirement traceability and model consistency.

In summary, the current issues in requirements modeling, traceability management, and verification in MBSE primarily stem from the fact that SysML, which supports MBSE, struggles to facilitate integrated modeling across the four levels of requirements, functionality, logic, and physics, and cannot support collaborative verification of both functional and non-functional requirements. Furthermore, there is a lack of an integrated approach to incorporate stakeholders, requirements, the system development process, system architecture models, and even simulation models, leading to difficulties in requirements traceability management and inefficiencies in system development.

2.4. X Language Family

X language [18,19,43,44], X-SEM [20], and XLab comprise a new set of languages, methods, and tools oriented towards MBSE system design and development. X language is an integrated modeling and simulation language for MBSE, based on DEVS and combining modeling concepts from SysML and Modelica. The specific classes established by X language, combined with its two modeling formats of graphics and text allow for integrated analysis and design at four levels: requirements, functions, logic, and physics. Furthermore, the text model based on X language supports simulation. This integration enables the verification and validation of models at all levels to be coordinated with the system design process. X-SEM encompasses the entire system design process through four distinct perspectives and three domains, facilitated by X language and its associated software, XLab. The four perspectives—requirements, structure, behavior, and parameters—provide a multifaceted view of the system’s core. The three domains—problem domain, solution domain, and verification domain—are designed to effectively manage, define, and verify varying levels of abstraction and granularity within the system, ensuring timely validation of both functional and non-functional requirements.

The primary focus of MBSE is multidisciplinary complex systems [45], defined as systems with numerous components and interconnections that are difficult to describe, understand, predict, manage, design, and change. Therefore, using X language for system modeling and simulation is highly suitable. However, the requirements diagrams of X language, derived from SysML, lack comprehensive descriptions of requirements and face difficulties in supporting bidirectional traceability of requirements relationships. Thus, by adopting the requirements modeling and automatic tracking methods proposed in this paper, in combination with the X language family, it is possible to achieve integrated modeling, tracing, and verification of complex product requirements effectively. This approach not only enhances system development efficiency but also ensures that when requirements change, the affected requirements and system components can be precisely located, enabling rapid iteration of system design.

3. ReqXL Overview

To effectively support the formalized description of stakeholder needs and system design requirements during the system design process, as well as to achieve traceability among requirements, the sources of requirements involved in the MBSE development process, functional models, and system components. To verify requirements, we have defined a formal modeling specification for requirements based on X language requirement diagram, named ReqXL. ReqXL adopts a model-based and structured language approach, including specifically designed semantics for describing stakeholder needs, system design requirements, and capturing their traceability links used in the MBSE development process. These requirement semantics are introduced in Section 3.1, and the abstract syntax of ReqXL is provided in Section 3.2.

Figure 1 shows how ReqXL is related to other elements of X language through import relationships. Therefore, functional models, structural models, and behavioral models established in X language can be imported into ReqXL, facilitating effective traceability between these models and requirements defined during the MBSE development process. Additionally, structural models and behavioral models defined using X language can automatically generate simulation models to verify requirements. This enables an integrated process of requirement modeling, traceability, and verification, thereby improving the efficiency of system design and development.

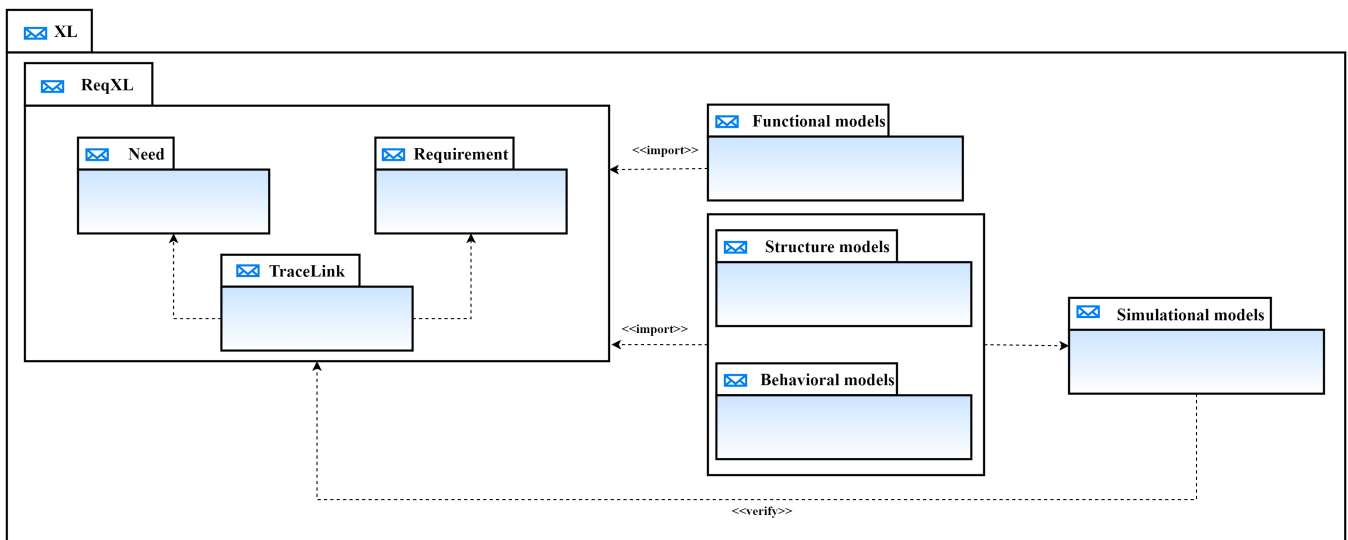


Figure 1. The relationship between ReqXL and existing elements of the X language.

3.1. Semantic Structure

The purpose of designing ReqXL is to enable requirements engineers and systems engineers to identify stakeholder needs, system design requirements, and effectively define the sources of requirements, authors, and responsible persons, as well as the hierarchy and types of requirements. The description of system design requirements must ensure unambiguity and verifiability. Additionally, tracking should be conducted in both forward and backward directions. ReqXL is an extension of X language tailored for the requirements domain, so the model file extensions are the same as existing X language models, which is .xl. In this section, we will define the semantics of ReqXL based on the goals we aim to achieve.

The semantics of ReqXL are designed to enable modelers to clearly define requirements. ReqXL aims to express requirements simply and understandably. It handles all types of requirements and records information about the requirements as attributes. Table 1 provides an informal detailed semantic design of ReqXL, defining the representation of requirements, types of requirements, requirement attributes, and requirement relationships.

First, ReqXL offers two modeling forms: requirement diagrams and requirement texts. Requirement diagrams facilitate the intuitive observation of relationships between requirements, model elements, and other entities, while also displaying specific details of the requirements. Requirement texts provide effective input for the automatic tracking and management of requirements, enabling data mining of implicit requirement relationships.

Second, ReqXL categorizes requirements into two main types: stakeholder needs and system design requirements. Stakeholder needs primarily originate from stakeholders and are often described vaguely and cannot be directly verified. System design requirements are derived by systems engineers using systems engineering methods based on stakeholder needs; they must be explicit and verifiable.

ReqXL records specific information about these two types of requirements as attributes. For example, the attributes for both types include the requirement ID, name, and description. Stakeholder needs also include the source of the need and the proposing stakeholder, while system design requirements include attributes such as level, priority, and type.

In addition to specifying requirements, ReqXL is used to establish traceability relationships between requirements and different elements from the system development process. Therefore, ReqXL provides ten types of requirement relationships to support the association between requirements, different personnel involved in the system development process, and model elements.

Table 1. ReqXL semantics definition.

ReqXL Semantics Definition	Description
Forms	
diagram	
text	
Types	
stakeholder need	Stakeholders' Expectations
system design requirement	Specifications for System Design
Attributes	
identifier	Unique identifier for stakeholder needs and system design requirements
name	Name of stakeholder needs and system design requirements
priority	Priority of System Design Requirements (High/Medium/Low)
type	Type of System Design Requirements (Functional/Performance/Design/Environmental/Suitability)
level	Level of System Design Requirements (System-level/Component-level)
source	Source of Stakeholder Needs
stakeholder	Presenter for Stakeholder Needs
responsible stakeholder	Responsible person for System Design Requirements
optimization role	Optimization Roles for System Design Requirements (Design variable/Design variable bound/Input parameter/Constraint/Objective)
description	Description of Stakeholders' Expectations for the Product Description of System Engineers for Product System Design Requirements
Traceability Relationships	
source	Correlation of the Stakeholder Needs and their Source
responsible	Correlation of Responsible Stakeholders and System Design Requirements
presenter	Correlation of Stakeholders and Stakeholder Needs
compose	Inclusion Relationship between Requirements
derive	Derivation Relationship between Requirements, as well as between Requirements and other Models
refine	Refinement Relationship between Requirements, as well as between Requirements and other Models
trace	Traceability Relationship between Requirements, as well as between Requirements and other Models
satisfy	Satisfaction Relationship between Requirements and Model Elements
verify	verification Relationship between Requirements and Simulation Test Cases
map	Correlation of Requirements and Specific Attributes of Model Elements

The most critical attribute in the requirements model is the description attribute. In ReqXL, stakeholder needs are mostly general expectations of the product proposed by stakeholders. Their description attributes do not strictly require a specific format. However, system design requirements ultimately need verification; therefore, their description attributes must be strictly defined. INCOSE provides relevant guidelines for standardized requirement descriptions. To define unambiguous and verifiable system design requirements, ReqXL adopts the requirement description format proposed by Carson [10]. Different types of requirements should use different description formats, as illustrated in Figure 2. For instance, functional requirements need to specify what functions the

system should possess; performance requirements indicate how well a function should perform. Design constraint requirements impose restrictions on the feasible design space, while environmental requirements specify how the system should behave when exposed to specific environments. Lastly, suitability requirements encompass all “abilities”, such as maintainability, and reliability.

Functional: "The **SYSTEM** shall [exhibit] **FUNCTION** [while in **CONDITION**]"

Performance: "The **SYSTEM** shall **FUNCTION** with **PERFORMANCE** [and **TIMING** upon **EVENT TRIGGER**] while in **CONDITION** "

Design: "The **SYSTEM** shall [exhibit] **DESIGN CONSTRAINTS** [in accordance with **PERFORMANCE** while in **CONDITION**]"

Environmental: " The **SYSTEM** shall [exhibit] **CHARACTERISTIC** during/after exposure to **ENVIRONMENT** [for **EXPOSURE DURATION**]

Suitability: "The **SYSTEM** shall exhibit **CHARACTERISTIC** with **PERFORMANCE** while **CONDITION** [for **CONDITION DURATION**]"

Figure 2. Specified patterns for expressing different types of requirements.

3.2. ReqXL Abstract Syntax

Here, we elaborate on the abstract syntax of ReqXL by introducing the ReqXL metamodel and BNF-based grammar.

3.2.1. ReqXL Metamodel

ReqXL incorporates concepts commonly used for specifying requirements. To express the requirement model in a structured manner, it must be defined through formalization. Figure 3 provides an overview of the ReqXL metamodel describing requirement constructs, along with model elements involving traceability. *ModelElement* is a highly abstract element, encompassing requirements, needs, and trace links in ReqXL, along with related model elements imported from the X language, such as couple class models and discrete class models. In ReqXL, needs can specify their sources and stakeholders, while requirements can delineate their levels, types, roles, and priorities. Furthermore, each need or requirement possesses a descriptive attribute known as “text”, which articulates the specific content of the need or requirement in natural language. This “text” attribute can also be linked to related requirements or model elements. Notably, the “text” attribute of a requirement provides a structured description template based on its requirement type. Model elements typically capture X language components that we want to trace to needs or requirements. *Tracelink* is utilized to establish the relationships between requirements and other captured elements. There are four categories of trace links, encompassing a total of ten corresponding trace relationships, which are associated with two traceable elements: supplier and customer. *Tracelink* defines four-link categories: pre-specification requirement traceability (*PreRS*), which refers to the connection between requirements and the preceding model elements upon which they are based; post-specification requirement traceability (*PostRS*), which pertains to the link between requirements and subsequent development activities and model element; traceability between function and requirement (*TraceBetFun_Req*) refers to traceability between requirements, use cases and functional activities in the system development process; and traceability between requirements (*TraceBetReq*) involves dependencies between requirements.

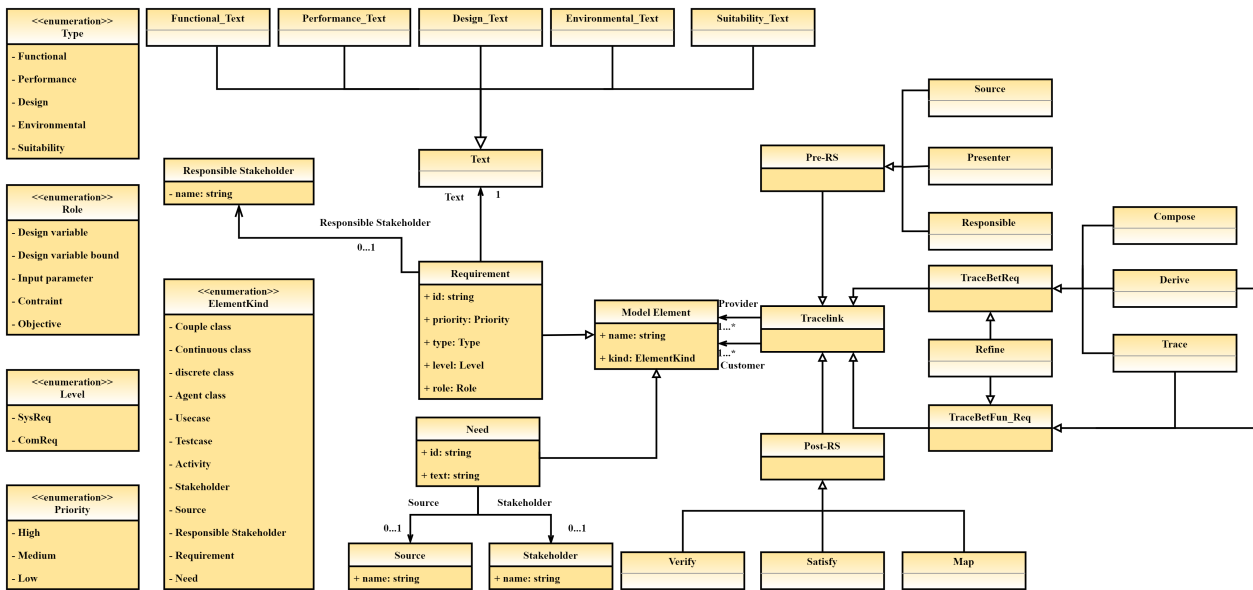


Figure 3. ReqXL metamodel.

3.2.2. ReqXL Grammar

The ReqXL metamodel depicted in Figure 3 is highly beneficial in guiding system engineers in writing requirements. For instance, it describes requirement content by specifying different types of requirements. Figure 4 specifies the corresponding grammar for describing requirements and model elements, capturing some traceability information.

```

1 <REQXL> ::= <Need> | <Requirement> | <Traciability>
2
3 <Need> ::= "need" <Need_name> ":"
4 <Need_attributes> "end;"
5
6 <Requirement> ::= "requirement" <Requirement_name> ":"
7 <Req_attributes>
8 "Text:"
9 <Text_description>
10 "end;"
11 <Traciability> ::= "traciability" ":"
12 <Traciability_relationships>
13 "end;"
14
15 <Need_attributes> ::= <Attribute> ":" <Need_attributes> | ε
16 <Attribute> ::= "Id : " <text_string> | "Source: " <text_string> | "Stakeholder: " <text_string> | "Text: " <text_string>
17 <Req_attributes> ::= <Attribute> ":" <Req_attributes> | ε
18 <Attribute> ::= "Id : " <text_string> | "Responsible stakeholder: " <text_string> |
19 "Priority: " ("High" | "Medium" | "Low") |
20 "Type: " ("Functional" | "Nonfunctional" | "Performance" | "Design(constraint)" | "Environmental" | "Suitability") |
21 "Level: " ("System requirement" | "Component requirement") |
22 "Role: " ("Design variable" | "Design variable bound" | "Input parameter" | "Constraint" | "Objective")
23 <Text_description> ::= "Functional: \The SYSTEM shall [exhibit] FUNCTION [while in CONDITION]" |
24 "Performance: \The SYSTEM shall FUNCTION with PERFORMANCE [and TIMING upon EVENT TRIGGER] while in CONDITION" |
25 "Design: \The SYSTEM shall [exhibit] DESIGN CONSTRAINTS [in accordance with PERFORMANCE while in CONDITION]" |
26 "Environmental: \The SYSTEM shall [exhibit] CHARACTERISTIC during/after exposure to ENVIRONMENT [for EXPOSURE DURATION]" |
27 "Suitability: \The SYSTEM shall exhibit CHARACTERISTIC with PERFORMANCE while CONDITION [for CONDITION DURATION]"
28
29 <Traciability_relationships> ::= <Traciability_relationship> ":" <Traciability_relationships> | ε
30 <Traciability_relationship> ::= "compose(" <Element> , <Element> ")" |
31 "trace(" <Element> , <Element> ")" | "derive(" <Element> , <Element> ")" |
32 "verify(" <Element> , <Element> ")" | "satisfy(" <Element> , <Element> ")" |
33 "refine(" <Element> , <Element> ")" |
34 "map(" <Element> , <Element> , <Element> ")" |
35 "source(" <Element> , <Element> ")" |
36 "present(" <Element> , <Element> ")" |
37 "responsible(" <Element> , <Element> ")"
38 <Element> ::= <text_string> | "Need" | "Requirement" | "class" | "usecase" | "testcase" | "activity"
    
```

Figure 4. ReqXL grammar.

ReqXL mainly consists of four primary rules. Firstly, there is the rule for importing other model elements from X language. During system development, functionalities and components defined based on X language can be referenced to support requirement traceability. The use of these references is demonstrated through the import rule. The rule for stakeholder needs starts with a mandatory name. Stakeholder needs have a set of attributes: identifier, source, stakeholders, and description. The rule for requirements also begins with a mandatory name. Requirements have a set of attributes: identifier, responsible stakeholders, priority, type, level, and optimization roles. The possible values for priority, type, level, and optimization roles are enumerated types of different kinds. In ReqXL, stakeholder needs are generally vague and broad. Therefore, their description has no restrictions. Requirements, on the other hand, guide system design and must be unambiguous and verifiable. Therefore, we structure requirements. Specifically, different structured description templates are provided for five types of requirements (functional requirements, performance requirements, design requirements, etc.). The core objective of the traceability rule is to establish direct traceability links between stakeholders, stakeholder needs, requirements, model elements, etc. ReqXL provides 10 traceability relationships supporting trace links for four types (*PreRS*, *PostRS*, *TraceBetFun_Req*, and *TraceBetReq*) of traceability. Therefore, the proposal of ReqXL provides an opportunity to enable requirements modeling, traceability management, and verification.

4. Integrated Approach for Requirement Modeling, Traceability Management, and Verification

In this section, an integrated approach to requirement modeling, traceability management, and verification is proposed. Firstly, an integrated framework is presented, defining a unified process for requirement modeling, traceability management, and verification based on X language and ReqXL. Secondly, the implementation methods and potential advantages of requirement modeling and verification, as well as requirement traceability management, are introduced in detail. Finally, an integrated platform based on the aforementioned methods is developed.

4.1. Integrated Framework

To efficiently support requirement modeling, traceability management, and verification, an integrated framework is proposed, as shown in Figure 5. Firstly, ReqXL is used to capture stakeholder needs and their sources. Secondly, the MBSE methodology based on X language and following the X-SEM can guide the creation of requirement models [18–20].

Specifically, in the problem domain, the functional analysis process can transform stakeholder needs into functional models and guide the formulation of system-level and component-level design requirements. Based on the functional analysis, the logical architecture of the system and the design of various logical subsystems can be implemented according to the component-level design requirements. After defining the logical architecture, the process first enters the verification domain, where the logical architecture model can generate simulation test cases for functional component-level design requirements to verify functional requirements.

In the solution domain, the logical architecture model defined in the problem domain, along with non-functional component requirements, guides the definition of the final architecture. The final architecture defines the design of logical subsystems oriented toward functional requirements and physical subsystems oriented toward non-functional requirements. Once the final architecture is defined, the process re-enters the verification domain. In the verification domain, the final architecture model can generate simulation test cases for non-functional component-level requirements to verify non-functional requirements.

Additionally, the functional models, system design models (logical and physical models), and simulation test case models for functional and non-functional requirements generated during the system design can be imported into the requirement model defined by ReqXL. Through the requirement relationships provided by ReqXL, complete traceability

from requirement sources to requirement verification is established. Finally, using the requirement relationships defined by ReqXL as input for the traceability generation algorithm, a complete requirement traceability chain is established, achieving the integrated goal of requirement modeling, traceability management, and verification.

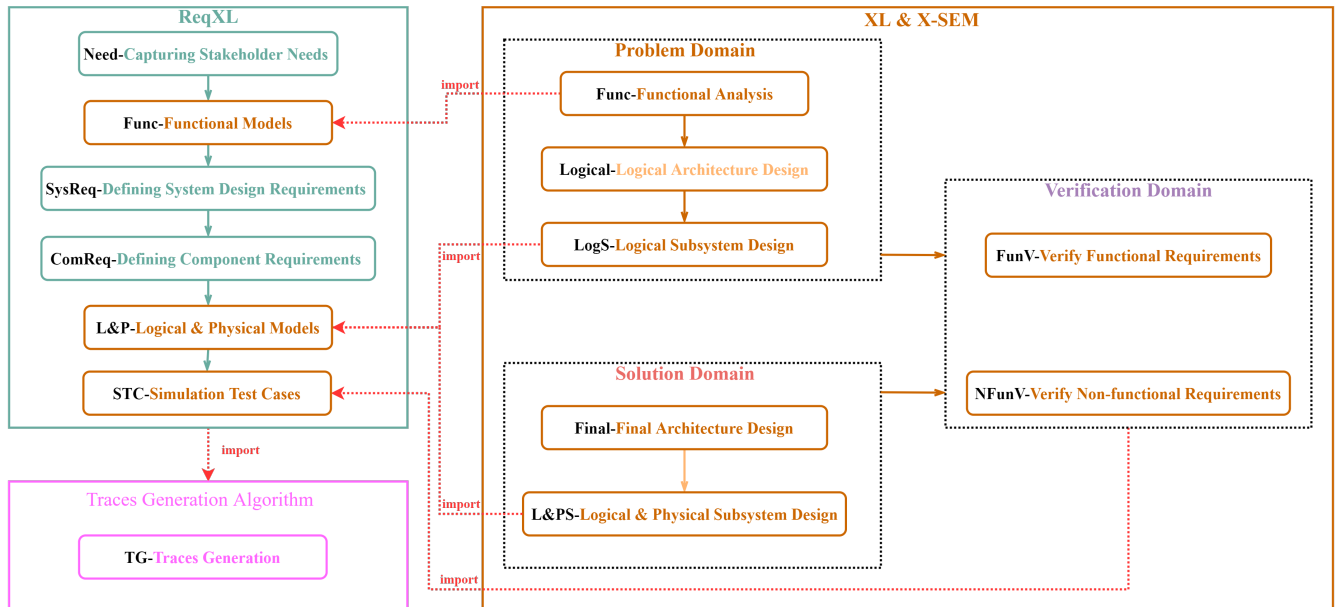


Figure 5. Integrated framework.

Requirements modeling relies on the implementation of ReqXL in the integrated approach. ReqXL offers two views (graphical and text views) for requirements modeling, as illustrated in Figure 6. The graphical view excels in intuitively displaying requirement relationships, providing a structured representation, and facilitating communication among engineers. The text view, on the other hand, allows engineers to easily modify the requirements model and provides a machine-readable format for the defined requirement relationships, which can be used as input for the traceability generation algorithm to infer implicit trace relationships. Additionally, ReqXL ensures the standardized modeling of stakeholder needs and system design requirements. Lastly, the “import” feature of ReqXL allows the functional models and system design models, which are established based on X language, to be imported into the ReqXL-based requirements model. This supports and guides the modeling of traceability relationships.

Requirement verification relies on the collaborative implementation of X language and ReqXL in the integrated approach. X language can integrate the modeling and simulation of both logical and physical models, ensuring that functional and non-functional requirements can be verified collaboratively through the definition of simulation test cases. Additionally, as shown in Figure 6b, ReqXL can define simulation test cases and establish traceability relationships, linking simulation test cases with requirements.

Requirements traceability management relies on the collaborative implementation of X language and ReqXL in the integrated approach. Requirements traceability management includes two aspects: first, maintaining the requirements traceability chain established during requirements modeling and verification; and second, deriving implicit traceability chains based on the established traceability chain and expressing them graphically. As shown in Figure 6b, ReqXL’s “traceability” feature can store the captured traceability relationships. Additionally, the graphical and text views of ReqXL can be converted into each other. Therefore, when changes occur in the requirement relationships, the modifications made in the graphical view simultaneously reflect in the text view. This brings great convenience to modelers for managing and maintaining the requirements

model. Furthermore, ReqXL’s text view also facilitates the automatic capture of implicit traceability chains. The method for acquiring implicit requirement relationships and the automatic generation of traceability relationships will be elaborated in the next subsection.

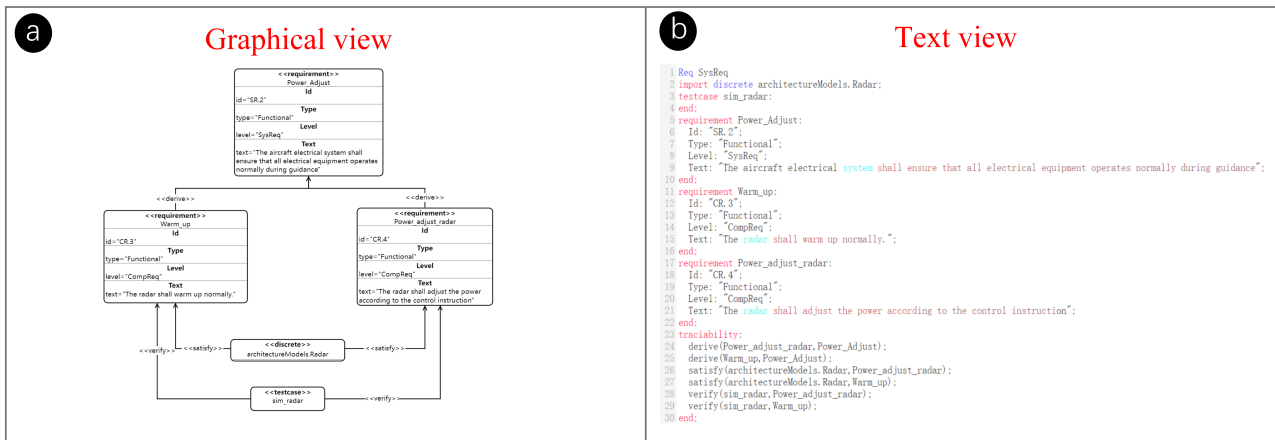


Figure 6. Requirement models based on ReqXL’s graphical and text views.

In summary, the consistency between the requirements model, the functional model, the system design model, and the system simulation model defined by this unified modeling and simulation language based on ReqXL and X language is ensured. At the same time, this integrated approach avoids the establishment of requirement-tracking relationships between different platforms, which is beneficial to improve the efficiency of system development.

4.2. Auto-Generation Method for Requirement Traces

In Figure 7, we depict the overall process for the automated generation of requirement traceability. First, a requirements engineer uses ReqXL to describe a set of requirements either graphically or textually. Then, during the MBSE-based system development process, it is necessary to capture other model elements established based on X language. These model elements are responsible for executing the specified actions within each requirement, particularly in the case of component requirements. Finally, during the system development process, systems engineers need to manually create all possible direct requirement links based on the different requirement relationships defined by ReqXL. These explicitly manually established requirement links are used to infer implicit links between stakeholder needs, system design requirements, system components, and test cases. The auto-generation algorithm for requirement traceability automatically generates these implicit links based on the requirement relationship inference rules and produces a complete graphical requirements traceability model. The following sections will focus on the derivation rules for requirement relationships and the auto-generation algorithm for requirement traceability.

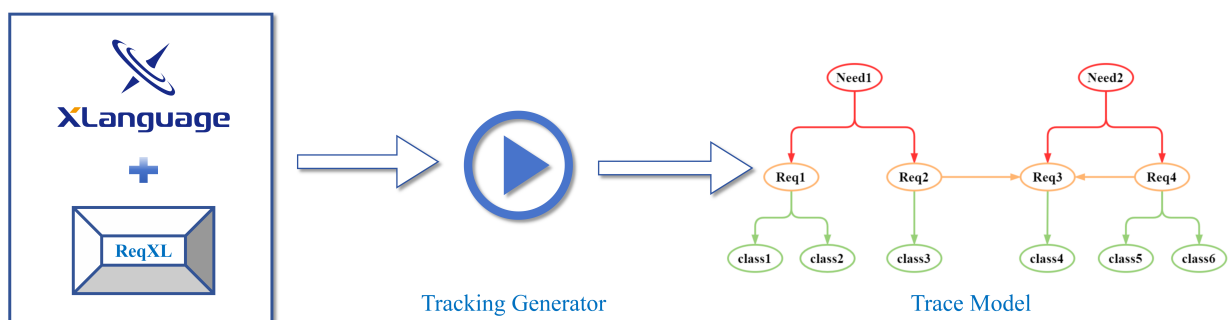


Figure 7. Overall process for generating requirement traceability Links.

4.2.1. Derivation Rules for Requirement Relationships

During the system development process, requirement changes are inevitable. Therefore, establishing effective traceability between stakeholders, stakeholder needs, requirements, and system design is crucial for the rapid iterative development of subsequent systems. In ReqXL, we can establish explicit links during the system development process. Additionally, implicit links need to be derived based on the semantics of requirement traceability relationships. This subsection focuses on various requirement traceability relationships in ReqXL and provides the derivation rules for implicit requirement relationships.

Ten types of requirement relationships are utilized to facilitate requirement traceability in ReqXL. The *source*, *presenter*, and *responsible* relationships belong to the *PreRS* type. The *source* and *presente* relationships are used to establish forward traceability between stakeholder needs, stakeholders, and source of stakeholder needs. The *compose*, *derive*, *refine*, and *trace* relationships belong to either the *TraceBetFun_Req* or *TraceBetReq* types. They are mainly used to establish traceability relationships between stakeholder needs, system design requirements, and system functional elements (use cases, activities). The *satisfy*, *map*, and *verify* relationships belong to the *PostRS* type. The *satisfy* and *map* relationships are used to establish traceability between stakeholder needs or system design requirements and system model elements or specific attributes of model elements. The *verify* relationship is used to establish verification relationships between stakeholder needs or system design requirements and test cases. The following section provides the derivation rules for implicit requirement relationships based on the brief classification.

In ReqXL, forward traceability relationships are primarily used to help system engineers identify the initiators or organizations behind stakeholder needs and the responsible parties for system design requirements. Establishing such explicit traceability relationships allows system developers to effectively correct and re-establish associations with the initiators or organizations behind forward stakeholder needs and the responsible parties for system design requirements when facing requirement changes. Therefore, the core of requirement derivation should focus on the other three types of traceability relationships: *TraceBetFun_Req*, *TraceBetReq*, and *PostRS*.

1. *compose*

The *compose* is primarily used to define the inclusion relationship between requirements. If there are requirements R_1, R_2, \dots, R_n , where $n \geq 2$, and R_1 is a composite requirement while R_2, \dots, R_n are atomic requirements, and R_1 is composed of R_2, \dots, R_n , then there is a *compose* relationship between R_1 and R_2, \dots, R_n . That is, $compose(R_1, R_n), n \geq 2$. According to the definition of the *compose* relationship, the *compose* relationship has the following characteristics:

- (a) Anti-reflexivity: the *compose* relationship is anti-reflexive, meaning there is no decomposition from a requirement to itself. That is,

$$\forall a \in R, \neg compose(a, a) \quad (1)$$

- (b) Anti-symmetry: the *compose* relationship is anti-symmetric, meaning there is no mutual compose between requirements. That is,

$$\forall a, b \in R, compose(a, b) \rightarrow \neg compose(b, a) \quad (2)$$

- (c) Transitivity: the *compose* relationship is transitive. That is,

$$\forall a, b, c \in R, compose(a, b) \wedge compose(b, c) \rightarrow compose(a, c) \quad (3)$$

2. *derive*

The *derive* is commonly used to define an inheritance or generalization between requirements. Given two requirements, R_1 and R_2 , where R_1 is the provider requirement and R_2 is the client requirement, if requirement R_2 is derived from requirement R_1 and the implementation of R_1 is a prerequisite for R_2 , then there exists a *derive*

relationship between R1 and R2. That is, $derive(R2, R1)$. The *derive* relationship also exhibits the following characteristics:

- (a) Anti-reflexivity: the *derive* relationship is anti-reflexive, meaning there is no derivation from a requirement to itself. In other words,

$$\forall a \in R, \neg derive(a, a) \quad (4)$$

- (b) Anti-symmetry: the *derive* relationship is anti-symmetric, meaning there is no mutual derivation between requirements. In other words,

$$\forall a, b \in R, derive(a, b) \rightarrow \neg derive(b, a) \quad (5)$$

- (c) Transitivity: the *derive* relationship is transitive. That is,

$$\forall a, b, c \in R, derive(a, b) \wedge derive(b, c) \rightarrow derive(a, c) \quad (6)$$

3. *refine*

The *refine* relationship is used to define a specialization between a client and a target. Here, the client or provider can be a requirement, use case, or activity. If the provider element is R1 and the client element is R2, and if R2 refines R1 by providing a more specific description, then there exists a *refine* relationship between R1 and R2. That is, $refine(R2, R1)$. The *refine* relationship also exhibits the following characteristics:

- (a) Anti-reflexivity: the *refine* relationship is anti-reflexive, meaning there is no refinement from a requirement to itself. In other words,

$$\forall a \in R, \neg refine(a, a) \quad (7)$$

- (b) Anti-symmetry: the *refine* relationship is anti-symmetric, meaning there is no mutual refinement between requirements. In other words,

$$\forall a, b \in R, refine(a, b) \rightarrow \neg refine(b, a) \quad (8)$$

- (c) Transitivity: the *refine* relationship is transitive. That is,

$$\forall a, b, c \in R, refine(a, b) \wedge refine(b, c) \rightarrow refine(a, c) \quad (9)$$

4. *trace*

The *trace* relationship is a generalized requirement relationship. In ReqXL, we generally recommend using the aforementioned three relationships to establish connections between requirements, as well as between requirements and other model elements. The *trace* relationship is primarily used to establish implicit cross-layer relationships. Typically, the *trace* relationship exhibits the following characteristics:

- (a) When there is a *compose* (*derive*) relationship between a and b , and a *derive* (*compose*) relationship between b and c , a *trace* relationship is established between a and c . In other words,

$$\forall a, b, c \in R, derive(a, b) \wedge compose(c, b) \rightarrow trace(a, c) \quad (10)$$

$$\forall a, b, c \in R, compose(a, b) \wedge derive(c, b) \rightarrow trace(c, a) \quad (11)$$

- (b) When there is a *compose* (*refine*) relationship between requirements a and b , and a *refine* (*compose*) relationship between b and c , a *trace* relationship is established between a and c . In other words,

$$\forall a, b, c \in R, refine(a, b) \wedge compose(c, b) \rightarrow trace(a, c) \quad (12)$$

$$\forall a, b, c \in R, compose(a, b) \wedge refine(c, b) \rightarrow trace(c, a) \quad (13)$$

- (c) When there is a *derive* (*refine*) relationship between requirements a and b , and a *refine* (*derive*) relationship between b and c , a *trace* relationship is established between a and c . In other words,

$$\forall a, b, c \in R, \text{refine}(a, b) \wedge \text{derive}(b, c) \rightarrow \text{trace}(a, c) \quad (14)$$

$$\forall a, b, c \in R, \text{derive}(a, b) \wedge \text{refine}(b, c) \rightarrow \text{trace}(a, c) \quad (15)$$

- (d) When there exist $R1$ and $R2$, $R2$ and $R3, \dots, R_{n-1}$ and R_n , with $n \geq 3$, and the above three types of relationships exist between them without the same type of relationship being consecutive, a *trace* relationship is established between $R1$ and R_n .

5. *satisfy*

The *satisfy* relationship is used to define the fulfillment relationship between system design components and requirements. Similar to the previous three types of requirement relationships, the satisfaction relationship also exhibits anti-reflexivity and anti-symmetry. However, since the satisfaction relationship spans both the requirement layer and the design layer, its transitivity is not considered. Satisfaction relationships are generally established between system component requirements and system design components. Therefore, it is often difficult to observe how stakeholder needs or top-level system design requirements are fulfilled. In ReqXL, we have defined four types of requirement relationships oriented towards *TraceBetFun_Req* and *TraceBetReq*, along with their derivation rules. Consequently, the implicit *satisfy* relationship can also be easily derived. Specifically:

When there is a *compose/derive/refine/trace* relationship between a and b , and a *satisfy* relationship between b and c , a *satisfy* relationship is established between a and c (where the four types of relationships between a and b can be either explicit relationships or implicitly derived relationships). In other words,

$$\forall a, b, c \in R, \text{compose}(a, b) \wedge \text{satisfy}(c, b) \rightarrow \text{satisfy}(c, a) \quad (16)$$

$$\forall a, b, c \in R, \text{derive|trace}(b, a) \wedge \text{satisfy}(c, b) \rightarrow \text{satisfy}(c, a) \quad (17)$$

6. *verify*

The *verify* relationship is used to define the verification relationship between test cases and requirements. The *verify* relationship also exhibits anti-reflexivity and anti-symmetry. The *verify* relationships are generally established between system component requirements and test cases. In ReqXL, we have defined four types of requirement relationships oriented towards *TraceBetFun_Req* and *TraceBetReq*, along with their derivation rules. Consequently, implicit *verify* relationships can also be easily derived. Specifically:

When there is a *compose/derive/refine/trace* relationship between requirements a and b , and a *verify* relationship between b and c , a *verify* relationship is established between a and c (where the four types of relationships between a and b can be either explicit relationships or implicitly derived relationships). In other words,

$$\forall a, b, c \in R, \text{compose}(a, b) \wedge \text{verify}(c, b) \rightarrow \text{verify}(c, a) \quad (18)$$

$$\forall a, b, c \in R, \text{derive|trace}(b, a) \wedge \text{verify}(c, b) \rightarrow \text{verify}(c, a) \quad (19)$$

4.2.2. Auto-Generation Algorithm for Requirement Traces

An auto-generation algorithm for requirement traces based on the ReqXL specification was designed to generate traceability links automatically. It is based on collecting system elements associated with these requirements. In Algorithm 1, the input is a .xl file established using ReqXL. Firstly, the algorithm extracts the sources set S_0 , the stakeholders set S , the responsible stakeholders set RS , the stakeholder needs set N , system design requirement

set R , explicitly defined traceability relationship set T_0 , and model elements E imported from X language based on the established $.xl$ file. Secondly, the algorithm retrieves all possible requirement trace paths from stakeholder needs to system components and test cases through a recursive method based on the acquired requirement traces. Then, based on the existing requirement traces and derivation rules, it iteratively traverses all elements on each path to obtain the implicit requirement traces. Finally, the algorithm returns a set $TM(S_o, S, RS, N, R, E, R_t)$ containing all elements and their traces.

Algorithm 1 Requirement traces generation

Require: $.xl$ files based on ReqXL

Ensure: Trace Model: $TM(S_o, S, RS, N, R, E, R_t)$

```

1:  $S_o = .xl.getAllsources();$ 
2:  $S = .xl.getAllstakeholders();$ 
3:  $RS = .xl.getAllresponsible\_stakeholders();$ 
4:  $N = .xl.getAllstakeholderneeds();$ 
5:  $R = .xl.getAllsystemdesignreqs();$ 
6:  $E = .xl.getAllmodelelements();$ 
7:  $T_0 = .xl.getAlltraces();$  ▷ triad (A,B,relationship)
8: for each  $trace \in T_0$  do ▷ get child and parent nodes for all elements
9:    $trace.A \leftarrow CreateNewChild(B, relationship);$ 
10:   $trace.B \leftarrow CreateNewParent(A, relationship);$ 
11: end for
12: function  $path\_search(node)$  ▷ get all paths for requirement traces by recursion
13:   if  $node.child == NULL$  then
14:      $All\_Path \leftarrow CreateNewPath(path);$ 
15:   else
16:     for each  $child \in node.child$  do
17:        $path \leftarrow CreateNewNode(child);$ 
18:        $path\_search(child);$ 
19:     end for
20:   end if
21: end function
22: for each  $stakeholder\_need \in N$  do ▷ get all paths relevant to stakeholder needs
23:    $path\_search(stakeholder\_need);$ 
24: end for
25: for each  $path \in All\_Path$  do ▷ get all implicit traces
26:   for each  $e \in path$  do
27:     for each  $e' \in path$  do
28:       if The derivation rule holds then
29:          $T_1 \leftarrow CreateNewTrace(e, e');$ 
30:       end if
31:     end for
32:   end for
33: end for
34:  $R_t \leftarrow T_0 \cup T_1;$ 
35: return  $TM(S_o, S, RS, N, R, E, R_t)$ 

```

Currently, some SysML-based approaches require converting SysML diagrams into XML files to extract model elements and requirement relationships, which increases the difficulty of obtaining input information for algorithms. Additionally, some domain-specific modeling language-based methods establish requirement relationships by modeling each requirement individually, which also complicates the extraction of requirement relationships.

As shown in Figure 6b, the ReqXL-based requirement modeling method separates the definition of model elements and requirement relationships, and the custom $.xl$ text file format of ReqXL can be directly used as input for Algorithm 1. Therefore, extracting both model elements and requirement relationships is very straightforward, effectively reducing

the implementation complexity of requirement traceability generation algorithms and increasing the efficiency of automatically generating traceability chains to a certain extent.

4.3. Integrated Platform for ReqXL and X Language

To effectively support system design engineers in integrated requirement modeling, traceability management, and verification, we have customized the development based on our self-developed XLab tool, which is oriented toward X language [18–20]. XLab is a B/S architecture modeling and simulation platform that supports requirement modeling, functional modeling, system architecture design, and simulation verification based on X language and X-SEM. The original requirement modeling functionality of X language was derived from SysML’s requirement diagrams.

Therefore, in XLab, we have extended the requirement diagram to develop graphical modeling functions, text modeling functions, bidirectional conversion between graphical and text representations, and requirement traceability analysis functions oriented towards ReqXL, as shown in Figure 8a,e. From a technical implementation perspective, the development of ReqXL-based graphical sources, line types, and related attributes is achieved using the *LogicFlow* framework. The ReqXL-based text editing, code suggestion, and highlighting functions are implemented using the *CodeMirror* framework. Utilizing the XML intermediate data format, the bidirectional conversion between graphical and textual models based on ReqXL is realized using the *ANTLR4* conversion framework and ReqXL syntax. The requirement traceability analysis function based on ReqXL is implemented using Algorithm 1.

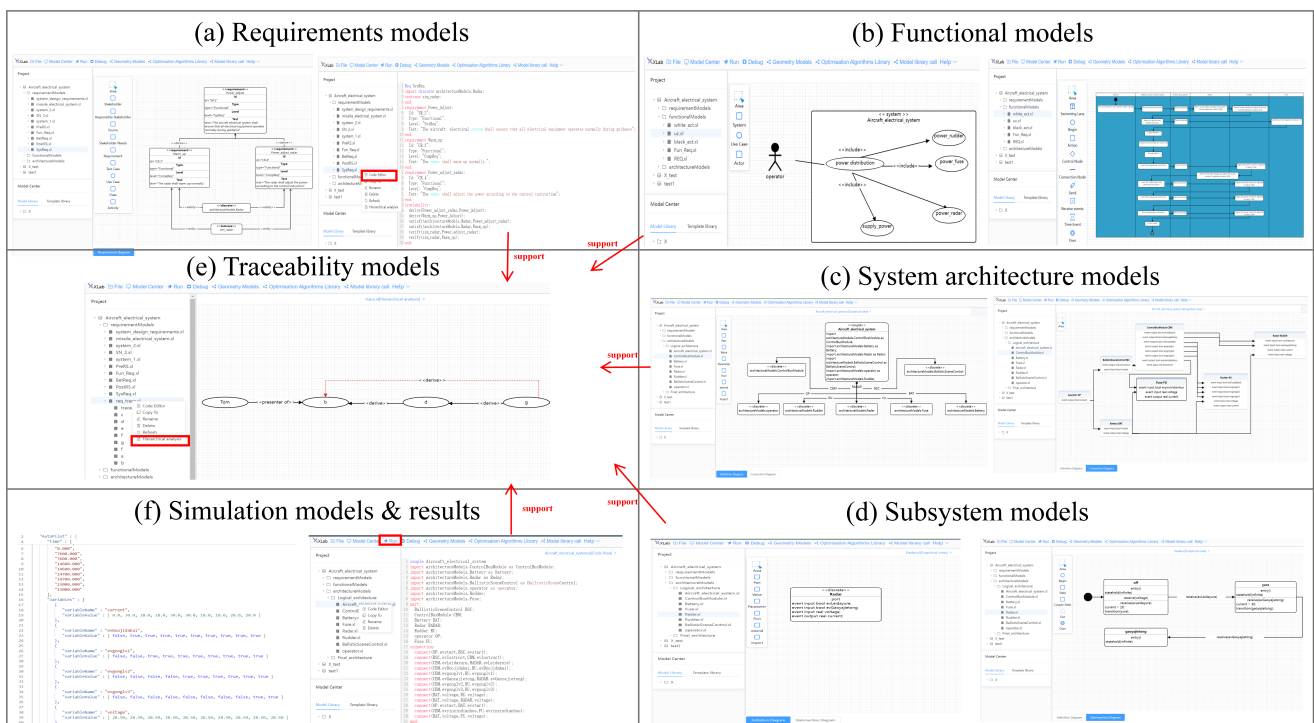


Figure 8. Integrated platform.

As shown in Figure 8, the existing functions of XLab and the customized development functions for ReqXL enable integrated requirement modeling, traceability management, and verification on a unified platform. This approach, based on a unified language and platform, ensures consistency across models at various stages and can effectively enhance system development efficiency.

5. Case Study

The application case focuses on an aircraft electrical system to implement an integrated process of requirements management, traceability, and verification. In this context, the aircraft electrical system employs an MBSE approach based on X language, combined with ReqXL, to achieve an integrated process of requirements traceability management. This process, starting from capturing stakeholder needs and continuing through system verification, demonstrates the superiority of the proposed method.

5.1. Requirement Modeling and Verification

5.1.1. Capturing Stakeholder Needs

In this section, ReqXL is used to describe stakeholder needs. In the design of the aircraft electrical system, we begin with two specified stakeholder needs. We use the acronym SN to denote stakeholder needs. The descriptions are as follows:

SN1: lithium battery supplies power to all electrical equipment, ensuring the normal operation of the aircraft electrical system.

SN2: during the different phases of guidance, the power of all electrical equipment is adjusted according to predetermined instructions.

In Figure 9, ReqXL specifies these stakeholder needs. These needs are proposed by the aircraft electrical system design team and originate from a design document. In ReqXL, forward traceability is captured based on the source and stakeholder attributes. Additionally, stakeholder needs are relatively general, vague, and not directly verifiable. Therefore, we need to transform these stakeholder needs into verifiable system design requirements through X language and its MBSE methodology, X-SEM, to achieve complete system design and closed-loop verification of the top-level stakeholder needs. On this basis, comprehensive traceability management is achieved using ReqXL.

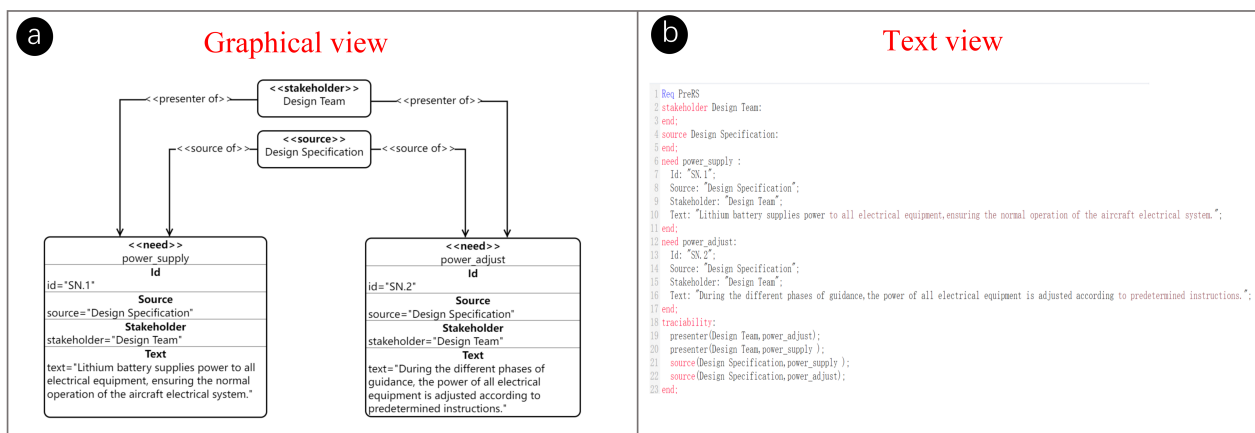


Figure 9. Modeling stakeholder needs and their sources based on ReqXL.

5.1.2. Problem Domain: Black Box Stage

The first step in the black box stage is to transform stakeholder needs into use cases. For the aircraft electrical system, the relevant use case is “power distribution”. Since the operator needs to allocate the appropriate power to various electrical devices, the “power distribution” use case encompasses four sub-use cases: *power_rudder*, *power_radar*, *power_fuse*, and *supply_power*, as illustrated in Figure 10d. Once the use case diagram is established, the system context is also defined, as shown in Figure 10b. Additionally, based on the use case diagram, a black box activity diagram can be constructed to depict the interaction behaviors between the aircraft electrical system and the operator. The black box activity diagram captures the use case scenarios by creating a flow of functional activities between the operator and the aircraft electrical system, as illustrated in Figure 10c. The operator initiates the aircraft electrical system, and upon receiving the signal, the aircraft’s electrical system sequentially completes power startup and allocates power to

each electrical device according to internal commands until the target is hit. By combining the stakeholder needs captured in Figure 10a, the physical parameters (voltage, current, and power) that the aircraft electrical system must meet to satisfy the stakeholder needs can be analyzed and obtained, as shown in Figure 10e.

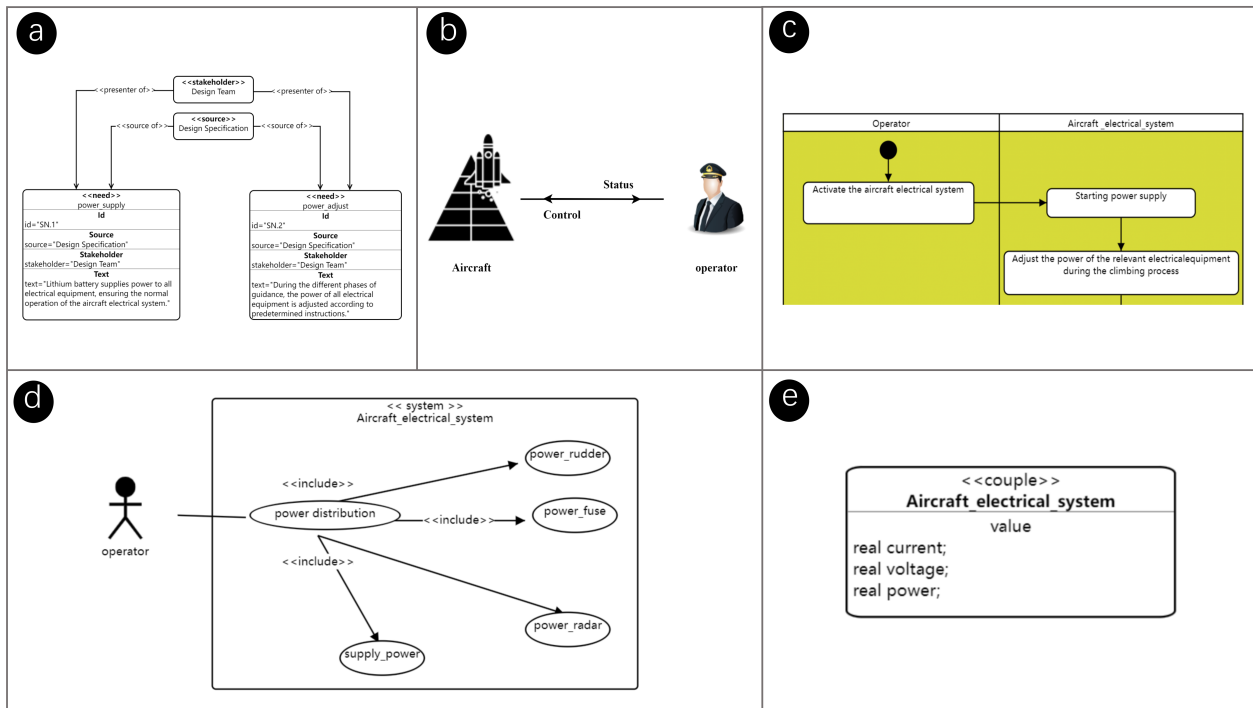


Figure 10. Problem domain with the aircraft electrical system considered as a black box: (a) Stakeholder needs; (b) System context; (c) Use case scenarios; (d) Use cases; (e) Measurements of effectiveness.

In the black box stage, the stakeholder needs for the aircraft electrical system are refined into use cases. Based on the aforementioned analysis, the relationships between stakeholder needs and use cases can be detailed using ReqXL, as illustrated in Figure 11.

```

1 Req Fun_Need
2 import usecase power_rudder;
3 import usecase supply_power;
4 import usecase power_radar;
5 import usecase power_fuse;
6 need power_supply :
7   Id: "SN.1";
8   Source: "Design Specification";
9   Stakeholder: "Design Team";
10  Text: "Lithium battery supplies power to all electrical equipment, ensuring the normal operation of the aircraft electrical system.";
11 end;
12 need power_adjust:
13   Id: "SN.2";
14   Source: "Design Specification";
15   Stakeholder: "Design Team";
16   Text: "During the different phases of guidance, the power of all electrical equipment is adjusted according to predetermined instructions.";
17 end;
18 traciability:
19   refine(supply_power, power_supply );
20   refine(power_fuse, power_adjust);
21   refine(power_radar, power_adjust);
22   refine(power_rudder, power_adjust);
23 end;
    
```

Figure 11. Modeling the relationships between stakeholder needs and functional models based on ReqXL.

5.1.3. Problem Domain: White Box Stage

In the white box stage, it is necessary to refine the black box functional activity flows into a white box implementation to further detail the use cases. Here, the functional activity

logic executed by each subsystem during the aircraft’s flight is established using a white box activity diagram with swimlanes, as illustrated in Figure 12a. The definition of the white box activity diagram not only guides system engineers in defining system design requirements but also helps to outline the logical architecture of the aircraft electrical system.

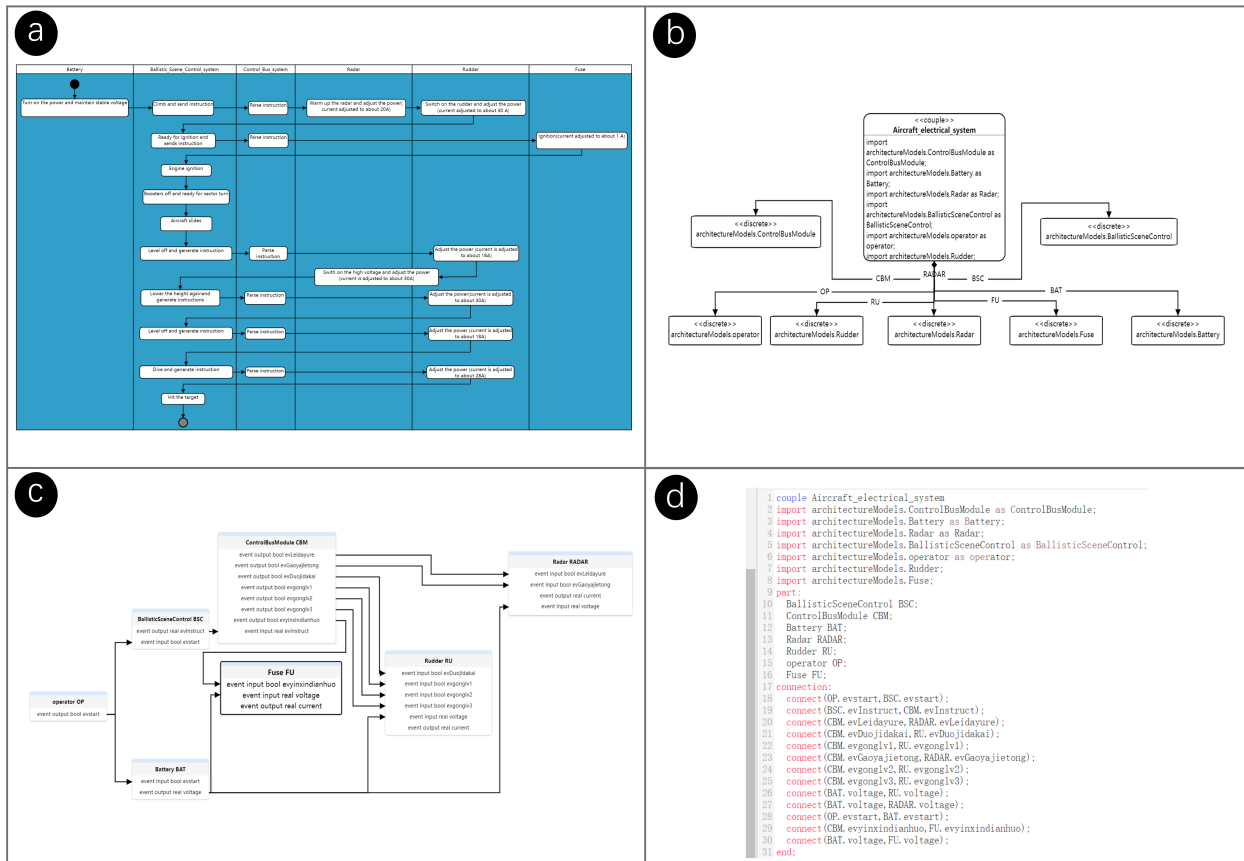


Figure 12. Problem domain with the aircraft electrical system considered as a white box: (a) Functional analysis; (b) Logical architecture definition; (c) Logical subsystems communication; (d) Logical architecture text.

Based on the use cases and functional activity analysis within the problem domain, system design requirements can be derived. We use acronyms to formulate the system design requirements. The detailed descriptions are as follows:

SR1: functional requirement: the aircraft electrical system shall be continuously powered during guidance.

SR2: functional requirement: the aircraft electrical system shall ensure that all electrical equipment operates normally during guidance.

Our goal is to address downstream traceability at a fine-grained level through ReqXL. Therefore, based on the analysis process of the problem domain, component-level requirements are derived. These requirements can be directly traced to individual classes or small groups of classes that contribute to fulfilling the requirement. We use acronyms to specify the component requirements. The detailed descriptions are as follows:

CR.1: design requirement: the SOC of the lithium battery shall be no less than 0 during guidance.

CR.2: design requirement: the voltage of the lithium battery shall be maintained at 27 ± 3 V during guidance.

CR.3: functional requirement: the radar shall warm up normally.

CR.4: functional requirement: the radar shall be on high voltage normally.

CR.5: functional requirement: the radar shall adjust the power according to the control instruction.

CR.6: functional requirement: the rudder shall start up normally.

CR.7: functional requirement: the rudder shall adjust the power according to the control instruction.

CR.8: functional requirement: the fuse shall ignite normally.

CR.9: functional requirement: the fuse shall adjust the power according to the control instruction.

As shown in Figure 13, ReqXL is used to describe use cases, activities, system design requirements, component requirements, and the relationships between them. Next, it is necessary to establish the logical architecture of the aircraft electrical system and the internal behaviors of each logical subsystem to satisfy and verify the component requirements. The logical architecture model of aircraft electrical systems generally only achieves satisfaction and verification of functional requirements. Non-functional requirements such as CR.1 and CR.2 must be addressed and validated in the solution domain. Therefore, in the problem domain, it is necessary to first establish the logical architecture of the aircraft electrical system and utilize ReqXL to achieve satisfaction and verification of functional component requirements.

```

1 Req Fun_Req
2 import usecase power_supply;
3 import usecase power_radar;
4 import usecase power_rudder;
5 import usecase power_fuse;
6 import action w.1;
7 import action w.4;
8 import action w.15;
9 import action w.5;
10 import action w.14;
11 import action w.18;
12 import action w.20;
13 import action w.8;
14 requirement Power_Supply:
15   Id: "SR.1";
16   Type: "Functional";
17   Level: "SysReq";
18   Text: "The aircraft electrical system shall be continuously powered during guidance";
19 end;
20 requirement Power_Adjust:
21   Id: "SR.2";
22   Type: "Functional";
23   Level: "SysReq";
24   Text: "The aircraft electrical system shall ensure that all electrical equipment operates normally during guidance";
25 end;
26 requirement SOC:
27   Id: "CR.1";
28   Type: "Design";
29   Level: "CompReq";
30   Text: "The SOC of the lithium battery shall be no less than 0 during guidance";
31 end;
32 requirement voltage:
33   Id: "CR.2";
34   Type: "Design";
35   Level: "SysReq";
36   Text: "The voltage of the lithium battery shall be maintained at 27±3V during guidance";
37 end;
38 ...
39 traciability:
40   refine(w.1, power_supply);
41   refine(w.4, power_radar);
42   refine(w.15, power_radar);
43   refine(w.5, power_rudder);
44   refine(w.14, power_rudder);
45   refine(w.18, power_rudder);
46   refine(w.20, power_rudder);
47   refine(w.8, power_fuse);
48   derive(SOC, Power_Supply);
49   derive(voltage, Power_Supply);
50   derive(Power_Supply, w.1);
51   derive(Power_Adjust, w.4);
52   derive(Power_Adjust, w.15);
53   derive(Power_Adjust, w.5);
54   derive(Power_Adjust, w.14);
55   derive(Power_Adjust, w.18);
56   derive(Power_Adjust, w.20);
57   derive(Power_Adjust, w.8);
58   derive(Warm_up, Power_Adjust);
59   derive(Power_adjust_radar, Power_Adjust);
60   derive(High_voltage, Power_Adjust);
61   derive(Start, Power_Adjust);
62   derive(Power_adjust_rudder, Power_Adjust);
63   derive(Ignite, Power_Adjust);
64   derive(Power_adjust_fuse, Power_Adjust);
65 end;

```

Figure 13. Modeling the system requirements, component requirements and the relationships between system requirements, component requirements, and functional models based on ReqXL.

In X language, the logical architecture of the aircraft electrical system is designed following a top-down modeling approach, using the *couple* class to define the system's logical architecture. The definition diagram in Figure 12b describes the logical subsystems included in the aircraft system, while the connection diagram in Figure 12c depicts the signal and data interactions among the six subsystems (radar, rudder, fuse, scenario control system, etc.). Additionally, an operator subsystem is added as an external stimulus to the aircraft electrical system. Figure 12d provides a textual representation of the logical architecture.

Based on the determined logical architecture, behavior modeling and simulation of each logical subsystem need to be conducted to satisfy and verify component requirements CR.3 to CR.9. From the descriptions of component requirements CR.3 to CR.9, it can be seen that CR.3 to CR.5 are fulfilled by the radar subsystem, CR.6 to CR.7 are fulfilled by the rudder subsystem, and CR.8 to CR.9 are fulfilled by the fuse subsystem. Here, the radar subsystem is selected as an example. The structure and behavior of the radar subsystem are defined using discrete class in X language. The definition diagram of the discrete class specifies the interaction ports, parameters, and variables of the radar subsystem with other

subsystems, as shown in Figure 14a. The state machine diagram defines the transitions of the radar subsystem's states (off, preheat, and high-voltage activation) according to control commands, as shown in Figure 14b. Figure 14c provides a textual representation of the radar subsystem. The modeling process for the other subsystems is similar to that of the radar subsystem and will not be repeated here.

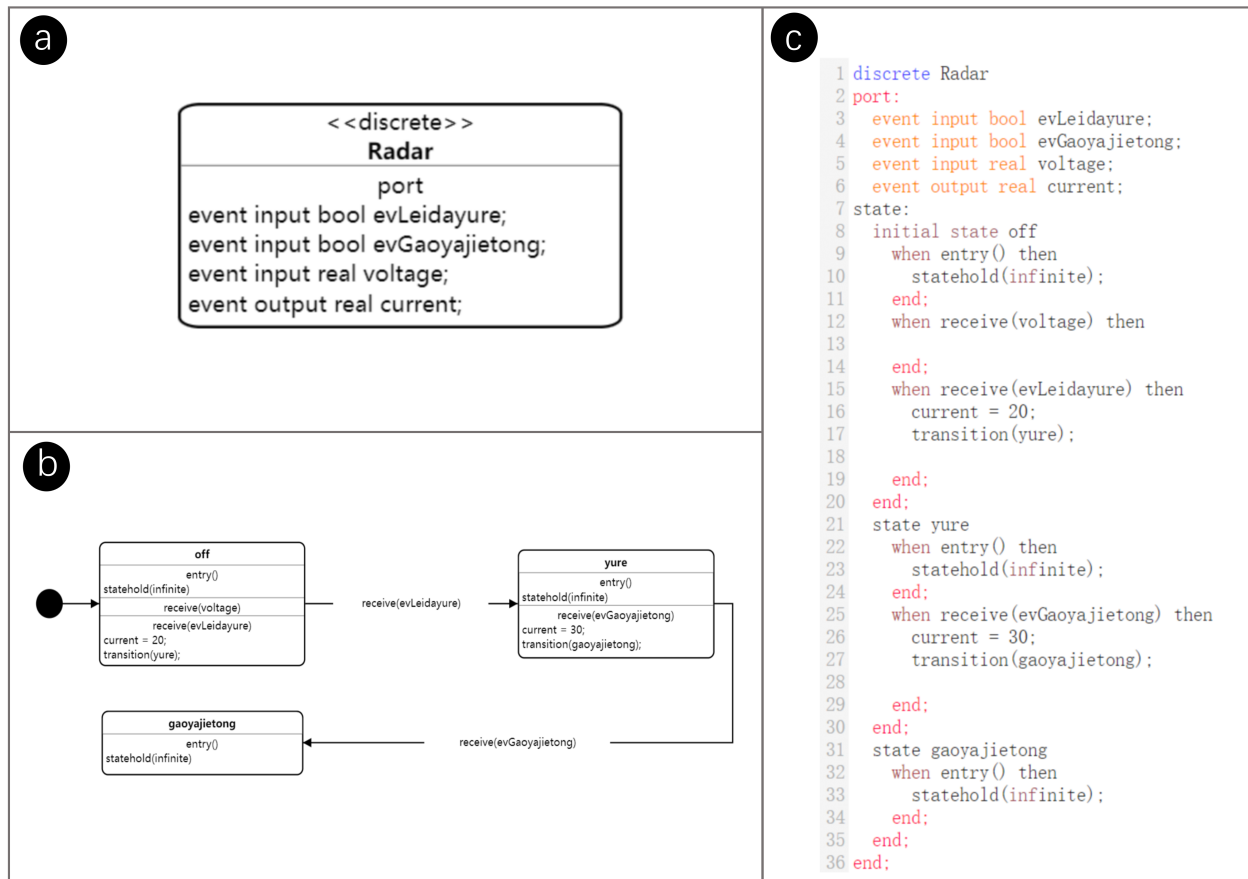


Figure 14. Graphical and textual models for the radar: (a) Definition diagram for the radar; (b) State machine diagram for the radar; (c) Text for the radar.

5.1.4. Verification Domain for Logical Architecture

After completing the logical models for all subsystems, the logical architecture model based on *couple* class and all subsystem models based on *discrete* class are merged into a simulatable project file. Compilation and simulation are then performed using X language compiler and simulator to verify the requirements. Here, three simulation test cases (sim_radar, sim_rudder, and sim_fuse) are defined to verify component requirements CR.3 to CR.9. As shown in Figure 15, sim_radar verifies CR.3 to CR.5, sim_rudder verifies CR.6 to CR.7, and sim_fuse verifies CR.8 to CR.9.

Based on the above analysis, ReqXL is used to describe the relationships between component requirements, relevant subsystems, and simulation test cases, as shown in Figure 16.

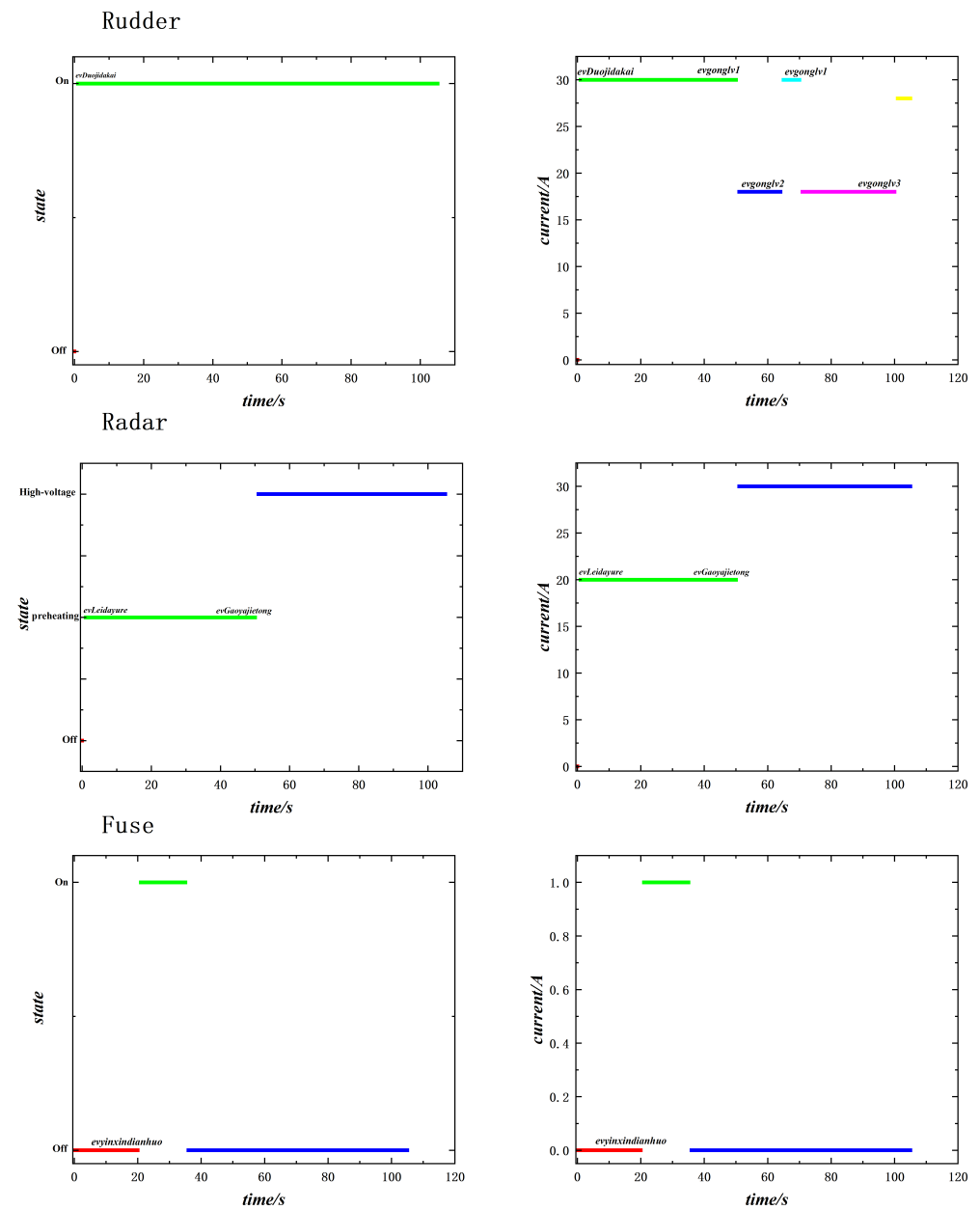


Figure 15. Simulation results of sim_rudder,sim_radar and sim_fuse.

```

1 Req: PostRS
2 import discrete architectureModels.Radar;
3 import discrete architectureModels.Fuze;
4 import discrete architectureModels.Rudder;
5 testcase sim_radar:
6 end;
7 testcase sim_rudder:
8 end;
9 testcase sim_fuse:
10 end;
11 requirement SOC:
12 Id: "CR.1";
13 Type: "Design";
14 Level: "CompReq";
15 Text: "The SOC of the lithium battery shall be no less than 0 during guidance ";
16 end;
17 requirement voltage:
18 Id: "CR.2";
19 Type: "Design";
20 Level: "SysReq";
21 Text: "The voltage of the lithium battery shall be maintained at 27±3V during guidance";
22 end;
23 requirement Warm_up:
24 Id: "CR.3";
25 Type: "Functional";
26 Level: "CompReq";
27 Text: "The radar shall warm up normally.";
28 end;
29 requirement Power_adjust_radar:
30 Id: "CR.4";
31 Type: "Functional";
32 Level: "CompReq";
33 Text: "The radar shall adjust the power according to the control instruction";
34 end;
35 .....
36 traciability:
37 satisfy(architectureModels.Radar, Warm_up);
38 satisfy(architectureModels.Radar, Power_adjust_radar);
39 satisfy(architectureModels.Radar, High_voltage);
40 satisfy(architectureModels.Fuze, Ignite);
41 satisfy(architectureModels.Fuze, Power_adjust_fuse);
42 satisfy(architectureModels.Rudder, Power_adjust_rudder);
43 satisfy(architectureModels.Rudder, Start);
44 verify(sim_radar, High_voltage);
45 verify(sim_radar, Warm_up);
46 verify(sim_radar, Power_adjust_radar);
47 verify(sim_rudder, Power_adjust_rudder);
48 verify(sim_rudder, Start);
49 verify(sim_fuse, Power_adjust_fuse);
50 verify(sim_fuse, Ignite);
51 end;
    
```

Figure 16. Modeling the relationships between component requirements, designed system and simulation test cases based on ReqXL.

5.1.5. Solution Domain

In the previous subsection, component requirements CR.1 and CR.2 are non-functional requirements that serve as design constraints for the battery subsystem. In the solution domain, the final architecture of the aircraft electrical system is realized by considering the physical characteristics of each subsystem based on the logical architecture model. Since component requirements CR.1 and CR.2 are aimed at the battery subsystem, we will illustrate the establishment of the final architecture model of the aircraft electrical system using the battery subsystem as an example. The physical model of the battery system is designed based on the equivalent model framework referenced in the literature [46], with the State of Charge (SOC) calculation using the ampere-hour integration method. In X language, *couple* class is used to model the battery subsystem.

As shown in Figure 17a, the definition diagram of the couple class defines the composition of the battery subsystem. The battery subsystem consists of the SOC calculation module, Resistance–Capacitance (RC) calculation module, voltage calculation module, and current calculation module. The connection diagram of the coupling class defines the interactions between these modules, as shown in Figure 17b. Figure 17c provides a textual representation of the battery subsystem. The construction of other subsystems follows a similar modeling process to that of the battery subsystem, and will not be elaborated here.

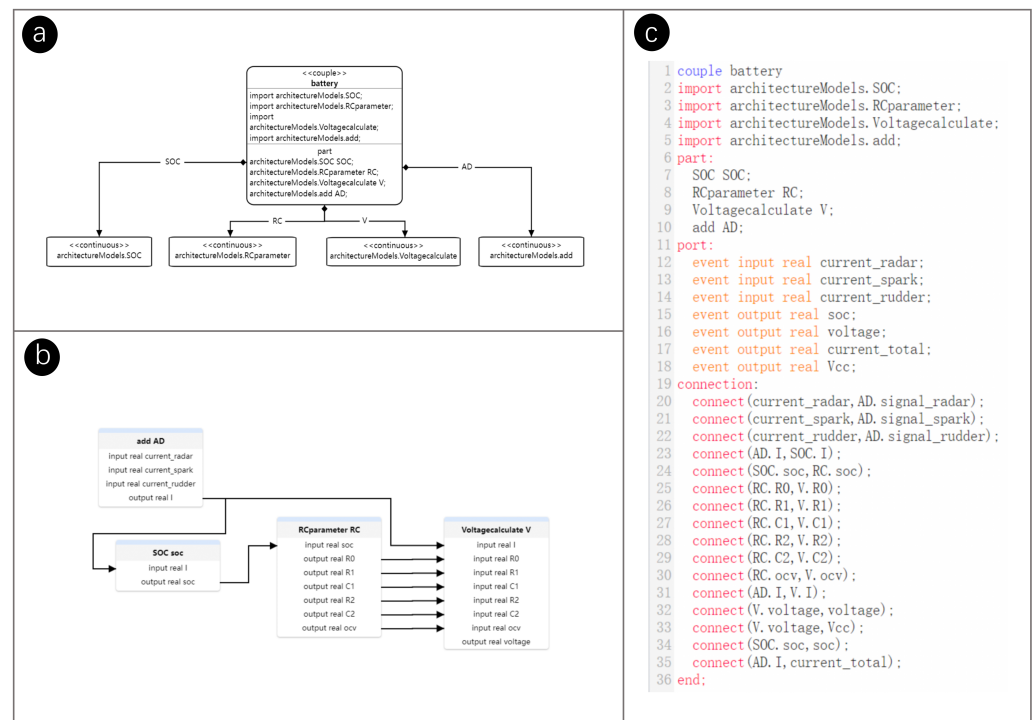


Figure 17. Graphical and textual models for the battery: (a) Definition diagram for the battery; (b) State machine diagram for the battery; (c) Text for the battery.

5.1.6. Verification Domain for Final Architecture

After establishing all subsystems, all subsystem models are merged into a simulatable project file to verify the requirements. Here, we define a simulation test case (sim_battery) to verify component requirements CR.1 and CR.2, as shown in Figure 18.

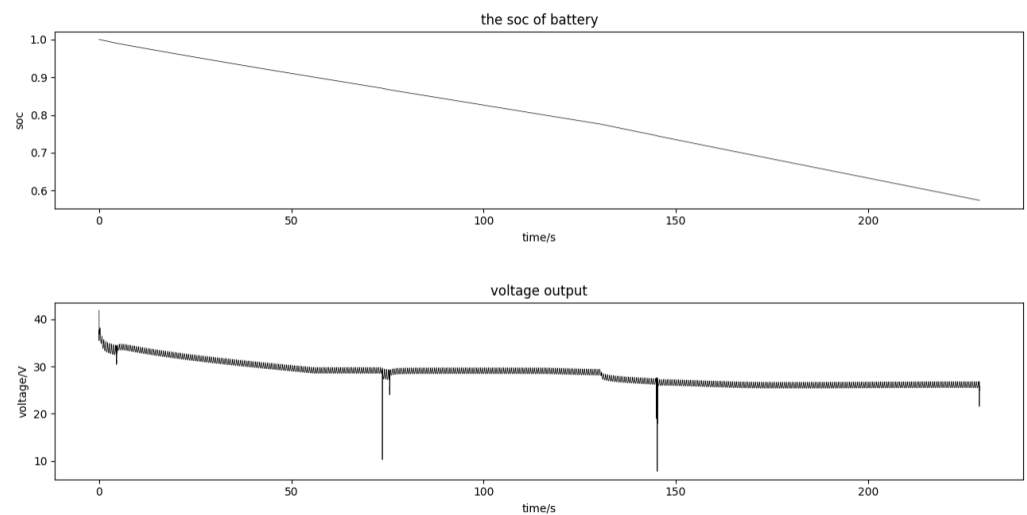


Figure 18. Simulation results of sim_battery.

Based on the above analysis, ReqXL is used to describe the relationships between component requirements, related subsystems, and simulation test cases. As shown in Figure 19, the relationships between component requirements CR.1 and CR.2, related subsystems, and simulation test cases are supplemented.

```

1 Req PostRS
2 import discrete architectureModels.Radar;
3 import discrete architectureModels.Fuze;
4 import discrete architectureModels.Rudder;
5 import compile architectureModels.Final_architecture_battery;
6 testcase sim_radar;
7 end;
8 testcase sim_rudder;
9 end;
10 testcase sim_fuse;
11 end;
12 testcase sim_battery;
13 end;
14 requirement SOC;
15 Id: "CR_1";
16 Type: "Design";
17 Level: "CompReq";
18 Text: "The SOC of the lithium battery shall be no less than 0 during guidance";
19 end;
20 requirement voltage;
21 Id: "CR_2";
22 Type: "Design";
23 Level: "SysReq";
24 Text: "The voltage of the lithium battery shall be maintained at 27±3V during guidance";
25 end;
26 requirement Warm_up;
27 Id: "CR_3";
28 Type: "Functional";
29 Level: "CompReq";
30 Text: "The radar shall warm up normally.";
31 end;
32 requirement Power_adjust_radar;
33 Id: "CR_4";
34 Type: "Functional";
35 Level: "CompReq";
36 Text: "The radar shall adjust the power according to the control instruction";
37 end;
38 .....
39
40 traciability:
41 satisfy(architectureModels.Radar, Warm_up);
42 satisfy(architectureModels.Radar, Power_adjust_radar);
43 satisfy(architectureModels.Radar, High_voltage);
44 satisfy(architectureModels.Fuze, Ignite);
45 satisfy(architectureModels.Fuze, Power_adjust_fuse);
46 satisfy(architectureModels.Rudder, Power_adjust_rudder);
47 satisfy(architectureModels.Rudder, Start);
48 verify(sim_radar, High_voltage);
49 verify(sim_radar, Warm_up);
50 verify(sim_radar, Power_adjust_radar);
51 verify(sim_rudder, Power_adjust_rudder);
52 verify(sim_rudder, Start);
53 verify(sim_fuse, Power_adjust_fuse);
54 verify(sim_fuse, Ignite);
55 verify(sim_battery, voltage);
56 verify(sim_battery, SOC);
57 satisfy(architectureModels.Final_architecture_battery, voltage);
58 satisfy(architectureModels.Final_architecture_battery, SOC);
59 map(architectureModels.Final_architecture_battery#SOC, SOC);
60 map(architectureModels.Final_architecture_battery#voltage, voltage);
61 end;

```

Figure 19. Modeling the relationships between component requirements CR.1 and CR.2, designed system and simulation test cases based on ReqXL.

5.2. Requirement Traceability Management

While completing the modeling and verification of the requirements for the aircraft's electrical system, the comprehensive requirements model (including requirement definitions and relationships) has been progressively defined based on ReqXL, as illustrated in Figures 9, 11, 13, 16 and 19. When these requirements models are integrated, they form a complete requirements model. This requirements model encompasses all model elements and their explicit traceability links, from the source of the requirements to their verification.

At this stage, we can use Algorithm 1 to produce explicit and inferred links between stakeholder needs for the aircraft electrical system, system design requirements, system design components, and simulation test cases. After generating potential traceability links between all model elements, we can describe the created traceability links using a graphical model. In XLab, by right-clicking the corresponding ReqXL file and selecting the hierarchy analysis button, the visualization of the requirements traceability links can be achieved.

Here, we select the stakeholder need SN.1 to demonstrate the effectiveness of the Algorithm 1. As shown in Figure 20, stakeholder need SN.1 can be effectively traced back to its source and to the corresponding stakeholders. Additionally, using the design method-

ology based on X language and X-SEM, it can be clearly shown how SN.1 establishes traceability relationships with system design requirement SR.1 and component requirements CR.1 and CR.2. In the figure, black links represent explicit links, while red links are automatically generated by Algorithm 1, associating the top-level stakeholder need SN.1 with the designed battery subsystem and the simulation test case sim_battery. This is very useful for detecting the impact of requirements changes on all related requirements and model elements.

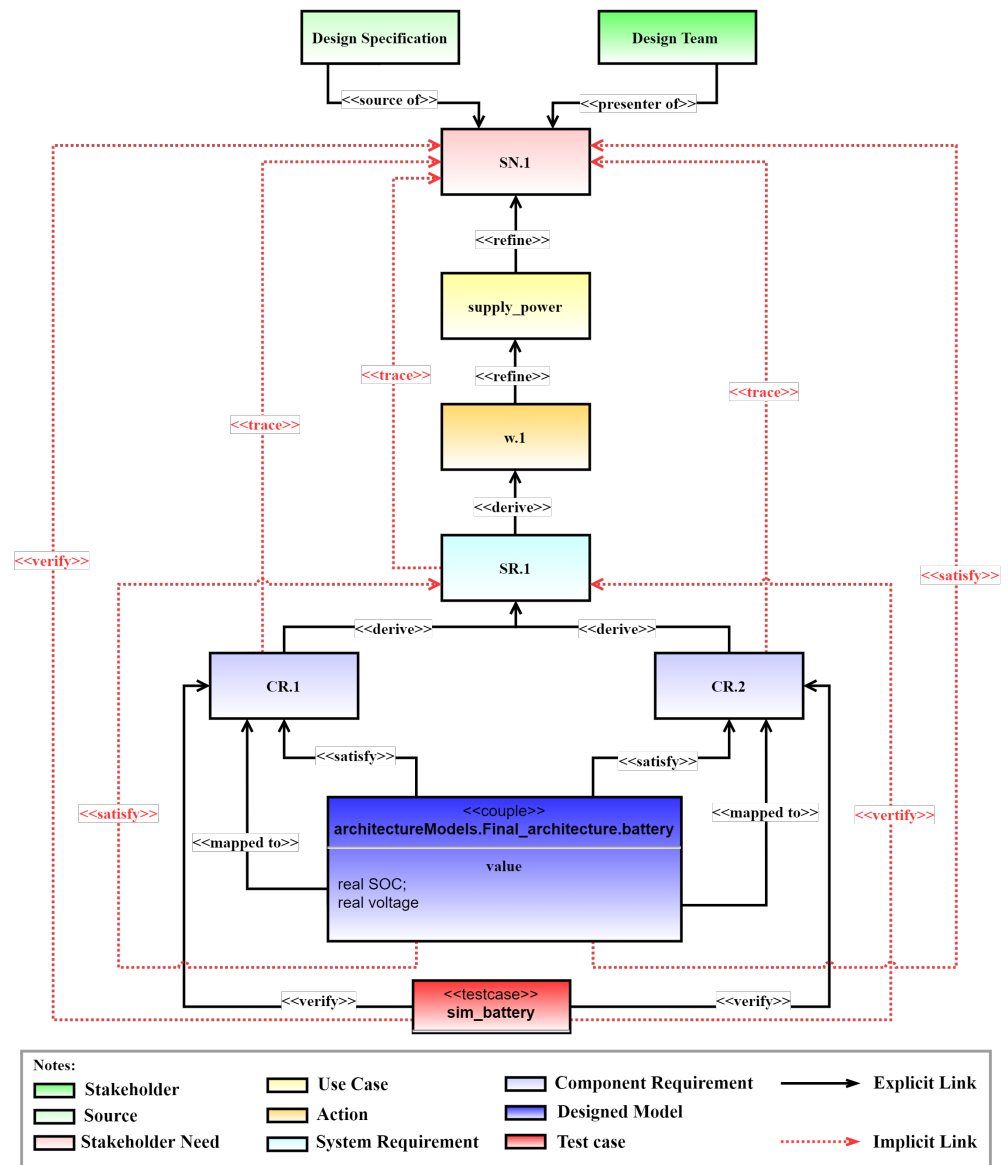


Figure 20. The traces model for SN.1.

5.3. Discussion

Based on the integrated framework proposed in Section 4.1, an integrated process for requirements modeling, traceability management, and verification of the aircraft electrical system was achieved by combining ReqXL with the previously developed X language and methodology, X-SEM. Specifically, during the system design process, stakeholder needs, corresponding stakeholders, and the sources of these needs are initially defined using ReqXL. Subsequently, under the guidance of X-SEM, comprehensive traceability links are progressively established between stakeholder needs, use cases, activities, system-level

requirements, component-level requirements, design components, and simulation test cases through the integration of the X language with ReqXL. Finally, the automatic traceability process is executed using the proposed Algorithm 1. When stakeholder needs change, the predefined traceability links, along with the implicit traceability links derived from Algorithm 1, facilitate the rapid identification and localization of how these changes impact the system's functionality, architectural design, and verification process. This integrated approach significantly enhances the development efficiency of the aircraft electrical system.

Based on previous experience, it appears that integrating ReqXL with SysML can, to some extent, achieve an integrated process for requirements modeling, traceability management, and verification of aircraft electrical systems. However, as previously mentioned, SysML cannot describe the system's physical characteristics and perform simulations. This limitation results in the inability to model and simulate the physical architecture of the aircraft within the solution and verification domains, such as the battery subsystem. Consequently, the integration of ReqXL and SysML struggles to establish complete traceability links, making it difficult to achieve an integrated approach to requirements modeling, traceability management, and verification.

A feasible approach is to integrate ReqXL, SysML, and simulation languages such as Modelica and MATLAB/Simulink to realize an integrated process for the aircraft electrical system's requirements modeling, traceability management, and verification. However, this multi-language and multi-platform integration approach faces challenges in maintaining consistency between requirements and other models, in addition to the high learning curve for modelers. In contrast, the X language is a unified modeling and simulation language that combines SysML's system modeling capabilities with Modelica's multi-domain unified modeling and simulation capabilities. Therefore, the integration of ReqXL with the X language effectively avoids the aforementioned drawbacks.

It is worth noting that the integrated approach proposed in this paper is a general method designed for complex multidisciplinary products and is not limited to product development in the aerospace field. In the future, we plan to consider applying this integrated method to industries such as automotive and marine.

6. Conclusions and Future Works

Currently, the methods for requirements modeling, traceability management, and verification in Model-Based Systems Engineering (MBSE) encounter several challenges. First, the MBSE-oriented modeling language, SysML, lacks the rich syntax and semantics necessary to precisely model requirements and fully capture the traceability from the source of requirements to system components and their specific attributes. Second, the absence of capabilities within SysML to describe physical characteristics and perform simulations necessitates the integration of external simulation languages, such as Modelica and MATLAB/Simulink, to achieve a cohesive process for requirements modeling, traceability management, and verification. However, this multi-language and multi-tool integration approach presents significant challenges in maintaining consistency between requirements and system design models, as well as ensuring traceability during changes to requirements. Furthermore, current approaches to requirement traceability are predominantly manual, lacking an automated method for effective traceability.

To address the aforementioned issues, this paper proposes an integrated approach for modeling requirements, managing traceability, and verification within an MBSE environment, building on a previously introduced unified modeling and simulation language, X language.

The approach begins by defining a requirement modeling specification called ReqXL, tailored to the requirements diagram of the X language and the unique characteristics of MBSE development. ReqXL is structured to depict stakeholder needs, system design requirements, and to establish syntax and semantics necessary for generating traceability links during the MBSE development process. Based on ReqXL, it is possible to achieve precise modeling of requirements. Additionally, it ensures the modeling of complete

traceability relationships from the source of the requirements to the system components and their attributes. Subsequently, the paper outlines the derivation rules for requirement traceability based on ReqXL and proposes an algorithm for automated traceability generation. This algorithm can automatically generate the upstream and downstream traceability chains for selected requirements. This ensures that when requirements change, it automatically identifies how the changes impact the system's functionality, architectural design, and verification process. This study also proposes an integrated framework that achieves unified requirements modeling, traceability, and verification by combining ReqXL with the previously introduced X language and the X-SEM methodology. This integrated approach effectively ensures the consistency of models across different levels during the system development process through language-level unification. Additionally, it facilitates more convenient automatic traceability and verification of requirements.

Currently, we have implemented custom development functions for requirements modeling and automatic traceability analysis using ReqXL on the XLab integrated modeling and simulation platform tailored for the X language. The effectiveness of this approach has been demonstrated through a case study involving the development of an aircraft electrical system.

Although ReqXL is an extension of the requirements modeling specification based on the X language, integrating ReqXL with SysML and its related software tools, such as Rhapsody and Cameo System Modeler, holds practical significance to a certain extent.

In the future, we will explore the integration of ReqXL with existing software tools that support SysML. Additionally, we will focus on achieving the synergy between MBSE and Multidisciplinary Design and Optimization (MDO) by combining X language with ReqXL and its defined optimization role attributes. This approach aims to create a cohesive framework for requirements, system design, verification, and optimization, utilizing a unified language and software to enhance the development efficiency of complex systems. Furthermore, the textual representations of X language and ReqXL provide opportunities for the automatic generation of requirements models and system architecture models based on a large language model.

Author Contributions: All authors contributed to the study conception and design. P.G.: Methodology, Writing—Original draft, Writing—review and editing, Project administration. Y.Z.: Methodology, Software. Z.C.: Visualization, Investigation. C.Z.: Conceptualization, Supervision. K.X.: Conceptualization, Data curation. Z.W.: Validation, Data curation. L.Z.: Supervision, Resources, Funding acquisition. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the National Key R&D Program of China, No. 2023YFB3308201.

Data Availability Statement: The data that support the findings of this study are available upon request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kotonya, G.; Sommerville, I. *An Introduction to Requirements Engineering*; Pearson Education: Princeton, NJ, USA, 1998; Volume 25, p. 2010.
2. Sage, A.P.; Rouse, W.B. *Handbook of Systems Engineering and Management*; John Wiley & Sons: Hoboken, NJ, USA, 2014.
3. Clancy, T. *The Chaos Report*; The Standish Group: Centerville, MA, USA, 1995.
4. Efremov, A.A.; Gaydamaka, K.I. IncoSE guide for writing requirements. Translation experience, adaptation perspectives. In Proceedings of the CEUR Workshop Proceedings, Como, Italy, 9–11 September 2019; pp. 9–11.
5. Tufail, H.; Masood, M.F.; Zeb, B.; Azam, F.; Anwar, M.W. A systematic review of requirement traceability techniques and tools. In Proceedings of the 2017 2nd International Conference on System Reliability and Safety (ICSRS), Milan, Italy, 20–22 December 2017; pp. 450–454.
6. Madni, A.M.; Sievers, M. Model-based systems engineering: Motivation, current status, and research opportunities. *Syst. Eng.* **2018**, *21*, 172–190. [[CrossRef](#)]
7. Zeigler, B.P. DEVS and MBSE: A review. *Int. J. Model. Simul. Sci. Comput.* **2022**, *13*, 2230001. [[CrossRef](#)]
8. Friedenthal, S.; Moore, A.; Steiner, R. OMG systems modeling language (OMG SysML) tutorial. *INCOSE Intl. Symp.* **2006**, *9*, 65–67. [[CrossRef](#)]

9. Génova, G.; Fuentes, J.M.; Llorens, J.; Hurtado, O.; Moreno, V. A framework to measure and improve the quality of textual requirements. *Requir. Eng.* **2013**, *18*, 25–41. [[CrossRef](#)]
10. Carson, R.S. Implementing Structured Requirements to Improve Requirements Quality. *INCOSE Int. Symp.* **2015**, *25*, 54–67. [[CrossRef](#)]
11. Boggero, L.; Ciampa, P.D.; Nagel, B. An MBSE architectural framework for the agile definition of system stakeholders, needs and requirements. In Proceedings of the AIAA Aviation 2021 Forum, Virtual Event, 2–6 August 2021; p. 3076.
12. Hull, E.; Jackson, K.; Dick, J.; Hull, E.; Jackson, K.; Dick, J. DOORS: A tool to manage requirements. *Requir. Eng.* **2002**, 187–204.
13. de Gea, J.M.C.; Nicolás, J.; Alemán, J.L.F.; Toval, A.; Ebert, C.; Vizcaíno, A. Requirements engineering tools. *IEEE Softw.* **2011**, *28*, 86–91. [[CrossRef](#)]
14. Haidrar, S.; Anwar, A.; Roudies, O. A SysML-based Approach to manage stakeholder requirements traceability. In Proceedings of the 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), Hammamet, Tunisia, 30 October–3 November 2017; pp. 202–207.
15. Haidrar, S.; Anwar, A.; Bruel, J.M.; Roudies, O. A Domain-Specific Language to manage Requirements Traceability. *J. Softw.* **2018**, *13*, 460–480. [[CrossRef](#)]
16. Zeigler, B.P.; Mittal, S.; Traore, M.K. MBSE with/out Simulation: State of the Art and Way Forward. *Systems* **2018**, *6*, 40. [[CrossRef](#)]
17. Lemazurier, L.; Chapurlat, V.; Grossetête, A. An MBSE approach to pass from requirements to functional architecture. *IFAC-PapersOnLine* **2017**, *50*, 7260–7265. [[CrossRef](#)]
18. Zhang, L.; Ye, F.; Xie, K.; Gu, P.; Wang, X.; Laili, Y.; Zhao, C.; Zhang, X.; Chen, M.; Lin, T.; et al. An integrated intelligent modeling and simulation language for model-based systems engineering. *J. Ind. Inf. Integr.* **2022**, *28*, 100347. [[CrossRef](#)]
19. Zhang, L.; Ye, F.; Laili, Y.; Xie, K.; Gu, P.; Wang, X.; Zhao, C.; Zhang, X.; Chen, M. X language: An integrated intelligent modeling and simulation language for complex products. In Proceedings of the 2021 Annual Modeling and Simulation Conference (ANNSIM), Fairfax, VA, USA, 19–22 July 2021; pp. 1–11.
20. Gu, P.; Chen, Z.; Zhang, L.; Zhang, Y.; Xie, K.; Zhao, C.; Ye, F.; Tao, Y. X-SEM: A modeling and simulation-based system engineering methodology. *J. Manuf. Syst.* **2024**, *74*, 198–221. [[CrossRef](#)]
21. Cooper, K.; Ito, M. 1.6. 2 formalizing a structured natural language requirements specification notation. In Proceedings of the INCOSE International Symposium, Las Vegas, NV, USA, 28 July–1 August 2002; Volume 12, pp. 1025–1032.
22. Garcia, I.; Pacheco, C.; León, A.; Calvo-Manzano, J.A. A serious game for teaching the fundamentals of ISO/IEC/IEEE 29148 systems and software engineering–Lifecycle processes–Requirements engineering at undergraduate level. *Comput. Stand. Interfaces* **2020**, *67*, 103377. [[CrossRef](#)]
23. Friedenthal, S.; Moore, A.; Steiner, R. *A Practical Guide to SysML: The Systems Modeling Language*; Morgan Kaufmann: London, UK, 2014.
24. Bernard, Y. Requirements management within a full model-based engineering approach. *Syst. Eng.* **2012**, *15*, 119–139. [[CrossRef](#)]
25. Salado, A.; Wach, P. Constructing true model-based requirements in SysML. *Systems* **2019**, *7*, 19. [[CrossRef](#)]
26. Rahman, A.; Amyot, D. A DSL for importing models in a requirements management system. In Proceedings of the 2014 IEEE 4th International Model-Driven Requirements Engineering Workshop (MoDRE), Karlskrona, Sweden, 25 August 2014; pp. 37–46.
27. Liumeng, D.; Guohua, S.; Zhiqiu, H.; Fei, W.; Xiaoyu, G. Extended SysML for Supporting Requirements Trace Model Automatic Generation. *J. Front. Comput. Sci. Technol.* **2019**, *13*, 950.
28. Haidrar, S.; Bencharqui, H.; Anwar, A.; Bruel, J.M.; Roudies, O. REQDL: A requirements description language to support requirements traces generation. In Proceedings of the 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW), Lisbon, Portugal, 4–8 September 2017; pp. 26–35.
29. Taromirad, M.; Paige, R.F. Agile requirements traceability using domain-specific modelling languages. In Proceedings of the 2012 Extreme Modeling Workshop, Innsbruck, Austria, 1–5 October 2012; pp. 45–50.
30. Chandrasekaran, B.; Josephson, J.R.; Benjamins, V.R. What are ontologies, and why do we need them? *IEEE Intell. Syst. Their Appl.* **1999**, *14*, 20–26. [[CrossRef](#)]
31. Adithya, V.; Deepak, G. OntoReq: An ontology focused collective knowledge approach for requirement traceability modelling. In Proceedings of the European, Asian, Middle Eastern, North African Conference on Management & Information Systems, İstanbul, Turkey, 19–20 March 2021; pp. 358–370.
32. Murtazina, M.S.; Avdeenko, T. An ontology-based approach to support for requirements traceability in agile development. *Procedia Comput. Sci.* **2019**, *150*, 628–635. [[CrossRef](#)]
33. Jinzhi, L.; Zhaorui, Y.; Xiaochen, Z.; Jian, W.; Dimitris, K. Exploring the concept of Cognitive Digital Twin from model-based systems engineering perspective. *Int. J. Adv. Manuf. Technol.* **2022**, *121*, 5835–5854. [[CrossRef](#)]
34. Wu, S.; Wang, G.; Lu, J.; Hu, Z.; Yan, Y.; Kiritsis, D. Design ontology for cognitive thread supporting traceability management in model-based systems engineering. *J. Ind. Inf. Integr.* **2024**, *40*, 100619. [[CrossRef](#)]
35. Wang, H.; Zhong, D.; Zhao, T.; Ren, F. Integrating model checking with SysML in complex system safety analysis. *IEEE Access* **2019**, *7*, 16561–16571. [[CrossRef](#)]
36. Staskal, O.; Simac, J.; Swayne, L.; Rozier, K.Y. Translating sysml activity diagrams for nuxmv verification of an autonomous pancreas. In Proceedings of the 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), Los Alamitos, CA, USA, 27 June–1 July 2022; pp. 1637–1642.

37. Hu, J.; Chen, S.; Chen, D.; Kang, J.; Wang, H. Model-based safety analysis for an aviation software specification. *Int. J. Perform. Eng.* **2020**, *16*, 238.
38. Nigischer, C.; Bougain, S.; Riegler, R.; Stanek, H.P.; Grafinger, M. Multi-domain simulation utilizing SysML: State of the art and future perspectives. *Procedia CIRP* **2021**, *100*, 319–324. [[CrossRef](#)]
39. Paredis, C.J.; Bernard, Y.; Burkhart, R.M.; de Koning, H.P.; Friedenthal, S.; Fritzson, P.; Rouquette, N.F.; Schamai, W. An overview of the SysML-modelica transformation specification. In Proceedings of the INCOSE International Symposium, Chicago, IL, USA, 12–15 July 2010; Volume 20, pp. 709–722.
40. Shuhua, Z.; Yue, C.; Zheng, Z.; Yusheng, L. System design and simulation integration for complex mechatronic products based on SysML and modelica. *J.-Comput.-Aided Des. Comput. Graph.* **2018**, *30*, 728–738.
41. Elmqvist, H.; Gaucher, F.; Mattsson, S.E.; Dupont, F. State machines in modelica. In Proceedings of the 9th International Modelica Conference, Munich, Germany, 3–5 September 2012; pp. 3–5.
42. Palachi, E.; Cohen, C.; Takashi, S. Simulation of cyber physical models using SysML and numerical solvers. In Proceedings of the 2013 IEEE International Systems Conference (SysCon), Orlando, FL, USA, 15–18 April 2013; pp. 671–675.
43. Xie, K.; Zhang, L.; Laili, Y.; Wang, X. XDEVs: A hybrid system modeling framework. *Int. J. Model. Simul. Sci. Comput.* **2022**, *13*, 2243001. [[CrossRef](#)]
44. Gu, P.; Zhang, L.; Chen, Z.; Ye, J. Collaborative Design and Simulation Integrated Method of Civil Aircraft Take-off Scenarios Based on X Language. *J. Syst. Simul.* **2022**, *34*, 929–943.
45. Yi, G.; Yi, L.; Zhang, S. A multidisciplinary design method and application for complex systems. *Int. J. Model. Simul. Sci. Comput.* **2023**, *14*, 2350015. [[CrossRef](#)]
46. Huang, M.; Zhao, J. Research on Constant Power Charging and Discharging of Battery Based on LCL Filter. In *Information Technology and Intelligent Transportation Systems*; IOS Press: Beijing, China, 2017; pp. 193–202.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.