Exploiting augmented intelligence in the modeling of safety-critical autonomous systems

Zhibin Yang, Yang Bao, Yongqiang Yang, Zhiqiu Huang, Jean-Paul Bodeveix, M Filali, Zonghua Gu

Under consideration for publication in Formal Aspects of Computing

Exploiting Augmented Intelligence in the Modeling of Safety-Critical Autonomous Systems

Zhibin Yang[1], Yang Bao[1], Yongqiang Yang[1], Zhiqiu Huang[1], Jean-Paul Bodeveix[2], Mamoun Filali[2] and Zonghua Gu[3]

[1]School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
[2]IRIT-University of Toulouse, Toulouse, France
[3]Department of Applied Physics and Electronics, Umea University, Sweden

Abstract.

Machine Learning (ML) is used increasingly in safety-critical systems to provide more complex autonomy to make the system to do decisions by itself in uncertain environments. Using ML to learn system features is fundamentally different from manually implementing them in conventional components written in source code. In this paper, we make a first step towards exploring the architecture modeling of safety-critical autonomous systems which are composed of conventional components and ML components, based on natural language requirements. Firstly, Augmented Intelligence (AuI) for restricted natural language requirement modeling is proposed. In that, several AI technologies such as natural language processing and clustering are used to recommend candidate terms to the glossary, as well as machine learning is used to predict the category of requirements. The glossary including data dictionary and domain glossary and the category of requirements will be used in the restricted natural language requirement specification method RNLReq, which is equipped with a set of restriction rules and templates to structure and restrict the way how users document requirements. Secondly, automatic generation of SysML architecture models from the RNLReq requirement specifications is presented. Thirdly, the prototype tool is implemented based on Papyrus. Finally, it presents the evaluation of the proposed approach using an industrial Autonomous Guidance, Navigation and Control (AGNC) case study.

Keywords: Safety-critical autonomous system; Augmented intelligence; Restricted natural language requirements; Natural language processing; Machine learning; SysML

Correspondence and offprint requests to: Zhibin Yang e-mail: yangzhibin168@163.com

1. Introduction

Safety-critical systems (SCS) are systems whose failure could result in loss of life, significant property damage, or damage to the environment. There are many well-known examples in different domains such as aircraft flight control, space missions, and nuclear systems. The SCS has the characteristics of high reliability, high safety, and strong real-time.

In the last decade, Artificial Intelligence (AI) is used increasingly in safety-critical systems, especially the aerospace engineering [JGMG19][TG18]. While a minimum level of autonomy is required for every aerospace system, past experiences have shown that it requires more complex autonomy to make the system to do decisions by itself in uncertain environments. For instance, deep space missions always need autonomous operations to deal with long communication delays and resource-constrained environments. There have been a number of examples of autonomous operations, such as vision-based sensing, attitude control [DIP19], communication with Earth, docking, soft-landing, anomaly detection[JGMG19], fault detection isolation and recovery (FDIR)[JGMG19], etc., which can be implemented as machine learning components (called as ML components). Using ML to learn system features is fundamentally different from manually implementing them in conventional components written in source code. Compared to human-readable source code, ML components are intrinsically hard to interpret. Thus, there has been a growing concern on the risks of using ML. Trustworthiness is becoming increasingly relevant as humans are progressively sidelined from the decision/control loop of intelligent and learning-enabled machines [Sif19, ESA20]. There are several categories for classifying and discussing AI/ML trustworthiness, such as modeling [Sif19], specification, formal verification of AI systems [KHI$^+$19, LAL$^+$19, AAHR16], testing [HSNB20], runtime monitoring, process assurance, and certification [ESA20], and so on. For instance, [LAL$^+$19] surveys the methods that have emerged recently for formal verifying whether a particular deep neural network satisfies certain input-output properties. More recently, European Union Aviation Safety Agency (EASA) gave an AI Roadmap [ESA20] to discuss the implication of AI on aviation systems. They also proposed four aspects such as learning assurance, AI explainability, AI safety risk mitigation, and trustworthiness analysis, to have an importance in gaining confidence in the trustworthiness of an AI/ML application. [FGH$^+$20] shows how the probabilistic model checker PRISM is used for optimal strategy synthesis for a sequence of scenarios relevant to UAVs (Unmanned Aerial Vehicle) and potentially other autonomous agent systems. Ryan Kirwan et al [KMP16] proposed model checking learning agent systems using Promela with embedded C code and abstraction, to help ensure the safety of autonomous systems.

In this paper, we make a first step towards exploring the architecture modeling of safety-critical autonomous systems which are composed of conventional components and ML components, based on natural language requirements.

1.1. Research Problems

Model-Driven Development (MDD) is generally accepted as a key enabler for the design of safety-critical systems. For example, in the guidance of civil avionics software certification (DO-178C) [FF17], MDD (DO-331), and formal methods (DO-333) were considered as vital technology supplements. There are several MDD languages and approaches covering various modeling demands, such as UML for generic modeling, SysML [1] for system-level modeling, AADL [FG13] for the architectural modeling and analysis of embedded systems, SCADE [2] and Simulink for functional modeling, and Modelica for multi-disciplines modeling. However, the increasing amount of ML in safety-critical autonomous systems demands MDD engineers to adapt their work [VB19]. One of the difficult activities is the architecture modeling of such systems, based on natural-language requirements.

Systems Modeling Language (SysML) was designed by the International Council on Systems Engineering (INCOSE) and the Object Management Group (OMG). As a profile of the UML 2.0, it was created specifically for the systems engineering domain, in order to integrate multiple views of large complex systems engineering consisting of hardware, software, requirements, data, people, and processes. Moreover, SysML provides several extension mechanisms such as stereotypes, diagram extensions, and model libraries. Thus, SysML is considered as the architecture modeling language in this paper.

[1] OMG. SysML. http://www.omgsysml.org

[2] https://www.ansys.com/products/embedded-software/ansys-scade-suite

However, when aligning to the principle of automation of MDD, automatically deriving architecture models directly from requirements in the context of developing safety-critical autonomous systems is a very challenge. One reason is that requirements are mostly written as free natural language text, which is often ambiguous, and difficult to be processed automatically and generate meaningful downstream design artifacts (e.g., models) from such requirements [YBL13].

Restricted natural language requirement templates serve as a simple and yet effective approach for increasing the quality of requirements by avoiding complex structures, ambiguity, and inconsistency in requirements [ASBZ15]. For instance, Tao Yue et al. proposed Restricted Use Case Modeling (RUCM)[YBL15][YBL13] to reduce ambiguity and facilitate the automated analysis of use case models. EARS (Easy Approach to Requirement Syntax) [MWHN09] specifies requirements with several sentence templates to improve the quality of requirements. Spacecraft Requirement Description Language (SPARDL) [WLZ+10] describes requirements with mode diagrams and interval temporal logic, for modeling and analyzing periodic control systems. In our previous work, we proposed a restricted natural language requirement specification method for AADL [WYH+20].

To promote the application of MDD in safety-critical autonomous systems and bridge the gap between textual requirements and SysML models, we propose a requirements specification methodology, named as RNLReq, which is equipped with a set of restriction rules and templates to structure and restrict the way how users document requirements. To the best of our knowledge, it is the first contribution towards a restricted natural language requirement modeling for SysML. Compared with the existing restricted natural language requirement specification approaches such as [YBL15, MWHN09, WLZ+10, WYH+20], this work mainly focuses on the following two research questions:

- *Are there any ML-specific requirements that need to be considered when ML is deployed in a safety-critical system?* There is not much work on Requirements Engineering (RE) for ML-based safety-critical autonomous systems. [Hor19] states challenges and research directions for non-functional requirements of ML systems such as fairness, transparency, privacy, security, and testability. Andreas Vogelsang et al. [VB19] conclude that the development of ML systems demands requirements engineers to: (1) understand ML performance measures to state good functional requirements, (2) be aware of new quality requirements such as explainability, freedom from discrimination, or specific legal requirements, and (3) integrate ML specifics in the RE process. However, the existing work hasn't considered ML-specific requirements when ML is deployed in a safety-critical context. For instance, the characterization of ML models must encompass more than just accuracy, i.e., it should consider the resource constraints in terms of memory, power, compute, and so on. Moreover, the system in a safety-critical application always consists of both conventional components and ML components.
- *How to use AI technologies to assist the restricted natural language requirement modeling?* In the existing restricted natural language requirement specification approaches such as [YBL15, MWHN09, WLZ+10, WYH+20], it always needs to extract requirements glossary terms and to classify the requirements by hand. Recently, the OMG AI-SECT (Augmented Intelligence for Systems Engineering challenge team) [3] introduced AI to model-driven system engineering in order to improve system model accuracy, content exploration, team productivity, and so on. They claimed that AI is already beginning to impact various system engineering aspects, such as system modeling, verification, process management, etc., and AI becomes a means to augment rather than replace human capability, that is "Augmented Intelligence (AuI)" [Mad20]. AI can collaborate with human systems engineers to measurably improve the system engineering effort. Thus, we would like to explore AuI in restricted natural-language requirement specification for safety-critical autonomous systems, which includes glossary terms recommendation and requirements classifications.

In addition, we propose RNLReq2SysML to automatically derive SysML architecture models from requirements specified with RNLReq. Besides, we implemented a tool for both RNLReq and RNLReq2SysML in Papyrus [4]. An industrial Autonomous Guidance, Navigation and Control (AGNC) as a case study for the proposed approach has been carried out and it provides some positive feedback and lessons.

[3] https://www.omgwiki.org/MBSE/doku.php?id=mbse:augmented
[4] https://www.eclipse.org/papyrus/

1.2. Main Contributions

The main contributions of this paper are presented as follows:

- *Augmented Intelligence for restricted natural language requirement modeling of safety-critical autonomous systems.* Firstly, in order to assist the RNL requirement modeling method, the IR4RNL (Intelligent Recommendation for RNL) method is proposed, including term recommendation and requirement classification. Especially, the term recommendation method recommends data dictionary and domain glossary, which are parts of RNLReq. The requirement classification method predicts the category of requirements. Secondly, the RNLReq method is proposed, which is equipped with a set of restriction rules and templates to structure and restrict the way how users document requirements.
- *Automatic generation of SysML architecture models from the RNLReq requirement specifications.* Firstly, in order to support the description of the specific requirement types of safety-critical autonomous systems (data quality, quality of resulting prediction, safety, reliability, resource consumption, etc.), an extension of SysML is given. Secondly, based on the meta-model technology, we propose RNLReq2SysML to automatically derive SysML architecture models from requirements specified with RNLReq.
- *Prototype tool and industrial case study.* The prototype tool for both RNLReq and RNLReq2SysML is implemented in Papyrus. In addition, the case study, i.e., Autonomous Guidance, Navigation and Control (AGNC), has been carried out.

1.3. Outline

The rest of this paper is organized as follows. Section 2 briefly introduces the concept of augmented intelligence in system engineering and the SysML language. Section 3 gives the requirement analysis of the target system of this paper, i.e., safety-critical autonomous systems. A global view of the approach is presented in Section 4. Section 5 gives the details of augmented intelligence for natural language requirement modeling. Section 6 presents the automatic generation of SysML architecture models. The prototype tool is presented in Section 7. Section 8 gives a real-world aerospace industrial case study to show the effectiveness of the proposed approach in this paper, and the threat to the validity of our approach is also presented. Section 9 discusses related work, and Section 10 provides concluding remarks and plans for future work.

2. Preliminaries

This section first briefly introduces the concept of AuI in model-driven system engineering. Then, the SysML language is described.

2.1. Augmented Intelligence in Model-Driven System Engineering

As claimed by Azad M. Madini [Mad20], the underlying technologies powering AI and AuI are the same, but the goals, usage, and application context are fundamentally different. AI creates systems operating without the need for humans, whereas AuI uses AI to augment human performance, which for many activities will be superior to the individual human or AI performance. It is also important to realize AuI is not a new technology category; rather, it is a different way to think about AI technology's purpose. While researchers pursue AI to see how far it can go is an admirable goal to realize key technologies proving invaluable in the future, the real world today is better served by AuI.

From the point of view of system engineering, AI is already beginning to impact various system engineering aspects. These include systems modeling, verification, process management, search and information retrieval, dynamic content management, and human-system integration. As shown in Figure 1, AuI systems engineering encompasses the system, the infrastructure support facilities, and processes. However, compared with [Mad20], we have added safety analysis and requirement specification as the system engineering activities which are vital for safety-critical autonomous systems. In each case, the AI is AuI. This paper will focus on the AuI in natural language requirement modeling and the SysML architecture models generation from such requirement specifications.

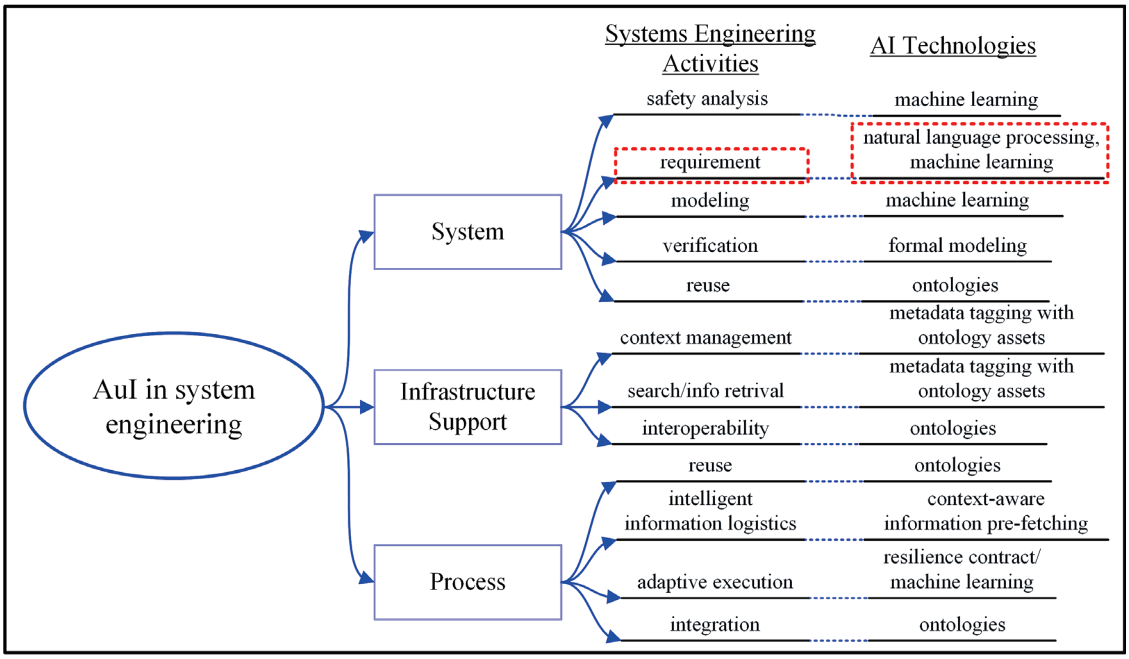

Systems Engineering Activities
AI Technologies
AuI in system engineering
System
Infrastructure Support
Process
safety analysis
machine learning
requirement
natural language processing, machine learning
modeling
machine learning
verification
formal modeling
reuse
ontologies
context management
metadata tagging with ontology assets
search/info retrival
metadata tagging with ontology assets
interoperability
ontologies
reuse
ontologies
intelligent information logistics
context-aware information pre-fetching
adaptive execution
resilience contract/ machine learning
integration
ontologies

Fig. 1. AuI in system engineering [Mad20]

The AI technologies which are used in our AuI consist of natural language processing, text similarity calculation, clustering algorithm, and classification algorithm for machine learning. Especially, in our empirical evaluation (Section 8), firstly, nine syntactic measures including Block distance[GF+13], Cosine[GF+13], Dice's coefficient[GF+13], Levenshtein[MRS10], Euclidean[GF+13], Jaccard[CRF+03], Jaro[CRF+03], Jaro-Winkler [CRF+03] and SimHash-Hamming[SL11] are considered. Secondly, for semantic similarity, we consider two strategies based on Hownet [Liu02] and CiLin[JW10], which can support the Chinese language. Thirdly, for clustering, we considered K-means, EM, and hierarchical clustering. Finally, classical machine learning algorithms including logistic regression, naive bayes, support vector machine, K-nearest neighbors, and random forest are considered in the requirement classification.

2.2. SysML

As UML is insufficient in capturing all the concepts that are meaningful in systems engineering, Systems Modeling Language (SysML) was designed by the International Council on Systems Engineering (INCOSE) and the Object Management Group (OMG). In addition to reusing a subset of UML 2.0, SysML provides new features towards system engineering including the ability to represent requirements and constraints of systems and to support trade-off analysis. A SysML model is made up of several diagrams that can be specified into three categories. Block definition diagrams(BDD) and internal block diagrams(IBD) capture the architecture of the system, with parametric diagrams describe constraints on property values to support future analysis. Requirement diagrams capture requirement, and requirement relationships. Activity diagrams, state machine diagrams and use case diagrams describe the behavior of the system. By extending a metaclass or generalizing an existing stereotype, SysML allows users to extend and adjust existing metamodels for different domains.

3. Requirement Analysis of Target Systems

Autonomous systems, such as deep space missions, autopilots, mobile robots, and autonomous vehicles belong to the spectrum of safety-critical applications. In this paper, we mainly focus on the safety-critical autonomous systems in the space domain. Observability and controllability of space systems are naturally limited due to several factors, such as communication delays, power budgets, and so on. Introducing more complex autonomy can increase the efficiency of many on-board missions of the space systems. On-board autonomy management addresses all aspects of on-board autonomous functions that provide the space system with the capability to continue mission operations and to survive critical situations without relying on ground system intervention. The implementation of on-board autonomy depends on the specific mission requirements and constraints, and can therefore vary between a very low level of autonomy involving a high level of control from the ground to a high level of autonomy, whereby most of the functions are performed on-board. For instance, the European Cooperation for Space Standardization (ECSS) [ECS08] defines four levels of such autonomous capabilities.

3.1. AGNC

The Guidance, Navigation, and Control (GNC) system is a core system supporting orbiting operations of spacecraft, which undertakes the tasks of determining and controlling spacecraft attitude and orbit. GNC is composed of navigation sensors (such as navigation cameras, star sensors, gyroscopes, and accelerometers), actuators (such as reaction flywheels, nozzles, orbit-controlled engines), and control computers (such as attitude and orbit control systems, AOCS) which process the guidance and control tasks of various sensors, perform orbit determination, orbit control, attitude determination, and attitude control. In addition, a data process unit (DPU) is added between navigation sensors and AOCS to preprocess the data sent by navigation sensors. Generally, AOCS has more than 20 modes such as stable, maneuver, safe, autonomous, and so on. The so-called mode is an operational state of a spacecraft, subsystem, or payload in which certain functions can be performed.

Figure 2 shows a simplified GNC structure with stable and autonomous modes, which is named AGNC (Autonomous Guidance, Navigation, and Control). In the stable mode, AOCS provides the missions by using conventional components. In the autonomous mode, to achieve autonomy, it uses an architecture including five complementary stages, i.e., perception, reflection, goal management, planning, and self-adaptation. Note that, the first two stages deal with “understanding” the situation of the environment. The third and forth aspects deal with the autonomy of decision. Self-adaptation ensures the adequacy of decisions with respect to the environmental situation. For instance, one can use ML to implement perception and autonomous decisions. Based on a data or knowledge base, a model is trained off-line on the ground and then can be queried by an on-board mission. Although insufficient training data exists for many applications, significant data from Earth, the Moon, and Mars is already available for machine learning applications. Moreover, data for off-line training can be obtained either by simulation or experiment or the past experiences.

Machine learning, particularly deep learning, is being increasingly utilized in different space applications [KLL20]. Here, we present three scenarios that can be implemented as ML components.

- *Imagery transmissions.* The images captured by the spacecraft need to be transmitted to the ground station for aggregation and analysis. However, sending and receiving large volumes of imagery is power consuming. In addition, the high-latency and low-bandwidth communication channels make this prohibitive. Thus, a spacecraft can reduce the amount of imagery transmitted by employing neural networks: on-board pre-processing can discard parts of the image of no interest. For instance, the images used as inputs to the neural networks can be 224×224 or 256×256 RGB images. Transmission costs can be further reduced by employing a neural network to compress image data.
- *Rendezvous and docking.* Autonomous rendezvous and docking are defined as one spacecraft approaching another one autonomously, from close proximity, with hard constraints on the final position, the final speed, and the final attitude of the spacecraft. For instance, the active spacecraft (usually referred to as *chaser*) observes the passive spacecraft (*target*) from almost 1km away and gather the spacecraft trajectory data. The AGNC system of the chaser controls the state parameters required for entry into the docking interfaces of the target vehicle. A simplified deep neural network structure is shown in Figure 3. The input layer receives the state information including the position of the chaser, the attitude and angular velocity of the chaser, the speed of the chaser, the position of the target, the attitude and angular velocity of the target, and the speed of the target. The hidden layers can be more than 20 layers. And the output layer controls the thrusters with two possible actions: left and right.
- *Fault detection, isolation, and recovery (FDIR).* The spacecraft may suffer from faults physically and/or by interfering with communications. In such situations, it is imperative that the spacecraft should have an

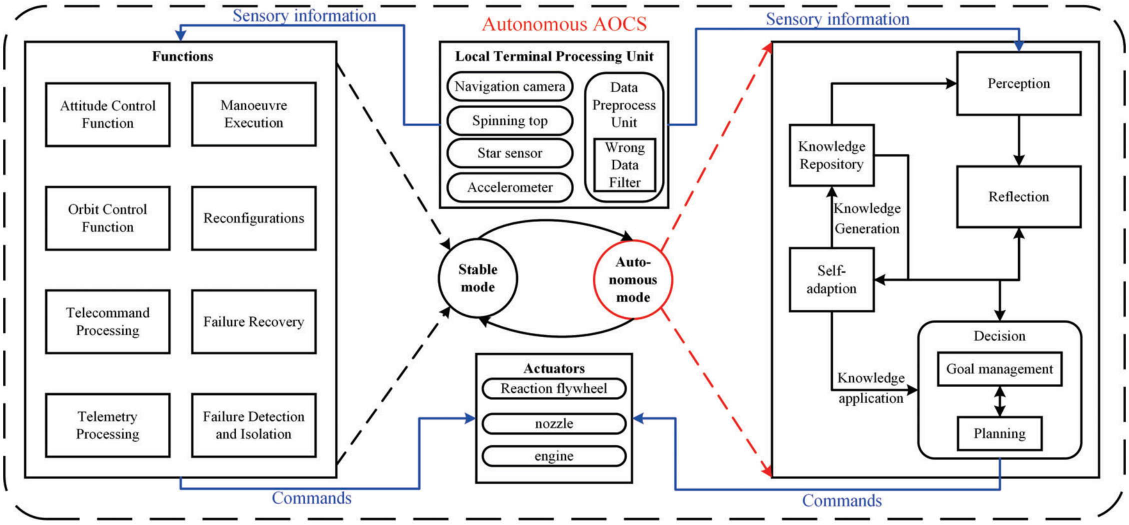


Fig. 2. AGNC Framework

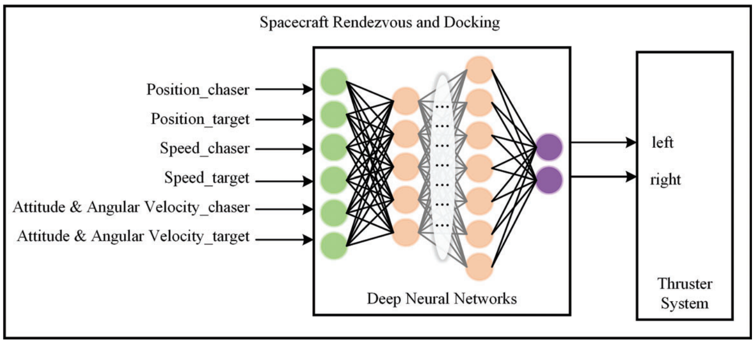

Spacecraft Rendezvous and Docking
Position_chaser
Position_target
Speed_chaser
Speed_target
Attitude & Angular Velocity_chaser
Attitude & Angular Velocity_target
Deep Neural Networks
left
right
Thruster System

Fig. 3. Spacecraft Rendezvous and Docking

autonomous ability to detect faults and other anomalies. Stottler et al [SRBM20] proposed an ML-based fault detection approach, in which a two-layer artificial neural network (ANN) is introduced. Once trained on normal system behavior, the ANN model is used at detecting behavior previously not encountered in the training data (i.e., anomalies). Upon detecting anomalous behavior, it uses a supervised classification approach to determine a subset of measurands that characterize the anomaly.

3.2. Requirement Categories of AGNC

In a recent survey, [Hor19] and [IY19] reported that requirement specification was listed as the most difficult activity for the development of ML-based systems. Andreas Vogelsang et al [VB19] provided the first contribution towards a requirement engineering methodology for ML systems, that is, they concluded that developing ML systems demands requirements engineers to: (1) understand ML performance measures to state good functional requirements, (2) be aware of new quality requirements such as explainability, freedom from discrimination, or specific legal requirements, and (3) integrate ML specifics in the requirement engineering process. However, they haven't considered ML-specific requirements when ML is deployed in a safety-critical context.

Based on the previous cooperation and the communication with our industrial partners, we conclude the requirement categories of the AGNC system shown in Table 1. Here, we mainly explain the requirements of ML components which are given as follows.

Table 1. Requirement Categories of AGNC

Component Types	Requirement Types	Explanation
Conventional components	Functional requirements	A description of the service that the system must offer.
	Interface requirements	Various interface relations between system and environment.
	Data requirements	Describe the various data used by the system and the requirements for data collection.
	Non-functional requirements	Including safety, reliability, real-time, performance, environmental constraints, etc.
ML components	Goal requirements	The goal given by the stakeholder.
	AI Functional requirements	Describe the learning process of machine learning, such as problem type, algorithm (or model) description, etc.
	AI Non-functional requirements	Including safety/reliability, real-time, performance (such as the quality of the resulting prediction, such as accuracy, precision, recall, etc.), resource consumption, environmental constraints, explainability, robustness, etc.
	Training/testing/operational data requirements	Including the quantity and quality of the training/testing/operational data, such as consistency, completeness, correctness, diversity.

- *From the point of view of stakeholders*: it is hard to specify the detailed functional behaviors of an ML component with explicit code. However, the decision of ML components should be based on the stakeholder needs, that is the goal requirements. In addition, requirement engineers always give a basic specification of the process of machine learning, such as the problem type, the used ML algorithms, etc. We call it a functional requirement of the ML component. The ML components also need to satisfy some non-functional attributes such as safety, reliability, real-time, explainability, robustness, and so on, in that, ML components always interact with an uncertain environment, so sometimes it needs several environmental constraints, for instance, temperature, wind speed, and so on. In addition, the safety-critical autonomous system is always resource-constrained, thus the characterization of ML models must encompass more than just accuracy, i.e., it should consider the resource constraints in terms of memory, power, compute, and so on.
- *From the point of view of ML implementation*: Firstly, data is an integral part of ML components, thus the requirements of training/testing/operational data quality play an important role for specifying components than for conventional components. Secondly, based on a data or knowledge base, a model is trained that can then be queried by an application. Therefore, the resulting prediction, such as prediction precision, accuracy, and recall, that is, performance requirements, need to be well understood by system engineers.

Please note that this paper explores the architecture modeling of safety-critical autonomous systems based on natural language requirements, instead of directly specifying and verifying the characteristics of AI components such as their safety and robustness.

4. Global View of the Approach

A global description of the approach proposed in this paper is shown in Figure 4. It mainly contains two parts:

- AuI for restricted natural language requirement modeling. Firstly, we propose the IR4RNL (Intelligent Recommendation for RNL) method to assist the RNL requirement modeling. Especially, the term recommendation method recommends a glossary and the requirement classification method predicts the category of requirements. Finally, the glossary and the category of requirements will be used in the restricted natural language requirement template RNLReq, to specify the natural language requirements in a controlled format.
- Automatic generation of SysML architecture models from the RNLReq requirement specifications. Firstly, in order to support the description of the specific requirement types of safety-critical autonomous systems

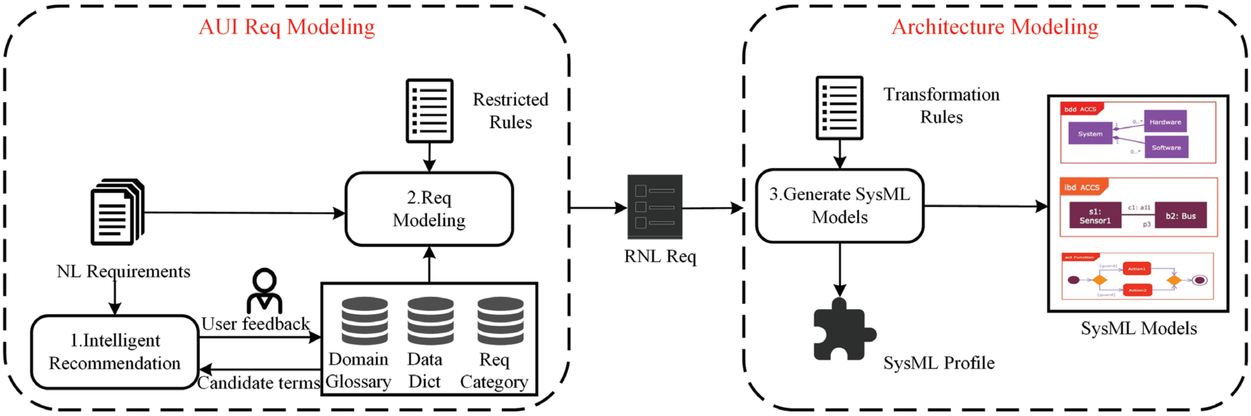


Fig. 4. Global view

(data quality, quality of resulting prediction, safety, reliability, resource consumption, etc.), an extension of SysML is given. Secondly, based on the meta-model technology, we propose RNLReq2SysML to automatically derive SysML architecture models from requirements specified with RNLReq.

In the following sections, we will present the two parts respectively. Please note that we mainly consider Chinese natural language requirement text in the IR4RNL (Intelligent Recommendation for RNL) method. However, in the restricted natural language requirement template RNLReq, each term in glossary and data dictionary contains both a Chinese name and an English name, thus the SysML model in English can be generated and the user interface can be adapted to Chinese.

5. Augmented Intelligence for Natural Language Requirement Modeling

This section gives the details of augmented intelligence for natural language requirement modeling. First of all, Section 5.1 presents the IR4RNL (Intelligent Recommendation for RNL) approach, which including term recommendation method and requirement classification method. Second, Section 5.2 presents the RNLReq method. Finally, Section 5.3 introduces the meta-model of RNLReq.

5.1. IR4RNL: Intelligent Recommendation for RNL

This section presents the intelligent recommendation for restricted natural language (IR4RNL), as shown in Figure 5. IR4RNL mainly consists of two parts: term recommendation and requirement classification. The data dictionary and the domain glossary, which are parts of RNLReq, can be recommended by the terminology recommendation method. The requirement classification method can assist in the establishment of the requirement template, which is also a part of RNLReq.

5.1.1. Terminology Recommendation Method

Given a natural language requirement, it first recommends candidate terms by interacting with engineers, then computes a similarity matrix for the candidate terms, and finally clusters the candidate terms based on their similarity.

Phase 1: Candidate term extraction. Candidate term extraction method includes text pre-processing, candidate terms extraction based on part-of-speech rule, candidate terms extraction based on dependency rule and domain filtering. Part-of-speech rule and dependency rule are used to extract candidate terms based on POS (part-of-speech) tagger and dependency parsing respectively, resulted from text pre-processing.

- *Text pre_processing*. The process of text pre-processing contains sentence splitter, word segmentation, POS tagger, and dependency parsing. Sentence splitter divides the natural language requirements into sentences. Word segmentation determines the word boundaries in a sentence. Then, the POS tagger

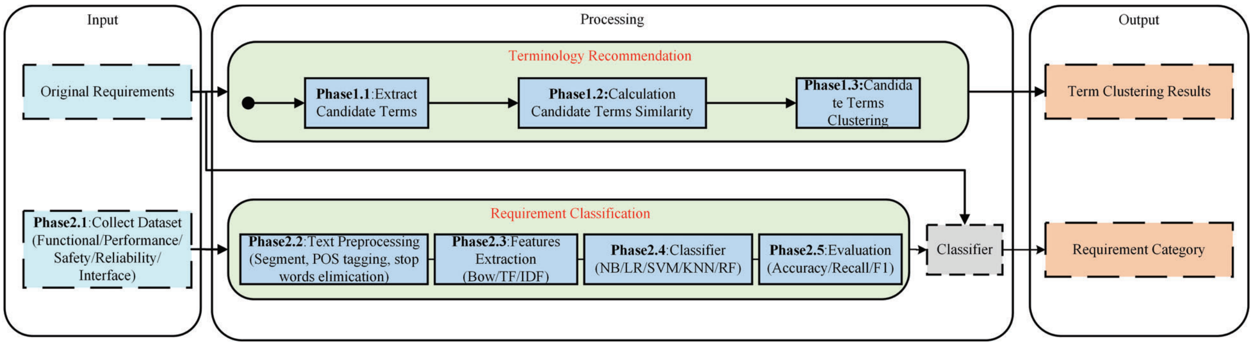

Input
Processing
Output
Original Requirements
Terminology Recommendation
Phase1.1:Extract Candidate Terms
Phase1.2:Calculation Candidate Terms Similarity
Phase1.3:Candidate Terms Clustering
Term Clustering Results
Phase2.1:Collect Dataset (Functional/Performance/ Safety/Reliability/ Interface)
Requirement Classification
Phase2.2:Text Preprocessing (Segment, POS tagging, stop words elimication)
Phase2.3:Features Extraction (Bow/TF/IDF)
Phase2.4:Classifier (NB/LR/SVM/KNN/RF)
Phase2.5:Evaluation (Accuracy/Recall/F1)
Classifier
Requirement Category

Fig. 5. IR4RNL method

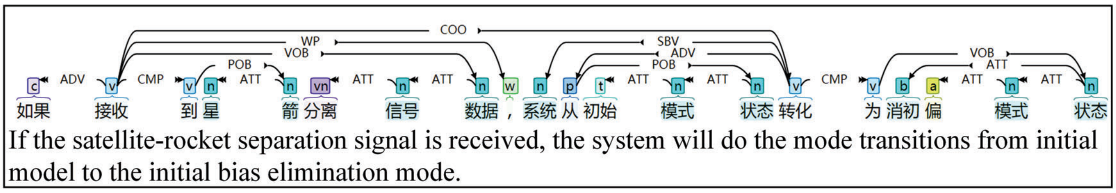

COO
WP
VOB
SBV
ADV
POB
ADV
CMP
ATT
VOB
ATT
如果 接收 到 星 箭 分离 信号 数据 ， 系统 从 初始 模式 状态 转化 为 消初 偏 模式 状态
If the satellite-rocket separation signal is received, the system will do the mode transitions from initial model to the initial bias elimination mode.

Fig. 6. The text pre-processing results of a requirement of AGNC in the normal modes

annotates each word with a part-of-speech tag. The final step is dependency parsing, where an attempt is to recognize a sentence and assign a syntactic structure. In this paper, text pre-processing is achieved through the existing natural language processing tool HanLP [He14] (Han Language Processing). Figure 6 and Figure 7 show the text pre-processing results of one of requirements of AGNC in normal modes and in the autonomous mode respectively. In that, ATT, ADV, COO, SBV, RAD, POB, LAD and VOB represent attribute relationship, adverbial relationship, coordinate relationship, subject-verb relationship, right additional relationship, preposition-object relationship, left additional relationship and verb-object relationship respectively. For instance, "*autonomous*", "*management*" and "*module*" are tagged as "vn", "vn" and "n" respectively. The relation between "*autonomous*" and "*module*", and the relation between "*management*" and "*module*", are ATT (attribute relation).

- *Candidate terms extraction based on part-of-speech rule*. Chinese domain terms have special linguistic characteristics: 1) From the perspective of external associations, domain terms are mostly noun phrases (NPs), which are often used as subjects, objects, and attributions in domain texts; 2) From the perspective of their internal grammatical composition, their composition forms including nouns + nouns (for instance "security agency"), adjectives + nouns ("combustible materials"), verbs + nouns ("launching devices"), verbs (nouns) + single-word nouns ("airtight doors"), etc; 3) The length of Chinese domain term is always two to six words.
  In this step, we introduce engineer feedback loop which is described in our AuI approach. Firstly, it formulates part-of-speech pattern matching rules based on the preliminary glossary extracted by the engineers. Please note the preliminary glossary should comply with the Chinese linguistic characteristics shown above. Secondly, candidate terms are extracted according to the part-of-speech pattern matching

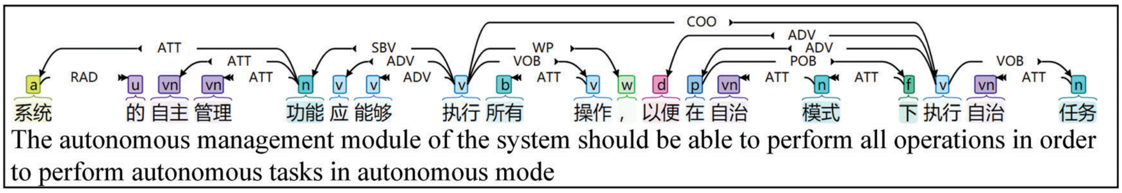

COO
ADV
ATT
SBV
WP
ADV
POB
ATT
RAD
ADV
VOB
系统 的 自主 管理 功能 应 能够 执行 所有 操作 ， 以便 在 自治 模式 下 执行 自治 任务
The autonomous management module of the system should be able to perform all operations in order to perform autonomous tasks in autonomous mode

Fig. 7. The text pre-processing results of a requirement of AGNC in the autonomous mode

rules. Thirdly, the engineers filter the candidate terms (addition, deletion, and modification), and then update the part-of-speech pattern matching rules. Finally, it repeats the second and the third steps until the engineers are satisfied. Regarding the requirement sentence given in Figure 7, for instance, engineers give part-of-speech pattern matching rules such as "vn+vn+n" and "vn+n" based on the preliminary glossary, thus there are three candidate terms that meet the part-of-speech pattern matching rules, such as "*autonomous management module*", "*autonomous task*" and "*autonomous mode*".

- *Candidate terms extraction based on dependency rule*. It includes three sub-steps: generating dependency tree, pruning, and generating candidate terms. Firstly, the requirement sentence is analyzed for dependency parsing to obtain the dependency tree. Secondly, Chinese domain terms are generally NPs, so we mainly consider four types of dependency relations such as the attribute relation (ATT), the coordinate relation (COO), the right additional relation (RAD), and the left additional relation (LAD). With the four types of dependency relations, the dependency tree will be pruned as a set of dependency sub-trees. The definition of the dependency sub-tree is given as Definition 1. Finally, according to the generated dependency sub-trees, we select $|V| > 1$ and continuous strings as candidate terms. For example, "*system module*" and "*autonomous module*" are discontinuous strings, and the remaining dependency sub-trees are all continuous strings, so four candidate terms are generated, "*autonomous management module*", "*management module*", "*autonomous task*" and "*autonomous mode*".

Definition 1. Given a dependency tree $T = (V, A)$, its dependency sub-tree $T' = (V', A')$ satisfies the following constraints:

(i) $V' \subseteq V$, $A' \subseteq A$.
(ii) There is a single designated root node that has no incoming arcs.
(iii) With the exception of the root node, each vertex has exactly one incoming arc.
(iv) There is a unique path from the root node to each vertex in V.
(v) The root node is a noun or verb.

The candidate terms extraction algorithm based on dependency rules is shown in Algorithm 1. Firstly, each requirement is analyzed for dependency parsing and a dependency tree is generated; Secondly, generate dependency sub-trees that contains only four types of dependency relationships: ATT, COO, LAD, and RAD; Finally, each string in each dependency sub-tree is analyzed, to extract continuous strings with length greater than 1 as candidate terms.

Algorithm 1 Terminology extraction algorithm based on dependency rules

```
Input: Requirements
Output: CandidateTerms

CandidateTerms = list()
for each Requirement in Requirements do
    compute DepandencyTree of the Requirement;
    compute DepandencySubtrees of only DepandencyTree of ATT, COO, LAD, RAD;
    for each DepandencySubtree in DepandencySubtrees do
        for String in DepandencySubtree do
            if String is continuous and $|String| > 1$ then
                CandidataTerms.add(String);
            end if
        end for
    end for
end for
```

- *Domain filtering*. The domain characteristics of a term mean that the term usually appears only in one or a few domain-specific texts. It is determined by comparing the number of occurrences of term candidates in our domain requirements with the occurrences in a non-domain-specific text corpus. For example, we eliminate all candidate terms that occur less frequently in our requirement corpus than in a subset of the THUCNews[5] corpus. It ensures that all candidate terms have a minimum degree of domain specificity. In addition, stop words refer to some very common words, and removing them will not affect the semantics of the requirement. Thus we further filter candidate terms with stop words and finally extract four candidate terms, including "*autonomous management module*", "*management module*", "*autonomous task*" and "*autonomous mode*".

Phase 2: Computing similarities between terms. In this step, it computes a similarity matrix to capture the degree of relatedness between every pair of candidate terms extracted in the previous step. To compute this matrix, we consider three alternative strategies[ASBZ16]:

(i) syntactic only, where a similarity, $\mathscr{S}_{syn}(t, t')$, is computed for every pair (t, t') of terms using a syntactic measure, e.g., Levenshtein;

(ii) semantic only, where a similarity, $\mathscr{S}_{sem}(t, t')$, is computed for every pair (t, t') of terms using a semantic measure, e.g., Hownet;

(iii) combined syntactic and semantic, where, given a syntactic measure *syn* and a semantic measure *sem*, we take, for every pair (t, t') of terms, $\max\left(\mathscr{S}_{syn}(t, t'), \mathscr{S}_{sem}(t, t')\right)$. Using max is motivated by the complementary nature of syntactic and semantic similarity measures.

Phase 3: Clustering candidate terms. We cluster the candidate terms based on their degree of relatedness. The inputs of this step are the similarity matrix, the choice of the clustering algorithm to use, and the number of clusters, and K. As the result of clustering, the candidate terms are grouped into K partitions[ASBZ16]. For example, "*GNC*" and "*AOCS*" are treated as the candidate terms in the same cluster.

In the case study shown in Section 8, we use K-means, Expectation-Maximization (EM) and, Hierarchical Clustering algorithms to cluster candidate terms to investigate the key questions related to tuning clustering for use in our application context. Specifically, identifying the most accurate clustering algorithm(s) is addressed in RQ5, see Section 8.2.2.

5.1.2. Requirement Classification Method

The requirement classification method is composed of five phases:

Phase 1: Data-sets collection. Firstly, according to the national standard *Software Requirements Analysis*: requirement analysis documents are divided into eight types: functional, performance, interface, data, environment, safety, modifiability, and assumptions and constraints. Secondly, according to the *Aerospace Product Reliability Assurance Requirement*, the reliability metrics of aerospace products are required. We select five typical requirement categories to collect data-sets, which are functional, performance, safety, reliability, and interface. The scale of the data-sets is shown in Section 8.2.2.

Phase 2: Text pre-processing. We preprocess requirement by segment, POS tagging, and stop words elimination.

Phase 3: Features extraction. Two feature extraction techniques BoW and TF-IDF are used to extract textual features and calculate the wright of features. We use these textual features and the weight of features to train classifiers.

Phase 4: Classifier. Five machine learning algorithms, including Naive Bayes, Logistic Regression, Support Vector Machine, K-Nearest Neighbors, and Random Forest, are used to train classifiers. The requirements are automatically classified by the trained classifiers.

Phase 5: Evaluation. We use precision, recall, and F-score to evaluate the performance of the classifiers which are trained in Phase 4. The calculation formula is shown in Section 8.2.2.

5.2. RNLReq: Restricted Natural Language Requirement Template

This section presents the Restricted Natural Language Requirement Specification Method (RNLReq), which can reduce ambiguities of natural language requirements without changing the requirements writing habits

[5] http://thuctc.thunlp.org/

of system engineers. A typical requirement document from industries consists of such as sections of functionalities, tables of domain terms, and tables of data. Each functionality is organized as IPOC (Input-Process-Output-Constraint) structure. Thus, RNLReq consists of three parts: Data Dictionary, Domain Glossary and RNL templates with well-defined sentence patterns based on restriction rules.

5.2.1. Data Dictionary & Domain Glossary

In the domain of safety-critical systems, requirements engineers are always recommended to construct a data dictionary and a domain glossary first. A well-organized domain glossary and structured data dictionary can enhance the requirement engineering processes in a number of ways. For instance, they can help improve the understanding of requirements and reduce the vagueness and ambiguity of requirements, which can promote better communications among stakeholders and provide a stepping stone for further requirement elaboration.

Therefore, data dictionary and domain glossary are included in RNLReq, and they are organized as follow:

- *Data dictionary*, refers to a dictionary covering the data and its attributes scattered in the requirements documents. All the data should be specified with name, id, data structures (can be comprised of simple data types, e.g. integer, character, etc., and complex structure, e.g. arrays, struct, union), unit, range, rate, quality, quantity, and description. Table 2 shows the Backus-Naur Form grammar (BNF) of the data dictionary. In order to present the continuous and discrete dynamic characteristics in SCS systems, Rate is specified with a numerical value or a distribution. In order to express specific requirements of ML components, we need to consider data quality and data quantity. Data quantity includes the size and diversity of data. Data quality includes completeness, consistency, and correctness. Five restriction rules D1-D5 constraining data quality are presented in Table 3. In order to ensure completeness, Rule D1 checks the sparsity of data within each characteristic. Rules D2, D3 are used to ensure the consistency of data. Rule D4 emphasizes on data collection process which has a strong influence on the correctness of data. Rule D5 reduces the discrimination and bias in data resulting from humans.

According to the actual needs of the industry, the data dictionary can include static data, input data, output data and so on. Among them, static data refers to the constants in the system; input data refers to the input of each component; and output data refers to the output of each component.

- *Domain glossary*, refers to a dictionary covering the domain terms scattered in the requirements documents, such as the names of systems, functions, interfaces, properties, modes, hardware, states, etc. All terms documented should be specified with name, id, category, and description. Table 4 shows the BNF grammar of the domain glossary.

Here, we give several examples to show how to build data dictionary and domain glossary from requirement sentences.

Example 1: Given a functional requirement "*The Perception Function sends the data collected by the sensor to the Decision Function every 10ms, and the data collected by the sensor includes angular velocity and angular position.*", we first register the data in the data dictionary: [name: Sensor_angular_data, ID: d001, struct: (angular velocity, angular position), unit: NA, range: NA, rate: 10ms, quality: NA, quantity:(size: 1), description: NA]. Here, NA means Not Applicable. It should be noted that the angular velocity and angular position need to be registered in the same way. Then, we register two terms in the domain glossary: [name: Perception_Function, id: g001, category: function, description: NA], [name: Decision_Function, id: g002, category: function, description: NA].

Example 2: In order to express safety and reliability requirements, we can register static data like FailureSeverity and FailureProbability in the data dictionary. For instance, given a safety requirement "*The failure of rocket propulsion has four levels of severity: NoEffect, Minor, Major, Hazard*", we register the static data: [name: FailureSeverity, ID: s001, struct: NA, unit: NA, range: (NoEffect, Minor, Major, Hazard), rate: NA, quality: NA, quantity: NA, description: NA]. To describe probability distributions of failure, we register the data FailureProbability: [name: FailureProbability, ID: s002, struct: NA, unit: NA, range: (Normal[mean, stdDev], Uniform, Basic[min,max]), rate: NA, quality: NA, quantity: NA, description: NA].

Example 3: For an ML component, given the data quality requirement "*The size of the data-set is 100k, and 60% of the training data comes from Source A and the remaining is from Source B*", we register the data: [name: TrainingSet, ID: d003, struct: NA, unit: NA, range: NA, rate: NA, quality: NA, quantity:(size: 100k), diversity: (SourceA=60%, SourceB=40%)].

Table 2. BNF grammar of data dictionary

```
<data-dictionary> ::= (<data>)+
<data> ::= <name> <ID> <struct> <unit> <range> <rate> <quality> <quantity> <description>
<name> ::= (<letter> | <digit>)+
<rate> ::= <digit> | <distribution>
<distribution> ::= <uniform> | <normal>
<quality> ::= <completeness> <consistency> <correctness>
<quantity> ::= <size> <diversity>
```

Table 3. Rules of data quality

No.	Description
D1	Data must cover the whole range of possible values.
D2	Additional data sources that satisfy hypotheses can be used to augment the original data to get a richer set of training data.
D3	Each data in the data-set must be in the same format and representation.
D4	Data must be validated and cleaned before use.
D5	Don't collect or label data by humans.

With the help of terminology recommendation methods in Section 5.1, we can get the recommendations of data dictionary and domain glossary.

5.2.2. Requirement Templates

It is a common practice in industry to decompose requirements hierarchically, and each decomposed requirement is organized as a typical IPOC structure. Therefore, we defined a set of requirement specification templates in RNLReq, which can be organized hierarchically.

Requirement specification templates are organized as four levels according to the decomposition of requirements in industry: System Requirement Template (SRT) presents the highest level requirement of the system, and engineers can decompose it as several sub-system requirements which can be described in Sub System Requirement Template (SubSRT). The decomposition of SubSRT can be iterative. Then, each SubSRT can be decomposed as several Function Requirement Template (FRT), and FRT can also be decomposed as Sub Function Requirement Template (SubFRT). The standard for the granularity of templates, especially whether SubSRT should be decomposed as FRT instead of lower-level SubSRT, is the strategy of implementations for each sub-system and function.

Besides, it is possible that some functions or sub-functions from different systems present the same or similar functionality. To simplify the requirement specification for such a situation, we defined a Share Function Block Template (SFBT), which only belongs to systems or sub-systems, and can be invoked from any template in these systems or sub-systems.

Table 5 shows the basic requirement template which can support the requirements of both conventional components and ML components. To be consistent with the requirement types presented in Table 1, in general, the template includes *ID*, *Name*, *Input/Output* (including *Data* and *Event*), *Functional Requirement*, *Non-Functional Requirement* (including *Safety/Reliability Requirement*, *Performance Requirement*, *Environmental Requirement*, etc.), *AI Requirements* (optional), and *Description*. When we use the requirement template to specify the requirements of ML components, the *Functional Requirement* and *Non-Functional Requirement* are noted as NA, and *AI Requirements* include the requirement types which has been shown in Table 1, such as goal requirements, training data quality requirements, and so on. Each category of the requirements, such as *Functional Requirement*, *Non-Functional requirements* and so on, should follow their specific templates.

Table 4. BNF grammar of domain glossary

```
<domain glossary> ::= (<noun>)+
<noun> ::= <name> <ID> <category>
<name> ::= (<letter> | <digit>)+
<category> ::= 'system' | 'function' | 'interface' | 'property' | 'hardware' | 'mode' | 'states'
```

Table 5. RNLReq Requirement Template

Elements	Explanations
ID	The unique identifier of this template
Name	The name of the module described in this template
Input	The input data and event of this module
Output	The output data and event of this module
Functional Requirement	The dynamic behavior of this module, such as sending data
Non-functional Requirement	Performance such as period, deadline, Safety/Reliability such as failure mode, failure case, Environmental Requirement such as transmission rates of ports (Not in FRT, SubFRT & SFBT)
AI Requirement	The description of ML components (Optional)
Description	Extra description of this module. (No restriction rule here)

Table 6. General Restriction Rules of Sentence Patterns

No.	General Restriction Rules
R1	The subject of a sentence must be the name of modules or “this module”.
R2	The sentence must be a declarative sentence.
R3	The sentence must be in the present tense.
R4	Do not use passive voice.
R5	Only complete sentences are allowed.
R6	Don’t use modal verbs, pronouns, adverbs and adjectives that express negative meanings.
R7	Same words should be used to represent the same object.
R8	Describe interactions between systems without dismissing subjects or objects.

5.2.3. General Restriction Rules

For each category of requirements shown in Table 5, a set of general restriction rules (R1-R8) in Table 6 are designed to constrain the use of natural language. R1 is to ensure the feasibility of hierarchy modeling and behavior modeling. R2-R5 are good practices for writing requirements and reduce the ambiguity. R6 can help reduce the uncertainty in requirements. R7 ensures the same concept will be expressed in a consistent way. R8 facilitates the automated generation of models.

We further refine the rules for specific requirements such as functional requirements, non-functional requirements and AI requirements and illustrate with examples.

5.2.4. Functional requirement Restriction Rules

Functional requirements (FR) focus on what the system does and emphasizes the inputs, outputs, sequences, and conditions for coordinating behaviors. For functional requirements, each sentence pattern represents a complex sentence structure, which is composed of sequences of simple sentences connected with a set of keywords (e.g., *IF THEN ELSE*, *AND*, *LOOP FOR*, *NOT*). Five simple sentence patterns are designed for functional requirements as shown in Table 7. For a better understanding, we also give an example for each sentence pattern in the table.

SimpleBehaviorSentence is used to express basic functional behavior, while *SimpleBehaviorSentence*

Table 7. Simple Behavior Sentences for Restricted NL

No.	Sentence Pattern	Explanation	Examples
S1	*SimpleBehaviorSentence*	A single sentence presenting one action, including S2-S5.	See examples of S2-S5.
S2	Send	Send <Data> at <Port>	DPU sends processed data to AOCS at Port-1573B.
S3	Assign	Assign <Data-A> to <Data-B>	Orbit control subsystem assigns orbitMode to LOW-EARTH.
S4	Call	Call <System> <SharedFunction>	AOCS calls the orbit determination function.
S5	Jump	Jump <Function>	AOCS jumps to orbit determination function each 15s when in Stable state.

Table 8. Compound Behavior Sentences for Restricted NL

No.	Sentence Pattern	Description	Examples
S6	SimpleValueCondition	IF <P1> <Comparator> <Value> <Unit>	IF Attitude Deviation Angle is bigger than 26 degrees.
S7	ValueCondition	More than one SimpleValueCondition connected with AND/OR	IF conditionA AND conditionB.
S8	EventCondition	IF receiving <Event>	IF receiving Handshake Success Information, THEN AOCS sends GNCC Controlling Data to GNCC.
S9	TimeCondition	DURING (T1, T2), IN D1	The Control Module sends Handshake Information to GNCC in 5ms.
S10	ConditionnalSentence	IF Condition, THEN SentenceA, [ELSE SentenceB]	IF receiving Handshake Success Information, THEN the Control Module sends GNCC Controlling Data to GNCC, ELSE the Control Module sends Handshake Information to GNCC.
S11	LoopSentence	LOOP SentenceA EACH T UNTIL E	LOOP (The Control Module sends data) UNTIL POWER_OFF.
S12	ConcurrentSencence	MEANWHILE (SentenceA, SentenceB, SentenceC, ..)	Meanwhile (The Control Module sends data, the Control Module counts times).

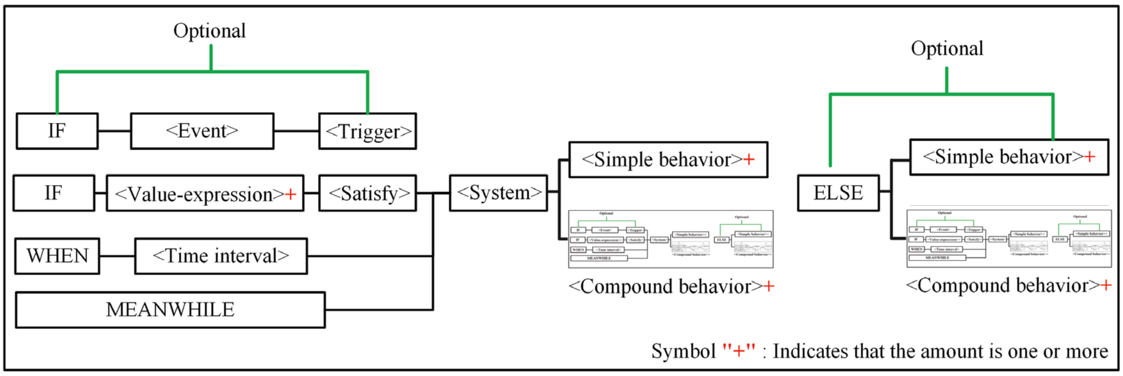


Fig. 8. Structure of Functional Requirement Template

with keywords (e.g., *IF THEN ELSE*, *AND*, *LOOP FOR*, *NOT*) are used to specify *CompoundBehaviorSentence* which are shown in Table 8.

For example, for a requirement of the AGNC system, "*IF Attitude Deviation Angle is bigger than 26 degrees in 30ms, THEN the Control Module sends Earth Capturing Success Information to AOCS every 5 ms until POWER_OFF*", firstly, *Attitude-Deviation-Angle* and *Earth-Capturing-Success-Information* are recommended into Data Dictionary. Secondly, *Control Module* and *AOCS* are recommended into the domain Glossary with type "system", and *POWER_OFF* is recommended with type "state". Thirdly, Rule S2, S6 and S9 are applied for the specification of the *Send* behavior, $SimpleValueCondition$ and $TimeCondition$ respectively. Finally, the requirement is rewritten with RNL templates as "[In 30ms, [IF *Attitude-Deviation-Angle* >26, [Loop [Send : *Earth-Capturing-Success-Information* to *Earth-Capturing-Success-Information-Port*] EACH 5ms UNTIL *POWER_OFF*]]]". Furthermore, if *Control Module* needs to send data together and count times at the same time, we can apply S12 to specify "MEANWHILE [Send : *Earth-Capturing-Success-Information-Signal* to *Earth-Capturing-Success-Information-Port* , Call : CountFunction]".

Therefore, with the combination of multi-level compound structures, RNL templates can represent complex behaviors. The structure of functional requirements is shown in Figure 8: From the structure of the template, we can see that the whole functional requirement template is a composite behavior. The behavior slot in the figure can be filled with a list of simple behaviors or nested composite behaviors, making the functional requirement template robust and scalable to express complex functional behavior.

Table 9. Non-functional Requirement Sentences for Restricted NL

No.	Sentence Pattern	Explanation	Examples
S13	ConstraintSentence	<property> <comparator> <value> <unit> <margin of error>	The period of SelfCheck function is 2ms. (Periodicity) The maximum allowed failure probability of Star Sensor A is 5%. (Failure probability) The main freq of CPU is less than 90% of the maximum. (Resource) The mass of the device is 4kg, with a 1% margin of error. (Resource) The severity of Losing Power in Control Subsystem is Level I. (Safety)
S14	RangeSentence	<property> within [min,max]	The normal temperature range of CPU is within [45,70] ° C.
S15	PeriodicitySentence	LOOP SentenceA EACH T UNTIL E	In working state, the motor receives instructions every 10ms. (Periodicity)
S16	TradeOffSentence	<property> optimizedBy <obj1>,<obj2>,<obj3> with <method>	The bandwidth of sending data optimizedBy the power, performance and failure rate with FunctionA. (Cost reduction)

5.2.5. Non-functional requirement Restriction Rules

Non-functional requirements (NFR) focus on how good the system does and the constraints it faces. Note that NFRs do not exist independently, they are a supplement to functional requirements.

NFR encompasses the following attributes: performance, safety, reliability, environmental constraints and so on. Among them, the NFR attributes that can be quantitatively described and analyzed includes performance, reliability, and safety. The time performance of the system, namely real-time requirements, include periodic/aperiodic execution, deadline, time-out, jitter, response time, schedulability, etc. Resource performance of the system includes resource consumption (e.g. power consumption, CPU, storage, bus) and constraints of the physical environment (e.g. temperature, quality, and pressure). Reliability requirements include such as the probability of system failure.

For an on-board ML component which lives in resource-constrained environment, in order to obtain the optimal solution, an NFR constraint on the property may vary over time and be influenced by objective cost, performance and failure rate. Thus, the property value is constrained by the trade-off of several objectives.

As shown in Table 9, we present the restricted sentences applicable to NFRs. On the one hand, we can give quantitative NFRs with *ConstraintSentence*, *RangeSentence* (a variation on *ConstraintSentence*). On the other hand, we can collaborate qualitative NFRs with corresponding System/Functional Templates, such as *ConditionnalSentence* for fault handling and *PeriodicitySentence* for periodicity, allowing system engineers to take functional and non-functional aspects into consideration at the design stage. Besides, the Trade-Off sentence describes the fact that the property has a relationship “optimizedBy” with the mentioned objectives and can be calculated by a method/function.

5.2.6. AI requirement Restriction Rules

Compared with conventional components, ML components automatically obtain a machine learning model by executing a training algorithm with training data-sets, and then evaluate trained models using testing data-sets. However, ML techniques are not applicable for behavioral modeling and should be treated as “black boxes”. A lack of requirements specification in ML components has a great impact on the whole system. Thus, as shown in Table 10, new requirement types are introduced and AI requirements are divided into six categories including training data, testing data, operational data, goal, AI functional requirements, AI non-functional requirements. Each category is described and further decomposed as follows.

Training Data & Testing Data. ML approaches generate models based on a set of examples (training data) and evaluate models on another set of examples (testing data). Both data are collected at development time. This category is refined into quantity, completeness, consistency and correctness.

- *Quantity.* It is obvious that “We expect a better performance of the model if we have more data and more diverse data”, thus, properties of data quantity like size, diversity are included in the data specifications.

Due to the fact that training data and testing data originate from the same source, they should share the same data specifications.

- *Completeness.* Completeness specifies the sparsity of data within each characteristic.
- *Consistency.* Consistency is guaranteed by the consistency between input data and test data (assuming the same environment).
- *Correctness.* Correctness is strongly influenced by the way how the data is cleaned, collected and stored.

Operational Data. Just like for conventional components, ML components take data as input at operation time, namely input data, or operational data. Though training data and testing data are approximately equally distributed, their relationship to the input data which tends to change with time or environment is unclear. Thus, it is important to check for consistency between input data and training, testing data, that is why we need to specify the distributions.

Goal. Goals are defined as well defined criteria for satisfaction such as evaluation metrics, an expected state of a system, an expected range of predictions. When a model makes bad predictions (outlier), it is natural to ask humans for information which may be unsuitable for time-critical tasks, so returning fallback values is a potential option in that case.

AI Functional Requirement. ML encompasses over a dozen problem types (e.g., Binary or Multiclass Classification, Regression, Clustering), with many more specific algorithm types (e.g., Logistic Regression, Linear Regression, Naive Bayes, Nearest Neighbor, Deep Neural Networks). In order to get a global view of ML solutions used, our specifications include problem types, algorithm types, layer information in Deep Learning systems and characteristics of models when updating.

AI Non-functional Requirement. To ensure the robustness of ML components, several dimensions of non-functional quality such as safety/reliability, real-time, performance and explainability are considered. Meanwhile, in respect that this article focuses on safety-critical systems, fairness which prevents the phenomenon that ML bias stems from prejudicial training data are ruled out.

- *Safety/Reliability.* If the safety/reliability estimation of individual predictions reach the threshold, changes such as retraining model or refining solutions should be made.
- *Real-time & Performance.* The accuracy (how correct the output is) and real-time (how fast the process is) properties of ML components need to be guaranteed, and fallback plans need to be specified.
- *Resource consumption.* The on-board computation and communication are resource-constrained, such as bandwidth for sending/receiving data, power consumption, memory and storage for data buffer.
- *Explainability.* Although ML models are opaque constructs that are intrinsically hard to interpret, engineers are able to give specifications to explain the behavior of the model around a given data point or features.

For example, a requirement in AGNC system, "*The Decision Subsystem needs to detect small celestial body in large images provided by the navigation camera, while a model with precision of 80% is needed.*", introduces a goal requirement "*The Decision Subsystem need to detect small celestial body in large images*" and a performance requirement "*Model satisfy a ConstraintSentence with precision >80%*" to specify an AI requirement of the Decision Subsystem.

5.3. The Meta-Model of RNLReq

The process of requirement specifications in RNLReq essentially contains three parts: data dictionary & domain glossary, requirement templates and sentence patterns. Thus, we define the meta-model of RNLReq as shown in Figure 9:

- *Data Dictionary & Domain Glossary*, used for specifying data and terminologies in requirements. Each data refers to an instance of $DataWord$ and each terminology refers to an instance of $Term$ in the meta-model.
- $Templates$, are organized as requirement templates for systems, sub-systems, functions, sub-functions and shared function blocks. Each template inherits from a super class *AbstractTemplate*, and higher-level templates can take lower-level ones as their composition.
- *Sentence Patterns*, each type of requirements in templates is organized as a list of sentences, which

Table 10. AI Requirement Template

Elements	SubElements	Explanations
ID		The unique identifier of AI template.
Name		The name of the module described in this template.
Training Data	Quantity	Specify sources of training data as a set like [A, B, C], and the corresponding diversity index as a set like [30%, 40%, 30%]. Specify the size of data-set. Specify the feature set. Specify the distribution of running data.
	Completeness	Specify the coverage of the data on the whole range of possible values in percent.
	Consistency	Ensure that format and representation of data that should be the same in the data-sets.
	Correctness	Specify the data format standard that must be followed. Specify the database technology to use. Specify the augmenting rules for data. Specify the purging rules for data. Specify the mechanism to store meta-data.
Testing Data		Same structure as Training Data.
Operational Data		Specify the distribution of input data at operation time.
Goal		System should be in an acceptable state set S. Specify the expected result range. Specify the fallback values in cases of an outlier predicted value.
AI Functional		Specify the problem type, e.g. binary or multiclass classification, regression, clustering. Specify the algorithm name. Specify the algorithm type, e.g. Logistic Regression, Linear Regression, Naive Bayes. Specify the set of layers, including [layer name, input shape, kernel size, padding method, activation], e.g. Convolutional layer 1: The input dimension is (1, 32, 32); the size of the convolution kernel is (5, 5); the padding method is valid; the number of convolution kernels is 6; the activation function is tanh. Specify the frequency of model updating or creation. Specify the strategy to make predictions when the model is being updated. Specify whether the model supports making multiple predictions at the same time.
AI Non-functional	Safety/Reliability	Specify reliability estimation of individual predictions in machine learning.
	Performance	The evaluation metrics (like precision, recall) should be in an acceptable range[min,max].
	Real-time	The time spent should be in an acceptable range[min,max]. Specify the fallback plans.
	Resource consumption	The resource spent should be in an acceptable range[min, max].
	Explainability	Specify the features that can derive the desired output or decisions. Specify the features that have a negative effect on the result. Give explanations on the behavior of the model around a given data point.
Description		Extra description of this module. (No restriction rule here)

satisfy predefined sentence patterns. Each sentence pattern inherits from a super class $Sentence$, and their transformation rules are defined in the $gen()$ operation of $Sentence$. $FunctionalSentence$ is composed of $ConditionalSentence$ and $CompoundBehaviorSentence$. $NonfunctionalSentence$ is composed of $ConstraintSentence$ and its variations. Most of the new AI requirement types reuse existing sentences, and Goal requirements are fulfilled with $GoalSentence$.

6. Automatic Generation of SysML Architecture Models

As safety-critical autonomous systems feature a tight combination of the system's computational, physical and communication parts which are heterogeneous, we propose a multi-view approach based on SysML for

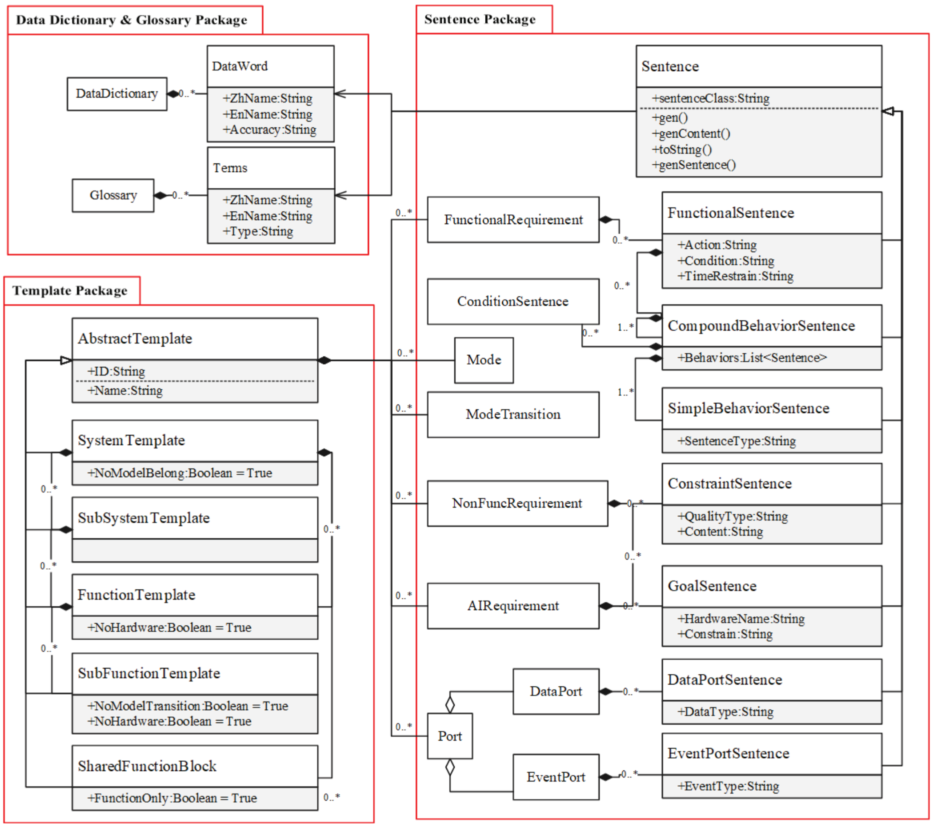


Fig. 9. The Meta-Model of RNLReq

modeling of these systems, which allows designers from different views to focus on particular aspects of the system. Firstly, we present the subset of SysML: SubSysML; then, the transformation from RNLReq to SysML architecture models is given.

6.1. Subset of SysML

The meta-model of SubSysML is summarized in Figure 10. SubSysML consists of a block definition diagram (BDD), internal block diagrams (IBD), activity diagrams (ActD) and stereotypes such as ConstraintRequirement and MLComponent. This paper considers several complementary views of safety-critical autonomous systems: hierarchy view, inter-communication view, functional view, non-functional view, and data view. Each view highlights certain features of the system and are illustrated as follows.

- *Hierarchy view.* Due to the fact that most systems are built compositionally and made of sub-systems or components which are connected in some way, hierarchy view reflects the compositionality of the systems. Hierarchy view is represented by a BDD and its elements such as blocks and MLcomponents (nodes in yellow color). Blocks are the key elements of BDDs and can represent basic structural decompositions for modeling static structures of systems. Properties of blocks represent the states and attributes of systems. MLComponent is a sub-stereotype of SysML block stereotype and has properties describing characteristics of data and algorithms in AI Requirement Template. Given a system, SubSysML specifies the system model as a BDD. Inside the BDD, it uses blocks or MLcomponents to express the hierarchical structure of the system.
- *Inter-communication view.* Inter-communication view reflects the interactions of different sub-components in the systems. Inter-communication view is represented by IBDs and its elements such as Port and

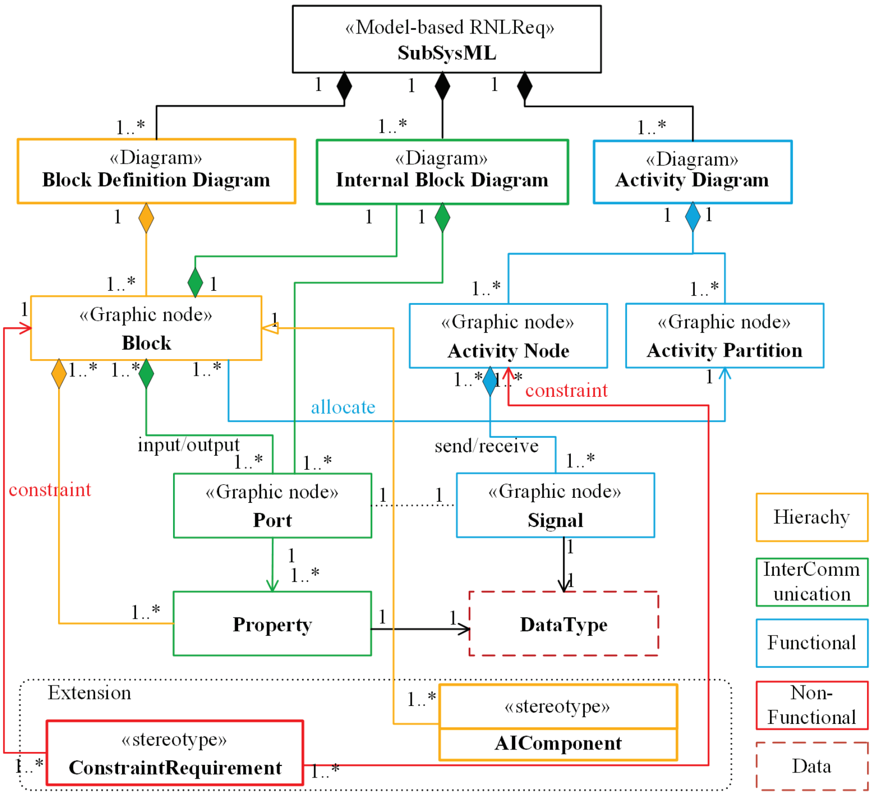


Fig. 10. Meta Model of SysML Model

Property (nodes in green color). For instance, given a parent block, we use an IBD to present the communication among its son blocks via ports. IBDs and BDDs provide complementary views of a block.

- *Functional view.* Functional view reflects the dynamic behavior of a system. Functional view is represented by an activity diagram and its elements (nodes in blue color). SubSysML contains several kinds of Activity Nodes such as call behavior actions (represent a basic unit of functionality within the activity), initial/final node (mark the start/end of an activity), fork/join nodes (mark the start/end of concurrent sequences), decision/merge node (mark the start/end of alternative sequences), send signal actions (send a signal instance to a target), accept event actions (wait for a signal to continue the activity). ActivityPartition allocates activity nodes to blocks. ConstraintRequirement for an activity node can represent some constraints such as timing that should be satisfied by a running component.

- *Non-functional view.* In non-functional view, we extend the existing SysML *Constraint* stereotype to specify the non-functional properties, including safety, reliability, performance and so on. Figure 11 is the definition of the ConstraintRequirement of the system. ConstraintRequirement can contain an OCL expression that formally states an acceptance condition, threshold for a traditional component, or a data requirement for an ML component. For example, Figure 12 shows the use of ConstraintRequirement to describe the property of the working temperatures of a sensor.

- *Data view.* Data view captures the data structure of the system using objects, attributes, operations, and relationships. We extend SysML's *ValueType* stereotype to model the data structure of the system which is specified by Data Dictionary. The *BaseType* package contains the predefined primitive value types such as Integer_16, Float_64 and Boolean. The *DataType* package contains all data types defined by system engineers.

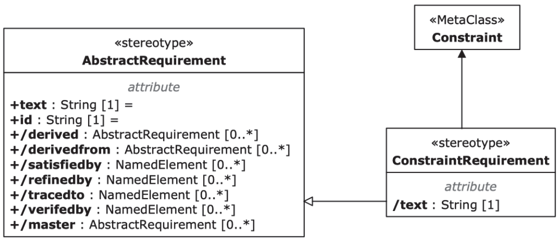

«stereotype»
AbstractRequirement
attribute
+text : String [1] =
+id : String [1] =
+/derived : AbstractRequirement [0..*]
+/derivedfrom : AbstractRequirement [0..*]
+/satisfiedby : NamedElement [0..*]
+/refinedby : NamedElement [0..*]
+/tracedto : NamedElement [0..*]
+/verifedby : NamedElement [0..*]
+/master : AbstractRequirement [0..*]
«MetaClass»
Constraint
«stereotype»
ConstraintRequirement
attribute
/text : String [1]

Fig. 11. Definition of ConstraintRequirement

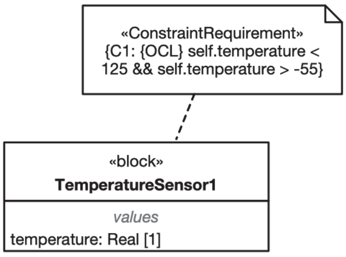


Fig. 12. Threshold represented by ConstraintRequirement

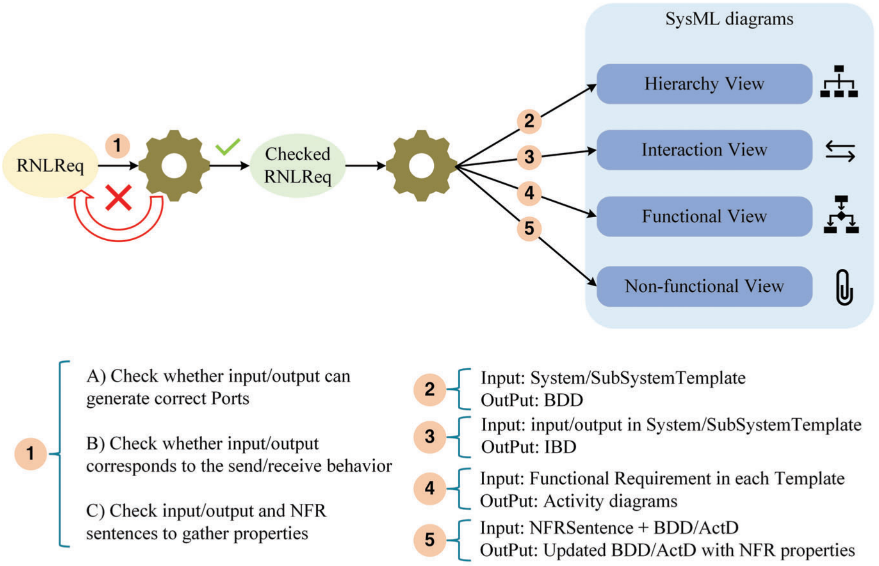

SysML diagrams
Hierarchy View
Interaction View
Functional View
Non-functional View
RNLReq
1
Checked RNLReq
2
3
4
5
1
A) Check whether input/output can generate correct Ports
B) Check whether input/output corresponds to the send/receive behavior
C) Check input/output and NFR sentences to gather properties
2
Input: System/SubSystemTemplate
OutPut: BDD
3
Input: input/output in System/SubSystemTemplate
OutPut: IBD
4
Input: Functional Requirement in each Template
OutPut: Activity diagrams
5
Input: NFRSentence + BDD/ActD
OutPut: Updated BDD/ActD with NFR properties

Fig. 13. Generation of SysML Model

6.2. Transformation from RNLReq to SubSysML

As shown in Figure 13, the transformation from RNLReq to SubSysML is composed of five parts. The first part is intended for identifying incorrect templates and preprocessing the RNLReq. For example, if a SystemTemplate has an input port while the corresponding output port cannot be discovered, the RNLReq is incorrect and needs to be repaired. Part 2-5 transform information in checked RNLReq to separate diagrams with mapping rules. The process is achieved by the following algorithms.

As described in the Algorithm 2, it takes RNLReq meta-model (given in Section 5.3) as input. In Line 1-3, each data defined in the Data Dictionary will be transformed into the value types in SubSysML. In Line 4-6, a block definition diagram is generated for the SystemTemplate at the top level. In Line 7-11, each SubSystemTemplate considered as siblings of the SystemTemplate will be transformed into a block and attached to the BDD, and the properties of blocks are generated. The hierarchy of SystemTemplates is generated into a tree: SystemTemplate at the top level is the root block, and its sub-templates are modeled as blocks in the next level. In Line 10, each block connects with the top system block via connectors using a composition relationship.

The process of generating inter-communication view is defined by Algorithm 3. We first check whether each SubSystemTemplate (son) contains interface requirements. If it contains I/O information, then an IBD is generated and associated with the corresponding block of the SystemTemplate (parent) at the higher level.

Algorithm 2 Generate hierarchy/data view

```
Input: RNLReq
Output: BDD, BaseTypes

for each Dataword $DW$ in DataDictionary do
    Generate a Datatype $DT$ with $DW$'s name;
end for
Get top SystemTemplate $topST$ in RNLReq;
Generate a SysML model $Model$ with topST's name;
Generate an block definition diagram $BDD$ with $topST$'s name;
Generate a block with $topST$'s name and its corresponding attributes;
for each SubSystemTemplate $subT$ in topST with SYSTEM type do
    Generate a block $Block$ with $subT$'s name and its corresponding attributes;
    Generate a composition connector named $topST+subT$ and multiplicities of $subT$;
end for
```

Then, the block corresponding to each SubSystemTemplate will be added into the IBD, and ports will be used to express the communication between them.

Algorithm 3 Generate inter-communication view

```
Input: RNLReq
Output: IBDs

for each SystemTemplate $ST$ in RNLReq do
    Generate a set $S$, add a block $HostBlock$ named $ST$
    for each SubSystemTemplate $subT$ with SYSTEM type do
        if $subT$ contains Input/Output data then
            Establish a connection $c$ and attach it to Set $S$;
            Attach $subT$'s block to $HostBlock$ as a property;
        end if
    end for
    Generate an internal block diagram $IBD$ with $ST$'s name;
    Generate a block $HostBlock$ with $ST$'s name;
    Link $IBD$ with its corresponding $BDD$;
    for each connection $c$ in S do
        Generate ports with its corresponding data types and attach them to $subT$ property;
        Generate connectors between ports;
    end for
end for
```

All the functional requirements are translated into activity diagrams, which is shown in the Algorithm 4. The structure of Functional Requirement Template in Figure 8 is considered as a tree. Each SimpleSentence and ConditionSentence is considered as a leaf node (which is solvable and has a corresponding SysML element). Line 9-12 and Line 21-27 show the rule mapping ConditionSentence and SimpleSentence into corresponding Activity Nodes. Line 13-14 shows that ComplexSentence is a non-leaf node and contains several leaf nodes inside. We search the tree recursively, until all the sub-trees are searched and solved, which means the entire function requirements have been transformed into an activity diagram. Besides, as using swimlanes to notate ActivityPartitions is usually complex and impractical, Line 22-23 allocates nodes to partitions by placing the partition name in parenthesis above the ActivityNode name into the ActivityNode symbol.

In addition, Algorithm 7 shows the process that non-functional/AI requirements is inserted into the diagrams generated in the steps above. In Line 4-7, ConstraintSentence and RangeSentence described in Section 5.2.5 are transformed into OCL expression and then added into ConstraintRequirement. In Line 8-12, behavior constraints such as deadline and WCET are added into activity diagrams as a constraint of

Algorithm 4 Generate functional view

Input: SystemTemplates containing Functional Requirements
Output: ActDs

```
for each SystemTemplate $ST$ in RNLReq do
    Generate an activity diagram $ActD$ with $ST$'s name;
    Get behavior sentence $BS$ in $ST$;
    ProcessComplexSentence($BS$);
    function ProcessComplexSentence(Sentence $BS$)
        Get sentence list $SenList$ in $BS$;
        for each sentence $Sen$ in $SenList$ do
            if $Sen$ is compound then
                Extract ConditionSentence $Con$, IfSentence $Action1$, ElseSentence $Action2$ in $Sen$;
                If $SS$ is EventCondition, generate AcceptEventAction and Pin;
                If $SS$ is ValueCondition, generate DecisionNode and MergeNode;
                If $SS$ is ConcurrentSentence, generate ForkNode and JoinNode;
                ProcessComplexSentence($Action1$);
                ProcessComplexSentence($Action2$);
                Connect nodes generated in steps above;
            else
                ProcessSimpleSentence($BS$);
            end if
        end for
    end function
    function ProcessSimpleSentence(SimpleSentence $SS$)
        Let name of SystemTemplate $SS$ belongs to be partition name;
        Append partition name to node name;
        Generate CallBehaviorAction if $SS$ is Call, link it to the activity if $SS$ points to another FunctionTemplate;
        Generate CallBehaviorAction and Pin if $SS$ is Assign;
        Generate SendSignalAction and Pin if $SS$ is Send;
    end function
end for
```

corresponding ActivityNode. In Line 13-23, each SystemTemplate containing AIRequirement is transformed to blocks stereotyped with MLComponent in the BDD. The transformation of NFRs in AIRequirement (Line 17-21) is similar to the Line 4-7.

7. Prototype Tool

We have implemented our approach in a prototype tool named SCASAMTool (Safety-Critical Autonomous Systems Architecture Modeling Tool). It mainly includes the intelligent recommendation tool IR4RNL, the requirement specification tool based on restricted natural language RNLReq, and the SysML model automatic generation tool RNLReq2SysML. Figure 14 shows the overall architecture of SCASAMTool. It is developed on the Eclipse platform integrated with Papyrus plug-in. The open-source modeling tool Papyrus [6] provides complete modeling support for SysML. A brief description of each module is given below.

- *IR4RNL.* IR4RNL includes term recommendation and requirement classification methods. Especially, terminology recommendation method takes natural language requirements as input, and recommends candidate data dictionary and candidate domain glossary through interaction with requirement engineers. The requirement classification method predicts the category of requirements by the trained model.

[6] https://www.eclipse.org/papyrus/

Algorithm 5 Transform NFRs/AI requirements of system templates into SysML extensions

Input: RNLReq
Output: Model

```
for each SystemTemplate $ST$ in RNLReq do
    if $ST$ contains NFR $quality$ then
        Apply corresponding profile to $Model$;
        if $quality$ is attribute then
            Translate ConstraintSentence and RangeSentence into OCL expressions with the attribute and generate ConstraintBlock;
            Apply corresponding constraints to $Block$;
        end if
        if $quality$ is deadline or WCET then
            Apply corresponding NFR stereotype to Model;
            Apply corresponding constraints to $ActivityNode$;
        end if
    end if
    if $ST$ contains AIRequirement $AIreq$ then
        Apply MLComponent stereotype to corresponding Block $AIblock$;
        Generate properties in $AIblock$ for AI Functional Requirement;
        Generate properties in $AIblock$ for input data, training data and testing data;
        Process AI NFRs with ConstraintRequirement;
        if $AIreq$ contains goal then
            Translate GoalSentence into OCL expressions with the attribute and generate GoalRequirement $gr$;
            If $AIreq$ is called in a function, generate a postcondition with $gr$ to corresponding $ActivityNode$;
        end if
    end if
end for
```

Table 11. Statistics of functions of the prototype tool

Function/Module	Description	Java Code Lines
Intelligent Recommendation	IR4RNL module	1000+
ReqTemplate Viewer	Requirement view module, an external interface for user operations	2000+
ReqTemplate Controller	Requirement model control module	2000+
RNLReq2SysML Mapper	RNL to SysML element mapping	1000+
SysML Element Generator	Generate XML files corresponding to SysML model elements	1000+
SysML Layout Visualizer	Generate the XML file corresponding to the SysML graphic layout	1500+

- *RNLReq.* RNLReq further formulates the data dictionary, domain glossary, and category of requirements from IR4RNL, and specifies the RNL structured requirements with the requirement templates.

- *RNLReq2SysML.* RNLReq2SysML takes RNLReq requirement specification as input, and generates SysML models through transformation rules.

The SCASAMTool adopts a modular architecture, in which it includes 6 modules such as Terminology Recommendation, ReqTemplate Viewer, ReqTemplate Controller, RNLReq2SysML Mapper, SysML Element Generator and SysML Layout Visualizer. Table 11 shows the main function code statistics of the SCASAMTool.

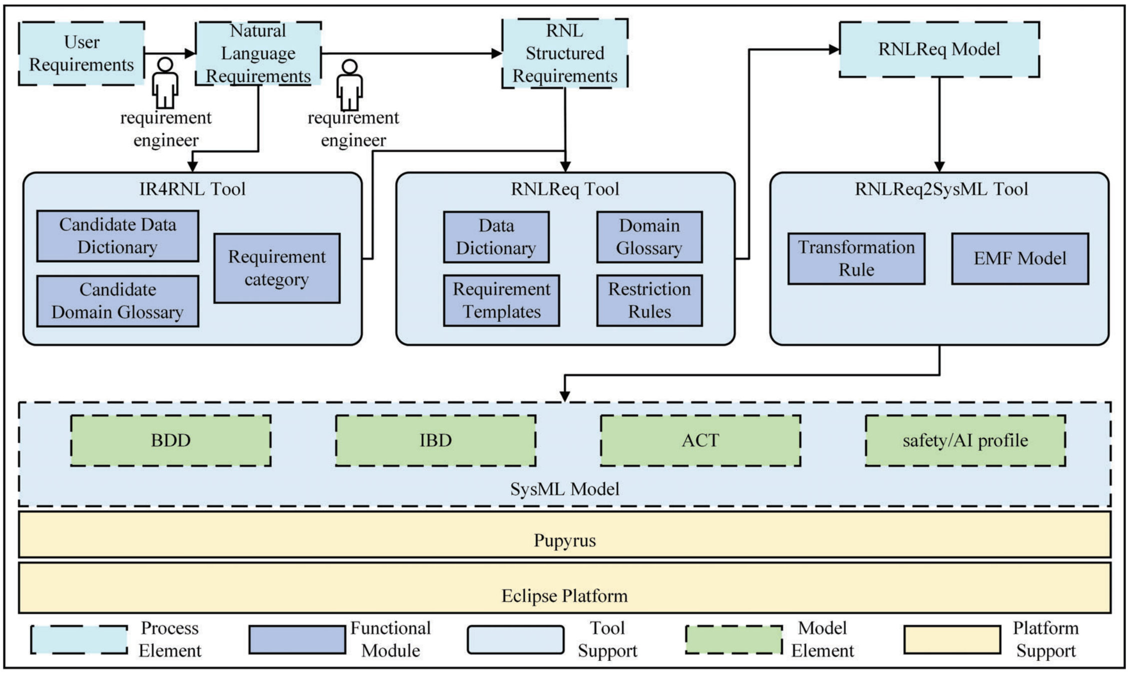


Fig. 14. SCASAMTool Achitecture

8. Evaluation

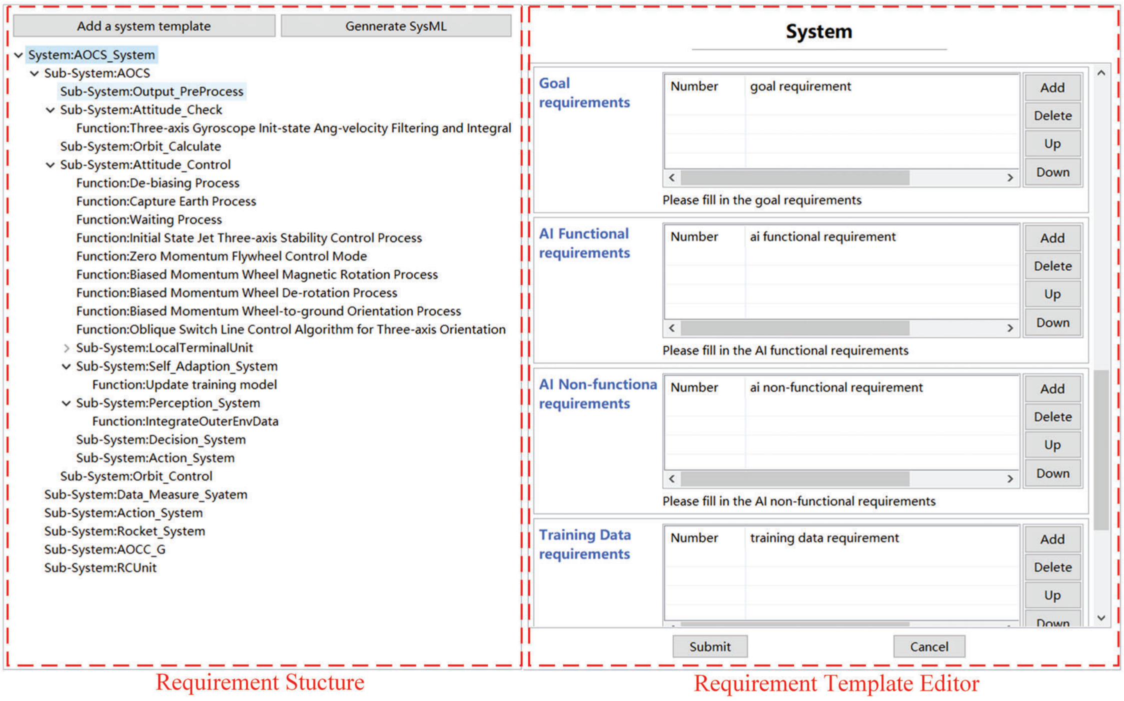
8.1. Industrial Case Studies

The requirement document of AGNC is obtained from the industrial partner. The document extends to more than 200 pages, written in Chinese and contains nine sections (such as Attitude Determination, Orbit calculation, Attitude Control, Orbit Control, etc.). For the sake of confidentiality, just a small portion of the requirement is allowed to present in this paper.

First, we recommend data dictionary, domain glossary, and requirement category based on the original requirement document with the prototype tool. Then, we specify the RNLReq modeling for AGNC system in Figure 15. The left side is the hierarchical structure of the requirements. The right side is the requirement template editor, the engineer can add and edit different categories of requirements through the restriction rules of the requirement template. After the entire RNL template has been completed, a SysML model of the AGNC system can be generated automatically.

Next, the generated SysML system architecture model is presented in Figure 16, in which the yellow blocks represent *device* stereotype and the blue ones represent the ML components in the autonomous mode. The architecture of AGNC system is a hierarchical control structure: 1) According to the working process of the system, the whole system is divided into several parts: data sampling sub-system, control function sub-system, action sub-system, etc. 2) For instance, control function sub-system can be further divided into attitude control sub-system, attitude check sub-system, orbit calculate sub-system, orbit control sub-system and output pre-processing sub-system. 3) Attitude control sub-system is further divided into Local Terminal Unit, perception sub-system, decision sub-system, self-adaption sub-system, and action sub-system. 4) The devices include sensor components (such as navigation cameras, star sensors, gyroscopes, and accelerometers) which collects signals from environment and data process unit which process the data and command.

Then, Figure 17 shows the system inter-communication model of the Attitude Control sub-system, which has a stable mode (filled with orange color) and an autonomous mode (filled with blue color). On the left side, interactions of the autonomous mode are illustrated, local terminal unit first collects data (such as navigation images and physical parameters) and sends them to the perception sub-system. Then the perception sub-system integrates the environment information and sends them to the decision sub-system. Decision sub-system delivers plans to the action sub-system which controls the actual executive agents, and



- System:AOCS_System
  - Sub-System:AOCS
    - Sub-System:Output_PreProcess
    - Sub-System:Attitude_Check
      - Function:Three-axis Gyroscope Init-state Ang-velocity Filtering and Integral
    - Sub-System:Orbit_Calculate
    - Sub-System:Attitude_Control
      - Function:De-biasing Process
      - Function:Capture Earth Process
      - Function:Waiting Process
      - Function:Initial State Jet Three-axis Stability Control Process
      - Function:Zero Momentum Flywheel Control Mode
      - Function:Biased Momentum Wheel Magnetic Rotation Process
      - Function:Biased Momentum Wheel De-rotation Process
      - Function:Biased Momentum Wheel-to-ground Orientation Process
      - Function:Oblique Switch Line Control Algorithm for Three-axis Orientation
      - Sub-System:LocalTerminalUnit
      - Sub-System:Self_Adaption_System
        - Function:Update training model
      - Sub-System:Perception_System
        - Function:IntegrateOuterEnvData
      - Sub-System:Decision_System
      - Sub-System:Action_System
    - Sub-System:Orbit_Control
  - Sub-System:Data_Measure_Syatem
  - Sub-System:Action_System
  - Sub-System:Rocket_System
  - Sub-System:AOCC_G
  - Sub-System:RCUnit

Goal requirements

Number	goal requirement

Please fill in the goal requirements

AI Functional requirements

Number	ai functional requirement

Please fill in the AI functional requirements

AI Non-functiona requirements

Number	ai non-functional requirement

Please fill in the AI non-functional requirements

Training Data requirements

Number	training data requirement

Fig. 15. Structure of RNLReq requirements of AGNC system

the feedback of executive agents will help improve the ability of self-adaption sub-system. On the right side, interactions of the stable mode are simpler: control components receive data from local terminal unit and send commands to executive agents directly. Compared to the process of autonomous mode, process of stable mode lacks the ability of self-adaption and a central decision module.

Besides, Figure 18 shows the activity diagram of a sub-system for eliminating the initial deviation in AGNC system. Eliminate Initial Deviation(EID) of Attitude Control sub-system eliminates the angular rate of attitude generated by the separation of satellites from launch vehicles by calling some three-axis attitude control algorithms of spacecraft. First, it sends the angular velocity of X, Y, Z axis gyros to the control sub-system. Depending on the values, it calculates the state of orbit. Then, it sends signals to executive agents and keeps a backup of data. If getting into orbit, it will send a signal for the success of first catch of the earth.

Finally, Figure 19 shows a safety requirement about losing control of the control function sub-system, "Control function sub-system may occur a current short-circuit fault, a memory fault, and an over-temperature fault. The probability of a current over-temperature fault is 1.0E-7". According to the Failure entry defined in the data dictionary, the corresponding user-defined stereotype Failure is generated. Failure level, failure type, failure probability, etc., in its internal structure are converted into the attributes such as FailureEffect, Severity, Likelihood, and ProbabilityValue in the Failure stereotype. For the block ControlFunction-init having the safety requirement, Failure stereotype is used as the type of the attributes CurrentFault, MemoryFault, and CurrentThermal in the block, and values of these attributes are acquired from the safety sentences in RNL templates, for example, the ProbabilityValue of CurrentThermal is 1.0E-7.

Statistics data of the case study is given in Table 12. It shows that the case study, 1) covers all the RNL templates mentioned in Section 5.2; 2) has a considerable number of RNL elements; 3) shows the correspondences between RNL elements and SysML model elements.

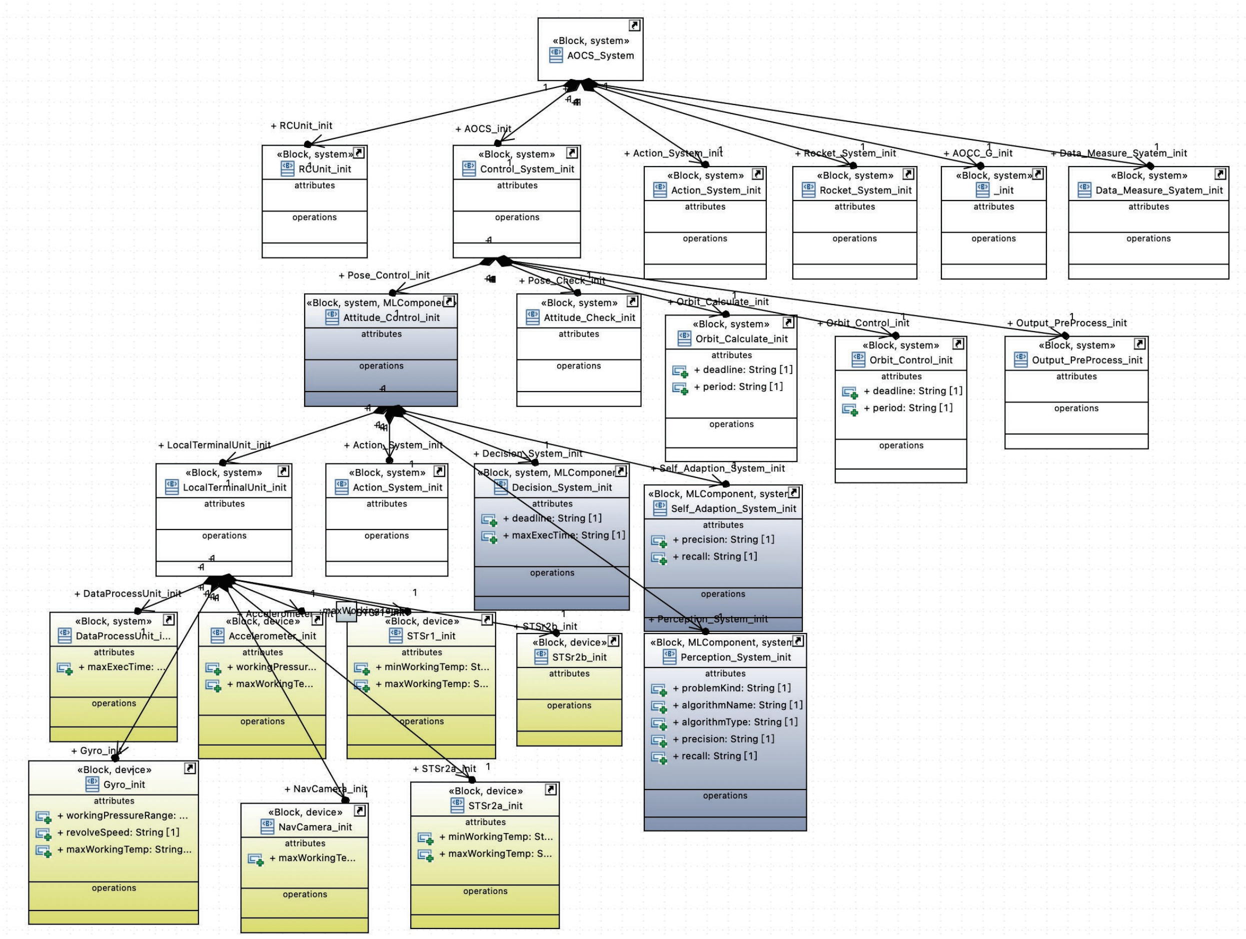


Fig. 16. System architecture model of AGNC system

Table 12. Statistics of Modeling elements

RNL elements	RNL counts	SysML Model Elements	SysML counts
System Template	1	Block	1
Subsystem Template	112	Block	112
Data Dictionary	172	DataType	179
Input/Output Data/Event between systems	104	Port	104
Simple Behavior (Call, Assign, Send)	172	CallBehaviorAction,SendSignalAction	172
Value Condition	32	DecisionNode,MergeNode	64
Static NFR attributes (e.g. deadline, WCET, performance)	102	properties of NFRBlock	102
Runtime NFR attributes (e.g. WCET)	15	ConstraintRequirement in Activity Diagram	15
Goal Requirement	10	ConstraintRequirement	10

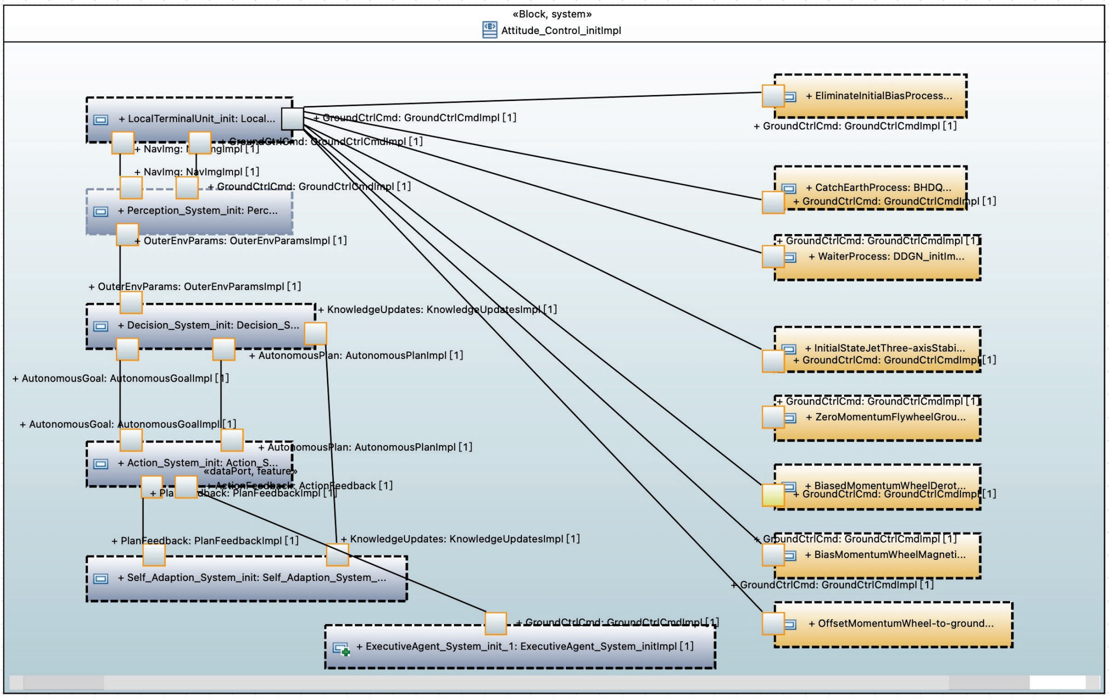

«Block, system»
Attitude_Control_initImpl
+ LocalTerminalUnit_init: Local...
+ GroundCtrlCmd: GroundCtrlCmdImpl [1]
+ NavImg: NavImgImpl [1]
+ GroundCtrlCmd: GroundCtrlCmdImpl [1]
+ NavImg: NavImgImpl [1]
+ GroundCtrlCmd: GroundCtrlCmdImpl [1]
+ Perception_System_init: Perc...
+ OuterEnvParams: OuterEnvParamsImpl [1]
+ OuterEnvParams: OuterEnvParamsImpl [1]
+ KnowledgeUpdates: KnowledgeUpdatesImpl [1]
+ Decision_System_init: Decision_S...
+ AutonomousPlan: AutonomousPlanImpl [1]
+ AutonomousGoal: AutonomousGoalImpl [1]
+ AutonomousGoal: AutonomousGoalImpl [1]
+ AutonomousPlan: AutonomousPlanImpl [1]
+ Action_System_init: Action_S...
«dataPort, feature»
+ ActionFeedback: ActionFeedback [1]
+ PlanFeedback: PlanFeedbackImpl [1]
+ PlanFeedback: PlanFeedbackImpl [1]
+ KnowledgeUpdates: KnowledgeUpdatesImpl [1]
+ Self_Adaption_System_init: Self_Adaption_System_...
+ GroundCtrlCmd: GroundCtrlCmdImpl [1]
+ ExecutiveAgent_System_init_1: ExecutiveAgent_System_initImpl [1]
+ EliminateInitialBiasProcess...
+ GroundCtrlCmd: GroundCtrlCmdImpl [1]
+ CatchEarthProcess: BHDQ...
+ GroundCtrlCmd: GroundCtrlCmdImpl [1]
+ GroundCtrlCmd: GroundCtrlCmdImpl [1]
+ WaiterProcess: DDGN_initIm...
+ InitialStateJetThree-axisStabi...
+ GroundCtrlCmd: GroundCtrlCmdImpl [1]
+ GroundCtrlCmd: GroundCtrlCmdImpl [1]
+ ZeroMomentumFlywheelGrou...
+ BiasedMomentumWheelDerot...
+ GroundCtrlCmd: GroundCtrlCmdImpl [1]
+ GroundCtrlCmd: GroundCtrlCmdImpl [1]
+ BiasMomentumWheelMagneti...
+ GroundCtrlCmd: GroundCtrlCmdImpl [1]
+ OffsetMomentumWheel-to-ground...

Fig. 17. Framework of Attitude Control Subsystem

8.2. Analysis Results

8.2.1. Analysis of RNLReq2SysML

- Research Questions

The major objective of this subsection is to evaluate the effectiveness of our approach. This objective is decomposed into the following research questions:

RQ1. Can we use our approach to cover the requirements from the aerospace domain?

As a model generation approach, the RNL templates proposed in this paper should be able to cover various kinds of requirements. In order to answer the question, we have applied our approach to requirements from different aerospace sub-systems.

Table 13 provides the name, the short description, the domain and the number and coverage of the requirements from three industry cases. For the description of Case-A, see section 3.1. For Case-B, the rocket control system refers to a system that uses sensors, on-board computers, and communication equipment to control the launch of a rocket, and provide stability for the rocket. Case-C is the aircraft air pressurization system (AAPS), which is one of sub-systems to ensure the normal operation of the aircraft. Its core component is the air pressurization controller, which is used to drive the motor to operate, thereby driving the air compressor to work. The coverage of RNL templates is nearly 91 percent and we believe the samples are enough for evaluation. Requirement statements that cannot be covered include one or more complex mathematical expressions. The predefined sentence patterns show good ability in expressing requirements, and if one wants to apply RNL to other domains, the predefined sentence patterns are designed with high flexibility and expansibility.

RQ2. Are our transformation rules from RNLReq to generated SysML models complete?

We use the following procedure to check the completeness of our approach: 1) Take the same requirements specifications (AGNC system) as input; 2) Generate the SysML models with RNLReq2SysML tool; 3) Check whether each template have been transformed into SysML elements; 4) Check whether Data Dictionary has been transformed as data types; 5) Check whether each SysML diagram is well-organized and shows up in the Papyrus environment.

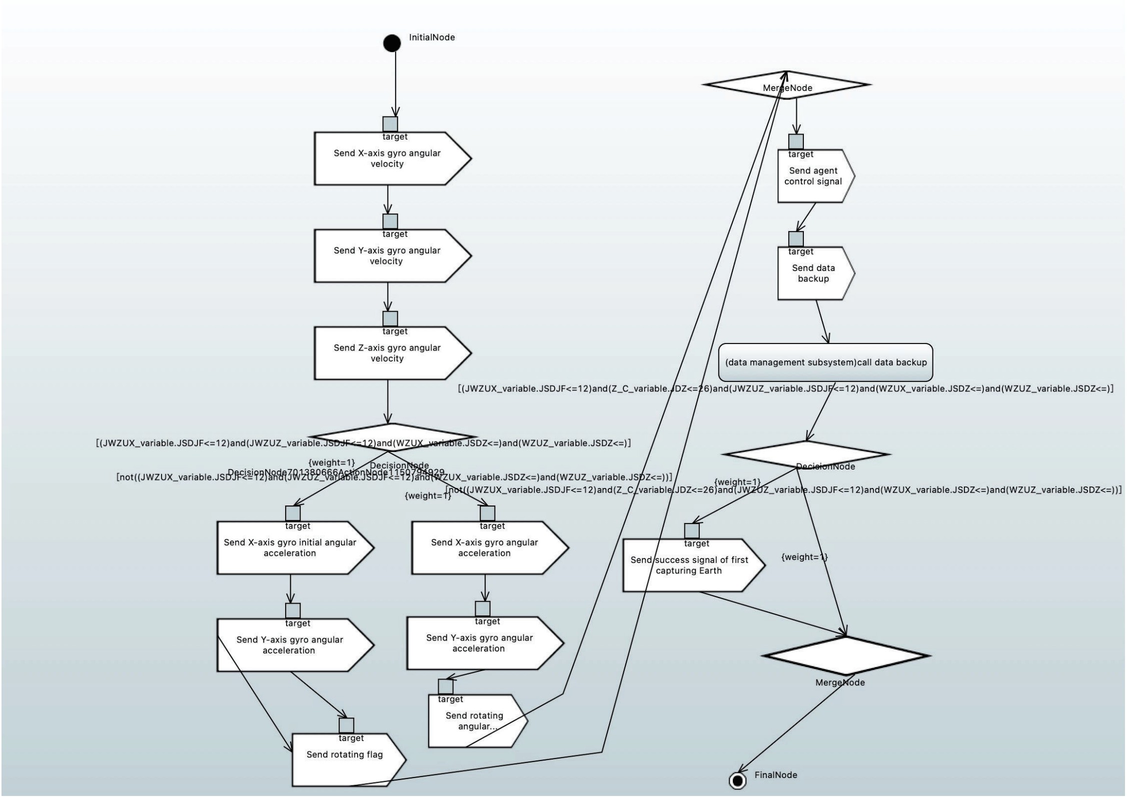
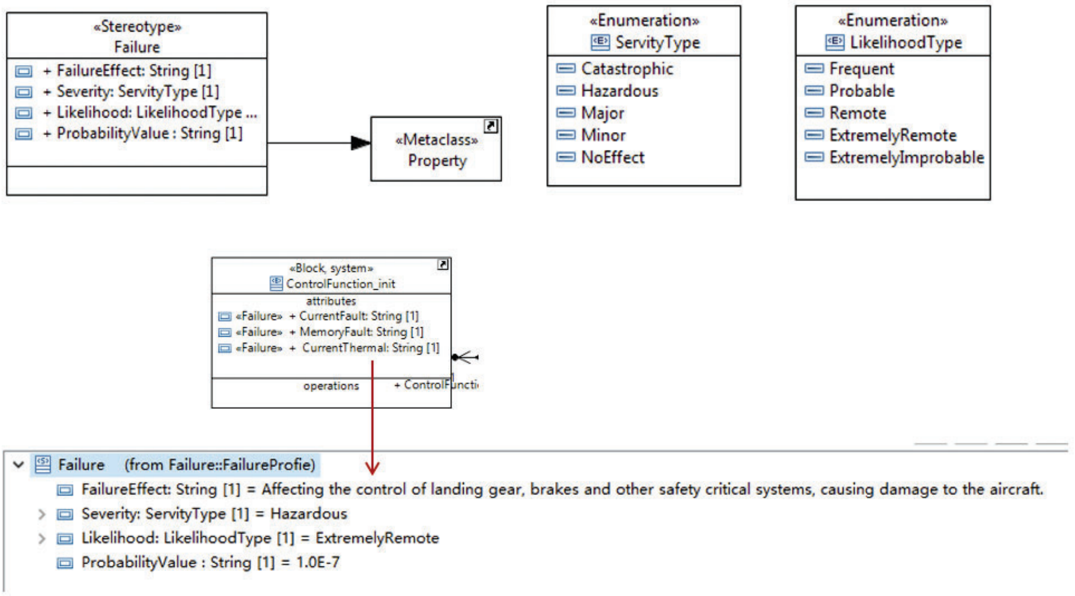

InitialNode
target
Send X-axis gyro angular velocity
target
Send Y-axis gyro angular velocity
target
Send Z-axis gyro angular velocity
[(JWZUX_variable.JSDJF<=12)and(JWZUZ_variable.JSDJF<=12)and(WZUX_variable.JSDZ<=)and(WZUZ_variable.JSDZ<=)]
{weight=1}
DecisionNode
DecisionNode701380666ActionNode1150794929
[not((JWZUX_variable.JSDJF<=12)and(JWZUZ_variable.JSDJF<=12)and(WZUX_variable.JSDZ<=)and(WZUZ_variable.JSDZ<=))]
{weight=1}
target
Send X-axis gyro initial angular acceleration
target
Send X-axis gyro angular acceleration
target
Send Y-axis gyro angular acceleration
target
Send Y-axis gyro angular acceleration
target
Send rotating angular...
target
Send rotating flag
MergeNode
target
Send agent control signal
target
Send data backup
(data management subsystem)call data backup
[(JWZUX_variable.JSDJF<=12)and(Z_C_variable.JDZ<=26)and(JWZUZ_variable.JSDJF<=12)and(WZUX_variable.JSDZ<=)and(WZUZ_variable.JSDZ<=)]
DecisionNode
{weight=1}
[not((JWZUX_variable.JSDJF<=12)and(Z_C_variable.JDZ<=26)and(JWZUZ_variable.JSDJF<=12)and(WZUX_variable.JSDZ<=)and(WZUZ_variable.JSDZ<=))]
target
Send success signal of first capturing Earth
{weight=1}
MergeNode
FinalNode

Fig. 18. Function of eliminating initial deviation



Failure (from Failure::FailureProfie)
- FailureEffect: String [1] = Affecting the control of landing gear, brakes and other safety critical systems, causing damage to the aircraft.
- Severity: ServityType [1] = Hazardous
- Likelihood: LikelihoodType [1] = ExtremelyRemote
- ProbabilityValue : String [1] = 1.0E-7

Fig. 19. Property of Control Function Subsystem

Table 13. Description of Case Studies

Case	Description	Domain	Number of Requirements	Coverage
Case-A	Autonomous Guidance, Navigation and Control (AGNC)	Space	280	257/280
Case-B	Rocket control system	Space	146	141/146
Case-C	Aircraft Air Pressurization System (AAPS)	Aviation	169	155/169

Table 14. Comparison with other existing Tools

Modeling Concepts	CM-Builder	DC-Builder	UMLG	aToucan	ABCD	RNLReq2SysML
Class	Y	Y	Y	Y	Y	Y
Composition	N	Y	N	Y	Y	Y
Attributes	Y	Y	Y	Y	Y	Y
Methods	N	N	N	Y	Y	Y
Multiplicity	Y	N	N	N	Y	Y
InterCommunication	N	N	N	N	N	Y
Activity	N	N	N	Y	N	Y
NFR	N	N	N	N	N	Y

Following the above procedure and engineers' feedback from industry, for all RNLreq requirements of the case study AGNC, we achieved 100% completeness of transformation with RNLReq2SysML. Statistics of RNL elements and SysML elements keeps consistent with each other as presented in Table 12, note that the number of entries in Data Dictionary and the predefined 8 primitive types constitutes the number of DataType. Besides, the generated SysML diagrams conforms to the SysML 1.6 standard.

RQ3. How many SysML modeling concepts can our approach cover when compared with other approaches?

We compare the proposed RNLReq2SysML tool with the international tools: CM-Builder[HG03], DC-Builder[HH12], UMLG[ISBM09], aToucan[YBL13], ABCD[WAD16]. Please note that, in the RNLReq2SysMLtool, Chinese natural language requirement text is mainly considered, and subsequently remarked that each term in glossary and data dictionary contains both Chinese and English names, therefore English SysML models can be generated. By conducting a survey on the modeling ability of these tools, we evaluate their coverage of the modeling concepts such as of SysML/UML. As shown in Table 14, compared to other tools, 1) with the help of SysML BDDs, RNLReq2SysML tool can capture more structure information such as multiplicity, compositions from NL requirements; 2) with the help of SysML IBDs, RNLReq2SysML can generate diagrams describing intercommunication and activity; 3) RNLReq2SysML supports the description of NFRs. By using RNL and SysML together, SysML's advantages in graphical modeling are enforced with RNL's efficiency in capturing requirements.

8.2.2. Analysis of IR4RNL

- Research Questions

IR4RNL method evaluation aims to answer the following research questions (RQs):

RQ4. How effective is the candidate term extraction method combined with engineer feedback loop? In the term recommendation method, we add engineer feedback loop to the candidate term extraction. The aim of RQ4 is to evaluate whether the engineer feedback loop can give better term recommendation.

RQ5. Which similarity measure(s) and clustering algorithms(s) yield the most accurate cluster? The choice of similarity measures and clustering algorithms can have a major impact on the quality of the generated clusters. The aim of RQ5 is to examine alternative combinations of similarity measures and clustering algorithms, and identify the best alternatives in terms of accuracy.

RQ6. Which feature extraction method(s) and classifiers(s) yield the most accurate results in requirement classification method? The choice of feature extraction methods and classification algorithms can have a major impact on the quality of the requirement classification method. The aim of RQ6 is to examine alternative combinations of feature extraction methods and classifiers, and identify the best alternatives in classification accuracy.

- Evaluation Procedure

Our case study is based on three evaluation procedures: the first is to evaluate the accuracy of candidate terms extraction, the second is to evaluate the accuracy of term clustering, and the third is to evaluate the accuracy of requirement classification.

RQ4. Accuracy of Candidate Terms. We use standard classification accuracy metrics, precision and recall, to evaluate the accuracy of candidate terms. The task at hand is to determine which candidate terms belong to the glossary and which ones do not. The candidate terms that belong to the glossary are True Positives (TP), and the ones that do not belong are False Positives (FP). The glossary terms that are not extracted by our tool are False Negatives (FN). Precision accounts for the quality of results, i.e., smaller number of FPs, and is computed as $TP/(TP+FP)$. Recall accounts for the coverage, i.e., smaller number of FNs, and is computed as $TP/(TP+FN)$. We use F-measure to combine precision and recall into one metric. F-measure is computed as: $2 \times \mathit{Precision} \times \mathit{Recall}/(\mathit{Precision}+\mathit{Recall})$.

RQ5. Accuracy of Clustering. In this paper, meaningful handling of overlaps is essential: while our clustering approach (Section 5.1.1) produces partitions, i.e., non-overlapping clusters, our ideal clusters are overlapping. To evaluate the accuracy of clustering, we use a simple set of accuracy metrics i.e., a standard generalization of precision, recall and F-measure for clusters which is straightforward to interpret in the presence of overlaps. The algorithm of calculating these accuracy metrics for clustering is shown in Algorithm 6[ASBZ16].

RQ6. Accuracy of Requirement Classification. For the multi-class classification problem, this paper uses macro metrics to meausure the performance of classification results. Precision, Recall, and F1 are defined as:

$$P_{macro} = \frac{1}{|G|}\sum_{i=1}^{|G}\frac{TP_i}{TP_i+FP_i} = \frac{\sum_{i=1}^{|G|} P_i}{|G|}$$

$$R_{\text{macro}} = \frac{1}{|G|}\sum_{i=1}^{|G|}\frac{TP_i}{TP_i+FN_i} = \frac{\sum_{i=1}^{|G|} R_i}{|G|}$$

$$F1_{\text{macro}} = \frac{1}{|G|}\sum_{i=1}^{|G|}\frac{2 \times P_i \times R_i}{P_i+R_i} = \frac{\sum_{i=1}^{|G|} F1_i}{|G|}$$

Algorithm 6 Accuracy metrics algorithm for clustering

Input: Let $I_1,\ldots,I_t$ denote the set of *ideal* clusters, and let $G_1,\ldots,G_u$ denote the set of *generated* clusters.
Output: *Precision*,*Recall*,*F-measure*

```
For every pair $(I_i, G_j)\ 1 \le i \le t; 1 \le j \le u$:
Let $n_{ij}$ be the number of common data points between $I_i$ and $G_j$.
Precision$(I_i, G_j) = \frac{n_{ij}}{|G_j|}$.
Recall $(I_i, G_j) = \frac{n_{ij}}{|I_i|}$.
$F$ -measure $(I_i, G_j) = \frac{2\times\text{Precision}(I_i,G_j)\times\text{Recall}(I_i,G_j)}{\text{Precision}(I_i,G_j)+\text{Recall}(I_i,G_j)}$.
For every *ideal* cluster $I_i 1 \le i \le t$:
Let $G_r$ be the best match for $I_i$ among generated
clusters, i.e.,$\mathit{F\text{-}measure}\,(I_i, G_r) \ge \mathit{F\text{-}measure}\,(I_i, G_j)$ for any $1 \le j \le u$. Let $P_{best\_match}(i)$ denote $\mathit{Precision}\,(I_i, G_r)$ and let $R_{best\_match}(i)$ denote $\mathit{Recall}\,(I_i, G_r)$.
Let $n = \sum_{i=1}^{t} |I_i|$.
Compute overall *precision* and *recall* as weighted-averages of the precisions and recalls of the *ideal* clusters:
$\mathit{Precision} = \Sigma_{i=1}^{t}\frac{|I_i|}{n} \times P_{\text{best }\_\text{match}}\ (i)$.
$\mathit{Recall} = \Sigma_{i=1}^{t}\frac{|I_i|}{n} \times R_{\text{best }\_\text{match}}\ (i)$.
```

- Results and Discussion

Table 15. Description of Experimental data-set

Case	Description	Domain	Number of Requirements	Number of standard terms (golden standard)
Case-A	Autonomous Guidance, Navigation and Control (AGNC)	Space	220	237

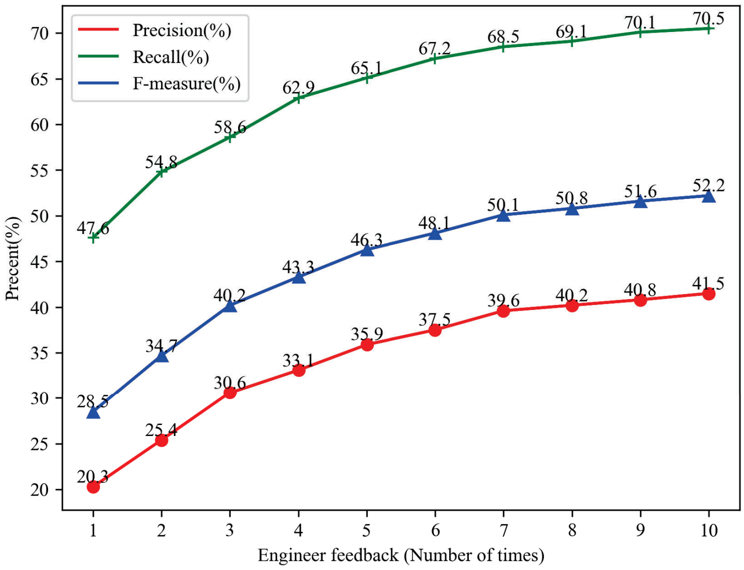


Fig. 20. Evaluation results of candidate term extraction methods combined with engineer feedback

RQ4. Candidate term extraction method. We take Case-A as the example, the experimental data-set is shown in Table 15. We manually identified 237 standard terms (golden standard) for 220 requirements. In order to get better recommendations, we set engineer feedback times to 10. Please note that we also count short terms that are included as parts of longer terms of the other set. The experimental results are shown in Figure 20. The experimental results show that as the number of engineer feedback increases, the accuracy, recall, and F-measure of the candidate term extraction method are all improving. Finally, the precision, recall and F-measure are 41.5%, 70.5%, 52.2% respectively, that is, candidate term extraction method extracts 455 candidate terms and 137 as standard terms.

The preliminary part-of-speech rules and the final part-of-speech rules are shown in Table 16. It can be seen that compared with the preliminary part-of-speech rules, the final part-of-speech rules are more abundant, which can better express the engineer's ideas.

For candidate term extraction, recall is a more important factor than precision. A low recall (i.e., a high

Table 16. Comparison of preliminary and final part-of-speech rules

Length	Preliminary Part-of-speech Rules	Final Part-of-speech Rules
Two	n+n, n+v, v+n	n+n, n+v, v+n, a+n, b+n, d+n
Three	n+n+n, v+n+n, n+v+n, v+v+n	n+n+n, v+n+n, n+v+n, v+v+n, b+v+n, n+m+n, vn+vn+n
Four	n+n+n+n, n+n+v+n	n+n+n+n, n+n+v+n, v+n+n+n, v+n+v+n, n+v+v+n, v+v+n+n, v+n+b+n
Five	v+v+n+n+n, d+v+n+n+n, n+n+v+n+n	v+v+n+n+n, d+v+n+n+n, m+v+m+n+n, b+v+n+v+n, n+n+v+n+n, a+n+v+n+n
Six	n+n+c+vn+n+n	n+n+c+vn+n+n, n+n+vn+c+vn+n, n+n+u+b+vn+n, vn+n+vn+c+vn+n, l+vn+k+n+vn+n, n+vn+u+n+vn+n

Table 17. Top-10 cluster computation alternatives

Grammatical Similarity Strategy	Semantic Similarity Strategy	Clustering Algorithm	Normalized F-measure
Jaro-Winkler	None	Hierarchical Clustering	0.4149
SimHash-Hamming Distance	Hownet	Hierarchical Clustering	0.4132
Jaro	None	Hierarchical Clustering	0.4119
SimHash-Hamming Distance	CiLin	Hierarchical Clustering	0.4105
SimHash-Hamming Distance	None	Hierarchical Clustering	0.4104
Block Distance	None	Hierarchical Clustering	0.4088
Euclidean	None	Hierarchical Clustering	0.4081
Jaro-Winkler	None	K-means	0.4004
Levenshtein	None	Hierarchical Clustering	0.3998
Jaro	None	K-means	0.3989

number of false negatives) means that the engineers will miss many of the terms that should be included in the glossary. Low precision is comparatively easier to address, as it entails only the removal of undesired items (false positives) from a list of extracted candidate terms.

For low accuracy, we consider it is caused by the by-produces of part-of-speech rules and dependency rules. For example, as shown in Figure 7 above, when the candidate term “autonomous management module” was extracted, “management module” was also extracted, which is not the standard term. For the standard terms that have not been extracted, this paper believes that most of the reasons are caused by word segmentation errors. For example: “tube bomb”, which is a part of the rocket control system, should be segmented as “tube bomb/n”, but the wrong segmentation is “tube/n, bomb/v”.

RQ5. Candidate term clustering method. As suggested by the discussion in RQ4, the number of candidate terms is more than the number of standard terms, that is, there are some non-standard terms. We note that outside an evaluation setting, one cannot distinguish terms that are relevant to the glossary from those that are not. To preserve the realistic behavior of clustering, it is thus paramount to compute the generated clusters for all the candidate terms first and then prune the results for evaluation.

To answer RQ5, we considered pairwise combinations of the nine syntactic similarity measures and two semantic similarity measures, which are introduced in Section 2.1. The total number of remaining combinations is $(9 + 1) \times (2 + 1) - 1 = 29$. These combinations include configurations where an individual syntactic or semantic measure is applied on its own. For clustering, we considered K-means, Hierarchical and EM. We evaluated each clustering algorithm in conjunction with all the 29 possible combinations of similarity measures, i.e., a total of $3 \times 29 = 87$ alternative cluster computations.

When the number of candidate terms is n and the selected number of clusters is K, increasing the value of K beyond $n/2$ will would have little practical meaning, as the average size of clusters would fall below 2 terms per cluster. Therefore, this paper restrict the upper range for K to a maximum of $n/2$ (upper bound). Specifically, firstly, for each alternative, this paper selects 10 points uniformly from 1 to $n/2$ as the value of K; Secondly, we compute the normalized F-Measure for each alternative.

In Table 17, it shows the top-10 alternatives that yield the best average accuracy, i.e., normalized F-measure. As shown in the table, the Jaro-Winkler similarity strategy in the best alternative does not produce stable results.

To provide a general recommendation, we need to further consider the effect of individual syntactic measures, semantic measures and clustering algorithms on accuracy across all the alternatives. More precisely, we would like to find similarity measures and clustering algorithms that yield good (but not necessarily the absolute-best) results, and at the same time, cause little variation. A standard way for performing such analysis is by building a regression tree [BFSO84]. Figure 21 shows the regression trees for the case study. The regression tree identifies at any level in the tree the most influential factor that explains the variation between the data points. In our context, there are three factors: (1) the syntactic measure, (2) the semantic measure, and (3) the clustering algorithm. Once the most influential factor is identified, the regression tree partitions the data points into two sets in a way as to minimize variations within the resulting sets.

In each node of the tree, it presents the count (number of alternatives), the mean and the mean-square error(MSE) for Normalized F-measure. By convention, sibling nodes in the tree are ordered from left to right based on their mean values. For every expanded (i.e., non-leaf) node, the node shows the difference between the mean values of its right and left children.

According to the regression tree results, we get the following observations: The first layer shows that for the mean of Normalized F-measure, the hierarchical clustering algorithm is 3.700 higher than the K-means

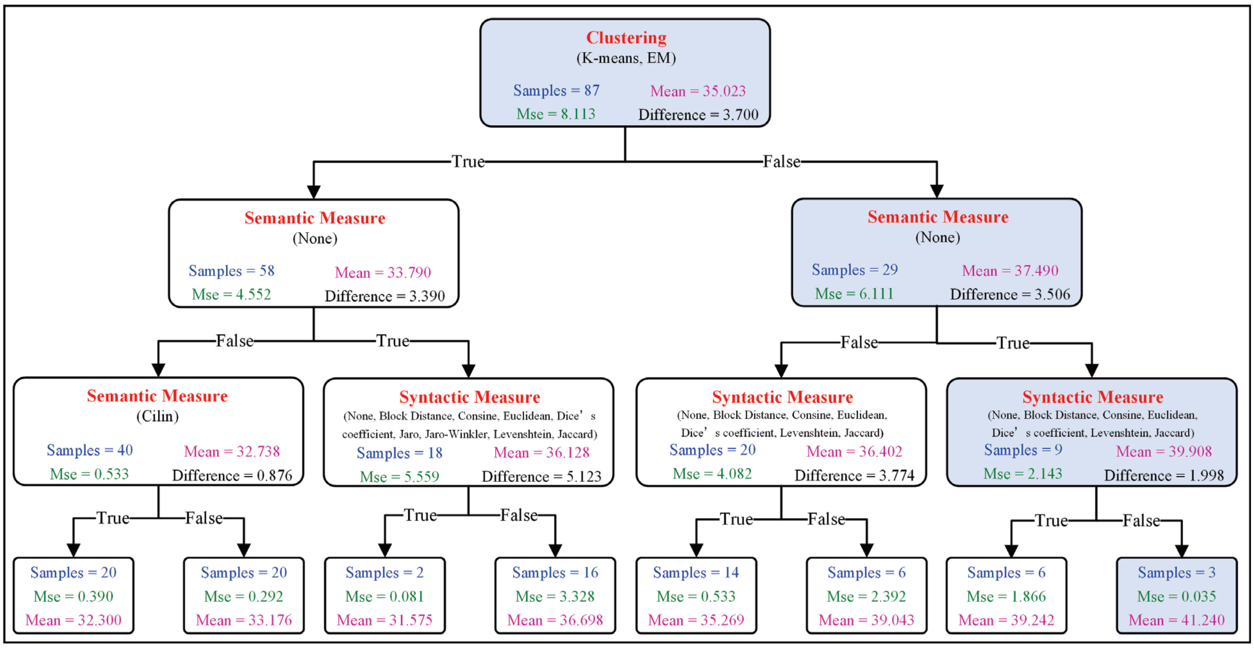


Fig. 21. Regression tree results

Table 18. Good choices of the similarity measures and the clustering algorithm

Strategy/Algorithm	Good Choices
Grammatical Similarity Strategy	Jaro, Jaro-Winkler, SimHash-Hamming
Semantic Similarity Strategy	Semantic measures do not have a significant impact on clustering accuracy.
Clustering Algorithm	Hierarchical Clustering

algorithm and the EM algorithm, thus ruling out the K-means algorithm and the EM algorithm. For the second layer, the experimental results without the semantic similarity strategy are overall higher than using semantic similarity algorithm. The third layer shows that the experimental results of Jaro, Jaro-Winkler and SimHash-Hamming distance are higher than other grammatical similarity algorithms, thus we narrow the choices of syntactic measures to Jaro, Jaro-Winkler and SimHash-Hamming distance. To summarize our discussion of RQ5, we conclude that good choices of the similarity measures and the clustering algorithm, in Table 18.

We note that despite semantic measures not being influential in improving clustering accuracy, we do not discourage the use of these measures. A potential reason why we did not find semantic measures useful is that semantic synonyms are not prevalent in our case studies. Semantic measures, when combined with syntactic ones, may be useful if the likelihood of (semantic) synonyms being present is high.

RQ6. Requirement classification method. We collect 250 requirements for each types of safety-critical autonomous software, including functional, performance, safety, reliability, and interface. a total of 1,250 requirement are collected. The data-sets covers national standard documents in most safety-critical areas, including software design requirements, aerospace software design requirements, and safety-critical software analysis and guidelines, etc.

Different feature extraction methods and classification algorithms are combined for comparative experiments. Specifically, we consider two feature extraction methods and five classification algorithms, a total of ten combination strategies. The experiment is based on the Scikit-learn machine learning library and five-fold cross-validation method. The experimental results of the combination method of the Bow-based feature extraction method and multiple classification models are shown in Table 19. The experimental results of the combination of the feature extraction method based on TF-IDF and the multiple classification models are shown in Table 20. The experimental results show that the combination of TF-IDF feature extraction method and SVM classification algorithm has achieved the highest evaluation result.

Table 19. Bow-based feature extraction method requirement classification experiment results

Combination Strategy		Experimental Results					
		Exp 1	Exp 2	Exp 3	Exp 4	Exp 5	Average
Bow + LR	Precision(%)	94.52	94.44	96.09	93.24	94.85	94.63
	Recall(%)	94.40	94.40	96.00	93.20	94.80	94.56
	F-measure(%)	94.41	94.36	96.02	93.20	94.81	94.56
Bow + NB	Precision(%)	94.27	96.93	97.28	94.75	96.60	95.96
	Recall(%)	94.00	96.80	97.20	94.40	96.40	95.76
	F-measure(%)	93.98	96.80	97.19	94.34	96.39	95.74
Bow + SVM	Precision(%)	94.44	93.66	96.53	93.73	94.50	94.57
	Recall(%)	94.40	93.60	96.40	93.60	94.40	94.48
	F-measure(%)	94.41	93.55	96.41	93.62	94.41	94.48
Bow + KNN	Precision(%)	79.88	77.30	80.09	77.71	78.08	78.61
	Recall(%)	78.00	76.40	75.60	75.60	73.20	75.76
	F-measure(%)	77.63	75.99	74.86	75.44	72.98	75.38
Bow + RF	Precision(%)	87.39	90.62	92.88	88.88	93.06	90.56
	Recall(%)	87.20	90.40	92.80	88.40	92.80	90.32
	F-measure(%)	87.17	90.38	92.80	88.46	92.81	90.32

Table 20. TF-IDF-based feature extraction method requirement classification experiment results

Combination Strategy		Experimental Results					
		Exp 1	Exp 2	Exp 3	Exp 4	Exp 5	Average
TF-IDF + LR	Precision(%)	97.28	96.81	98.42	97.65	96.02	97.24
	Recall(%)	97.20	96.80	98.40	97.60	96.00	97.20
	F-measure(%)	97.21	96.79	98.40	97.60	96.00	97.20
TF-IDF + NB	Precision(%)	95.01	96.12	97.65	95.43	95.86	96.01
	Recall(%)	94.80	96.00	97.60	95.20	95.60	95.84
	F-measure(%)	94.78	95.97	97.60	95.14	95.58	95.81
TF-IDF + SVM	Precision(%)	**96.46**	**96.46**	**99.22**	**97.62**	**98.45**	**97.64**
	Recall(%)	**96.40**	**96.40**	**99.20**	**97.60**	**98.40**	**97.60**
	F-measure(%)	**96.40**	**96.36**	**99.20**	**97.60**	**98.40**	**97.59**
TF-IDF + KNN	Precision(%)	93.67	94.13	94.45	92.60	92.69	93.51
	Recall(%)	93.60	94.00	94.40	92.40	92.40	93.36
	F-measure(%)	93.55	93.96	94.41	92.42	92.37	93.34
TF-IDF + RF	Precision(%)	88.11	90.61	90.46	88.13	93.35	90.13
	Recall(%)	88.00	90.40	90.40	87.60	93.20	89.92
	F-measure(%)	87.98	90.31	90.40	87.63	93.20	89.91

Since experimental data-sets basically covers the requirements of safety-critical autonomous areas, and the division process of the data-sets adopts a random extraction method, the experimental results have a certain degree of versatility. The single experiments listed in Table 19 and Table 20 can also reflect the stability of different combination strategies. In summary, for the requirements of safety-critical autonomous software, the above combination strategies have achieved relatively good results.

8.2.3. Threats to Validity

We discuss threats to validity of our empirical results and the steps we have taken to mitigate these threats.

Internal validity. First, both the standard terms (golden standard) and standard term cluster (golden standard) are subjective processes. In order to reduce this threat, we request two researchers to mark the benchmark data and then review by domain experts to determine the benchmark data. Second, the review of domain experts may contribute to the experimental results of our method. In order to reduce this threat, domain experts are not informed of our method before and after the review process.

External validity. First, the data-sets in this paper covers most of the requirements of AGNC, and terminology recommendation method successfully generates a reasonable list of candidates terms. However, further case studies are necessary to improve external effectiveness. Second, the analysis results of our candidate term extraction method need to be improved. One feasible way is to use existing term extraction tools to process the cases in this paper, and systematically compare the results with our method.

9. Related Work

Section 9.1 presents several surveys on the AI-based safety-critical autonomous systems in the domains of such as space application and autonomous vehicles, and surveys on the challenges of developing of these systems, such as modeling, verification, and so on. This paper mainly focuses on the architecture modeling of safety-critical autonomous systems, based on natural language requirements, thus we present the related work on the requirement modeling of such systems in section 9.2. In section 9.3, it describes the existing work on AuI for natural language requirement modeling.

9.1. AI-based Safety-Critical Autonomous Systems and its Challenges

[JGMG19] shows a survey of the techniques of AI for space applications, in which it mainly focuses on the application of artificial intelligence in autonomous planning and scheduling of operations, self-awareness, anomaly detection and Fault Detection Isolation and Recovery (FDIR), on-board data analysis as well as on-board operations and processing of earth-observation data.

Dario Izzo et al. [DIP19] give a survey on AI trends in spacecraft guidance dynamics and control systems. They mainly focus on evolutionary optimisation, tree searches and machine learning, including deep learning and reinforcement learning, as the key technologies and drivers for current and future research in the field of spacecraft guidance and control.

Autonomous Vehicles (AV) are expected to bring considerable benefits to society, such as traffic optimization and accidents reduction. They rely heavily on advances in many AI approaches and techniques. [NVM$^+$19] presents a systematic literature review about the impact of AI on autonomous vehicle safety.

Joseph Sifakis et al.[DHS20] propose that developing trustworthy autonomous systems requires addressing fundamental issues that have not been dealt with adequately by present research or industrial experience. Three main challenges are presented: 1) how to specify autonomous system behavior in the face of unpredictability, and behavioral specification is needed in virtually all stages and activities of the development process: requirements, design, simulation, testing, verification, and validation; 2) how to carry out faithful analysis of system behavior with respect to rich environments that include humans, physical artifacts, and other systems, it includes simulation, testing, formal verification, and system validation against the tacit needs of the stakeholders; and 3) how to build such systems by combining executable modeling techniques from software engineering with artificial intelligence and machine learning, for instance, combining "model-based" and "data-driven" approaches.

There is a growing body of research on the verification and testing of AI-based safety-critical autonomous systems. For instance, [AAHR16] presents the formal verification of safety-critical autonomous systems in dynamic environments. The authors adopt a two phase process which combines static verification methods (with UPPAAL), used at design time, with dynamic ones, used at run time (using monitors). Lionel Briand et al. [HSNB20] compare offline and online testing of deep neural networks used in an autonomous car. Offline testing where DNNs are tested as individual units based on test data-sets obtained independently from the DNNs under test, and online testing where DNNs are embedded into a specific application and tested in a close-loop mode in interaction with the application environment.

Compared with the existing work, this paper mainly focuses on the architecture modeling of safety-critical autonomous systems with SysML, based on the natural language requirements.

9.2. Requirements Engineering for AI-based Systems

There is not much work on Requirements Engineering (RE) for AI-based systems.

[IY19] shows how practitioners currently perceive the difficulties and their causes in the engineering of ML-based systems. The authors report that RE was listed as the most difficult activity for the development of machine learning systems.

[Hor19] states that much recent attention has been paid to certain qualities of machine learning solutions, particularly the non-functional requirements such as fairness, transparency, privacy, security, and testability.

Andreas Vogelsang et al. [VB19] describe the characteristics and challenges unique to RE for ML-based systems. They conclude that development of ML systems demands requirements engineers to: (1) understand ML performance measures to state good functional requirements, (2) be aware of new quality requirements such as explainability, freedom from discrimination, or specific legal requirements, and (3) integrate ML specifics in the RE process.

[LFCL16] explores the steps that healthcare organizations may take to elicit data analytics requirements, to specify functional requirements for using data to improve business and clinical outcomes, to improve service quality while reducing costs.

However, the existing works have not considered ML-specific requirements when ML is deployed in a safety-critical context. Moreover, we envision that systems in safety-critical application are unlikely to be pure ML systems. Therefore, it needs to consider the requirements of both conventional components and ML components.

9.3. AuI for Natural Language Requirement Modeling

9.3.1. AI for Requirement Engineering

There are several work on how to use AI/ML to improve RE tasks such as requirement prioritization, categorization, domain models extraction, etc.

[PSA13] presents a machine learning approach to software requirements prioritization. Deciding which, among a set of requirements, are to be considered first and in which order is a strategic process in software development. This task is commonly referred to as requirements prioritization. The authors describe a requirements prioritization method called Case-Based Ranking (CBRank), which combines project's stakeholders preferences with requirements ordering approximations computed through machine learning techniques, bringing promising advantages.

[WV16] proposes an approach for automatic classification of requirements based on convolutional neural networks. Besides the actual requirements, these documents contain additional content such as explanations, summaries, and figures. It is important to be able to differentiate between legally relevant requirements and other auxiliary content. The authors present an approach to automatically classify content elements of a natural language requirements specification as "requirement" or "information (auxiliary content)". They also do evaluation on a real-world automotive requirements specification.

Lionel Briand et al [ASNB19] proposed an active learning approach for improving the accuracy of automated domain model extraction. The objective of this article is to develop automated assistance for filtering the superfluous information extracted by NLP during domain model extraction. The authors devise an active-learning-based approach that iteratively learns from analysts' feedback over the relevance and superfluousness of the extracted domain model elements and uses the feedback to provide recommendations for filtering superfluous elements.

9.3.2. Restricted Natural Language Requirement Modeling

Restricted natural language requirement templates serve as a simple and yet effective approach for increasing the quality of requirements by avoiding complex structures, ambiguity, and inconsistency in requirements [ASBZ15].

Tao Yue et al. proposed Restricted Use Case Modeling (RUCM)[YBL15][YBL13] to reduce ambiguity and facilitate the automated analysis of use case models. RUCM is composed of a set of well-defined restriction rules and a modified use case template. The goal is two-fold: (1) restrict the way users can document use case models in order to reduce ambiguity and (2) facilitate the manual derivation of initial analysis models which, when using the Unified Modeling Language (UML), are typically composed of class diagrams, sequence diagrams, and possibly other types of diagrams.

EARS (Easy Approach to Requirement Syntax) [MWHN09] specifies requirements with five sentence templates to improve the quality of requirements. A small set of structural rules was developed to address eight common requirement problems including ambiguity, complexity and vagueness. The ruleset allows all natural language requirements to be expressed in one of five simple templates. Moreover, the ruleset was applied whilst extracting aerospace engine control system requirements from an airworthiness regulation document.

[WLZ+10] presents a formal modeling approach SPARDL(Spacecraft Requirement Description Language) as a concise and precise way to specify embedded systems, which can improve the quality of embedded systems in aerospace.

Compared with existing work, this paper provides a first contribution towards a restricted natural language requirement modeling for SysML, augmented with AI, that is AuI for restricted natural language requirement modeling approach.

10. Conclusion and Future Work

This paper made a first step towards exploring the architecture modeling of AI-based safety-critical autonomous systems, based on natural language requirements. Firstly, Augmented Intelligence for restricted natural language requirement modeling is proposed. In that, several AI technologies such as natural language processing and clustering are used to recommend candidate terms to the glossary. The glossary (including data dictionary and domain glossary) is used in the restricted natural language requirement template RNLReq, which is equipped with a set of restriction rules and templates to structure and restrict the way how users document requirements. Secondly, in order to support the description of the specific requirement types of safety-critical autonomous systems, an extension of SysML is given, and then automatic generation SysML architecture models from requirements specified with RNLReq is presented. Thirdly, the prototype tools for both RNLReq and RNLReq2SysML are implemented based on Papyrus. Finally, an industrial Autonomous Guidance, Navigation and Control (AGNC) as case study for the proposed approach has been carried out and it provides some positive feedback and lessons.

In the future, we would like to consider more new AI technologies such DNN in our AuI method. In addition, as mentioned before, AI is already beginning to impact various system engineering aspects. These include safety analysis, requirement engineering, systems modeling, verification, process management, search and information retrieval, dynamic context management, and human-system integration. Therefore, we would like to explore AuI in other phases of system engineering, such as RNL template recommendation, safety analysis, formal verification, and so on. As safety-critical autonomous systems are often resource-constrained, we are currently working on the compositional verification on the SysML models to evaluate the trade-off between objectives such as performance, resource consumption, failure tolerance rate so as to determine the optimal solutions.

Acknowledgement Supported by the National Natural Science Foundation of China (62072233), Aviation Science Fund of China (201919052002), and the Fundamental Research Funds for the Central Universities (NP2017205).

References

[AAHR16] Adina Aniculaesei, Daniel Arnsberger, Falk Howar, and Andreas Rausch. Towards the verification of safety-critical autonomous systems in dynamic environments. In Mehdi Kargahi and Ashutosh Trivedi, editors, *Proceedings of the The First Workshop on Verification and Validation of Cyber-Physical Systems, V2CPS@IFM 2016, Reykjavík, Iceland, June 4-5, 2016*, volume 232 of *EPTCS*, pages 79–90, 2016.

[ASBZ15] Chetan Arora, Mehrdad Sabetzadeh, Lionel C. Briand, and Frank Zimmer. Automated checking of conformance to requirements templates using natural language processing. *IEEE Trans. Software Eng.*, 41(10):944–968, 2015.

[ASBZ16] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Automated extraction and clustering of requirements glossary terms. *IEEE Transactions on Software Engineering*, 43(10):918–945, 2016.

[ASNB19] Chetan Arora, Mehrdad Sabetzadeh, Shiva Nejati, and Lionel C. Briand. An active learning approach for improving the accuracy of automated domain model extraction. *ACM Trans. Softw. Eng. Methodol.*, 28(1):4:1–4:34, 2019.

[BFSO84] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees.* CRC press, 1984.

[CRF+03] William W Cohen, Pradeep Ravikumar, Stephen E Fienberg, et al. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, volume 2003, pages 73–78, 2003.

[DHS20] Assaf Marron David Harel and Joseph Sifakis. Autonomics: In search of a foundation for next-generation autonomous systems. *PNAS*, 117(30):17491–17498, July, 2020.

[DIP19] Marcus Martens Dario Izzo and Binfeng Pan. A survey on artificial intelligence trends in spacecraft guidance dynamics and control. *Astrodyn*, 3:287–299, 2019.

[ECS08] Space Segment Operability (ECSS-E-ST-70-11C). Research report, European Cooperation for Space Standardization (ECSS), 2008.

[ESA20] EASA Artificial Intelligence Roadmap 1.0. A human-centric approach to AI in aviation. Research report, EASA, February 2020.

[FF17] Thomas K Ferrell and Uma D Ferrell. RTCA DO-178C/EUROCAE ED-12C. *Digital Avionics Handbook*, 2017.

[FG13] Peter H. Feiler and David P. Gluch. Model-based engineering with AADL: An introduction to the SAE architecture analysis & design language. *Pearson Schweiz Ag*, 2013.

[FGH$^+$20] Douglas Fraser, Ruben Giaquinta, Ruth Hoffmann, Murray Ireland, Alice Miller, and Gethin Norman. Collaborative models for autonomous systems controller synthesis. *Formal Aspects Comput.*, 32(2):157–186, 2020.

[GF$^+$13] Wael H Gomaa, Aly A Fahmy, et al. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18, 2013.

[He14] Han He. Hanlp: Han language processing. *URL: https://github. com/hankcs/HanLP*, 2014.

[HG03] H.M. Harmain and R. Gaizauskas. Cm-builder: A natural language-based case tool for object-oriented analysis. *Autom. Softw. Eng.*, 10(2):157–181, 2003.

[HH12] Wahiba Ben Abdessalem Hatem Herchi. From user requirements to uml class diagram. *arXiv preprint*, 2012.

[Hor19] Jennifer Horkoff. Non-functional requirements for machine learning: Challenges and new directions. In *27th IEEE International Requirements Engineering Conference, RE 2019, Jeju Island, Korea (South), September 23-27, 2019*, pages 386–391. IEEE, 2019.

[HSNB20] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel C. Briand. Comparing offline and online testing of deep neural networks: An autonomous car case study. In *13th IEEE International Conference on Software Testing, Validation and Verification, ICST 2020, Porto, Portugal, October 24-28, 2020*, pages 85–95. IEEE, 2020.

[ISBM09] Ali Samad Imran Sarwar Bajwa and Shahzad Mumtaz. Object oriented software modeling using nlp based knowledge extraction. *European Journal of Scientific Research*, 2009.

[IY19] Fuyuki Ishikawa and Nobukazu Yoshioka. How do engineers perceive difficulties in engineering of machine-learning systems?: questionnaire survey. In *Proceedings of the Joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice, CESSER-IP@ICSE 2019, Montreal, QC, Canada, May 27, 2019*, pages 2–9. IEEE / ACM, 2019.

[JGMG19] Frank Dannemann Jan-Gerd Meß and Fabian Greif. Techniques of artificial intelligence for space applications - a survey. In *European Workshop on On-Board Data Processing (OBDP2019)*, 2019.

[JW10] Tian Jiule and Zhao Wei. Method for calculating similarity of words based on cilin. *Journal of Jilin University (Information Science*, (6):602–608, 2010.

[KHI$^+$19] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. The marabou framework for verification and analysis of deep neural networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 443–452. Springer, 2019.

[KLL20] Vivek Kothari, Edgar Liberis, and Nicholas D. Lane. The final frontier: Deep learning in space. In Padmanabhan Pillai and Qin Lv, editors, *HotMobile '20: The 21st International Workshop on Mobile Computing Systems and Applications, Austin, TX, USA, March 3-4, 2020*, pages 45–49. ACM, 2020.

[KMP16] Ryan F. Kirwan, Alice Miller, and Bernd Porr. Model checking learning agent systems using promela with embedded C code and abstraction. *Formal Aspects Comput.*, 28(6):1027–1056, 2016.

[LAL$^+$19] Changliu Liu, Tomer Arnon, Christopher Lazarus, Clark W. Barrett, and Mykel J. Kochenderfer. Algorithms for verifying deep neural networks. *CoRR*, abs/1903.06758, 2019.

[LFCL16] Lin Liu, Letong Feng, Zhanqiang Cao, and Jingdong Li. Requirements engineering for health data analytics: Challenges and possible directions. In *24th IEEE International Requirements Engineering Conference, RE 2016, Beijing, China, September 12-16, 2016*, pages 266–275. IEEE Computer Society, 2016.

[Liu02] Qun Liu. Word similarity computing based on hownet. *Computational linguistics and Chinese language processing*, 7(2):59–76, 2002.

[Mad20] Azad M. Madni. Exploiting augmented intelligence in systems engineering and engineered systems. *Insight*, pages 31–36, 2020.

[MRS10] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.

[MWHN09] Alistair Mavin, Philip Wilkinson, Adrian Harwood, and Mark Novak. Easy approach to requirements syntax (ears). In *Requirements Engineering Conference*, pages 277–282, 2009.

[NVM$^+$19] Alexandre Moreira Nascimento, Lucio Flavio Vismari, Caroline Bianca Santos Tancredi Molina, Paulo Sérgio Cugnasca, João Batista Camargo Jr., Jorge Rady de Almeida Jr., Rafia Inam, Elena Fersman, Maria V. Marquezini, and Alberto Y. Hata. A systematic literature review about the impact of artificial intelligence on autonomous vehicle safety. *CoRR*, abs/1904.02697, 2019.

[PSA13] Anna Perini, Angelo Susi, and Paolo Avesani. A machine learning approach to software requirements prioritization. *IEEE Trans. Software Eng.*, 39(4):445–461, 2013.

[Sif19] Joseph Sifakis. Autonomous systems - an architectural characterization. In Michele Boreale, Flavio Corradini, Michele Loreti, and Rosario Pugliese, editors, *Models, Languages, and Tools for Concurrent and Distributed Programming - Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday*, volume 11665 of *Lecture Notes in Computer Science*, pages 388–410. Springer, 2019.

[SL11] Sadhan Sood and Dmitri Loguinov. Probabilistic near-duplicate detection using simhash. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1117–1126, 2011.

[SRBM20] Dick Stottler, Sowmya Ramachandran, Christian Belardi, and Rocky Mandayam. On-board, autonomous, hybrid spacecraft subsystem fault and anomaly detection, diagnosis, and recovery. In *Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS)*, 2020.

[TG18] Massimo Tipaldi and Luigi Glielmo. A survey on model-based mission planning and execution for autonomous spacecraft. *IEEE Systems Journal*, 12(4):3893–3905, 2018.

[VB19] Andreas Vogelsang and Markus Borg. Requirements engineering for machine learning: Perspectives from data scientists. In *27th IEEE International Requirements Engineering Conference Workshops, RE 2019 Workshops, Jeju Island, Korea (South), September 23-27, 2019*, pages 245–251. IEEE, 2019.

[WAD16] Aarti Singh Wahiba Abdessalem, Zeineb Ben Azzouz and Nilanjan Dey. Automatic builder of class diagram (abcd): an application of uml generation from functional requirements. *Software Practice and Experience*, 2016.

[WLZ+10] Zheng Wang, Jianwen Li, Yongxin Zhao, Yanxia Qi, Geguang Pu, Jifeng He, and Bin Gu. Spardl: A requirement modeling language for periodic control system. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, pages 594–608, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[WV16] Jonas Winkler and Andreas Vogelsang. Automatic classification of requirements based on convolutional neural networks. In *24th IEEE International Requirements Engineering Conference, RE 2016, Beijing, China, September 12-16, 2016*, pages 39–45. IEEE Computer Society, 2016.

[WYH+20] Fei Wang, Zhibin Yang, Zhi-qiu Huang, Chengwei Liu, Yong Zhou, Jean-Paul Bodeveix, and Mamoun Filali. An approach to generate the traceability between restricted natural language requirements and AADL models. *IEEE Trans. Reliab.*, 69(1):154–173, 2020.

[YBL13] Tao Yue, Lionel C. Briand, and Yvan Labiche. Facilitating the transition from use case models to analysis models: Approach and experiments. *ACM Trans. Softw. Eng. Methodol.*, 22(1):5:1–5:38, 2013.

[YBL15] Tao Yue, Lionel C. Briand, and Yvan Labiche. atoucan: An automated framework to derive UML analysis models from use case models. *ACM Trans. Softw. Eng. Methodol.*, 24(3):13:1–13:52, 2015.

A. PLAIN-TEXT INSTRUCTIONS FOR AUTO-GENERATED SYSML MODELS

The following is the plain-text instructions which have the same process and effect as the automated tool implementation. A system engineer can read these low-level, detailed instructions and build the model manually.

Algorithm 7 Plain-Text Instructions for Auto-generated SysML Model

```
Input: SystemTemplate A0, SubSystemTemplate Ai in A, i=1,2,3...
Output: BDD

 1: Add a Block BA named A0.
 2: Add a Block Bi named Ai.
 3: Add a part property PPi named Ai to the Block BA.
 4: Apply the Bi classifier to the part property PPi.
Input: Nonfunctional Requirement sentences in Ai in Table 14
 5: Query DataWord DW in DataDictionary with <property> in Ai's sentences, i=0,1,2,3..
 6: Add a value type VT named DW.
 7: Add a part property named DW's struct.
 8: Add a value type VT2 for DW's struct.
 9: Apply the unit named DW's unit to the value type VT.
10: Repeat Step 5 for all structs in DW.
11: Add a value property VPi named <property> to Block Bi
12: Apply a value type VT to VPi.
13: If Ai contains NFR requirement, generate OCL expressions, add the ConstraintRequirement CBi.
14: Link the ConstraintRequirement CBi to Block Bi.
Input: SystemTemplate/SubSystemTemplate Ai in A, i=0,1,2,3...
Output: IBDs
15: Traverse Ai and find input/output with corresponding output/input, and add the pair into the portPairList.
16: For each Ai, add an IBD ibd and link ibd to Bi
17: For each pair PR in portPairList, find the part property ppA and ppB in block Bi.
18: Add a flow port portA on ppA with input data info, add a flow port portB on ppB with output data info.
19: Add a connector started with portA and ended with portB
Input: FunctionRequirement FRi in Ai in A, i=0,1,2,3..
Output: ACTs
20: Add a activity ACi named Ai.
21: Get sentenceList STList in FunctionRequirement FRi.
22: Add a InitialNode inode and FinalNode fnode into ACi,
23: Record inode and fnode as the entry point enNode and exit point exNode
24: Traverse STList in DFS, for each sentence ST
25: If ST is simple, add Node tNode with corresponding type into ACi, put tNode between enNode and exNode
26: If ST is Call, tNode is typed by CallBehaviorAction. IF ST call shared function SF, apply the classifier of SF's activity to tNode.
27: If ST is Send, tNode is typed by SendSignalAction
28: If ST is Receive, tNode is typed by AcceptEventAction
29: If ST has deadline,wcet, add the ConstraintRequirement to tNode.
30: If ST has rate, apply Rate stereotype to edge of tNode.
31: If ST has probability constraint, apply Probability stereotype to edge of tNode.
32: If ST is complex, record the new entry point enNode and exit point exNode before processing ST
33: If ST isMadeUpOf EventCondition+SentenceB, add AcceptEventAction aeaNode after enNode into ACi, assign aeaNode to enNode
34: If ST isMadeUpOf ValueCondition+SentenceB, add DecisionNode dNode after enNode into ACi, and MergeNode mNode before exNode into ACi, assign dNode to enNode, assign mNode to exNode
35: If ST isMadeUpOf ConcurrentSentence+SentenceB, add ForkNode fkNode after enNode into ACi, and jnNode mNode before exNode into ACi, assign fkNode to enNode, assign jnNode to exNode
36: assign SentenceB to ST, repeat Step 10-12d
37: Make ACi be owner of all nodes in 10-14
```