Journal of Intelligent System and Applied Data Science (JISADS)

Journal homepage : https://www.jisads.com
ISSN (2974-9840) Online

EMBEDDED SYSTEMS REQUIREMENTS: A SYSTEMATIC REVIEW OF DESCRIPTION LANGUAGES AND INTELLIGENT SOFTWARE SYNTHESIS

Maisam Khalil[1], *Dr.Eng.Ahmad Kurdi*[2 *]
[1] *Computer Engineering Department, Al-Wataniyah Private University, Syria.*
[2*] *Applied Faculty - Computer Technology Department, University of Hama, Syria*
maysamkhalil994@gmail.com, Dr.eng.ahmad_kurdi@hama-univ.edu.sy

ABSTRACT

With the widespread application of embedded systems, their requirements are becoming increasingly complex, and requirements analysis has become a key stage in the development of embedded systems. How to accurately model and describe requirements has become the primary issue. This paper systematically investigates the requirements description of embedded systems and conducts a comprehensive comparative analysis in order to gain a deeper understanding of the core concerns of embedded system requirements. First, a systematic literature review method is adopted to identify, screen, summarize, and analyze the relevant literature published between January 1979 and November 2023. Through automatic retrieval and snowballing, 150 papers closely related to the topic were selected to ensure the comprehensiveness of the literature review. Secondly, from the aspects of requirements description focus, requirements description dimension, and requirements analysis elements, the expressive ability of existing embedded requirements description languages is analyzed. Finally, the challenges faced by the current software requirements description of embedded systems are summarized, and the requirements for the expressive ability of the requirements description method for embedded systems are put forward for the task of intelligent synthesis of embedded software.

Keywords: embedded system, requirements description language, requirements analysis, Classification.

1. INTRODUCTION

Embedded systems have penetrated all aspects of human life and are widely used in various fields such as automotive electronics, aerospace, rail transit, medical equipment, and personal mobile devices [1]. The accompanying increase in demand complexity is a result of this. Each embedded system integrates a large number of functional requirements and non-functional requirements such as real-time, safety, and reliability. For example, a high-end car has more than 2,000 software-based functions [2]; and an aviation embedded system has as many as 139 functions [3]. In addition, the complexity of embedded system requirements is also reflected in the sharp increase in the number of connected devices. According to Mumtaz et al. in the literature [4], the number of wireless access devices in the fields of future transportation, smart cities, and factories will reach the level of tens of billions.

For the development of complex software systems, the requirements stage is the most error-prone and costly stage, especially for embedded software systems [5, 6]. Broy et al. found that more than 50% of the errors in the embedded field occurred at the time of system delivery and were related to the wrong understanding of the requirements [7]. Naumchev et al. pointed out that the requirements problems that lead to software disasters can be divided into two categories: one is the wrong understanding, updating, and implementation of the requirements, and the other is that the requirements are incomplete, inconsistent, and do not meet the user's needs [8]. Accurately describing the requirements has become a key task in the development of embedded systems [9, 10].

Compared with traditional information system software, embedded system requirements have their particularity. First, the embedded system is composed of software and hardware. Its software interacts directly with the physical world through various sensing devices and acting devices, and runs on a hardware platform with limited resources according to the system control logic. Therefore, its requirements not only include software and hardware behavior, but also cover constraints on performance, accuracy, reliability, etc., some of which are mandatory requirements. This requires us to accurately capture and express these constraints in the requirements description. Secondly, the development of embedded systems begins with the task intention. The requirements analysis gradually refines the task intention from system-level requirements to software and hardware-related requirements. The requirements analysis needs to clearly decouple the intertwined requirements into relatively simple operations. The relationship between different levels of requirements has become an important part of requirements analysis. How to express the requirements of various levels has become the primary issue.

Since the 1970s, researchers have begun to pay attention to the problem of embedded system requirements description and have proposed a series of requirements description languages, designing their language expression capabilities from different perspectives. In the 1980s, Davis et al. [6] and Melhart et al. [7] respectively reviewed the mainstream embedded requirements description languages at that time and explained the background, characteristics, and applications of these languages. However, their writing age is relatively long, the languages covered are few, and the computer and embedded technology at that time were far from mature and developed as they are today, and the connected devices were also few.

With the increasing complexity of embedded systems, their requirements are not only intricate, but also scattered in different stages of software and hardware development. Therefore, many new embedded system requirements description methods have appeared. Through a comprehensive investigation, comparison, and analysis of these existing works, this paper hopes to deeply understand the core concerns of embedded system requirements, as well as the capabilities of existing embedded requirements description languages, clarify the intertwining of embedded system requirements caused by device sharing, refine the key components of requirements description, and lay a foundation for proposing a requirements description and requirements analysis that has a certain degree of universality and can capture decentralized requirements.

This paper adopts the systematic literature review method to search and screen the relevant literature published from January 1979 to November 2023 and finally selected 150 papers from 3,442 documents as the research object. Due to the differences in research objects and terminology in various documents, and considering that the requirements involved in embedded systems are closely related to their system structure, in order to uniformly express the meaning of various types and levels of requirements and facilitate comparative analysis, before starting the investigation, we first gave a reference structure of a general embedded system as the basis for the investigation and analysis.

Next, from the aspects of requirements modeling focus, requirements description dimension, and requirements analysis elements, the expressive ability of existing embedded requirements description languages is analyzed. Finally, the challenges faced by the current software requirements description of embedded systems are discussed, and for the research hotspot the intelligent synthesis task of embedded systems the requirements for the expressive ability of its embedded system requirements description language are proposed

2. EMBEDDED SYSTEM REFERENCE STRUCTURE

In order to review the requirements description of embedded systems from a unified perspective, we give the reference structure of the embedded system, as shown in Figure 1. Among them, the embedded system is regarded as a computing software and hardware complex with specific functions. The software serves as a controller that coordinates system equipment to complete the design intent of the embedded system. The hardware equipment mainly includes various sensors, actuators, and other equipment that the system can manage and schedule. The controller communicates with the external environment through the system equipment. These external environments include super administrators, external software systems, natural environments, physical environments, etc.

Complex embedded systems usually have multiple software controllers. We distinguish between a system controller and a set of subsystem controllers. Among

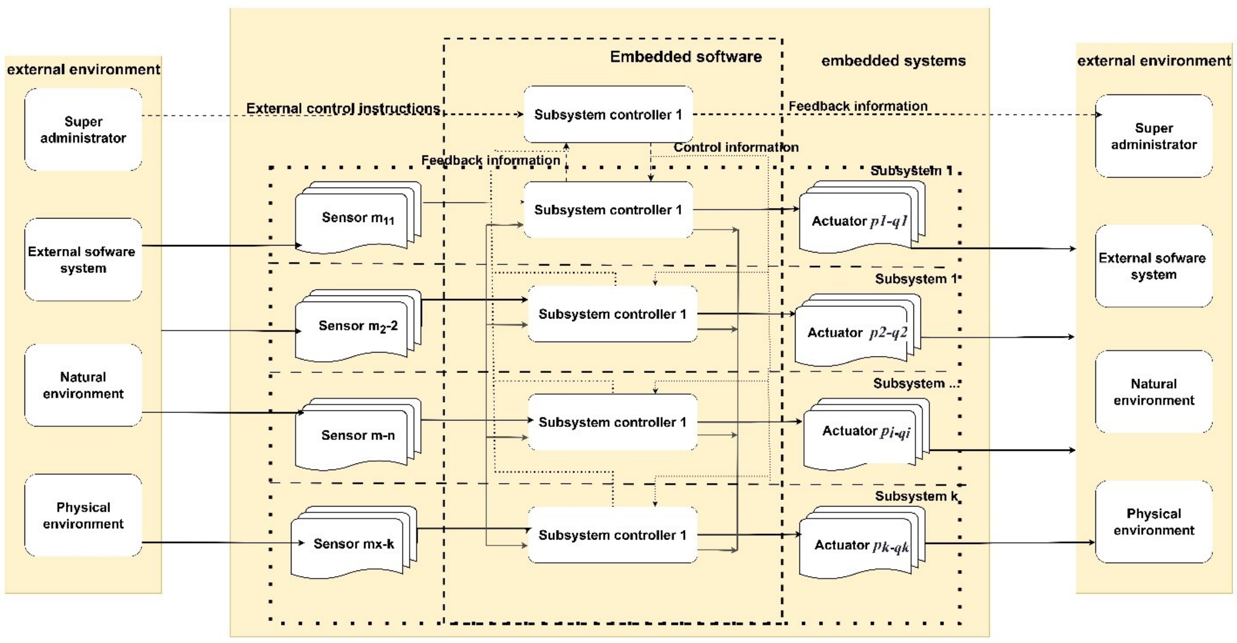

Figure1. Embedded system architecture

them, the system controller can receive external control instructions from the super administrator, monitor the status of each subsystem controller, and control and coordinate the subsystem controllers. The subsystem controller is responsible for coordinating the corresponding sensors and controllers, obtaining external environment information, and implementing control operations. The sensor can obtain the attributes of the external environment. In addition to being regarded as an external device, it is also regarded as a system interface. The actuator can apply the software controller instructions to the external environment.

For the expectations and requirements of different components in the above embedded system reference structure, different types of requirements descriptions will be obtained. These types include system task intent, system capability requirements, software capability requirements, hardware requirements, and software design constraints.

Among them, the system task intent expresses the goal of the system. Since the embedded system will act on the external environment, following the requirements engineering concept based on environmental modeling [8], the system task intent is usually reflected in the response to the expected changes in the external environment. For example, in the aerospace field, the task intent of the sun search control system is to perceive the position of the sun in the natural environment and the current satellite's body angle in the physical environment and adjust its own body angle to achieve sun-tracking cruise. In this example, both the sun and the star constitute the external environment.

System capability requirements describe the capabilities that the system must have. These capabilities are the specific conditions or requirements that must be met to achieve the system's task intent. Specifically, since the embedded system will interact with the external environment through the system equipment, its capabilities are based on the input and output relationship between the embedded system and the external environment, that is, the input and output of the system equipment and the external environment. For example, in the sun search control system, the system measures the satellite's speed and angle through sensors such as gyroscopes and sun sensors and perceives sun visibility information.

Software capability requirements refer to the capabilities that the software needs to have to schedule, manage, and control system equipment in order to meet the system capability requirements. It is often expressed in the input, output, and their relationship between the embedded software controller and the system equipment, also known as embedded software requirements, including various functional and non-functional requirements. For example, the sun search control software needs to send power control instructions and data communication instructions to devices such as gyroscopes and sun sensors and receive and process measurement data. These are all functional requirements; the period interval for the sun search control software to write control commands to the thruster must be less than 120 ms, etc., which belongs to non-functional requirements.

Hardware requirements are constraints on embedded hardware devices, which can be divided into system equipment requirements and software operating platform requirements. System equipment requirements mainly involve various sensors and actuators, such as the sensor gyroscope can measure the angular velocity of the satellite, each data acquisition of the sensor must be completed within 90 ms, the restart time of the actuator cannot exceed 500 ms, etc.; the software operating platform requirements refer to the requirements for the hardware platform that supports the operation of the embedded system software, such as the memory capacity is not less than 128 MB, the storage space is not less than 1 GB, and the processor must process 5 million instructions per second.

The above five different types of requirements descriptions are actually the products of different requirements development stages. The usual requirements development process is as follows: first, through system goal analysis, study the expected changes in the external environment, and obtain the system's task intent. Then, through system capability analysis, the task intent is specified as the input and output relationship between the system and the external environment, and the system capability requirements are derived. Next, software and hardware partitioning is performed. Through the analysis of the interaction between the software and external devices and the requirements for external devices and platforms, the software capability requirements and hardware capability requirements are obtained. Finally, the software design constraints can be obtained for the preliminary design stage of the software. The software requirements specification of the embedded system is a process of continuous deduction from the expectation of the external environment to the behavior of the software controller, which involves the refinement, decomposition, deduction, and evolution of different types of requirements.

3. LITERATURE REVIEW METHOD

3.1 Research Questions

The purpose of this paper is to deeply understand the core concerns of embedded system requirements and the capabilities of requirements description languages. For this purpose, two research questions were designed:

- RQ1: What are the types of requirements involved in the common description of embedded systems?

- RQ2: What are the capabilities of existing embedded system requirements description languages?

3.2 Literature Collection

This section describes the data collection process in the systematic literature review, including literature retrieval, literature screening, and snowballing. The three authors of this paper finally selected 150 papers from 3,442 documents as the research collection after automatic and manual retrieval and snowballing.

(1) Literature Retrieval

We first considered keywords related to requirements engineering, embedded systems, embedded software, and requirements description. In the search, we found that process control systems and real-time systems are also many embedded systems, so we added related keywords. The final Chinese keywords for literature retrieval were "(embedded system software + embedded system + embedded software + process control system + real-time control system) * (requirements description + requirements specification + requirements specification + requirements modeling + system description + system specification + system specification + system modeling + requirements standard + requirements format + requirements document)", and the English keywords were "("embedded system software" OR "embedded system" OR "embedded software" OR "process control system" OR "real time control system") AND ("requirements description" OR "requirements specification" OR "requirements modeling" OR "system description" OR "system specification" OR "system modeling" OR "requirements standard" OR "requirements format" OR "requirements documentation")".

The literature databases used for retrieval include Web of Science, IEEE Xplore, ACM Digital Library, Science Direct, Springer, Engineering Village, and CNKI. A total of 3,442 documents were obtained by searching in the above literature databases. Among them, Springer had the most documents, with 2,443, and CNKI had the fewest, with 13. The specific search results are shown in Figure 2. Due to the large data gap, we used a non-proportional vertical coordinate for display.

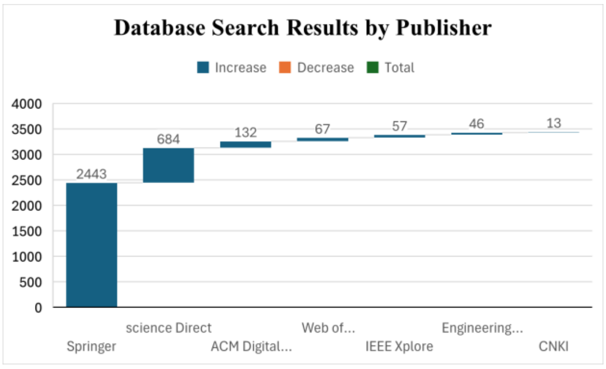

Figure 2. Database Search Results by Publisher

(2) Literature Screening

We screened the retrieved literature. The inclusion and exclusion criteria for the literature are as follows. Note that literature that meets any of the inclusion criteria is included, and literature that meets any of the exclusion criteria is excluded. The literature inclusion criteria (IC) are as follows:

- IC1: The literature involves the requirements description related to embedded systems. At this stage, our goal is to maximize the scope of the literature to ensure the integrity of the research.

- IC2: The literature successfully applies the general requirements description and specification language, method, and process to real cases of embedded systems.

The literature exclusion criteria (EC) are as follows:

- EC1: Literature for which the full text of the electronic version cannot be obtained.

- EC2: Literature written in languages other than English and Chinese.

- EC3: Literature that has not been peer-reviewed.

- EC4: There is more complete literature, that is, there are multiple documents for the same research, and only the most complete literature is included.

The specific screening process is as follows. First, the title, keywords, and abstract of the literature are screened to determine which literature meets the inclusion/exclusion criteria, and then the full text is read for screening. When assigning tasks, we ensure that each selection stage of each document is completed by at least two authors. Controversial documents must first be evaluated by three authors, and then a consensus is reached through discussion. After screening according to the above criteria, 106 documents were obtained. Then, the snowball method was used to screen and summarize the references of these documents, and finally 150 documents were selected.

4. TYPES OF EMBEDDED SYSTEM REQUIREMENTS DESCRIPTION

Table 1 summarizes the types of embedded system requirements involved in the searched literature, including system task intent, system capability requirements layer, software capability requirements, hardware requirements, and software design constraints, which involve various different levels. Among them, the description level of system task intent and system capability requirements is relatively high, but there is less existing work. Most of the work focuses on the description of software capability requirements, which is an indispensable basic component of embedded software requirements specification. Both functional and non-functional requirements are important in software capability requirements. There are few research works describing hardware requirements, and there are also many research works involving software design constraints.

Table 1: Requirement Types in Embedded Systems

Requirement Type	References
System Task Intent	[10-13]
System Capability	[10,11,14-20]
Software Capability	[2,5,10-12,14-18,21-97]
Non-Functional	[2,5,10,11,14-18,20-30,32,34-41,43,56,63,64,66-70,72-78,82-84,87,95,98-122]
Hardware	[13,14,16,19,20]
Software Platform Constraints	[18,22,84]
Software Design Constraints	[11,16,21,23,24,33,40,53,60,83,84,97,123-138]

It should be noted that there is a close relationship between these different types of requirements. From the perspective of implementation, system capability requirements must meet system task intent, and software capability requirements and hardware requirements must meet system capability requirements. Because of this relationship, many works involve multiple levels of requirements description. For example, the SCR (software cost reduction) method [14–16] and the ECSAM (embedded computer system analysis and modeling) method [11] both include descriptions of two levels: system capability requirements and software capability requirements; SDL (specification description language) [22], SOFL (structured-object-based-formal language) [23, 126, 127], and Hume [40] all contain descriptions of software capability requirements and software design constraints; the research of Pereira et al. involves system capability requirements, software capability requirements, and hardware requirements [19]. However, no research has yet involved the refinement process from system task intent to low-level requirements.

Next, we will explain the existing expression forms of these requirement types separately.

4.1 System Task Intent and System Capability Requirements

System task intent describes the high-level goals that the system task is to achieve. For example, Leveson et al. described the task intent of an aircraft collision avoidance system as “eliminating the danger of aircraft being too close to each other” or “minimizing the number of situations in which aircraft are too close to each other” [10]. The aircraft here is the external environment. Pereira et al. defined intent as a set of system goals defined by stakeholders [19], such as “serving more passengers.” Similarly, Ponsard et al. [12] and Braun et al. [13] also use goal models to express intent.

System capability requirements are descriptions of the system’s functions [20]. Since embedded systems include software and hardware, and the software needs to perceive information from the external environment through devices such as sensors and exert control over the external environment through devices such as actuators, the system capability requirements are often expressed as the interaction between the system equipment and the external environment or the relationship between the input and output from the external environment. For example, in the SCR method, the system capability requirements are expressed as the relationship between the monitored variables and the controlled variables of the external environment. Among them, the monitored variable is the input perceived by the system through the sensor, and the controlled variable is the output controlled by the system through the actuator. This relationship can be described by a mathematical formula. Leveson et al. defined the controllable part of the external environment as a process, and expressed the system requirements as the relationship or function between the input of the external environment and the output of the system [10]. Wang Fei et al. have a similar description [18]. Zave et al. [32] and Lavi et al. [11] expressed the system requirements as the interaction between the equipment and the external interactive environment. Lavi et al. expressed the system requirements as a task sequence of the system [11], for example, “when there is an intruder in the protected area, the alarm sounds.”

4.2 Software Capability Requirements

Software capability requirements include the functional and non-functional requirements of the software. Generally speaking, software functional requirements describe the relationship between the input and output of the software. Embedded software interacts with the external environment through sensors and actuators, and its input and output are obtained through sensors and actuators. Many studies have focused on the relationship between input events and output events. For example, some studies use state machines to express the relationship between input and output events [10, 11, 22, 25–27, 29, 30]. In EBS (event-based specification language) [33], this relationship is expressed through an event relationship description language [33], while in RSL (requirements statement language) [21], the relationship between input and output events is expressed through a stimulus-response path [21]. Other studies focus on the relationship between input data and output data. For example, the research of PAISLey (process-oriented applicative and interpretable specification language) [32], ESML (extended system modeling language) [31], SOFL, etc., all involve the conversion of input and output data. In ASLAN [36], the properties and constraints that the input and output variables must satisfy, and in SCR and SPARDL (spacecraft requirement description language) [41], the calculation relationship or functional relationship between the input and output variables are discussed.

There are many non-functional requirements for embedded software, mainly including time-related requirements, reliability, robustness, personal safety, and information security. Time-related requirements are the most common [10, 14, 21, 22, 25, 26, 32, 34, 37, 38, 40, 111,137]. They have many names, such as time requirements, real-time requirements, time constraints, performance, etc. According to the strictness of the time requirements, time-related requirements can be divided into three categories.

The first category is the requirement for time series [128]. The second category is the quantitative requirement for time periods and moments [32], which can be expressed by maximum response time (response time cannot exceed a certain value), minimum response time (response time cannot be less than a certain value), average response time, fixed value, etc. The third category is the requirement for real-time performance. For example, some constraints in RTRSM* (real-time requirements specification model*) [29], RT-FRORL (real-time frame and rule oriented requirements language) [35], RTASLAN [37], etc. can be violated, which belongs to soft real-time. Some are hard requirements that must be implemented, which belongs to hard real-time.

Personal safety requirements and information security requirements are also very important. Personal safety requirements refer to measures to prevent the system from harming personal safety or property safety [76, 98, 104, 106, 108, 109, 112, 114, 116, 117]. They are generally expressed by what the system is not allowed to do [129], describing system-specific statements about the existence of safety elements, and are combined with the minimum or maximum required thresholds of some quality measurement standards [130]. Some works also specify the degree of safety that the system must be in. For example, Medikonda et al. further divided personal safety requirements into three types: important safety requirements, pure safety requirements, and safety constraints [131]. Information security requirements [20, 118, 119, 122, 133,136,125] consider the attacks that the software system may face, such as leaking personal data or allowing attackers to gain unauthorized control of the vehicle, and introduce countermeasures that should be taken to deal with various threats of destroying or stealing system information and data.

In addition, other common non-functional requirements include reliability and robustness. Reliability refers to the ability or probability of the system to complete a specific function. For example, PAISLey uses probability to measure reliability. It divides the value range of a function into two parts: success and failure, and uses a random variable, related distribution information, or a fixed value to represent the probability of the two. Robustness usually refers to the ability of the system to operate normally in the face of illegal or incorrect input and unexpected environmental changes. Embedded systems usually need to provide response strategies for abnormal events (such as resource failure, incorrect input, etc.) [10, 14, 40, 78] to ensure the stable operation of the system.

4.3 Hardware Requirements and Software Design Constraints

Hardware refers to the physical equipment used to process, store, and transmit computer programs or data [19]. Hardware includes external devices such as sensors and actuators, and also includes the operating platform of the software. Since peripherals and software operating platforms have different requirements, hardware requirements are divided into requirements for peripherals and constraints on software operating platforms.

Peripheral requirements should provide requirements for the characteristics of peripherals [14, 16], and often include the functions of peripherals, user interaction, hardware characteristics (temperature range, humidity range, battery), action buttons, accuracy, memory specifications, response time, etc. [20]. Constraints on the software operating platform are the requirements for the operating platform such as the processor, memory, and data storage space during functional design. They are all limited resources in embedded systems, so they must be described. Many works have clearly expressed them. For example, Hume proposed a space overhead model to predict the upper limit of the stack and space usage of the program; MARTE (modeling and analysis of real-time and embedded systems) [132] contains many resource-related template attributes to express resource constraints.

Software design constraints generally include interface requirements for hardware devices, software structure, programming language, development standard requirements, confidentiality requirements, maintainability, usability, etc. [84]. Interface requirements elaborate on the requirements of each function at the interface level, which is often used to describe the input and output of hardware interfaces. Generally speaking, interface requirements will also include attributes such as accuracy, range, and time requirements. Here, accuracy refers to the accuracy of the system's output data, that is, it describes the acceptable error between the output data value and the ideal value [14]. In existing research work, the design of the internal structure of software has been studied in depth. For example, SDL uses structured concepts such as blocks and processes to describe the designed internal structure of the system. The SYSREM method [21] assigns the sub-functions after functional decomposition to each system component, and at the same time completes the design of the component interfaces. In addition, the SYSREM method also proposes a distributed computing design system (DCDS) and details the design process of the modules.

5. EXISTING EMBEDDED SYSTEM REQUIREMENTS DESCRIPTION LANGUAGES

After in-depth investigation, we have summarized 20 representative requirements description languages, as shown in Table 2. These languages only cover three types of requirements description: system capability requirements, software capability requirements, and software design constraints. Among them, system capability requirements are the highest level of description. Most of the work focuses on the description of software capability requirements. In contrast, there are relatively few descriptions involving software design constraints.

There is only SCR for system capability requirements, and only SDL for software design constraints. It is worth noting that both languages are involved at the software capability level. However, at different requirement levels, their focus is different. At the system capability requirement level, SCR mainly focuses on the measurable environmental attributes that affect the behavior of the system itself and the environmental attributes that the system can control. Among them, the measurable environmental attributes are called monitored variables, and the environmental attributes controlled by the system are controlled variables. The system capability requirements are expressed as the relationship between these environmental attributes or variables, and the system must ensure the realizability of these relationships. At the same time, SCR also describes the accuracy in measuring monitored variables and calculating controlled variables. In order to achieve acceptable system behavior, the input and output devices must measure the monitored attributes and set the controlled attributes with sufficiently high accuracy and sufficiently small time delay.

In the description of software design constraints, the SDL description language focuses on the organization and design of the internal structure or modules of the system. For example, SDL uses the concept of blocks to describe the system structure. Blocks are hierarchical and can be continuously decomposed or nested, and also contain various sub-structural concepts to describe complex structures.

Next, this section will elaborate on the most expressive software capability description languages from four dimensions: description focus, description dimension,

Table 2: Requirement Description Languages for Embedded Systems

Type	Description Language	Description Focus	Description Dimension	Requirement Analysis Elements	Non-Functional Aspects
System Capability	SCR [14,16]	Environmental attributes affecting system behavior and control	Relationship between monitored/controlled variables	—	Accuracy, Timing
	Statecharts [20], Modechart [27], RTRSM [28], Stateflow [29]	Input/output sequences, conditions, actions, timing constraints	Superstates, AND/XOR decomposition	Timing	—
	SDL [22]	External events/signals requiring system response	Mapping inputs/current state to outputs/updated state	Hierarchical representation	Timing
	RTRL [23]	Correspondence between input/output events	System decomposition into independent modules	Timing	—
	EBS [33]	Events/messages at interfaces and their relationships	—	Timing	—
	SCR [14-16]	Real-world attributes monitored by inputs and controlled by outputs	Output-input relationships across modes	Mode-based state organization	Timing, Performance, Exception handling
	PAISLey [32]	Physical objects, humans, and digital systems controlled by the system	State transitions between asynchronous processes	Hierarchical	Performance, Reliability

Type	Description Language	Description Focus	Description Dimension	Requirement Analysis Elements	Non-Functional Aspects
Software Capability	FRORL/RT-FRORL [34,35,33]	Interactions between real-world objects and their constraints	Rules modifying object interactions	Activity decomposition, refinement	Real-time
	RSML [10]	Controllable variables in real-world processes	Input/output sequences	Superstates, AND decomposition	Accuracy, Timing, Exception handling, Interfaces
	SPARDL [41], Giotto [39]	Response timing of controlled devices	Mode transitions and periodic tasks	Multi-task modes, superstates	Timing
	ASLAN/RT-ASLAN [36,37], ASTRAL [38]	Events triggering system state changes	State transition sets	Multi-level refinement	Real-time
	RSL [31]	External environment information for system processing	Input/output data processing	Functional decomposition	Performance, Accuracy, Timing, Exception handling
	Hume [40]	Shared resources and communication channels with environment	Communication events/resource consumption sequences	Composite resource/event decomposition	Real-time, Resource constraints, Exception handling
Software Design Constraints	SDL [22]	Internal system structure design	—	—	—

requirements analysis elements, and non-functional requirements.

5.1 Description Focus

The description focus is the language's perspective on the world, and it is closely related to how the language views the role of its own system. Different languages have different description focuses. Some languages view their own systems as responsive systems, and their own systems respond to events, signals, or stimuli from the external environment. Therefore, their focus is on the events, signals, or stimuli issued by the external environment. For example, Statecharts [26] continuously responds to stimuli through event-driven. Similarly, Stateflow [30] also describes how software reacts to signals, events, and time-based conditions. Other similar languages include Modechart [27], RTRSM, SDL, RTRL (real-time requirements language) [25], and EBS.

Some languages believe that software will observe and change the world. For example, SCR focuses on the real-world attributes monitored by input devices, such as barometric altitude, ground altitude measured by radar, etc., and the real-world attributes controlled by output devices, such as the coordinates of the flight path markers on the head-up display, radar antenna steering commands, and turn signals. Similarly, PAISLey focuses on modeling distributed and continuous (through discrete simulation) phenomena in a computer system environment. Its environment model can include input/output devices, physical objects controlled by its own system, communication links with humans and other digital systems, and so on. FRORL (frame-and-rule oriented requirements language) [34] and RT-FRORL focus on the interaction between real-world objects, the objects in the real application domain, their possible changes, constraints, and assumptions about this world. RSML (requirements state machine language) [10] is for process control systems and focuses on the variables that can be manipulated and controlled in the process to be controlled. SPARDL and Giotto [39] focus on the response and response time of the controlled device.

The other focuses are summarized as follows. For example, ASLAN and RTASLAN and ASTRAL (ASLAN based TRIO assertion language) [38] view their own systems as being in a set of states, and they focus on events that can cause state transitions in their own systems. RSL and Hume believe that their own systems are for processing data, so they focus on the external environment messages to be processed in the system. GCSR (graphical communicating shared resources) [78] views a real-time system as a set of communication components that execute on a limited set of serial shared resources and synchronize with the components through communication channels. It focuses on the resources shared with the external environment and their communication channels.

In short, from the current description focus, only focusing on the changes in the state of the system itself or only viewing itself as a responder to external stimuli will make it difficult to solve the problems that come with the increasing scale of embedded systems. As the number of devices increases, the external stimulus events and system state data will also increase, which will make it difficult for users to understand and accurately express their needs. The view that software will observe and transform the world will be strengthened. They can effectively limit the user's needs to the scheduling and control of these external worlds, and define the software itself by transforming the external world, so that users can clearly express their actual needs.

5.2 Description Dimension

Due to different description focuses, the description dimensions of languages will also be different. Many languages describe software requirements from the reaction to signals, events, and time conditions. They often use state machines or state transition diagrams as a basis for representation. For example, Statecharts regards software behavior as a collection of elements such as input and output, conditions, actions, and time constraints. It extends the ordinary state transition diagram and introduces hierarchical and concurrent states to express requirements. Similar ones include RTRSM and Stateflow. The only difference is that Stateflow uses the state machine representation method of Statecharts and flowcharts for description, and also provides state transition tables and truth tables.

Modechart regards software behavior as a sequence of specific actions that may be executed in a certain mode. It pays more attention to the time constraints related to the start and completion of actions. Similar expression content also includes SDL. SDL does not use the common form of state transition diagrams, but uses a syntax form that emphasizes the reception and sending of messages, but the meaning of the final description result is still the same as that of the state transition diagram.

Some languages describe the sequence of input and output, although their specific expression forms may be slightly different. For example, EBS regards software behavior as events or messages that occur on the interface and their interrelationships. It defines three event relationships: time sequence, concurrency, and enabling relationship, and uses the symbols of these three event relationships and first-order predicate logic to describe the behavior of the software. RTRL regards software behavior as the correspondence between input events and output events, where the output is not only related to the set of input signals, but also to the arrival order of these inputs. RSML uses a black-box method to describe the behavior of the controller, which also describes the input and output sequence of the controller. As the state of the controlled process changes, its behavior will also change accordingly. The representation method used is a hierarchical and concurrent state machine. Methods belonging to this category also include SCR and PAISLey.

SCR regards software as a set of functions associated with output data items, and each output data item is assigned a value by a function. These input data items and output data items both represent the interaction events with sensors and actuators. It uses tables to represent the relationship between modes, states, and outputs. The basic description unit of PAISLey is a function, which includes a state space and a successor function that defines the successor state for each state. The asynchronous interaction process between functions is defined by an exchange function. It uses a hierarchical data flow diagram and an embedded control state machine to represent function calls.

Some languages view software behavior as a set of rules that change the interaction of real-world objects, such as FRORL and RT-FRORL. Their descriptions mainly include objects and activities. Objects have some attributes, and activities have participating objects, preconditions, action sequences, alternative processes, etc. Actions can be other activities or assertions. Both activities and assertions are represented using first-order predicate logic.

Some languages view software behavior as a set of modes that transition and periodically driven computing tasks. For example, SPARDL uses modes and mode transitions to organize tasks, and the mode also contains a set of computing modules expressed by a control flow diagram. SPARDL uses the expression form of a state transition diagram, and the modes allow nesting, which is similar to the hierarchical state of Statecharts. Similarly, there is also Giotto, but Giotto does not care about the specific implementation of the task, but only represents the external interaction of the task.

Some languages view software behavior as a set of transitions that can change the state of their own system. For example, ASLAN and RTASLAN. They use states and state transitions to organize the description of software behavior. The state here refers to the value of the state variable at a certain moment in the software or its own system. This state is not suitable for representation by a state transition diagram. RT-ASLAN uses assertions and invariants based on first-order predicate logic to describe the state. ASTRAL retains the same representation as in ASLAN and RT-ASLAN, and its semantics are defined in TRIO logic.

There are also descriptions such as GCSR that view their own system as a set of communication processes that share limited resources. The behavior of the software consists of a set of execution step sequences, where each execution step represents a communication event or a time and resource consumption action. It uses graphics to represent these symbols, and its process defines communication events, events or resource consumption actions, and the relationship between them. Its semantics are defined as a labeled transition system or converted to the process algebra ACSR.

The main dimensions of software requirements description include behavior and data. Most languages focus more on the description of behavior, mainly focusing on its action behavior flow, and describing the reaction to signals, events, and event conditions. These languages usually use finite state machines, state transition diagrams, predicate logic, and other means for description. Some methods will use hierarchical concurrent state machines (mode transition diagrams) to describe complex behaviors. However, there are also a few languages, such as RSL and Hume, that focus more on data conversion. These languages focus on processing the external environment information within the system, use the input data and output data processing process to describe the function, and may consider using a hierarchical structure to represent complex functions. There are also some works, such as SPARDL, that focus on both behavior and data. They have both mode diagrams and computing tasks. For complex embedded systems, both their behavior and data are complex. Therefore, in their software requirements description, it is necessary to organically combine the models of these two dimensions.

5.3 Requirements Analysis Elements

After investigation, we found that in different requirements description languages, there are their own unique description elements to facilitate requirements analysis at different levels. As shown in Table 2, most languages can support the expression of requirements at different granularity levels. Only EBS, which only relies on events and the relationship between events for description, has no top-down analysis method.

In requirements analysis, there are mainly the following forms of analysis elements. Many languages have proposed the concept of a superstate to represent states hierarchically. For example, Statecharts not only proposes a superstate, but also proposes “AND-decomposition” and “XOR-decomposition” for the superstate, allowing cross-layer migration and parallel relationship representation of states, and supporting the continuous refinement of behavior description from high to low levels. This mechanism of Statecharts has been borrowed by many languages, including Modechart, RTRSM*, Stateflow, and RSML.

Some languages use modes to organize tasks. For example, the SCR method points out that after representing the system output data as a mapping relationship between the system state and input, modes are used to organize these mapping relationships, and the modes are different system state classes. SPARDL uses modes to organize tasks. Its mode diagram contains two levels. The high level is the mode and mode transition, and the low level is a set of computing tasks represented by a control flow in each mode. The Giotto language [39] is also similar. The difference is that the modes in SPARDL allow nesting, which is similar to the superstate in Statecharts. The tasks in Giotto allow concurrent execution, while the control flow in SPARDL is sequential execution.

Some languages are divided into many levels and define the relationship between their levels. For example, the ASLAN language, whose requirements description contains a sequence of levels, and each level is an abstract view of the system’s data types. The top-level view is a very abstract model of the system’s composition, and also includes what the system does (state transitions) and the key requirements that the system must meet (invariants, constraints). Lower levels will add more details, and the implementation of high-level requirements by low-level requirements is represented by an implementation relationship. The ASTRAL language retains the hierarchical representation of ASLAN. The Hume language supports a three-layer representation. The outermost layer represents the external (static) declaration/metaprogramming layer, the middle layer describes the static layout of the dynamic process, and the inner layer describes each process as a dynamic mapping from a pattern that matches the input to an expression that produces the output. The RTRL language uses features to express requirements, decomposes requirements into a set of features, and features contain many independently implemented functions.

Some languages support the decomposition of language elements. For example, the FRORL language uses the frame of objects and activities to express the requirements model. The concept of activity is an abstract concept. It represents each activity frame as a variable, and different instances of the same activity can be represented in the same frame. It provides a mechanism to decompose an activity into multiple sub-activities, and these sub-activities can be in a sequential or parallel relationship. At the same time, it also supports a stepwise refinement mechanism. The PAISLey specification is composed of a set of function definitions, and the call of functions is represented by a hierarchical data flow diagram and an embedded control table. In GCSR, an entity is a basic concept. It can contain many nodes (such as resources, events, etc.) and composite nodes (a group of resources and events), and can be decomposed into lower-level entities. The RSL language contains AND and OR nodes to decompose the conditions for processing requirements. The SDL language uses blocks to organize the system's behavior and structure. This block can be continuously decomposed into multiple blocks until a block only contains processes. SDL also provides a variety of substructure concepts: the substructure of a block is used to further describe the internal structure of the block; the substructure of a channel is used to describe the behavior within the channel; the signal refinement mechanism is the refinement of signals, the purpose of which is to hide the details of low-level signals to obtain a high-level abstraction, allowing the system's behavior to be described from top to bottom.

In addition, in order to support the requirements hierarchy in the language, some languages also have method support. For example, the SREM method [21] to which the RSL language belongs supports the decomposition of functions into a group of low-level functions. The SCR method and PAISLey also have method support. These methods provide a process for obtaining the specifications of these requirements description languages, which is also a very important part.

In short, the existing languages basically support the expression of different description granularities, which may be the same or different language elements, and support the description of requirements of different granularities, so that the requirements can be continuously improved from a higher abstract level of description of the same type to a lower level of specific description through decomposition, refinement, and other means. There are also some works that support the expression of requirements granularity of different requirement types and establish the refinement or decomposition relationship between them. Some languages will also provide decomposition and refinement mechanisms, and even process support, but decomposition and refinement all rely on requirements analysts and domain experts, and their quality seriously depends on human experience. In particular, the current decomposition is basically functional division, and there is no intertwining between requirements. For complex embedded systems, both decomposition and refinement are needed, but how to improve efficiency and quality is still a problem to be solved.

5.4 Supported Non-functional Requirements

According to the investigation, the non-functional requirements involved in the current software requirements description languages mainly include time-related requirements (timing, real-time, delay, performance, time constraints, etc.), accuracy, exception handling, and resource constraints. Time-related requirements are involved more.

Some languages use the concept of timers to represent delays. For example, Statecharts and SDL both use timers to specify the delay time on the basis of the state and transition of the functional requirements description. After this time, the timer generates a specific (timeout) event or signal, which in turn leads to a state transition.

Some languages express periodic time constraints and their corresponding sporadic time constraints. For example, the functional description in SCR is divided into demand functions and periodic functions, which correspond to these two time constraints. The demand function specifies a specific trigger event, and the periodic function specifies the start and stop events, as well as the execution period. The time constraint of Modechart is related to the execution of actions under a specific mode and conditions of the system. The sporadic time constraint is expressed as the completion deadline and interval time, and the periodic time constraint is expressed as the execution period and completion deadline. Similarly, the periodic time constraint in RT-FRORL is represented as a simple periodic attribute of a time activity, and the representation of the paroxysmal time constraint introduces the operation of a global clock variable. The time requirement in RT-ASLAN is expressed as the sequential relationship of state transitions and the periodic attribute of transitions.

Other time-related expressions are as follows. The timing in ESB is expressed by defining the time sequence relationship of events and is integrated into the functional description. The performance requirement in RSL is expressed by a specific data processing path in the R-net, and the performance requirement is expressed as the maximum and minimum response time of a specific path. The real-time requirement in GCSR is related to actions, and the execution of actions takes a certain amount of time. ASTRAL, on the basis of RT-ASLAN, provides operations for the start and end times of state transitions, and also adds the requirement of scheduling time for state transitions. The time requirements in Giotto and SPARDL are first expressed as their described computing tasks are all periodic. The mode in SPARDL has a periodic attribute, and the mode in Giotto has a conversion frequency. SPARDL also provides timing predicates and timing control flow relationships to describe the transition conditions of modes and control the computing tasks within the modes. The performance requirement in PAISLey is related to the function that describes its process behavior. The time requirement in RSML is expressed in the guarding condition of the state transition and is expressed as a time function. RSML describes three time functions: the variable value at the previous time point, the true value of the condition at a certain point in the past, and the event implicitly generated based on time (that is, the timeout relative to the state item), all of which are based on the guarding condition.

For exception handling in robustness, SCR records all abnormal events and the system’s response as a separate part, which is divided into three categories: resource (device) failure, incorrect input data, and incorrect internal data. The superstate and hierarchical state machine in Statecharts can flexibly represent local abnormal events and global abnormal events and their handling, and RSML is similar to it. GCSR clearly handles abnormal events through an exception edge, allowing cross-level migration of states, which is similar to Statecharts.

Other non-functional requirements are expressed as follows. Accuracy requirements are all related to data. For example, the accuracy requirement in RSL is related to the data stored at the verification point. The resource requirement in GCSR is related to actions, and the execution of actions consumes a set of resources. The reliability requirement in PAISLey is expressed by probability and is related to the function that describes its process behavior.

In short, from the perspective of non-functional requirements, the description of software capability requirements includes functional requirements. Its existing non-functional requirements descriptions only include some requirements that can be expressed on the basis of functions, such as time, reliability, exception handling, and accuracy. Other types of requirements that cannot be directly expressed on the basis of functions, such as personal safety, are not involved.

6. ANALYSIS OF THE DEVELOPMENT TREND OF REQUIREMENTS DESCRIPTION FOR EMBEDDED SYSTEMS

This section starts with the challenges faced by the requirements description of complex embedded systems, proposes the need to study new embedded software requirements description languages, and further proposes specific requirements for the requirements description for the intelligent synthesis of embedded software.

6.1 Challenges for Requirements Description of Complex Embedded Systems

The development of embedded systems is a process of “embedding” software into a “embeddable” computing device. First, the computing function is “embedded” into the application object, and then with the access of multiple forms of networks, the embedded system presents a networked feature. In recent years, embedded software has been integrated into more physical objects, forming a variety of embeddable digital devices. They have the ability of environmental perception and autonomous interaction, so that the computing device is deeply “embedded” in the application object and “disappears” in the physical world, promoting the deep integration of the ternary world of human, machine, and object. Such complex embedded systems bring the following challenges to the description of software requirements.

First, the description of complex embedded requirements cannot only involve software capability requirements, but should describe all types of requirements involved in embedded systems, that is, task intent, system requirements, peripheral requirements, operating platform constraints, and software design constraints. The development of embedded software starts from the task intent, and its task intent is closely related to the external interactive environment. The task intent and system capability requirements are its source. Embedded software will be directly placed in the physical world through a variety of sensing devices and acting devices, and its system equipment requirements must also be recorded. In addition, embedded software needs to run on a specific platform according to a specific strategy, and its resources, performance, etc. must be constrained. Software design constraints may also exist.

Secondly, it is necessary to define the description dimensions of each requirement level. Complex embedded systems involve complex control processes and calculation processes, including two major dimensions of behavior and data. But at different requirement levels, the dimensions may be completely different. At the task intent level, it is mainly for the expected effect on the external environment. The system capability requirement is as long as the input and output sequence with the external environment. The software requirement is the input and output sequence with the system equipment, and the software design constraint includes the calculation formula between the input and output.

Third, it is necessary to define a systematic software requirements specification method to establish a tracking relationship between different levels from task intent to software requirements (including functional and non-functional requirements) to cope with the increasing complexity of embedded system requirements changes. In embedded systems, the system task intent is the beginning of software development, the system capability requirements meet the task intent, and the software capability requirements and system equipment requirements jointly meet the system capability requirements. With the tracking relationship, the software requirements can be effectively changed according to the reasons for the requirements changes. Fourth, it is necessary to provide more efficient requirements analysis methods for the decomposition and refinement mechanism of requirements. In embedded systems, due to the strong device dependence of its software, the control requirements of the software requirements are intertwined, and the scheduling and control of the same device appear in different requirements, such as the two requirements of “automatic light off” and “manual light off” are both to control the light. This device intertwining makes the original partitioned requirements decomposition (no intertwining) no longer applicable. They do not pay attention to the characteristics of the device in the decomposition, which may lead to inconsistent requirements after decomposition, such as one requirement may require the light to be turned on, while another requirement requires the light to be turned off at the same time. In addition, with the increase of embedded devices and the complexity of embedded systems, manual requirements analysis will become a bottleneck in development, and automated requirements analysis methods are needed to improve efficiency.

Complex embedded systems need to design new requirements description languages. In the design, it is necessary to target the characteristics of embedded software, describe the task intent of embedded software, extract the main components of the embedded software language on the basis of requirements analysis at all levels and dimensions, and define the logic of the language to support the expression of embedded software requirements. This is a task that spans the entire requirements stage of embedded software. It is necessary to comprehensively learn from and expand existing requirements engineering methods, combine effective requirements extraction, modeling, analysis, simulation, and verification technologies, propose a language structure, and define a structured embedded system requirements description language with sufficient expressive power.

6.2 Requirements from the Intelligent Synthesis of Embedded Software

At present, software intelligent synthesis has become a software automation technology that has attracted much attention [134]. Software intelligent synthesis refers to the use of artificial intelligence foundations such as machine learning on the basis of traditional software synthesis technology to automatically synthesize software that meets user intent by using existing code knowledge. The software requirements specification that carries the user’s intent is the basis for software intelligent synthesis, and its importance is self-evident. The intelligent synthesis of embedded system software will also become a future research hotspot [135], which brings new requirements to the description language of embedded system software requirements specification.

First, from the perspective of description dimension, it needs to express various dimensions of requirements specifications, including behavior, data, constraints, etc. Embedded software is the same as general software, and each piece of code has a control flow and a data flow. Since most embedded software needs to schedule system equipment in real time, the description of system equipment must be added to the requirements description, such as ports, response time, communication protocols, etc. Not only that, it also needs to add constraints on performance, efficiency, safety, reliability, etc., so that software that meets the needs of equipment scheduling can be synthesized.

Secondly, from the perspective of description granularity, it is best to decompose the software requirements to a relatively small granularity, also known as atomic requirements. This granularity is best to be at the same description granularity as the function of the software asset, or a smaller granularity. Refinement technology is still needed to improve the description of requirements from a high abstract level to a low-level specific description. In addition, due to the high complexity of current embedded systems, the decomposition or decoupling of requirements is essential. Different from the software decomposition of traditional information systems—functional division, the software of embedded systems is closely related to hardware devices, and its decomposition also needs to consider that the same device may be called in different functions, and there is device sharing, which will bring new challenges to the decoupling of intertwined software requirements.

Third, from the perspective of the description scheme, a complete project requirements specification must also clarify the dependency relationship between atomic requirements, so as to synthesize complete project code. Due to the intertwining of devices between atomic requirements, it may bring control dependencies. For example, the requirement “automatic light on” requires the light to be in a powered state first, while the requirement “initialization” will make the light enter a powered state, then “automatic light on” is control-dependent on “initialization.” Due to the data sharing between atomic requirements, it may bring data dependencies, that is, one requirement produces data, and another requirement uses data. These control dependencies and data dependencies may be expressed as sequential or concurrent relationships between atomic requirements, which will affect the subsequent synthesis.

Finally, from the perspective of description form, the software requirements description language for intelligent synthesis must be machine-understandable to facilitate subsequent code synthesis. It is best to be formally expressed, with strict syntax and precise semantics. Its software requirements specification can be automatically converted into a simulation model or a verification model. Before synthesizing the software, the requirements can be simulated, confirmed, and verified. The test cases of the final synthesized software can be automatically generated, and the software can be tested after synthesis. After multiple simulations, verifications, and tests, the quality of the synthesized code is guaranteed.

7. CONCLUSION

This paper conducted a systematic literature review on the topic of requirements description for embedded systems, provided an overview of the current situation of requirements description types for embedded systems, comprehensively compared the capabilities of existing embedded system requirements description languages, summarized the challenges faced by the requirements description of complex embedded systems, predicted future trends, and discussed the capability requirements of the requirements description language for embedded systems for the task of intelligent software synthesis.

During the review and investigation process, our exclusion criteria may have some bias, which may cause our review paper to not cover all relevant fields. For example, we did not include references written in other languages. However, since most research results have corresponding English versions, this will not have a substantial impact on our investigation of the current situation of embedded requirements description types and requirements description languages. In addition, the search process may have a certain degree of instability, and the search engine may sometimes have a large number of documents that are not related to the search keywords. However, documents with higher relevance will usually be displayed first, so it will not affect our research results.

Currently, most requirements description languages only describe software capability requirements and do not describe various other possible types of requirements, such as system task intent, system capability requirements, etc. For the future intelligent synthesis of embedded software, the new requirements description language describes various types of requirements starting from the task intent, establishes their tracking relationship, can analyze the problem of control requirements intertwining caused by device sharing, and through decoupling, describes the requirements at a more appropriate granularity to facilitate subsequent code synthesis based on software assets.

REFERENCES

1. Broy M. Challenges in automotive software engineering. In: Proc. of the 28th Int'l Conf. on Software Engineering. Shanghai: ACM, 2006. 33–42. [doi: 10.1145/1134285.1134292]

2. Feng JC, Miao WK, Zheng HY, Huang YH, Li JW, Wang Z, Su T, Gu B, Pu GG, Yang MF, He JF. FREPA: An automated and formal approach to requirement modeling and analysis in aircraft control domain. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and the Symp. on Foundations of Software Engineering. ACM, 2020. 1376–1386. [doi: 10.1145/3368089.3417047]

3. Mumtaz S, Alsohaily A, Pang ZB, Rayes A, Tsang KF, Rodriguez J. Massive Internet of Things for industrial applications: Addressing wireless IIoT connectivity challenges and ecosystem fragmentation. *IEEE Industrial Electronics Magazine*, 2017, 11(1): 28–33. [doi: 10.1109/MIE.2016.2618724]

4. Broy M, Stauner T. Requirements engineering for embedded systems. 1999. https://wwwbroy.in.tum.de/publ/papers/femsys_boesswet_1997_Conference.pdf#:~:text=In%20requirements%20engineering%20we%20describe%20the%20required%20behaviour,be%20as%20simple%2C%20abstract%2C%20and%20suggestive%20as%20possible

5. Naumchev A, Meyer B, Mazzara M, Galinier F, Bruel JM, Ebersold S. Autoreq: Expressing and verifying embedded software requirements. arXiv:1710.02801v1, 2017.

6. Davis AM. The design of a family of application-oriented requirements languages. *Computer*, 1982, 15(5): 21–28. [doi: 10.1109/MC.1982.1654021]

7. Melhart BE. Specification languages for embedded systems: A survey. Technical Report, Irvine: UCI, 1988.

8. Jin Z. *Environment Modeling-based Requirements Engineering for Software Intensive Systems*. Amsterdam: Elsevier, 2018. [doi: 10.1016/C2014-0-00030-5]

9. Kitchenham B. Procedures for performing systematic reviews. Technical Report, 0400011T.1, Keele: Keele University, 2004. 1–26.

10. Leveson NG, Heimdahl MPE, Hildreth H, Reese JD. Requirements specification for process-control systems. *IEEE Trans. on Software Engineering*, 1994, 20(9): 684–707. [doi: 10.1109/32.317428]

11. Lavi JZ, Kudish J. Systems modeling & requirements specification using ECSAM: An analysis method for embedded & computer-based systems. *Innovations in Systems and Software Engineering*, 2005, 1(2): 100–115. [doi: 10.1007/s11334-005-0010-4]

12. Ponsard C, Massonet P, Molderez JF, Rifaut A, van Lamsweerde A, van Tran H. Early verification and validation of mission critical systems. *Formal Methods in System Design*, 2007, 30(3): 233–247. [doi: 10.1007/s10703-006-0028-8]

13. Braun P, Broy M, Houdek F, Kirchmayr M, Müller M, Penzenstadler B, Pohl K, Weyer T. Guiding requirements engineering for software-intensive embedded systems in the automotive industry. *Computer Science—research and Development*, 2014, 29(1): 21–43. [doi: 10.1007/s00450-010-0136-y]

14. Heninger KL. Specifying software requirements for complex systems: New techniques and their application. *IEEE Trans. on Software Engineering*, 1980, SE-6(1): 2–13. [doi: 10.1109/TSE.1980.230208]

15. Parnas DL, Madey J. Functional documents for computer systems. *Science of Computer Programming*, 1995, 25(1): 41–61. [doi: 10.1016/0167-6423(95)96871-J]

16. Heitmeyer CL. Software cost reduction. In: Marciniak JJ, ed. *Encyclopedia of Software Engineering*. Hoboken: John Wiley & Sons, 2002. [doi: 10.1002/0471028959.sof307]

17. Yoo J, Kim T, Cha S, Lee JS, Seong Son H. A formal software requirements specification method for digital nuclear plant protection systems. *Journal of Systems and Software*, 2005, 74(1): 73–83. [doi: 10.1016/j.jss.2003.10.018]

18. Wang F, Yang ZB, Huang ZQ, Zhou Y, Liu CW, Zhang WB, Xue L, Xu JM. Approach for generating AADL model based on restricted natural language requirement template. *Ruan Jian Xue Bao/Journal of Software*, 2018, 29(8): 2350–2370 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5530.htm [doi: 10.13328/j.cnki.jos.005530]

19. Pereira T, Sousa A, Silva R, Albuquerque D, Alencar F, Castro J. A metamodel to guide a requirements elicitation process for embedded systems. In: Proc. of the 11th Int'l Conf. on the Quality of Information and Communications Technology. Coimbra: IEEE, 2018. 101–109. [doi: 10.1109/QUATIC.2018.00023]

20. Apvrille L, Li LW. Harmonizing safety, security and performance requirements in embedded systems. In: Proc. of the 2019 Design, Automation & Test in Europe Conf. & Exhibition (DATE). Florence: IEEE, 2019. 1631–1636. [doi: 10.23919/DATE.2019.8715124]

21. Alford M. SREM at the age of eight; the distributed computing design system. *Computer*, 1985, 18(4): 36–46. [doi: 10.1109/MC.1985.1662863]

22. Belina F, Hogrefe D. The CCITT-specification and description language SDL. *Computer Networks and ISDN Systems*, 1989, 16(4): 311–341. [doi: 10.1016/0169-7552(89)90078-0]

23. Liu SY, Offutt AJ, Ho-Stuart C, Sun Y, Ohba M. SOFL: A formal engineering methodology for industrial applications. *IEEE Trans. on Software Engineering*, 1998, 24(1): 24–45. [doi: 10.1109/32.663996]

24. Liu S, Asuka M, Komaya K, Nakamura Y. An approach to specifying and verifying safety-critical systems with practical formal method SOFL. In: Proc. of the 4th IEEE Int'l Conf. on Engineering of Complex Computer Systems. Monterey: IEEE, 1998. 100–114. [doi: 10.1109/ICECCS.1998.706660]

25. Taylor B. A method for expressing the functional requirements of real-time systems. *Annual Review in Automatic Programming*, 1980, 10: 111–120. [doi: 10.1016/0066-4138(82)90015-5]

26. Harel D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 1987, 8(3): 231–274. [doi: 10.1016/0167-6423(87)90035-9]

27. Jahanian F, Mok AK. Modechart: A specification language for real-time systems. *IEEE Trans. on Software Engineering*, 1994, 20(12): 933–947. [doi: 10.1109/32.368134]

28. Shu FD, Wu GQ, Wang M. Embedded real-time software-oriented requirements engineering environment—SREE. *Computer Science*, 2002, 29(4): 4–8, 14 (in Chinese with English abstract). [doi: 10.3969/j.issn.1002-137X.2002.04.002]

29. Shu FD, Wu GQ, Li MS. An embedded real-time software oriented requirements specification language and checking methods. *Ruan Jian Xue Bao/Journal of Software*, 2004, 15(11): 1595–1606 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1595.htm

30. The MathWorks. *Stateflow® and Stateflow® CoderTM User's Guide*. Natick: The MathWorks, 2009.

31. Bruyn W, Jense R, Keskar D, Ward P. An extended systems modeling language (ESML). *ACM SIGSOFT Software Engineering Notes*, 1988, 13(1): 58–67. [doi: 10.1145/43857.43866]

32. Zave P. An operational approach to requirements specification for embedded systems. *IEEE Trans. on Software Engineering*, 1982, SE-8(3): 250–269. [doi: 10.1109/TSE.1982.235254]

33. Chen BS, Yeh RT. Formal specification and verification of distributed systems. *IEEE Trans. on Software Engineering*, 1983, SE-9(6): 710–722. [doi: 10.1109/TSE.1983.235434]

34. Tsai JJP. A knowledge-based system for software design. *IEEE Journal on Selected Areas in Communications*, 1988, 6(5): 828–841. [doi: 10.1109/49.634]

35. Tsai JJJP, Jang HC. A knowledge-based approach for the specification and analysis of real-time software systems. *Int'l Journal on Artificial Intelligence Tools*, 1992, 1(1): 1–35. [doi: 10.1142/S0218213092000119]

36. Auernheimer B, Kemmerer RA. ASLAN User's Manual. Santa Barbara: University of California, 1985.

37. Auernheimer B, Kemmerer RA. RT-ASLAN: A specification language for real-time systems. *IEEE Trans. on Software Engineering*, 1986, SE-12(9): 879–889. [doi: 10.1109/TSE.1986.6313044]

38. Ghezzi C, Kemmerer RA. ASTRAL: An assertion language for specifying realtime systems. In: Proc. of the 3rd European Software Engineering Conf. Milan: Springer, 1991. 122–146. [doi: 10.1007/3540547428_46]

39. Henzinger TA, Horowitz B, Kirsch CM. Giotto: A time-triggered language for embedded programming. In: Proc. of the 1st Int'l Workshop on Embedded Software. Tahoe City: Springer, 2001. 166–184. [doi: 10.1007/3-540-45449-7_12]

40. Hammond K, Michaelson G. Hume: A domain-specific language for real-time embedded systems. In: Proc. of the 2nd Int'l Conf. on Generative Programming and Component Engineering. Erfurt: Springer, 2003. 37–56. [doi: 10.1007/978-3-540-39815-8_3]

41. Gu B, Dong YW, Wang Z. Formal modeling approach for aerospace embedded software. *Ruan Jian Xue Bao/Journal of Software*, 2015, 26(2): 321–331 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4784.htm [doi: 10.13328/j.cnki.jos.004784]

42. Miao WK, Pu GG, Yao YB, Su T, Bao DZ, Liu Y, Chen SH, Xiong KP. Automated requirements validation for ATP software via specification review and testing. In: Proc. of the 18th Int'l Conf. on Formal Engineering Methods. Tokyo: Springer, 2016. 26–40. [doi: 10.1007/978-3-319-47846-3_3]

43. Feng JC. Detailed requirement of embedded system oriented formal modeling and analysis [Ph.D. Thesis]. Shanghai: East China Normal University, 2022 (in Chinese with English abstract). [doi: 10.27149/d.cnki.ghdsu.2022.004041]

44. Wang S, Feng JC, Zhu JY, Huang YH., Zheng HY, Xu XR, Miao WK, Zhang X, Pu GG. A dimensional analysis method for the requirements model of railway control software. *Chinese Journal of Computers*, 2020, 43(11): 2152–2165 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.02152]

45. Dietrich D, Atlee JM. A pattern for structuring the behavioural requirements of features of an embedded system. In: Proc. of the 3rd Int'l Workshop on Requirements Patterns (RePa). Rio De Janeiro: IEEE, 2013. 1–7. [doi: 10.1109/RePa.2013.6602664]

46. Hoang TS, Snook C, Salehi A, Butler M, Ladenberger L. Validating and verifying the requirements and design of a haemodialysis machine using the rodin toolset. *Science of Computer Programming*, 2018, 158: 122–147. [doi: 10.1016/j.scico.2017.11.002]

47. Park S. Software requirement specification based on a gray box for embedded systems: A case study of a mobile phone camera sensor controller. *Computers*, 2019, 8(1): 20. [doi: 10.3390/computers8010020]

48. Ghazel M, Yang J, El-Koursi EM. A pattern-based method for refining and formalizing informal specifications in critical control systems. *Journal of Innovation in Digital Ecosystems*, 2015, 2(1–2): 32–44. [doi: 10.1016/j.jides.2015.11.001]

49. Galinier F. A DSL for requirements in the context of a seamless approach. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering. Montpellier: IEEE, 2018. 932–935. [doi: 10.1145/3238147.3241538]

50. Bujorianu MC, Bujorianu ML. An integrated specification framework for embedded systems. In: Proc. of the 5th IEEE Int'l Conf. on Software Engineering and Formal Methods (SEFM 2007). London: IEEE, 2007. 161–172. [doi: 10.1109/SEFM.2007.6]

51. Lattemann F, Lehmann E. A methodological approach to the requirement specification of embedded systems. In: Proc. of the 1st IEEE Int'l Conf. on Formal Engineering Methods. Hiroshima: IEEE, 1997. 183–191. [doi: 10.1109/ICFEM.1997.630425]

52. Maraninchi F, Rémond Y. Argos: An automaton-based synchronous language. *Computer Languages*, 2001, 27(1–3): 61–92. [doi: 10.1016/S0096-0551(01)00016-9]

53. Roßkopf A, Tempelmeier T. Aspects of flight control software—A software engineering point of view. *Control Engineering Practice*, 2000, 8(6): 675–680. [doi: 10.1016/S0967-0661(00)00012-5]

54. Fuchs NE, Schwertel U, Schwitter R. Attempto controlled English—Not just another logic specification language. In: Proc. of the 8th Int'l Workshop on Logic Programming Synthesis and Transformation. Manchester: Springer, 1999. 1–20. [doi: 10.1007/3-540-48958-4_1]

55. Pettersson F, Ivarsson M, Öhman P. Automotive use case standard for embedded systems. *ACM SIGSOFT Software Engineering Notes*, 2005, 30(4): 1–6. [doi: 10.1145/1082983.1083193]

56. Shan JH, Zhao HY, Wang JB, Wang RX, Ruan CL, Yao ZX. An extended TASM-based requirements modeling approach for real-time embedded software: An industrial case study. In: Proc. of the 15th National Software Application Conf. on Software Engineering and Methodology for Emerging Domains. Kunming: Springer, 2016. 19–34. [doi: 10.1007/978-981-10-3482-4_2]

57. Jung H, Lee C, Kang SH, Kim S, Oh H, Ha S. Dynamic behavior specification and dynamic mapping for real-time embedded systems: Hopes approach. *ACM Trans. on Embedded Computing Systems*, 2014, 13(4s): 135. [doi: 10.1145/2584658]

58. Post A, Menzel I, Podelski A. Applying restricted english grammar on automotive requirements—Does it work? A case study. In: Proc. of the 17th Int'l Working Conf. on Requirements Engineering: Foundation for Software Quality. Essen: Springer, 2011. 166–180. [doi: 10.1007/978-3-642-19858-8_17]

59. Yue T, Briand LC, Labiche Y. A use case modeling approach to facilitate the transition towards analysis models: Concepts and empirical evaluation. In: Proc. of the 12th Int'l Conf. on Model Driven Engineering Languages and Systems. Denver: Springer, 2009. 484–498. [doi: 10.1007/978-3-642-04425-0_37]

60. Stachtiari E, Mavridou A, Katsaros P, Bliudze S, Sifakis J. Early validation of system requirements and design through correctness-by-construction. *Journal of Systems and Software*, 2018, 145: 52–78. [doi: 10.1016/j.jss.2018.07.053]

61. Mavin A, Wilkinson P, Harwood A, Novak M. Easy approach to requirements syntax (EARS). In: Proc. of the 17th IEEE Int'l Requirements Engineering Conf. Atlanta: IEEE, 2009. 317–322. [doi: 10.1109/RE.2009.9]

62. Mavin A, Wilkinson P. Ten years of EARS. *IEEE Software*, 2019, 36(5): 10–14. [doi: 10.1109/MS.2019.2921164]

63. Górski J. Formal specification of real time systems. *Computer Physics Communications*, 1988, 50(1–2): 71–88. [doi: 10.1016/0010-4655(88)90117-8]

64. Al-Fedaghi S. High-level representation of time in diagrammatic specification. *Procedia Computer Science*, 2015, 62: 478–486. [doi: 10.1016/j.procs.2015.08.519]

65. Denger C, Berry DM, Kamsties E. Higher quality requirements specifications through natural language patterns. In: Proc. of the 2003 Symp. on Security and Privacy. Herzlia: IEEE, 2003. 80–90. [doi: 10.1109/SWSTE.2003.1245428]

66. Maraninchi F, Rémond Y. Object-oriented logical specification of time-critical systems. *ACM Trans. on Software Engineering and Methodology*, 1994, 3(1): 56–98. [doi: 10.1145/174634.174636]

67. Rashid M, Anwar MW, Azam F, Kashif M. Model-based requirements and properties specifications trends for early design verification of embedded systems. In: Proc. of the 11th System of Systems Engineering Conf. (SoSE). Kongsberg: IEEE, 2016. 1–7. [doi: 10.1109/SYSOSE.2016.7542917]

68. Ravindran B, Edwards S. Palette: A reuse-oriented specification language for real-time systems. In: Proc. of the 6th Int'l Conf. on Software Reuse: Advances in Software Reusability. Vienna: Springer, 2000. 20–40. [doi: 10.1007/978-3-540-44995-9_2]

69. Kang KC, Ko KI. PARTS: A temporal logic-based real-time software specification and verification method. In: Proc. of the 17th Int'l Conf. on Software Engineering. Seattle: IEEE, 1995. 169. [doi: 10.1109/ICSE.1995.495030]

70. Videira C, Da Silva AR. Patterns and metamodel for a natural-language-based requirements specification language. In: Proc. of the 17th Conf. on Advanced Information Systems Engineering. Porto: CAiSE, 2005.

71. Mahmud N, Seceleanu C, Ljungkrantz O. ReSA: An ontology-based requirement specification language tailored to automotive systems. In: Proc. of the 10th IEEE Int'l Symp. on Industrial Embedded Systems (SIES). Siegen: IEEE, 2015. 1–10. [doi: 10.1109/SIES.2015.7185035]

72. Mahmud N, Seceleanu C, Ljungkrantz O. Specification and semantic analysis of embedded systems requirements: From description logic to temporal logic. In: Proc. of the 15th Int'l Conf. on Software Engineering and Formal Methods. Trento: Springer, 2017. 332–348. [doi: 10.1007/978-3-319-66197-1_21]

73. Welch LR, Ravindran B, Shirazi BA, Bruggeman C. Specification and modeling of dynamic, distributed real-time systems. In: Proc. of the 19th IEEE Real-time Systems Symp. Madrid: IEEE, 1998. 72–81. [doi: 10.1109/REAL.1998.739732]

74. Ebert C. Specifying, designing and rapid prototyping computer systems with structured Petri nets. In: Frey HH, ed. *Safety of Computer Control Systems 1992*. Amsterdam: Elsevier, 1992. 19–24. [doi: 10.1016/B978-0-08-041893-3.50008-2]

75. Damm W, Hungar H, Josko B, Peikenkamp T, Stierand I. Using contract-based component specifications for virtual

integration testing and architecture design. In: Proc. of the 2011 Design, Automation & Test in Europe. Grenoble: IEEE, 2011. 1–6. [doi: 10.1109/DATE.2011.5763167]

76. Camilli M, Gargantini A, Scandurra P. Zone-based formal specification and timing analysis of real-time self-adaptive systems. *Science of Computer Programming*, 2018, 159: 52–78. [doi: 10.1016/j.scico.2018.03.002]

77. Ben-Abdallah H, Lee I, Kim YS. The integrated specification and analysis of functional, temporal, and resource requirements. In: Proc. of the 3rd IEEE Int'l Symp. on Requirements Engineering. Annapolis: IEEE, 1997. 198–209. [doi: 10.1109/ISRE.1997.566870]

78. Doering D, Pereira CE, Denes P, Joseph J. A model driven engineering approach based on aspects for high speed scientific X-rays cameras. In: Proc. of the 16th IEEE Int'l Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2013). Paderborn: IEEE, 2013. 1–8. [doi: 10.1109/ISORC.2013.6913190]

79. Dutertre B, Stavridou V. Formal requirements analysis of an avionics control system. *IEEE Trans. on Software Engineering*, 1997, 23(5): 267–278. [doi: 10.1109/32.588520]

80. Faulk S, Brackett J, Ward P, Kirby J. The core method for real-time requirements. *IEEE Software*, 1992, 9(5): 22–33. [doi: 10.1109/52.1568714]

81. Ferrante O, Passerone R, Ferrari A, Mangeruca L, Sofronis C. BCL: A compositional contract language for embedded systems. In: Proc. of the 2014 IEEE Emerging Technology and Factory Automation (ETFA). Barcelona: IEEE, 2014. 1–6. [doi: 10.1109/ETFA.2014.7005353]

82. Pierce RH, Ayache S, Ward R, Stevens J, Clifton H, Galle J. Capturing and verifying performance requirements for hard real time systems. In: Proc. of the 1997 Int'l Conf. on Reliable Software Technologies. London: Springer, 1997. 137–148. [doi: 10.1007/3-540-63114-3_13]

83. Du G, Lin J. Real-time embedded software requirements description framework exploration. *Quality and Reliability*, 2008, (1): 47–50.

84. Goldsack SJ, Finkelstein ACW. Requirements engineering for real-time systems. *Software Engineering Journal*, 1991, 6(3): 101–115. [doi: 10.1049/sej.1991.0014]

85. Ravn AP, Rischel H, Hansen KM. Specifying and verifying requirements of real-time systems. *IEEE Trans. on Software Engineering*, 1993, 19(1): 41–55. [doi: 10.1109/32.1987.10057]

86. Ribeiro FGC, Misra S, Soares MS. Application of an extended SysML requirements diagram to model real-time control systems. In: Proc. of the 13th Int'l Conf. on Computational Science and Its Applications. Ho Chi Minh City: Springer, 2013. 70–81. [doi: 10.1007/978-3-642-39646-5_6]

87. Saiedian H, Kumarakulasingam P, Anan M. Scenario-based requirements analysis techniques for real-time software systems: A comparative evaluation. *Requirements Engineering*, 2005, 10(1): 22–33. [doi: 10.1007/s00766-004-0192-6]

88. Laouadi MA, Mokhati F, Seridi-Bouchelaghem H. A novel formal specification approach for real time multi-agent system functional requirements. In: Proc. of the 8th German Conf. on Multiagent System Technologies. Leipzig: Springer, 2010. 15–27. [doi: 10.1007/978-3-642-16178-0_4]

89. Siegl S, Hielscher KS, German R. Model based requirements analysis and testing of automotive systems with timed usage models. In: Proc. of the 18th IEEE Int'l Requirements Engineering Conf. Sydney: IEEE, 2010. 345–350. [doi: 10.1109/RE.2010.49]

90. Zhou JL, Lu Y, Lundqvist K, Lönn H, Karlsson D, Liwång B. Towards feature-oriented requirements validation for automotive systems. In: Proc. of the 22nd Int'l Requirements Engineering Conf. Karlskrona: IEEE, 2014. 428–436. [doi: 10.1109/RE.2014.6912294]

91. Zhu XN. A formal model for service-based behavior specification using stream-based I/O tables. In: Proc. of the 10th Int'l Workshop on Formal Aspects of Component Software. Nanchang: Springer, 2014. 369–383. [doi: 10.1007/978-3-319-07602-7_22]

92. Sinha R, Patil S, Pang C, Vyatkin V, Dowdeswell B. Requirements engineering of industrial automation systems: Adapting the cesar requirements meta model for safety-critical smart grid software. In: Proc. of the 41st Annual Conf. of the IEEE Industrial Electronics Society. Yokohama: IEEE, 2015. 002172–002177. [doi: 10.1109/IECON.2015.7392423]

93. Li R, Ma SL, Yao WT. Ontology-based requirements generation for credibility validation of safety-critical system. In: Proc. of the 2015 Int'l Conf. on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing. Liverpool: IEEE, 2015. 849–854. [doi: 10.1109/CIT/IUCC/DASC/PICOM.2015.126]

94. Ribeiro FGC, Pereira CE, Rettberg A, Soares MS. Model-based requirements specification of real-time systems with UML, sysML and MARTE. *Software & Systems Modeling*, 2018, 17(1): 343–361. [doi: 10.1007/s00766-016-0255-1]

95. Westman J, Nyberg M. Providing tool support for specifying safety-critical systems by enforcing syntactic contract conditions. *Requirements Engineering*, 2019, 24(2): 231–256. [doi: 10.1007/s00766-017-0286-6]

96. Blouin D, Giese H. Combining requirements, use case maps and AADL models for safety-critical systems design. In: Proc. of the 42th Euromicro Conf. on Software Engineering and Advanced Applications (SEAA). Limassol: IEEE, 2016. 266–274. [doi: 10.1109/SEAA.2016.15]

97. Fu RR, Bao XH, Zhao TD. Generic safety requirements description templates for the embedded software. In: Proc. of the 9th Int'l Conf. on Communication Software and Networks (ICCSN). Guangzhou: IEEE, 2017. 1477–1481. [doi: 10.1109/ICCSN.2017.8230353]

98. Chen XH, Han L, Liu J, Sun HY. Using safety requirement patterns to elicit requirements for railway interlocking systems. In: Proc. of the 24th Int'l Requirements Engineering Conf.

Workshops (REW). Beijing: IEEE, 2016. 296–303. [doi: 10.1109/REW.2016.55]

99. Hu ZM, Huang LT, Zhao YX. Hybrid modeling language for aerospace model software. *Aerospace Control and Application*, 2021, 47(2): 25–31 (in Chinese with English abstract). [doi: 10.3969/j.issn.1674-1579.2021.02.004]

100. Feyerabend K, Josko B. A visual formalism for real time requirement specifications. In: Proc. of the 4th Int'l AMAST Workshop on Aspects of Real-time Systems and Concurrent and Distributed Software. Palma: Springer, 1997. 156–168. [doi: 10.1007/3-540-63010-4_11]

101. Roop PS, Sowmya A. Hidden time model for specification and verification of embedded systems. In: Proc. of the 10th EUROMICRO Workshop on Real-time Systems. Berlin: IEEE, 1998. 98–105. [doi: 10.1109/EMWRTS.1998.7066607]

102. Khan AM, Mallet F, Rashid M. Natural interpretation of UML/MARTE diagrams for system requirements specification. In: Proc. of the 11th IEEE Symp. on Industrial Embedded Systems (SIES). Krakow: IEEE, 2016. 1–6. [doi: 10.1109/SIES.2016.7509429]

103. Kirner TG, Davis AM. Nonfunctional requirements of real-time systems. *Advances in Computers*, 1996, 42: 1–37. [doi: 10.1016/S0065-2458(08)60483-0]

104. Kirner TG, Davis AM. Requirements specification of real-time systems: Temporal parameters and timing constraints. *Information and Software Technology*, 1996, 38(12): 735–741. [doi: 10.1016/0950-5849(96)00104-4]

105. Martins LEG, Gorschek T. Requirements engineering for safety-critical systems: A systematic literature review. *Information and Software Technology*, 2016, 75: 71–89. [doi: 10.1016/j.infsof.2016.04.002]

106. Lee HK, Lee WJ, Chae HS, Kwon YR. Specification and analysis of timing requirements for real-time systems in the CBD approach. *Real-time Systems*, 2007, 36(1–2): 135–158. [doi: 10.1007/s11241-007-9017-2]

107. Wu X, Liu C, Xia QX. Safety requirements modeling based on RUCM. In: Proc. of the 2014 Computers, Communications and IT Applications Conf. Beijing: IEEE, 2014. 217–222. [doi: 10.1109/ComComAp.2014.7017199]

108. Han L, Liu J, Zhou TL, Sun JF, Chen XH. Safety requirements specification and verification for railway interlocking systems. In: Proc. of the 40th Annual Computer Software and Applications Conf. (COMPSAC). Atlanta: IEEE, 2016. 335–340. [doi: 10.1109/COMPSAC.2016.182]

109. Petters S, Muth A, Kolloch T, Hopfner T, Fischer F, Farber G. The REAR framework for emulation and analysis of embedded hard real-time systems. In: Proc. of the 10th IEEE Int'l Workshop on Rapid System Prototyping. Clearwater: IEEE, 1999. 100–107. [doi: 10.1109/IWRSP.1999.779038]

110. De Lemos R, Saeed A, Anderson T. Analysis of timeliness requirements in safety-critical systems. In: Proc. of the 1992 Int'l Symp. on Formal Techniques in Real-time and Fault-tolerant Systems. Berlin: Springer, 1992. 171–192. [doi: 10.5555/646842.706607]

111. Aoyama M, Yoshino A. AORE (aspect-oriented requirements engineering) methodology for automotive software product lines. In: Proc. of the 15th Asia-Pacific Software Engineering Conf. Beijing: IEEE, 2008. 203–210. [doi: 10.1109/APSEC.2008.59]

112. Frey HH, ed. Safety of Computer Control Systems 1992. Amsterdam: Elsevier, 1992.

113. De Lemos R, Saeed A, Anderson T. A train set as a case study for the requirements analysis of safety-critical systems. *The Computer Journal*, 1992, 35(1): 30–40. [doi: 10.1093/comjnl/35.1.30]

114. Tjell S, Fernandes JM. Expressing environment assumptions and real-time requirements for a distributed embedded system with shared variables. In: Proc. of the 2008 IFIP Working Conf. on Distributed and Parallel Embedded Systems. Milano: Springer, 2008. 79–88. [doi: 10.1007/978-0-387-09661-2_8]

115. Martins LEG, de Oliveira T. A case study using a protocol to derive safety functional requirements from fault tree analysis. In: Proc. of the 22nd Int'l Requirements Engineering Conf. (RE). Karlskrona: IEEE, 2014. 412–419. [doi: 10.1109/RE.2014.6912292]

116. Hansen KM, Ravn AP, Stavridou V. From safety analysis to software requirements. *IEEE Trans. on Software Engineering*, 1998, 24(7): 573–584. [doi: 10.1109/32.708570]

117. Markose S, Liu XQ, McMillin B. A systematic framework for structured object-oriented security requirements analysis in embedded systems. In: Proc. of the 2008 Int'l Conf. on Embedded and Ubiquitous Computing. Shanghai: IEEE, 2008. 75–81. [doi: 10.1109/EUC.2008.92]

118. Roudier Y, Idrees MS, Apvrille L. Towards the model-driven engineering of security requirements for embedded systems. In: Proc. of the 3rd Int'l Workshop on Model-driven Requirements Engineering (MoDRE). Rio de Janeiro: IEEE, 2013. 55–64. [doi: 10.1109/MoDRE.2013.6597264]

119. Saeed A, De Lemos R, Anderson T. A train set as a case study for the requirements analysis of safety-critical systems. In: Proc. of the 1992 Int'l Symp. on Formal Techniques in Real-time and Fault-tolerant Systems. Berlin: Springer, 1992. 171–192. [doi: 10.5555/646842.706607]

120. Sun JF, Feng JC, Zhu JY, et al. A dimensional analysis method for the requirements model of railway control software. *Chinese Journal of Computers*, 2020, 43(11): 2152–2165 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.02152]

121. Zhou Y, Huang YH, Zheng HY, et al. A dimensional analysis method for the requirements model of railway control software. *Chinese Journal of Computers*, 2020, 43(11): 2152–2165 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.02152]

122. Zhu JY, Huang YH., Zheng HY, et al. A dimensional analysis method for the requirements model of railway control software. *Chinese Journal of Computers*, 2020, 43(11): 2152–2165 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.02152]

123. Zhang WB, Xue L, Xu JM, et al. A dimensional analysis method for the requirements model of railway control software. *Chinese Journal of Computers*, 2020, 43(11): 2152–2165 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.02152]

124. Feng JC. Detailed requirement of embedded system oriented formal modeling and analysis [Ph.D. Thesis]. Shanghai: East China Normal University, 2022 (in Chinese with English abstract). [doi: 10.27149/d.cnki.ghdsu.2022.004041]

125. Darem, A., Alhashmi, A., Sheatah, H. K., Mohamed, I. B., Jabnoun, C., Allan, F. M., ... & Elmourssi, D. M. (2024). DECODING THE DECEPTION: A COMPREHENSIVE ANALYSIS OF CYBER SCAM VULNERABILITY FACTORS. Journal of Intelligent Systems and Applied Data Science, 2(1).

125. Wang S, Feng JC, Zhu JY, Huang YH., Zheng HY, Xu XR, Miao WK, Zhang X, Pu GG. A dimensional analysis method for the requirements model of railway control software. *Chinese Journal of Computers*, 2020, 43(11): 2152–2165 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.02152]

126. Liu SY, Offutt J, Ho-Stuart C, Sun Y, Ohba M. SOFL: A formal engineering methodology for industrial applications. *IEEE Trans. on Software Engineering*, 1998, 24(1): 24–45. [doi: 10.1109/32.663996]

127. Liu SY, Asuka M, Komaya K, Nakamura Y. An approach to specifying and verifying safety-critical systems with practical formal method SOFL. In: Proc. of the 4th IEEE Int'l Conf. on Engineering of Complex Computer Systems. Monterey: IEEE, 1998. 100–114. [doi: 10.1109/ICECCS.1998.706660]

128. Chen XH, Liu QQ, Mallet F, Li Q, Cai SB, Jin Z. Formally verifying consistency of sequence diagrams for safety critical systems. *Science of Computer Programming*, 2022, 216: 102777. [doi: 10.1016/j.scico.2022.102777]

129. Chen XH, Mallet F. Modeling timing requirements in problem frames using CCSL. In: Proc. of the 18th Asia-Pacific Software Engineering Conf. Ho Chi Minh City: IEEE, 2011. 381–388. [doi: 10.1109/APSEC.2011.30]

130. Vilela MA, Castro J, Martins LEG, Gorschek T. Integration between requirements engineering and safety analysis: A systematic literature review. *Journal of Systems and Software*, 2017, 125: 68–92. [doi: 10.1016/j.jss.2016.11.031]

131. Medikonda BS, Panchumarthy SR. A framework for software safety in safety-critical systems. *ACM SIGSOFT Software Engineering Notes*, 2009, 34(2): 1–9. [doi: 10.1145/150786.150788]

132. Selić B, Gérard S. *Modeling and Analysis of Real-time and Embedded Systems with UML and MARTE: Developing Cyber-physical Systems*. Amsterdam: Elsevier, 2014. [doi: 10.1016/C2012-0-13536-5]

Ali, W. A., Mangini, A. M., Júlvez, J., Mahulea, C., & Fanti, M. P. (2024). Toward Enhancing Security in Intelligent Transportation: A Simulation-Based Approach. IFAC-PapersOnLine, 58(4), 156-161.

134. Gu B, Yu B, Dong XG, Li XF, Zhong RM, Yang MF. Intelligent program synthesis techniques: Literature review. *Ruan Jian Xue Bao/Journal of Software*, 2021, 32(5): 1373–1384 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6200.htm [doi: 10.13328/j.cnki.jos.006200]

135. Yang MF, Gu B, Duan ZH, Jin Z, Zhan NJ, Dong YW, Tian C, Li G, Dong XG, Li XF. Intelligent program synthesis framework and key scientific problems for embedded software. *Chinese Space Science and Technology*, 2022, 42(4): 1–7 (in Chinese with English abstract). [doi: 10.16708/j.cnki.1000-758X.2022.0046

136. Tsai JJP, Liu AL. Experience on knowledge-based software engineering: A logic-based requirements language and its industrial applications. *Journal of Systems and Software*, 2009, 82(10): 1578–1587. [doi: 10.1016/j.jss.2009.03.019]

137. ÇAM, A. S., & YILDIZ, F. (2023). Privacy Threats Unveiled: A Comprehensive Analysis of Membership Inference Attacks on Machine Learning Models and Defense Strategies. Journal of Intelligent Systems and Applied Data Science, 1(2).

138. Wang S, Feng JC, Zhu JY, Huang YH., Zheng HY, Xu XR, Miao WK, Zhang X, Pu GG. A dimensional analysis method for the requirements model of railway control software. *Chinese Journal of Computers*, 2020, 43(11): 2152–2165 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.02152]