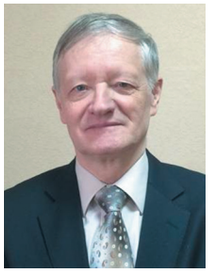


Kolchin A.F., Mikheev N.V.

ARCHITECTURE OF SAFETY RELATED SYSTEM SOFTWARE

The paper presents the results of analysis of architectures for safety related system software. The basic components of architecture are defined, and the definition of safety related system software architecture is offered. Quality criteria are proposed, and comparison of defined components by the criteria is carried out. The paper offers results of comparison of applicable structures and styles that are proposed for application in software development to satisfy to the functional safety requirements specified in GOST R IEC 61508.

Keywords: *dependability, functional safety, architecture, software.*

Introduction

The modern trend to place stricter safety requirements for hazardous technical systems has led to application of not only sufficient but also indeed necessary safety measures. However it requires some quantitative estimation of applied safety measures, which is possible and necessary to specify prior to the development of such systems.

For estimation of safety related system hardware, one generally uses statistical methods which allow us to obtain quantative estimation of hardware dependability. Similar methods are inapplicable for software, as errors arising in software during its development and operation are systematic.

One of the approaches integrating methods of dependability estimation both for hardware and software is the methodology of functional safety presented in GOST R IEC 61508 [1]. The given standard covers the development of relatively simple systems ensuring the performance, as a rule, of one function, i.e. safety function, and referring to as "safety related systems."

One of the key stages in software development for safety related systems is the development of software architecture. However, comprehensive guides on development of software architecture are not available at present. Existing studies consider either particular subtypes of architecture, or general methods of designing. One of the reasons hampering knowledge systematization in the area is absence of the generally accepted definition of software architecture.

This paper presents the results of analysis of architectures for safety related system software. The basic components of architecture are defined, and the definition of safety related system software architecture is offered. Quality criteria are proposed, and comparison of defined components by the criteria is carried out. The paper offers results of comparison of applicable structures and styles that are proposed for application in software development to satisfy to the functional safety requirements.

1. Safety related system

1.1. System structure

Sufficiently long experience of development and application of safety related systems in various areas allow us to assert that the structure of such system has already

Table 1. Advantages and disadvantages of program structures

Structure	Advantages	Disadvantages
Simple control loop	Simplicity	The worse time of reaction is equal to the sum of execution times of all subtasks. When adding new devices or supplementary processing, the worse time of reaction increases
Interrupt controlled system	Allows essentially increasing the time of reaction for high-priority subtasks in comparison with a simple control loop. It is development of idea of simple control loop, but it still remains sufficiently simple	The problem of split memory, as when there is an interruption, the basic stream can stop in the middle of calculations while the code carried out on interruption can change the given memory
Cooperative multitasking	There is no necessity to protect all shared data structures. Simplifies process of one-thread code transition into multithreaded environment. Simple enough expansion of functionality (by subroutine addition in execution queue)	In case of failure of a single thread, all others also fail as the operation "to give processor time" is not called. High complexity of multithreaded input-output
Preemptive multitasking / RTOS	The possibility of adequate realization of multithreaded input-output. The possibility of multiprocessing systems use. A single failed program does not affect other programs. In case of absence of shared memory use, a programmer can develop the software as one- thread program – the whole operation is executed by an operating system	Overhead resources for execution tasks by operating system itself. Strong complication of a system as a whole. A problem of shared memory

become well-established and consists of the following elements:

1. Programmable electronics:

а) controller, which is executing the logic and realizing the interaction between components;

б) software executing the logic.

2. Input /output devices and auxiliary means:

а) sensors and other input devices;

б) executive devices and other output devices;

в) power supplies, communication medium, etc.

1.2. System requirements

According to the life cycle model of a safety related system [1], by the start of the development process of its software architecture, a great volume of documents describing the controlled equipment, safety functions and various requirements for software system identified at previous stages is developed. Many requirements are typical for the majority of developed safety related systems. The analysis of requirements for the structure described above has allowed us to identify the following requirements and restrictions for a safety related system essential for software:

1. Strictly specified maximal time of system response.

2. Software under development is intended for execution on programmable controllers with limited resources.

3. Software is strongly connected with lower level tasks (for example, input/output), therefore it is strictly tied to the operating system (or even implements its functions).

4. Absence of necessity for storage of great volumes of data.

5. Absence of a complex business logic, as in most cases it is strictly not recommended to implement non-safety related functions on a controller[1].

6. Interaction with a user should be minimal, operation is basically autonomous.

1.3. Requirements to software architecture

It is possible to distinguish three groups of software architecture requirements [1]:

1. Requirements of safety integrity level (SIL) for software architecture and designing tools.

2. General requirements stipulated by a safety related system. These requirements can influence architecture directly or indirectly.

3. Special requirements for software stipulated by hardware architecture.

The first group includes various requirements, for example, for documentary registration of process, traceability, etc. The given group also includes methods of architecture development required for specific application depending on a safety integrity level.

The second group includes restrictions for resources, sizes, interfaces of input-output, etc., which depend on a specific task. However, it is possible to state that the given requirements basically limit architecture resource demand [2].

The source of the third group can be a type of development, for example, if it is necessary to build a system

[1] In [1] there are no restrictions as for the size or complexity of programs, however with increase of requirements for system fault tolerance, methods of its SIL ensuring become more complicated, therefore it is recommended to realize more simple (or to realize in part) safety functions on a separate safety related system.

Table 2. Comparison of considered program structures by quality criteria

	An opportunity of subroutines priority settings	Independence of the worse time of reaction from the number of subroutines	Independence of the worse time of reaction from a subroutine change	General simplicity of system implementation	Simplicity of subroutines' realization	Simplicity of subroutines' addition	Absence of the overhead expenses which are not connected to the basic logic	An opportunity of single-thread code writing	Independence of subroutines
Simple control loop	-	-	-	+	+	+	+	+	-
Interrupt controlled system	-	-	-	+	-	+	+	-	-
Cooperative multitasking	It is set by the programmer	-	-	-	-	+	-	-	-
Preemptive multitasking / RTOS	+	+	+	-	+	+	-	+	+
Designations: "+" – criterion is applied; "-" – criterion is not applied.									

Table 3. Advantages and disadvantages of style architectures

Architecture	Advantages	Disadvantages
Conveyors and filters	The possibility of representation of the program whole behavior as a simple sequence of separate filters. A reuse of filters. Simple addition of new functionality by adding a filter to a processing queue. It is possible to carrying out special checks, such as the analysis of mutual blocking or filter capacity. Potentially simple multisequencing of a code	The organization of batch processing is frequently required. In view of filters' independence requirement, the designer should believe that the data are completely processed by each filter. In particular, it can be demanded each time to reduce data to a common view and to assort them separately in each filter
Data abstraction and object-oriented organization	The possibility of implementation change without consequences for clients. Combination of data and functions which process them allows designers to decompose the task down to a set of entities cooperating among themselves	In view of specificity of the procedural calls, the calling object should have explicit access to the called object, as opposed to conveyors or to event-trigger system. As a consequence, when changing an object identifier, it is required to explicitly notify all calling objects. Presence of outside effects is possible (for example, if an object A uses an object B and an C also uses the object B, then the changing of the object B by the object C look as a outside effect for the object A and vice-versa)
Event-trigger system, implicit call	Ample possibilities for system reuse: the system can be extended by a new component by simple registration as events' handler. The implicit call simplifies system development: any component can be altered or replaced without influence on other components	The control over executed calculations belongs not to software components but to a system. There is no guarantee of reaction to an event. As consequence of the previous statement, confirmation of reaction to the event should be made explicitly. Complex procedures of mass data exchange
Level-sensitive (layer-wise) system	Easy escalating abstraction. Ample opportunities of reuse, similarly to abstract types of data. The opportunity of realization of separate levels in different ways by the declaration of interfaces of interaction between levels	Not all tasks can be decomposed simply enough down to a level structure. It is difficult enough to define a suitable level of abstraction. Presence of an additional overhead charge in view of translation of abstraction from one level to another

Table 4. Comparison of architecture styles

	Simplicity	Supportability	Design reuse	Efficiency	Scalability	Portability
Conveyors and filters	+	+ It is easy to replace a filter	+ Allows achieving different effects due to change of the order	- There is no interrelation between filters (it is impossible to transfer control from one filter to another	±	- Not portable
Data abstraction and object-oriented organization	±	+ Set of principles, allowing increasing abstraction (encapsulation, inversion of control and so on)	+	±	±	±
Event-trigger system, implicit call	- Sometimes the behavior is unpredictable, it is difficult to control	+ It is possible to replace or remove a component without influence on others	+ Components can be registered in a system for processing any events	- Components have no possibility to control calculations	- Outside effects can arise if two components use the third one	+ Components can react to any events
Level-sensitive (layerwise) system	±	+ Change of one level influences only two neighboring levels	+ Do not depend on overlying levels	±	+ Can be incorporated with other styles	
Designations: "+"– criterion is applied; "-" – criterion is not applied; "±"– criterion is applied partially.						

under development above the available base provided by a controller manufacturer. Besides, it can be connected to features of hardware when a chosen controller implements special technology optimized for the certain architecture of software.

Software is usually developed using three different ways [3]:

a) Development of new software based on a specification.

b) Software development for integration into an existing platform. This process is focused not on software designing and development, but rather on mapping required functions on the given program framework.

c) Existing software improvement. The given process is not considered in this study as actions on improvement substantially depend on an updated system.

Both alternatives (a) and (b) are acceptable for development of software for a new system, as integration into an existing platform is considered as reuse of already existing software, rather than its improvement.

2. Designing of software architecture

2.1. Definition of software architecture

Among developers of various systems, there is a general understanding of the importance the architectural level of a system under development. However, at present there is none established consent on the exact definition of a system architecture.

The analysis carried out so far has shown that software architecture of safety related systems can be presented and described with the help of the following three invariant constituents (which are named sometimes as *architecture patterns*):

1. *Program structure* [4] describes ways of organization of base functions, such as memory management, flow control, etc.

2. *Architecture style* [5] describes the method of logic division of a system, which is realized on the basis of system structure.

3. *Task solution*, described in terms of program structure and architecture style.

Thus, in this paper as architecture of safety related system software, we shall understand a set of conceptions and descriptions of:

program structure;

architecture style;

executed safety function in terms of program structure and architecture style.

2.2. Program structure

Safety related systems are in fact a subclass of embedded systems, and therefore, versions of program structure organization are rather similar to the versions used in common embedded systems. For implementation of safety

Table 5. Possible combinations of architecture structure and style

	Simple control loop	Interrupt controlled system	Cooperative multitasking	Preemptive multitasking/ RTOS
Conveyors and filters	There are developments showing the possibility to effectively process data in such a configuration, however the basic scope of the given combination of technologies is a network traffic control [7]	Generally such combination is not applied	Conveyors and filters can be considered as a cooperative multitasking alternative	A combination is possible
Data abstraction and object-oriented organization	A combination is possible	A combination is possible	A combination is possible	A combination is possible
Event-trigger system, implicit call				As a matter of fact, RTOS (Real-Time Operating System) is the operating system based on events. At least, it is one of the basic tools, allowing prioritizing separate processes. Also, transfer of event is the simplest way of interaction between streams.
Level-sensitive (layer-wise) system	A combination is possible	A combination is possible	A combination is possible	A combination is possible

related system software, it is possible to use the following program structures [6]:

1. Simple control loop.
2. Interrupt controlled system.
3. Cooperative multitasking.
4. Preemptive multitasking or multi-threading.
5. Other versions of real time operating systems.

Table 1 shows the advantages and disadvantages of the listed structures at development of safety related systems identified as a result of the analysis, which allowed us to define the following criteria of comparison of considered structures.

1. Possibility of setting of subprograms priorities.
2. Independence of the worst time of reaction:
а) from the number of subprograms;
б) from subprogram change.
3. General simplicity of system realization.
4. Simplicity of subprogram realization.
5. Simplicity of subprogram addition.
6. Absence of overhead expenses not related to the basic logic of the task under solution.
7. Opportunity of one-thread code writing.
8. Independence of subprogram s.

Table 2 presents comparison of the considered structures by defined criteria.

2.3. Architecture style

There is a good deal of various architectural designs frequently named as architecture patterns, which represent solutions within the framework of some repeating context. At present there is not any common list of similar patterns, or even a common opinion concerning their abstraction, however within the framework of the problem considered in the present study it is possible to identify the following set of architecture patterns[1] (or styles) [3]:

1. Pipes and filters.
2. Data Abstraction and Object-Oriented Organization.
3. Event-based, Implicit Invocation.
4. Layered system.

Table 3 considers the advantages and disadvantages of the listed program styles at development of safety related systems.

The following criteria of quality [7] have been used in this study for comparison of architecture styles:

1. Supportability – a degree of simplicity of change introduction (addition of new handlers, removal of old ones, updating existing ones, etc.).
2. Repeated usability – a degree of applicability of existing program constituents for creation of new ones.
3. Efficiency.
4. Simplicity – system understandability for new developers, possibility of quickly understanding existing modules.
5. Scalability.
6. Portability – absence of binding to certain tools (to the development environment, operating system, compiler, etc.).

Considered styles presented in Table 4 are compared by the listed criteria.

Undoubtedly, not all styles can be applied at implementation of the considered programs' structures. Table 5 shows

[1] Certainly, both the combinations of mentioned architectures and completely new ones are possible.

all possible combinations of the structures and program styles considered above. Application of the given table will allow enhancing correctness and efficiency of development of safety related system software, when selecting software architecture.

2.4. The task solution

As follows from the definition, when shaping architecture, the high-level representation of safety functions is used, and not its specification generated at the corresponding stage of safety related system life cycle. Necessary detailed elaboration is selected based on requirements for the safety related system under development. In particular, for software architecture the following things should be described:

1. The basic system components, their interfaces, arrangement and methods of interaction with each other.

The architecture should reflect a high-level representation of system (subsystem) components, the most important ones from the point of view of the majority of software development participants as they have to be taken into account during development of the majority of other system components. Some subsystems are required for operation of the majority of components (for example, input-output subsystem encapsulating a physical level of interaction with communication channels). Other subsystems are sources of events for miscellaneous components (for example, a watchdog timer). The third components can be the most resource-intensive and/or critical for performance of the system basic function – safety function.

2. Key patterns of designing and technologies used in the project.

As certain patterns can influence the program as a whole (for example, connections pool), they have to be described at the architecture level. Including the patterns which are allowable to use at lower levels.

3. Interfaces of interaction with external systems.

4 The services providing support for basic functional operation (journalizing, data storage, etc.).

Conclusion

The paper presents the results of analysis of software architectures and their development for safety related systems. Also, the basic components of architecture are defined.

The definition of software architecture for safety related is offered.

The paper offers results of comparison of applicable structures and styles that are proposed for application in software development to satisfy to the functional safety requirements [1].

The application of the considered approach will allow enhancing the correctness and efficiency of software development for safety related systems at the stage of software architecture selection.

References

1. GOST R IEC 61508-2012 Functional safety of electrical, electronic and programmable electronic safety related systems. Parts 1 – 7.

2. **Philip J. Koopman Jr.,** Design constraints on embedded real time control systems. Systems Design & Networks Conference, (1990).

3. **Douglas Densmore, Roberto Passerone, Alberto Sangiovanni-Vincentelli**, A Platform-Based Taxonomy for ESL Design. IEEE Software.

4. **David Stonier-Gibson**, Understanding embedded microcontroller multitasking RTOS alternatives.

5. **David Garlan, Mary Shaw**, An Introduction to Software Architecture.

6. http://en.wikipedia.org/wiki/Embedded_system#Embedded_software_architectures.

7. **Mary Shaw** (1995), Comparing Architectural Design Styles. IEEE Software.