Expressibility Required to Design Embedded and Real-time Systems

Shourong Lu¹, Qichun Jin²

¹ School of Mechanical & Electrical Engineering, Wuxi City College of Vocational Technology, 12 Qianou Road, Huishan District, Wuxi City, Jiangsu Province, 214153, P.R.China
² E-mail: ¹shouronglu@hotmail.com, ²694146000@qq.com

Abstract. Embedded and real-time systems are characterised by safety criticality, reliability, real-time, limited resources, low-cost and re-usability, etc. Development of such systems is a complex engineering process, involving many different disciplines with respect to their demanding requirements, in order to achieve desired system properties and reduce the complexity of the development process, we compile these requirements and criteria for design, specification and modelling of embedded real-time systems, it is benefit for rapid development of high quality systems.

1. Introduction

Embedded and real-time systems are widely used in vehicle systems, consumer electronics, industrial embedded applications and national defense. These systematic compositions generally contain several microcomputers as a part of a larger system and interact directly with external devices. The processing software is typically constructed of concurrently running programs (tasks), which respond to a series of external inputs, take appropriate decisions, and also generate output necessary to control the peripherals connected to them. Development of such systems is a complex engineering process, involving many different disciplines with respect to their demanding requirements which are dealing with timing constrains, safety, dependability, limited resources, cost, re-usability and so on. I.e., in developing such systems requires not only system functionality but also system non-functional requirements and quality properties which appear in system requirements description.

2. Requirements for design Embedded and Real-time System

System requirement is a mandatory characteristic (behavior or attribute) for a system, subsystem, software application, or component. It is expressed as a structured document setting out detailed descriptions of the system services. As shown in Fig.1, it comprises of stakeholder requirements, main mission requirements, quality requirements, subsystem requirements, software and hardware requirements, functional and non-functional requirements. Quality requirements specify a mandatory amount of a type of system quality, such as safety; Functional requirements capture a system's intended execution behaviour. Non-functional requirements are expressed on various attributes of these services, tasks, or functions. Stakeholder requirements deal with customer or clients, developers and users requirements.
3. Quality Requirements Specification

Quality requirement deals with safety-related requirements, dependability, maintainability, verifiability, flexibility and so on. The most important one is a safety-related requirement which is a requirement that has significant safety ramifications. As show in Fig.2, safety-related requirements can be specified with Safety-independent, safety-significant, safety, and safety constraints. Safety significant requirements specify mandatory minimum safety levels in terms of pairs of sub-factors of the safety quality factor; Safety requirements are enforced by a series of activities, and safety constraints are constraints intended to ensure a minimum level of safety integrity. Safety integrity levels (SIL) are categories of requirements based on their associated safety risk level. The 4 Safety Integrity Levels play an important role in IEC 61508[4], which give a certain form to safety requirements and their implication on the development of software. For each SIL, the appendices of the standard's Part 3 state a number of highly recommended development practices, techniques, and measures.

4. Functional Requirements Specification

An embedded real-time system is typically constructed of a number of resources (e.g., processors) and a number of concurrent tasks. They are designed to fulfill a number of timing constraints. If a resource is shared by a number of concurrent tasks, a scheduler is needed to arbitrate the access to the resource according to a scheduling policy [1].

Task: A task is a simple program, and each task corresponds to a certain part of the application. In actual operation, the task is divided into many subtasks, each of which is given a certain priority, and each task is in the state of running, waiting (suspended), ready and interruption. A task must satisfy a number of temporal constraints, the most important one of which is that it needs to meet a deadline, i.e., a point in time by which the task has to be completed.

Scheduler: The process of determining the order of task execution is known as scheduling [2][5]. It enables designers to predict the execution behaviour of embedded and real-time systems by ensuring that all tasks fulfill their execution requirements and meet their deadlines. A number of scheduling techniques exists, and each one is developed for a particular task model or environment. For instance, periodic scheduling takes into account periodic tasks, and is based on the concept of priority. A processor is allocated to the process with the highest priority, but tasks can be pre-empted by higher-priority tasks. There are Static scheduling techniques and Dynamic scheduling techniques.

Concurrency: In an embedded real-time system, multiple simultaneous task executions may occur, it is necessary to describe the strategies and mechanisms for co-ordinating the concurrent task activities. The dependencies existing for co-operation (e.g., synchronisation) of and competition (e.g., for shared resources) between tasks should be clearly described to enable schedulability analyses.

In addition to describe above, explicit specification of static architecture and dynamic behavior has also important consequences for the dependability of a system, and it is useful to reason about the behavior of their composition and their non-functional properties.
5. Non-Functional Requirements Specifications
The properties of an embedded and real-time system have strong implications on the non-functional requirements. The majority of the non-functional requirements are related to non-functional characteristics, for instance, strict timing constraints, resource consumption, fault-tolerance as well as application-specific design [6][8].

Timing constraints: Timing constraints are expressed explicitly as temporal constraints. The most common of temporal constraints is the task deadlines that must be observed. In addition, absolute instants and relative validity intervals are also temporal constraints.

Resource consumption: Many embedded systems have strict requirements for the use of resources, such as low power, controlled memory space, or execution time. The tasks specification should include resource requirements including dynamic resource consumption per service. There are three types of resources, i.e., computation resources, communication resources, and storages.

Fault tolerance: Fault tolerance is a means to achieve dependability, working under the assumption that a system contains faults (e.g., made by humans while developing or using systems, or caused by aging hardware), and aiming to provide specified services in spite of faults being present. Faults can be distinguished with permanent, intermittent, or transient. It is necessary to use fault-tolerant techniques to design hardware and software systems.

Re-usability: To reduce development cost and time, the re-use of existing solutions is indispensable. Therefore, also software is to be constructed from re-usable components. The realisation methodology should be composition with respect to functional and nonfunctional aspects of components, which also deals with predicting quality attributes based on resource consumption.

Portability: Portability is considered to be very important, mainly due to the fact that it is desired to keep hardware upgrading costs at an absolute minimum. Platform-independent design is useful to minimize the number of design decisions and of dependencies to system-specific resources necessary to construct executable systems.

6. QoS Criteria for Embedded and Real-time Systems
A number of quality-of-service criteria for embedded and real-time system design are identified as shown in Table 1: these criteria are divided into qualitative exclusive criteria, qualitative gradual criteria and quantitative criteria. Qualitative exclusive criteria are either fulfilled or not, qualitative gradual criteria are property that cannot be quantified and quantitative criteria are measurable numbers [3][7].

| Qualitative exclusive criteria | Qualitative gradual criteria | Quantitative criteria |
|-------------------------------|-----------------------------|----------------------|
| 1. Functional correctness      | 1. Safety,                  | 1. Worst-case response times |
| 2. Timeliness                 | 2. Dependability            | 2. Deadline           |
| 3. Safety licensability        | 3. Predictability,          | 3. Capacity reserves  |
| 4. Physical constraints met   | 4. Complexity,              | 4. Overall project costs |
|                               | 5. Robustness,              |                      |
|                               | 6. Fault tolerance,         |                      |
|                               | 7. Portability,             |                      |
|                               | 8. Flexibility              |                      |

Qualitative exclusive criteria: Functional correctness is difficult to prove it formally. It is generally checked by “black-box” or “grey-box” according its specification. Timeliness is quite important for real-time applications to be correct. Safety licensability means that a system can be certified with respect to safety. Various standards have been introduced during the development processes, such as IEC 61508 compliant with the requirements of a certain Safety Integrity Level (SIL). About safety licensability can only be reasoned on the basis of the standards applied during system design and development. All physical constraints relate to the hardware used, the memory,
power, the software application structure, and so on, must be met, because embedded real-time systems can only function properly if they make use of resources in an adequate way.

**Qualitative gradual criteria:** Safety is closely related to reliability and stability. Expressing safety in a gradual way, e.g., by SILs, offers a kind of assurance of the unharmed functioning of systems. Dependability comprises availability, reliability, safety, confidentiality, integrity, and maintainability [6]. These properties are dealing with separately according to different applications. The degree of dependability is always considered with regard to the specifications. Robustness refers to the degree of resilience of a system under pressure or when it encounters an invalid input or changes in internal structure or external environment. Fault tolerance means being able to provide the right service in the event of a failure. Flexibility represents a measure of how suitable a system is to operate in a different environment without changing it. The highest degree of flexibility has applications that consider portability and adaptability during the design process. However, it cannot be quantified.

**Quantitative criteria:** The timing constraints imposed on a task are, e.g., deadline and period/minimum inter-arrival time. The timing properties are, e.g., worst-case execution time. The execution of tasks must respect maximum delays; no task should finish after its absolute deadline. The absolute deadline of a task is the instant where its execution must be finished. Violating a deadline can entail a timing failure. The worst-case execution time is the longest time the execution of a task may take. The longest latency allowed for the completion of a task is from its nominal starting point to its deadline. Both timing constraints and properties can be defined as static task parameters, which describe characteristics applying independent of other tasks. The dynamic parameters of a task can be defined by describing effects that occur during its execution, for example, start time, completion time, and response time.

7. Conclusion

The properties of an embedded and real-time system have strong implications on the requirements. According to characteristics of real-time and embedded system the specifications of quality requirements, of functional requirements and of non-functional requirements as well as QoS criteria are discussed above. In order to achieve desired properties of embedded and real-time systems, the presented these requirements and QoS criteria should be captured during system design, specification and modelling.

References

[1] Burns, A., Wellings, A. (1994) HRT-HOOD: A Structured Design Method for Hard Real-Time Systems. Real-Time Systems, 6, 73–114.

[2] Burns, A., Wellings, A., (1997) Real-Time Systems and Programming Languages. International Computer Science Series, Addison-Wesley.

[3] Halang, W.A., Gumzej, R., Colnarić, M., and Družovec, M. (2000) Measuring the Performance of Real-Time Systems. Real-Time Systems, 18, 59–68.

[4] IEC 61508:2000) Functional Safety of Electrical/Electronic/Programmable Electronic Safetyrelated Systems. International Electrotechnical Commission, 2000.

[5] Liu, C.L., Layland, J.W. (1973) Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment. J. ACM, 20(1):46–61.

[6] Laprie, J.C. (1995) Dependability Attributes, Impairments and Means. In: Predictably Dependable Computing Systems, B. Randell, J.C. Laprie, H. Kopetz and B. Littlewood (eds.), pp. 3–18. Springer-Verlag.

[7] Lu, S-R. (2009) Engineering of Safety-related and Embedded Real-time Systems, Dissertation, ISBN: 978-3-89863-241-6, GCA-Verlag, Waabs.

[8] Lee, P.A., Anderson, T. (1990) Fault Tolerance — Principles and Practice. In: Dependable Computing and Fault-Tolerant Systems, 2nd ed. Springer-Verlag.