1

From the System to the Software

1.1. Introduction

The automation of numerous command systems (in railways, the aeronautics, automotive, nuclear industries, etc.) and/or process control systems (production, etc.), and the replacement of logical or analog systems involving little interaction by highly-integrated systems, have led to a considerable expansion of the domain of functional safety, taking account of the features and peculiarities of computer systems.

Dependability relates to applications for which it is crucial to ensure a continuous good level of service (reliability), because human lives are at stake (transport, nuclear energy, etc.), because of the high level of investment which would be lost were the calculation to go wrong (space, chemical production process, etc.), or indeed because of the cost of the problems that could be caused by failure (e.g. in the banking process, reliability of the transport network, etc.). It should be noted that for several years, account has been taken of the environmental impacts (e.g. with accidental spills of chemical products into the environment, impact on ecosystems, recycling, etc.).

Since the very beginning of research into such systems, the problems linked to validation of those systems have been at the heart of designers' concerns: it is useful to prove the mechanisms to react to the occurrence of failures are well designed, to check that design (by means of simulations, tests, evidence, etc.) and convincingly estimate projected, meaningful values measuring the performances of the functional safety devices.

The difficulty then lies in accurately identifying the various actors involved in the process (users, operators, managers, maintenance personnel, service providers, assessors, authorities, etc.), the different elements in the system, the interactions between those elements, the interactions with the users and the factors which have an impact on the operational safety, ultimately with identification of the electronic and/or programmable elements.

The aim of this first chapter is to offer an examination of the software in the context in which it is used, which is a system, and recap on the links and the constraints which need to be taken into account in creating software.

1.2. Command/control system

Figure 1.1[1] shows an example of a railway system. The Operation Control Center (OCC – photo *a*) controls the whole of the line and passes operational commands to the trains and to the signaling management system (photo *c* shows a manual operation control center).

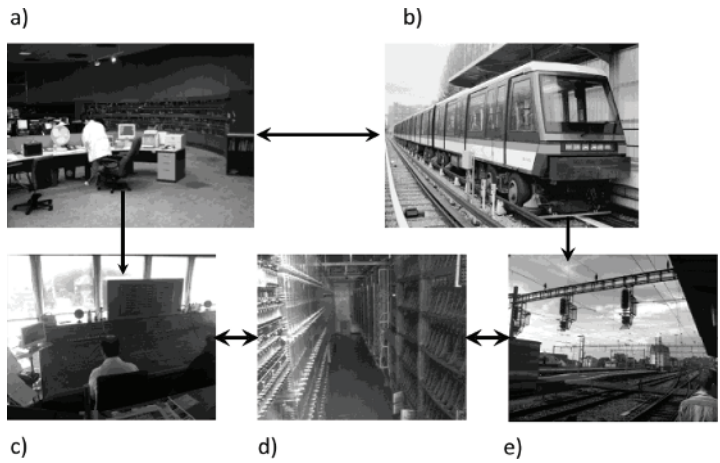

Figure 1.1. *The system in its environment*[2]

1 The picture shows an old-generation operating control center (OCC); new OCCs are stored in PCs and have developed from a physical technology (TCO – optical control view) to display by a video projector.
2 Photos taken by Jean-Louis Boulanger.

The operation control center[3] sends commands to the ground via a set of relays (photo *d* shows an example of a room containing the relays linked to the signaling system). In response to the commands, the ground equipment adopts the desired behavior (in photo *e*, we can see maneuver signals).

Figure 1.1 demonstrates the complexity associated with the concrete system, and highlights the point that a complex system is based not on one piece of software, but on many. Each of these software programs is associated with safety objectives which likely differ from one program to another.

The software involved in supervision does not have as much impact on people's safety as does the software relating to automated control of the trains. For this reason, in the context of systems requiring certification (aeronautics, railways, the nuclear sector, systems based on programmable electronics, etc.), we assign a given level of safety to each software application.[4]

This level of safety is associated with a scale, ranging from "non-critical" to "highly critical". The concept of safety assurance levels and the scales associated therein will be presented in Chapters 2 and 3.

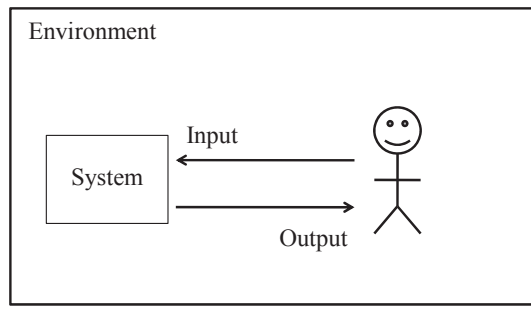

Figure 1.2. *The system in its environment*

Figure 1.2 highlights the fact that the system being constructed is closely linked with an environment which responds to the commands issued by the

3 Figure 1.1 shows a manual operating control center. However, these have now become computerized, and are referred to as PMIs ([BOU 10a – Chapter 5]); PIPCs and PAINGs ([BOU 10a – Chapter 4]).

4 For instance, in the field of aeronautics, the level of safety is called the *Design Assurance Level*; in railways, we speak of the Safety Integrity Level (SIL); and in the automobile sector we have the Automotive Safety Integrity Level (ASIL).

system. It is therefore necessary to acquire a view of the state of the process to be controlled and to have a means of command which is capable of relaying the commands to the environment. The environment may be composed of physical elements, but as a general rule, there are interactions with human parties (operators, users, maintenance personnel, etc.).

During the requirements analysis phase, it is essential to clearly identify all the actors (operators, maintenance personnel, customers, etc.) and identify all the devices which interact with the system. The requirements analysis phase is essential, but can still give rise to numerous omissions and misunderstandings.

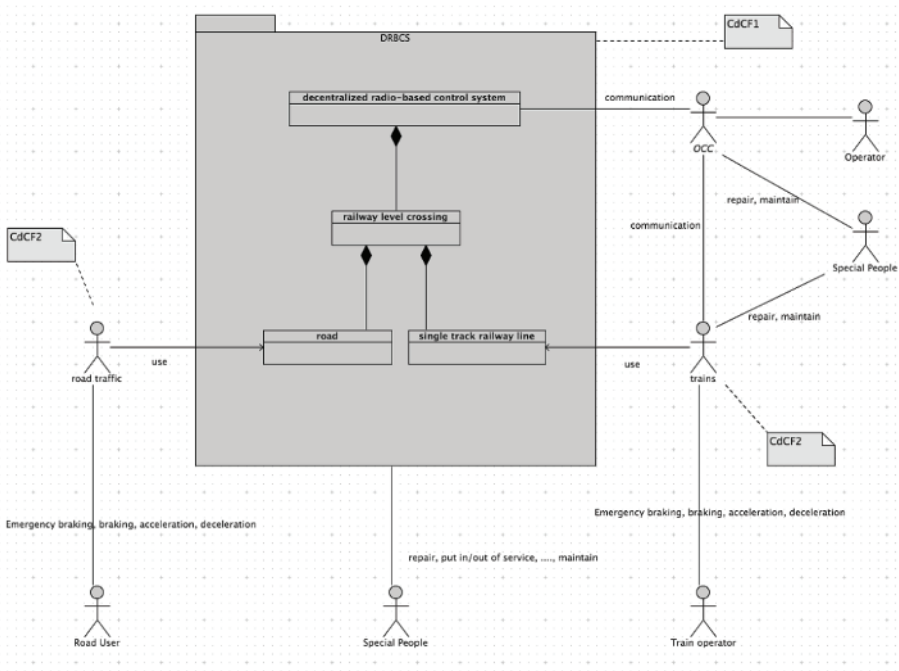

Figure 1.3. *Example of modeling of the system in its environment*

Figure 1.3 presents an example of the modeling of a system to control a level crossing. This system can control the intersection of at least one road with a railway track. This system interacts with various actors (both human

and machine): an OCC (as shown in the Figure 1.1), the road users (trucks, cars, etc.), railway users and operators in charge of operation and/or maintenance.

We have chosen to construct a class diagram which models the fact that the decentralized level-crossing management system using a communication system (DRBCS) comprises a level crossing which is itself made up of a railway and a roadway.

The important point in Figure 1.3 lies in identifying the actors which interact with the management system, including the road users, the trains, the OCC and especially the maintenance operators or other personnel (whom the model identifies as "special people").

It is crucial to identify all the actors involved at system level; otherwise there is a risk of forgetting actions – e.g. maintenance activities – but it is also possible to overlook disturbances or malfunctions. We can point to the classic example[5] of the efficiency of a Wi-Fi network, which may correlate to the density of auxiliary networks connected to the system.

Hereinafter, we shall not discuss how to deal with the human factor, because whilst the human factor is an essential one, it does not directly relate to the critical software-based equipment, except for:

– the activities of creation of the software application – hence the need to formalize the skills and responsibilities of the people in charge of the software, as indicated in Chapter 5 of the standard;

– the activities of maintenance and rollout, which are dealt with by Chapter 9 of CENELEC 50128:2011 [CEN 11a].

As regards the identification of the actors involved, it is more usual to speak of identification of the stakeholders; for further information, see Chapter 11 of [BOU 14c].

5 The use of so-called "open" networks (see the standards [CEN 01a] and [CEN 11a]) such as Wi-Fi is attended by a certain number of difficulties, such as network densification (the number of private networks is constantly increasing) and/or interference caused by nearby equipment. It should be noted that, for a very long time, the issue of open networks has not been approached from the standpoint of functional safety, because it relates to aspects such as confidentiality, intrusion, etc., which are covered by the term "security".

1.3. System

Our aim in this section is to lay down the vocabulary relating to the creation of a software-based device. To begin with, we must remember that a software application is directly linked to a device, and that without hardware architecture, there can be no software. Indeed, the validation of a program (see Chapter 5) requires the hardware architecture, and the results are applicable only to that particular hardware. For this reason, the first definitions we shall give relate to the concept of a system and of a software-based system.

DEFINITION 1.1 (System).– *A system is a set of elements interacting with one another, which is organized in such a way as to achieve one or more predetermined results*.

The "organized" part of Definition 1.1 can be seen in the system's organization into different levels, as illustrated by Figure 1.4.

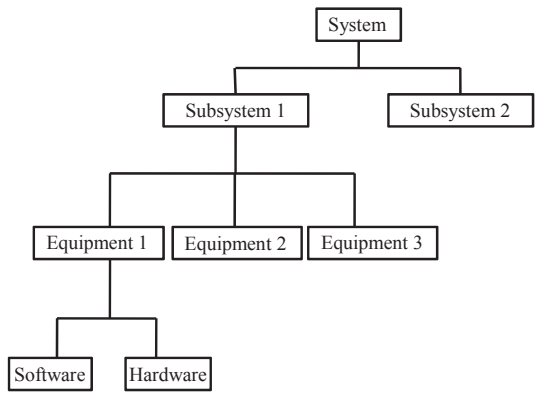

Figure 1.4. *From the system to the software*

Figure 1.4 offers a hierarchical view of the system. This is the view which is used in the railway domain. Hence, a railway line is viewed as a system, which is divided into a number of subsystems: the signaling control subsystem, the passenger transfer subsystem, etc. The signaling control subsystem, for its part, is divided into a number of devices, or classes of equipment: onboard equipment, ground equipment and line equipment.

A system performs several functions. A system function can be subdivided into a variety of subsystems, with each subsystem performing functions which are subfunctions of the whole system's functions. At system level, this representation needs to be accompanied by models which illustrate the interactions between the functions, as shown by the example given in Figure 1.5.

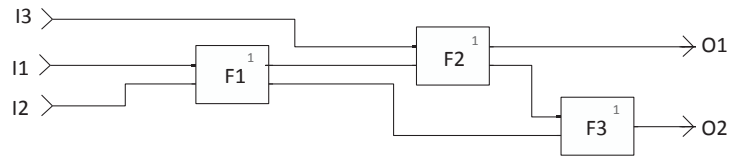

Figure 1.5. *Example of the subdivision of a system*

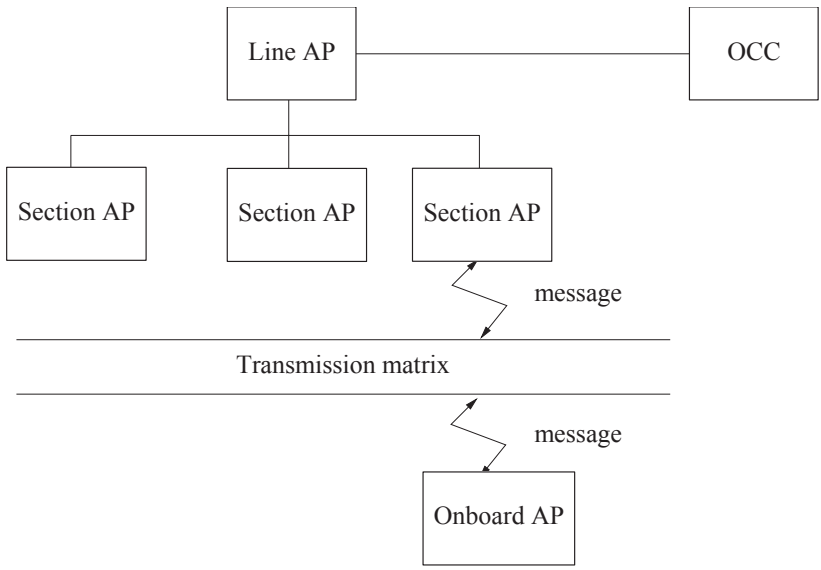

Figure 1.6. *Example of distribution*[6]

6 The example of a subsystem presented is the control system "SAET" used on the "METEOR" line (the rapid-transit East/West Line 14 ("METEOR" is a backronym for this) on the Paris Metro). For further information, see Chapter 2 of [BOU 12].

Thus, a subsystem hosts a variety of functions, which can then be divided between several different pieces of equipment. A piece of equipment is not a functional element in itself; it must be joined by other equipment in order to perform a subsystem-level function.

In terms of a railway system, the difficulty lies in the fact that a train is home to many system functions, and therefore that it contains equipment which contributes to these different functions. For example, the installation of an auto-pilot subsystem involves installing devices on the ground, which communicate with an onboard component on the train, as shown by Figure 1.6.

DEFINITION 1.2 (Software-based system).– *Elements of the system may be totally or partially software-based.*

Figure 1.7 shows that a system is a structured entity (comprising computer systems, processes and use contexts) which must form an organized, coherent whole. Hereinafter, we shall examine the software applications which are found in the computer/automated system component.

In this chapter, we have shown that a system based on programmable electronic equipment is a complex object, which needs to be carefully analyzed in each of its component parts.

1.4. Software application

1.4.1. *What is software?*

In the context of this chapter, the so-called “software” element is a set of computation/processing elements which are executed on a physical hardware architecture so that the system, as a whole, can render the services associated with a device (see Figure 1.4).

Later on in this book, we shall look at the software aspects, so it is necessary, at this point, to define exactly what software is – see Definition 1.3. This definition is slightly different from the one given by ISO 90003:2004 [ISO 04a].

DEFINITION 1.3 (Software).– *Set of programs, processes and rules, and possibly documentation as well, relating to the performance of a set of operations on the data.*

Definition 1.3 does not differentiate between the means (the methods, processes, tools, etc.) used to create the software application, the products created by its execution (documents, analytical results, models, sources, test scenarios, test results, specific tools, etc.) and the software application itself.

This definition is generally associated with the concept of a software application. The concept of software itself is associated with that of executable files.

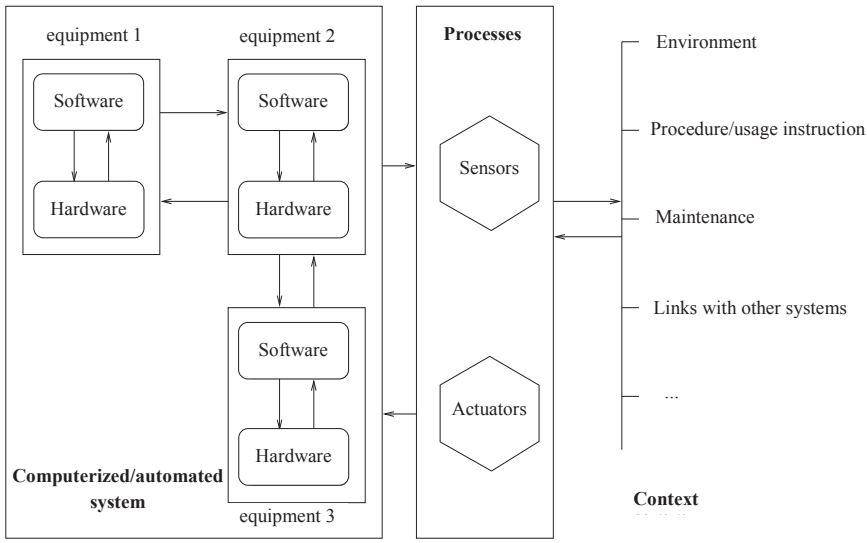

Figure 1.7. *System and interaction*

1.4.2. *Different types of software*

Definition 1.3 shows what the concept of software involves, but it should be noted that there are a variety of different types of software:

– operational software: this term refers to any software delivered to an external customer as part of a program or a product. Test arrays for external usage fall into that category;

– demo: a demonstrator (demo) is a piece of software used by an external customer to help refine their expression of their needs and measure the level of service which could potentially be delivered. These programs are not intended for operational use;

– development tool: a development tool is an internal software application, which is not delivered to an external customer, designed to help development in the broadest sense (editor, compilation chain, etc.), including at the test stage and integration stage;

– model: a model is an internal program for study, not delivered to any external parties, which serves to check a theory, an algorithm or the feasibility of a technique (e.g. by simulation), without the objective of a result or of completeness.

1.4.3. *The software application in its proper context*

In spite of the long-standing monolithic view, we feel it is important to look at a software application as a set of components (see Definition 1.4), which interact to process a set of data. Thus, a component may be a part of the software application, a reused part, a library, a commercial off-the-shelf (COTS[7] – see Definition 1.5) component, etc.

DEFINITION 1.4 (Component).– *A component is an element of software which performs a set of predefined services; these services (or tasks) conform to a clear set of requirements; a component has clear interfaces and is managed in configuration as a separate element in its own right*.

DEFINITION 1.5 (Commercial off-the-shelf – COTS).– *A software product which is available to buy and use without carrying out development activities*.

As Figure 1.8 shows, a software application generally uses an abstraction of the hardware architecture and of its operating system by way of a base layer known as the "base software". In principle, the base software should be written in low-level programming languages such as an assembly language and/or C [KER 88]. It is used to encapsulate the services of the

7 COTS are products that are commercially available and can be bought "as is" (without specification, V&V elements, etc.).

operating system and its utilities, but it also provides relatively direct access to hardware resources.

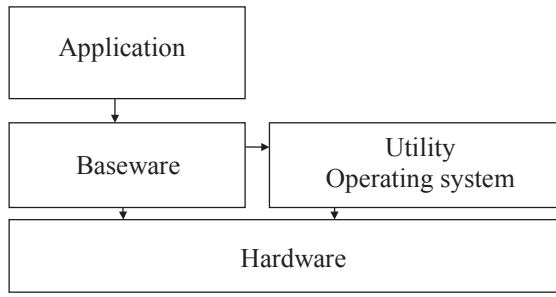

Figure 1.8. *A software application in its environment*

If the software application is associated with a high level of safety, then the base layers (baseware, utilities and operating system) are also associated with safety objectives. The safety objective of the lower layers will depend on the hardware architecture in place (monoprocessor, 2oo2, nOOm, etc.) and on the safety measures employed. In [BOU 10a], we presented real-world examples of safe functional architectures. For the rest of this book, we shall assume that the safety analyses have been performed and that for all the software applications (including the base layer), a level of safety has been allocated.

1.5. Conclusion

In this chapter, we have shown the underlying complexity of programmable electronics-based systems. We have demonstrated the existence of a hierarchy which ranges from the system to the software, through the subsystems, devices and the electronic hardware.

In Chapter 3, we shall show how to control the dependability of a system by controlling the electronic part (see [BOU 10a]) and the software part.