A Roadmap and Some Directions Towards the Engineering of Interactive Systems Deployable in Safety Critical Contexts

David Navarre[1], Philippe Palanque[1-2] & Célia Martinie [1]

[1] ICS-IRIT, University of Toulouse, 118 Route de Narbonne, F-31062, Toulouse, France
[2] Technical University Eindhoven, Department of Industrial Design, Eindhoven, Netherlands
{navarre, palanque}@irit.fr

Keywords: Engineering Interactive Systems, Critical systems, Formal Models, Fault-Tolerance, Development Assurance Levels, Technology Readiness Levels.

1 Introduction

This position paper presents how the engineering of interactive system may be impacted when these systems are designed to be deployed in safety critical contexts. These domains are specific as they are usually driven by constraining standards defining precisely best practices and mandatory processes to be applied. We thus discuss on how it is possible to take into account these constraints in real-life domains where critical and non-critical components may coexist. Engineering approaches in this context must jointly exploit formal description techniques (for the critical parts) and standard software production techniques (for the non-critical part). More precisely, we will show how Engineering Interactive Systems in these domains impact particularly design and development processes, systems architectural aspects and developers' tasks.

In the next section, we present the constraints introduced by the safety critical context on development processes that embed certification activities as well as standards (with a focus on aviation industry). Other domain like space, nuclear or Air Traffic Management propose specific standards with stronger or softer constraints but the overall approach remains the same. The third section presents how software and (more globally) system architectures must encompass specificities of current interactive systems and how they are deeply impacted by their critical nature. The fourth section focuses on the human aspect highlighting the specific need for usable tools to support software and system engineers. Last section concludes the paper and provides directions for future work.

2 Constraints from Safety Critical Context

When dealing with safety critical systems, it is important to notice that all components of the whole system do not have necessarily the same level of criticality, requiring to carefully identify each of them in the early stages of the development process.

In this section we present how this identification of levels of criticality is performed and more precisely how it is handled in civil aviation.

2.1 Development Assurance Level (DAL) identification

In civil aviation, the criticality level of a system (or component) is classified by means of five Development Assurance Levels (DAL), introduced in the DO-178C standard [6] for software components and in the DO-254 standard [7] for electronic hardware components. These levels are directly linked to failure condition categories defined by certification authorities such as EASA (European Aviation Safety Agency) or FAA (Federal Aviation Administration). Table 1 presents the five Development Assurance Levels associated with their failure condition category and its description (summarized from the EASA CS-25 standard [5]). As presented in the first row of Table 1, a failure having catastrophic consequences (failure condition column) must not occur more often than once per 10^9 hours of functioning (failure rate column). Such software must be developed following DAL A processes and methods as defined in DO-178C [6].

Development Assurance Level	Failure condition categories	Description of the failure conditions	Failure rate (failures/hour)
A	Catastrophic	Failure conditions that may cause a crash	Extremely improbable 10-9 +fail safe
B	Hazardous	Failure has a large negative impact on performance, or reduces the ability of crew to operate the plane	Extremely remote 10-7
C	Major	Failure is significant, but has lesser impact than hazardous	Remote 10-5
D	Minor	Failure is noticeable, but has lesser impact than Major	Probable 10-3
E	No safety effect	No impact on dependability	Any range

Table 1. System Development Assurance Level for civil aircrafts

According to this standard the first two rows of Table 1 (colored in grey) correspond to so-called **critical** systems while systems in the lower rows are called **non-critical**.

The main difference between critical and non-critical software is that a lot of verification activities must be satisfied with independence, thus leading to very expensive software and system development. Identifying accurately the correct level of DAL is thus very important in order to invest adequately development resources (not overspend on low DAL systems and not underspend on critical software).

ARINC specification 661 [1] aims at providing a common ground for building interactive software in the field of aeronautical industry. All interactive software in cockpits (following ARINC 661 specification) is categorized as DAL C and thus can only be used for the command and control of non-critical systems. On such systems, assuring that operators will be able to perform their activities (even if the interactive system exhibits failures) is an important issue to address. The zero-defect approaches (usually based on formal approaches such as [2]) try to guarantee that the interactive system will be defect free and if it is a good mean for removing faults and bugs at development time, natural faults (such as bit-flips due to radiations) are beyond their reach. Fault tolerance mechanisms are thus to be added to guarantee that both input and output devices are not becoming single points of failure, requiring to add redundancy, diversification and segregation of these devices/software as proposed in [12] following ARINC 653 portion management standard as in [13].

2.2 Certification

As explained previously, the CS-25 standard [5] is the standard for certification specification of the EASA for large aeroplanes. This document identifies requirements dedicated to the cockpit in section 1302 called "*Installed systems and equipment for use of the flight crew*". Following this standard, the cockpit must allow the crew to safely perform all their tasks and avoid error prone behaviors.

However, during operations, some of the pilot tasks may involve several aircraft systems with different DALs. For instance, after perceiving an alarm on the ECAM display the pilot might decide to decrease flight level. In that case, he will successively use information displayed by the Flight Warning System (DAL C) and trigger commands using the Flight Control Unit (DAL A). In order to comply with CS 25 1302 it is thus important to develop methods and tools capable of ensuring that goals can be reached and tasks can be performed on systems of various DALs involving different techniques (formal and non-formal) for their development as presented in [11].

2.3 Standards

An important task in processes for critical systems is to ensure that high-level requirements (HLR) have been explicitly (and in an exhaustive manner) gathered and described. Beyond, the designs and developments must demonstrate that they take into account each of these requirement and that such traceability is also part of the certification processes (e.g. in Air Traffic Management [9]). Taking into account all HLR into development is an objective and this is assessed by external bodies (for DAL A and DAL B software). DO-178C only identifies what must be done while annexes to that standard identify how thing should/could be done. For instance, DO-333 annex [8] (page 101) states that high-level requirements should be expressed in temporal logic (and more precisely Computational Tree Logic [10]) while low-level requirements (LLR) should be expressed by state-based description techniques. Compliance between these two representations must be done using model-checking techniques [3] (as illustrated by **Fig. 1**).

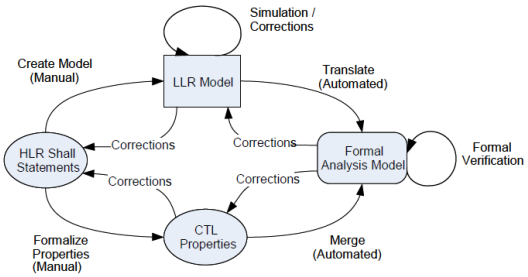

Fig. 1. Analysis Process (DO-178 C [6])

Dealing with interactive systems introduces specific HLRs that mix together a system point of view (hardware and software) and an operation point of view (users' tasks). This would require the extension of such standards with the use of specific formal description techniques covering both the system side and the operation side.

Such approach has been proposed by [11], highlighting (only for standard WIMP interaction techniques), that a process should ensure completeness and consistency between a

mixed-criticality interactive application and users' tasks to support design of multiple DAL systems:

- **Completeness:** Aims at assessing if there is a system artifact dedicated to each expected supported user tasks. For verification purpose, a focus must be put on having a correspondence for each critical task.
- **Consistency:** As tasks may be critical, they should be performed using a critical part of the application (higher DALs), while standard tasks may be performed without any attention put on the criticality of the interactive software part they are performed with (lower DALs).

3 Architectures to deal with complexity

The software architecture of an interactive critical system must be able to follow the evolution of systems (input devices, output devices, interaction techniques…). Such an ever-changing context makes it very difficult to provide generic approaches to support design, development and deployment. Designers need to go beyond the interactive application design by providing new interaction techniques that encompass new input and output deices which can be very cumbersome to design and evaluate. Developers of these systems are repetitively facing the same issues of new devices integration, software redesign (due to device drivers' evolution) and above all poor reliability of the resulting system due to the low level of maturity of the various components to integrate. Such constraints are even stronger with high DAL critical components.

As proposed in [4], ensuring the modularity of the entire interactive system (including hardware and software parts), leads to identify the building bricks that must be developed for integrating new devices as well as the building bricks for merging information from these devices in order to offer multimodal interactions to users. The proposed architecture thus enables the separation of a complex system into several components that are then easier to apprehend, allowing the separation of concerns and the locality of modification.

4 Usable Tools to cope with complexity

Providing support for process and architecture of complex interactive critical systems requires the tools to cope with three challenges: they must be mature enough to support industrial processes, they must support developers' tasks, they must support models engineering.

4.1 Assessing the maturity of tools

The Technology Readiness Levels (TRLs), defined in ISO 16290:2013 [15] is a scale from 1 (*Basic principles observed and reported*) to 9 (*Actual system "flight proven" through successful mission operations*) that defines conditions to be met to qualify the maturity of an element within a system. Initially designed for space system hardware, it can be used in many domains, including tool support.

While industrial processes may require specific TRL for their tool support, it is important to assess the maturity of proposed tools. However, in the field of interactive software engineering, there is not any communication about the TRL of tools involved in engineering interactive systems. We know for instance that PetShop [19] has gone through TRL3 (*Analytical and experimental critical function and/or characteristic proof-of-concept*) at Airbus, but this did not lead to any publication or advertisement.

4.2 Supporting models engineering

Several tasks must be performed when producing models during the development of an interactive system:

- When designing and developing interactive systems, the scope of the modeling activities must be identified in order to determine which elements (related to the interactive system to be developed) will be modeled and which types of models will be used. Then, the user/engineer must analyze the extant system, as well as the needs and requirements for the interactive system to be developed. After this analysis step, the production of the models starts with the editing step. That step produces a first version of models.
- The verification step is then performed for checking the models in order to ensure that they are complete and consistent.
- The validation step will then be initiated to check whether the model meets specified needs and requirements.

This editing-verification-validation loop will be performed again until the models are complete, consistent and meet needs and requirements. When introducing several kinds of models (e.g. task models and system models), the whole process thus must include the cross validation of these models in order to check if they are complete and consistent. While these steps are presented in an abstract way, the tools must be tuned for each notation in order to best support models engineering. In [14] authors report how action theory model can be used to design and evaluate modelling tools but only for notations based on Petri nets.

4.3 Supporting developer's tasks

In [17], Brad A. Myers stated that "Programmers are Users Too…", arguing that is it necessary to promote specific design processes and evaluation method to developers.

Such approach must be extended to modelling activities. For instance, applying (in that context too) the seven stages of Norman action theory [18] to modeling work highlights specific activities to consider. The two main stages we are focusing on are:

- On the execution side, the activity of going from an intention to its actual transcription into some model,
- On the evaluation side, the activity of perceiving model behavior and interpreting this perception.

One of the main advantages of this model is that it provides a generic framework for investigating where the main difficulties can occur during system modeling which is a

highly demanding task on the engineer's side. This framework thus provides design rules for environments to support user's activities and reduce their difficulties.

Norman action theory provides a generic framework for investigating where the main difficulties can occur during system modeling activities performed by an engineer. Each engineer task described in the previous section (editing, verification, validation and cross-type validation of models) is a goal that must be accomplished during the development of interactive systems. And for each of these goals, the seven stages of Norman action theory can be used to analyze the low-level activities that the engineer will execute.

5 Conclusion

This position paper has advocated the fact engineering interactive system is far from being a mature domain. This is due to the intrinsic nature of these systems that are directly handled by human beings but also due to the fact that their deployment to critical contexts is more and more of importance.

Some work has been proposed over the years by different research groups of different origins but integration is still needed so that engineers of such systems have processes, notations, languages and tools adapted to their work.

Lastly, interactive systems are currently of very low quality and exhibit defects and failure at a very high rate. This is due to the fact that interaction techniques and interaction technologies are a constant moving target trying to support better users, their needs and their experience. In the future, interactive systems engineering should thus progress at a higher pace than interaction technologies in order, first to catch up will current gap and second to have interactive systems engineers adequately equipped for future challenges.

References

1. ARINC 661-6 specification: Cockpit Display System Interfaces To User Systems, Prepared by AIRLINES ELECTRONIC ENGINEERING COMMITTEE, Published by AERONAUTICAL RADIO, INC, sept, 2016.
2. Barboni E., Conversy S., Navarre D., Palanque P. Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification. DSV-IS 2006, LNCS, Springer Verlag, 25-38
3. Clarke E., Grumberg O. & Peled P. Model Checking, The MIT Press, Cambridge, Massachusetts, 2001.
4. Cronel M., Dumas B., Palanque P., Canny A. (2019) MIODMIT: A Generic Architecture for Dynamic Multimodal Interactive Systems. In: Bogdan C., Kuusinen K., Lárusdóttir M., Palanque P., Winckler M. (eds) Human-Centered Software Engineering. HCSE 2018. Lecture Notes in Computer Science, vol 11262. Springer, Cham
5. CS-25 – Amendment 17 - Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes. EASA, 2015.
6. DO-178C / ED-12C, Software Considerations in Airborne Systems and Equipment Certification, published by RTCA and EUROCAE, 2012.
7. DO-254/ED-80. Design Assurance Guidance for Airborne Electronic Hardware, published by RTCA and EUROCAE, 2000.
8. DO-333 Formal Methods Supplement to DO-178C and DO-278A, published by RTCA and EUROCAE December 13, 2011.

9. ESARR 6. EUROCONTROL Safety Regulatory Requirement. Software in ATM Systems. Edition 1.0. http://www.eurocontrol.int/src/public/standard_page/esarr6.html (2003)
10. Emerson E.A., Srinivasan J. Branching Time Temporal Logic, in LNCS 354 p.122-172, Springer-Verlag 1988.
11. Fayollas, C., Martinie, C., Navarre, D., and Palanque, P. 2016. Engineering mixed-criticality interactive applications. In Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '16). ACM, New York, NY, USA, 108-119. DOI: https://doi.org/10.1145/2933242.2933258
12. Fayollas C., Fabre J-C., Palanque P., Cronel M., Navarre D., Deleris Y. A Software-Implemented Fault-Tolerance Approach for Control and Display Systems in Avionics. IEEE Pacific Rim Dependable Computing conference 2014: 21-30
13. Fayollas C., Fabre J-C., Navarre D., Palanque P., Deleris. Fault-Tolerant Interactive Cockpits for Critical Applications: Overall Approach. SERENE 2012: LNCS, Springer Verlag, 32-46
14. Fayollas C., Martinie C., Palanque P., Barboni E., Fahssi R., Hamon A.Exploiting Action Theory as a Framework for Analysis and Design of Formal Methods Approaches: Application to the CIRCUS Integrated Development Environment. Handbook of Formal Methods in Human-Computer Interaction 2017: 465-504
15. ISO/IEC IS 16290:2013: Space Systems - Definition of the Technology Readiness Levels (TRLs) and their criteria of assessment. International Organization for Standardization, Geneva, Switzerland, 2013.
16. Morris M., Huang A., Paepcke A., & Winograd T. Cooperative Gestures: Multi-user Gestural Interactions for Colocated Groupware. ACM CHI Conf., 2006. 1201-1210.
17. Myers, B. A., Ko, A. J., LaToza, T. D. and Yoon, Y. "Programmers Are Users Too: Human-Centered Methods for Improving Programming Tools," IEEE Computer, Special issue on UI Design, 49, issue 7, July, 2016, pp. 44-52.
18. Norman, D. A. (2013). The design of everyday things: Revised and expanded edition. Basic books.
19. Sy, O, Bastide, R, Palanque, P, Le, D-H, Navarre, D. “PetShop: a CASE Tool for the Petri Net Based Specification and Prototyping of CORBA Systems.” In 20th International Conference on Applications and Theory of Petri Nets, ICATPN’99, Williamsburg, VA, USA. 1999.