Similarity as a Design Driver for User Interfaces of Dependable Critical Systems

David Navarre, Philippe Palanque(✉), Arnaud Hamon, and Sabrina Della Pasqua

ICS-IRIT, Université Toulouse III, Toulouse, France
{Navarre, palanque}@irit.fr

Abstract. Assuring that operators will be able to perform their activities even though the interactive system exhibits failures is one of the main issues to address when designing and implementing interactive systems in safety critical contexts. The zero-defect approaches (usually based on formal methods) aim at guaranteeing that the interactive system will be defect free. While this has been proven a good mean for detecting and removing faults and bugs at development time, natural faults (such as bit-flips due to radiations) are beyond their reach. One of the way to tackle this kind of issue is to propose redundant user interfaces offering multiple ways for the user to perform operations. When one of the interaction mean is failing, the operator can select another functional one. However, to avoid errors and increase learnability, it is important to ensure that the various user interfaces are "similar" at presentation and interaction levels. This paper investigates this relation between dependability and similarity for fault-tolerant interactive systems.

Keywords: UI properties · Similarity · Dependability · Usability
Learnability

1 Introduction

Usability [9] and user experience [7] properties have received (and are still receiving) a lot of attention in the area of Human-Computer Interaction to the extent that they are perceived as the main properties to study and consider while designing interactive systems or while performing research activities in HCI.

Beyond this main stream of research and design, other more marginal approaches have tried to investigate the relationship between these properties and other ones such as security [18], accessibility [16, 19], dependability [3] or privacy [6] (among many others).

Each of these specific domains bring specific issues in order to ensure that the associated properties have been taken into account. Taking into account these properties usually requires identifying and managing trade-off i.e. favoring one property above the other. For instance, adding an undo function to an interactive system will improve usability by make it more efficient for users to recover from errors. However, adding undo functionality to a system increases significantly the number of lines of code and thus the likelihood of bugs. This paper focuses on dependability related issues

T. Clemmensen et al. (Eds.): INTERACT 2017, LNCS 10774, pp. 114–122, 2018.
https://doi.org/10.1007/978-3-319-92081-8_11

and how dealing with them might bring additional concerns for the design of user interfaces and their associated interaction techniques. However, despite this specific focus on one property, similar constraints would apply to other conflicting properties.

Assuring that operators will be able to perform their activities even though the interactive system exhibits failures is one of the main issues to address when designing and implementing interactive systems in safety critical contexts. Exploiting methods, techniques and tools from the dependable computing field [10] can ensure this even though they have not been designed and developed to meet the challenges of interactive systems [4]. Such approaches can be dived into two main categories:

- **The zero-defect approaches** (usually based on formal methods [21]) that aim at guaranteeing that the interactive system will be defect free. While this has been proven a good mean for detecting and removing faults and bugs at development time, natural faults (such as bit-flips due to radiations) are beyond their reach.
- **The fault-tolerant approaches** that promote the use of **redundancy** (multiple versions of the system), **diversity** (the various versions are developed using different means, technologies and providers) and **segregation** (the various versions are integrated in the operational environment by independent means e.g. executed on different computers, using different communication means, …). Segregation ensures that a fault in one of the versions will not induce a fault in another version – usually called common point of failure.

One of the way to apply dependability principles to the user interface of the interactive system is to propose redundant user interfaces offering multiple ways for the user to perform operations. This can be displaying the same information on different screens or offering multiple input devices for triggering the same action. This can also be performed at the interaction technique level as presented in [15] where mouse failures were mitigated by the use of "similar" configurations based on use of multiples keys on the keyboard. However, to avoid user errors (such as capture errors [17]) and increase **learnability**, it is important to ensure that the various user interfaces are "similar" at presentation and interaction levels. This concept of **similarity** has already been used in the field of web engineering [8] but only with a focus of designing new web systems being consistent with legacy non-web systems.

This paper refines the concept of similarity and shows how this concept is relevant at different levels of the architecture of interactive systems. The paper then presents a set of examples from the avionics domain where dependability is a major concern and where development of fault-tolerant mechanisms is a requirement from standardization authorities such as DO 178C standard [1]. These examples present how similarity has been driving the design of multiple user interfaces even though they are as different as hardware only (interaction taking place through knobs and dials) and software mainly using WIMP interaction techniques. Conclusions and discussions for the workshop are presented in the last section.

2 Conflicts and Congruence Between Similarity, Diversity and Redundancy in the Area of Interactive Systems

In order to increase resilience to failures, fault-tolerance (i.e. guaranteeing the continuity of service), requires **duplicated user interfaces** for the command and control of a single system. This ends up with **redundant user interfaces** serving the same purpose. If those interfaces are built using the same processes and offer the same interaction techniques, it is possible that a single fault could trigger failures in both user interfaces. This could be the case for instance when using the idea of cloning the UI as proposed by [20]. In order to avoid such common points of failure the redundant user interfaces must ensure **diversity**. Diversity can be guaranteed if the user interfaces have been developed using diverse means such as different programming languages, different notations for describing their specification, executed on top of different operating systems, exploiting different output and input devices, … Such diversity is only efficient if the command and control system offers confinement mechanisms avoiding cascading faults i.e. the failure of one user interface triggering a failure in the duplicated one.

Such fault tolerant basic principles raise **conflicting** design issues when applied to user interfaces. Indeed, diversity requires the user interfaces to be very different in terms of structure, content and in terms of interaction techniques they offer, even though they must guarantee that they support the same tasks and the same goals of the operators [5]. Another aspect is that they must be located in different places in the system i.e. distributed as this is one of the most efficient way of ensuring confinement of faults.

In that context, distribution of user interface does not concern the presentation of complementary information in different contexts (as presented in [12]) but the presentation of redundant information in those contexts.

In terms of design, it is important to be able to assess that the various user interfaces make it possible for the operators to reach their goals (this would be called similarity in terms of **effectiveness**). Beyond that, it is also important to be able to assess the relative complexity and diversity of these interfaces in order to be sure that operations will not be drastically degraded when a redundant user interface has to be used after a failure has occurred on another one. Studying the effective **similarity** (in terms of **efficiency**) at the level of input and output is thus required even though different type of displays and different types of input devices have to be used. This goes beyond the study of similarity at effectiveness level, but both contribute to the usability of the systems. It is important to note that all the other properties mentioned previously are intrinsic or extrinsic properties of a given interactive system. Similarity is very special as it only has a meaning when two interactive systems are considered (or two different versions of a same interactive system). Such relative properties are usually less studied than absolute properties as the focus of interest is usually to favour a given property of a given system.

3 Examples from the Avionics Domain

The case study presents (in the area of aircraft cockpits) examples of redundant user interfaces. More precisely, we present in the context of the cockpit of the A380 (see Fig. 1) aircraft. In this new generation of large civil aircrafts, the cockpit presents display units (that can be considered here as computers screens) of which some of them are offering interaction via a mouse and a keyboard by means of an integrated input device called KCCU (Keyboard Cursor Control Unit). Applications are allocated to the various display unit (DU).

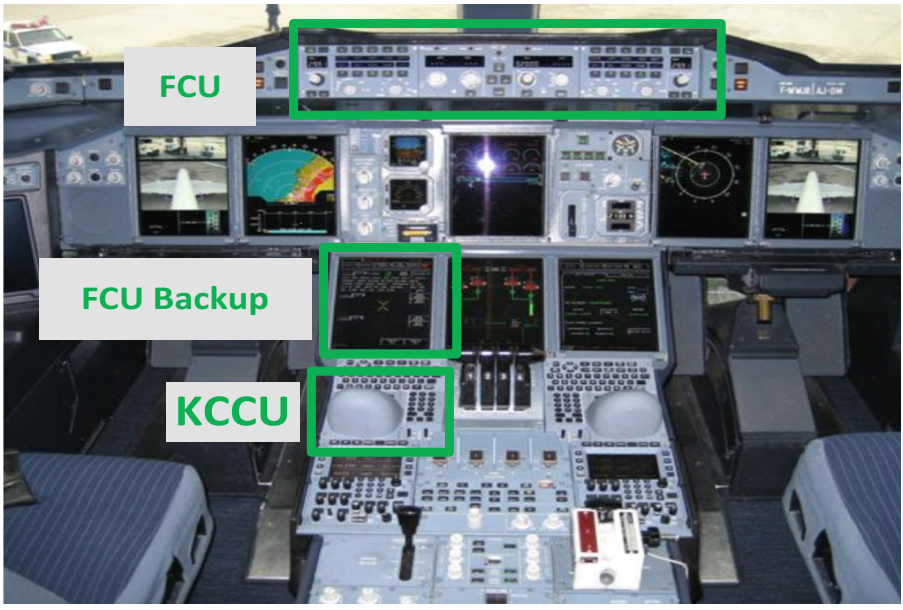

Fig. 1. Two possible means to control flight heading within the A380 interactive cockpit, one using the FCU and the other using the FCU Software application and the KCCU

In the A380, two redundant ways of using the autopilot are offered to the pilot in order to change the heading of the aircraft. One is performed using the electronic user interface of the Flight Control Unit (FCU on top of Fig. 1) while the other one exploits the graphical user interface of the Flight Control Unit Backup interface and the KCCU (bottom of Fig. 1).

3.1 Example One: Entering a New Value for Heading

Figure 2 presents a zoomed view on the two means for entering a new heading of the aircraft. On the left-hand side of the figure, the editing of the heading is performed using a physical knob, which may be turned to set a heading value (this value ranges from 0 to 360). The selected value can be sent to the autopilot (called “engaged”) by

pressing the physical LOC push button below the knob. On the right-hand side, the heading is set using the keyboard of the KCCU and engaged by using the KCCU and its manipulator to click on the dedicated software LOC push button.

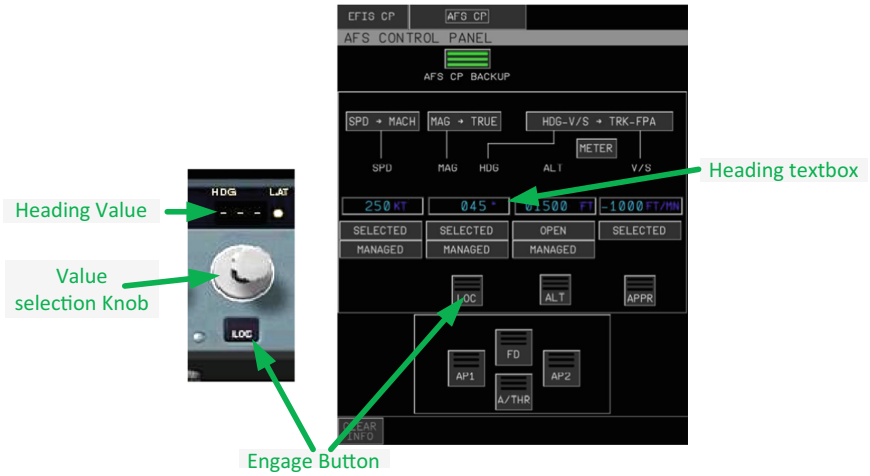

Fig. 2. Heading selection.

At a high level of abstraction (i.e. not taking into account the input and output devices), the task of setting a new value for the heading is the same on both user interfaces (they are similar at the effectiveness level). If described at a lower level, the description of these two tasks would be different, as they would require different physical movements from the pilots (they are thus not similar at the effectiveness level as for instance, the pilot would have to execute the FCUS application while the hardware FCU is directly reachable). It is important to note that there are other additional means to perform the same task (for instance controlling directly the aircraft using the sidestick) that are not presented here.

3.2 Example Two: Entering a Set of Parameters for the Navigation Display

Figure 3 presents two different means to handle both barometer settings and parameters of the navigation display (ND – pilot ND is the second screen on the left in Fig. 1 while first officer ND is the second screen on the right). It illustrates how physical input devices (on the left-hand side of Fig. 3) have been transposed into software components (right-hand side of Fig. 3) handled using the KCCU (as in the FCUS presented in Fig. 2). The general layout of both interface is quite close to that one, but the translation into a software application leads to different design options:

- On the physical interface, the **two barometer settings** options (highlighted in yellow and on the bottom left part of both physical and software interfaces) are handled using two physical labelled push buttons (LS and VV) that are lighted on

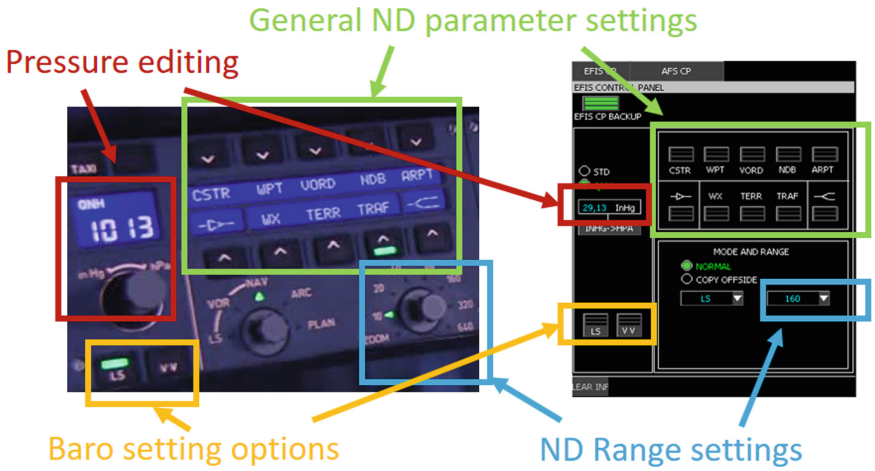

Fig. 3. Baro settings and Navigation Display configuration

with a single light when the option is selected. The transposition of these two buttons in the software user interface results is a set of two software buttons that may be highlighted by changing the color of three horizontal lines. In this case, the two design options are quite similar.

- The **General ND parameter settings** (highlighted in green and on the top right part of both physical and software interfaces) are physically handled using physical push buttons without labels associated to labels displayed on a dedicated screen. These buttons behave in the same way as the two previous buttons. The software transposition is similar to the previous one, using both software push buttons and labels, and following the same layout constraints (relative position and size) as the physical interface.
- The **Pressure editing** (highlighted in red and located on the left-hand side of both physical and software interfaces) consists in the editing of a numeric value. The physical and software representations of this function follow two distinct design option. With the physical interface, this value is modified using a physical knob and the edited value is displayed on a dedicated screen while on the software transposition, this editing is performed using a classical text field that embed both editing and display of the value. It is thus possible on the software UI to use the arrow keys to navigate into the text box and modify one specific digit of the pressure, which is not feasible on the hardware UI.
- The **ND range setting** (highlighted in blue and on the bottom right part of both physical and software interfaces) is performed by selecting a range amongst a finite set of predefined values. In this case, the two design options are quite different too. On the physical interface, the task is performed using a knob that can rotate between the set of values, these values being physically written around the knob (making it visible at any time). The software translation of this interface is made up using a drop down combo box that embed both the display and selection of the value. In this case, the selectable values are only displayed while using the software component.

3.3 Example One: Visualization of Aircraft Pitch and Roll

Figure 4 presents two different design of the gyroscope instrument that aims at providing the pilot with information about the position of the aircraft relatively to the horizon (both pitch and roll). At the bottom right-hand side of Fig. 4 the cockpit presents the physical analog display of these values. This device is also called the artificial horizon as the information it displays is similar to the view the pilots have when they look outside through the windshield. The software transposition of this instrument (on the left-hand side of Fig. 4 – called Primary Flight Display) embeds several other functions such as an altimeter or a speed controller. The graphical layout of the software UI is clearly inspired by the physical one which was, in the early days of aviation only a physical ball emerged in a container filled with liquid.

Fig. 4. Physical and software representation of the aircraft gyroscope.

4 Research Directions

While the examples above focus on the presentation and interaction aspects of interactive systems, we are investigating other means to support similarity analysis and assessment also at user level:

- Investigating means of describing past experiences and practice of users to understand the level of familiarity a user may have with a given interaction
- Investigating means of describing tasks (including knowledge, information and objects used to perform the tasks) such as with the HAMSTERS tool [14] to assess similarity of tasks and goals
- Investigating means of describing users' errors (including causes called genotypes and manifestation called phenotypes) in order to identify potential unexpected types of errors that could occur see [2].

In the area of aviation, the design driver for cockpit has been on targeting at similarity of command and displays even though this is not clearly stated. Indeed, looking at training, most airlines propose Cross Crew Qualification programs for pilots [11]. As training is mainly based on tasks execution [13] such a goals and task-based approach is critical and is the only way of designing and evaluating training programs evolutions.

5 Discussions and Conclusion

This paper has presented the similarity property for interactive systems offering redundant ways for the users to enter and perceive information. In order to ensure diversity and segregation (that are required for building dependable interactive systems) the similarity property may be violated. We have shown on the first example that the hardware and the software user interface are similar at the effectiveness level but distinct at interaction level. The following examples have shown bigger gaps in terms of similarity as the use of computing systems and graphical interfaces provides designers and developers with more advanced communication and interaction means. Digital devices are thus more informative and more efficient than the hardware ones. However, they are also less reliable than hardware systems and must not be used if failures are detected [3]. This means that the design and the evaluation of the training program is a complex and expensive activity requiring tools and technique to assess (and explain to trainees) gaps in similarity.

References

1. DO-178C/ED-12C, Software Considerations in Airborne Systems and Equipment Certification, published by RTCA and EUROCAE (2012)
2. Fahssi, R., Martinie, C., Palanque, P.: Enhanced task modelling for systematic identification and explicit representation of human errors. In: Abascal, J., Barbosa, S., Fetter, M., Gross, T., Palanque, P., Winckler, M. (eds.) INTERACT 2015. LNCS, vol. 9299, pp. 192–212. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22723-8_16
3. Fayollas, C., Martinie, C., Palanque, P., Deleris, Y., Fabre, J-C., Navarre, D.: An approach for assessing the impact of dependability on usability: application to interactive cockpits. In: Tenth European Dependable Computing Conference (EDCC 2014), pp. 198–209. IEEE Computer Society (2014)
4. 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. In: 20th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2014, pp. 21–30. IEEE Computer Society (2014)
5. Fayollas, C., Martinie, C., Navarre, D., Palanque, P., Fahssi, R.; Fault-tolerant user interfaces for critical systems: duplication, redundancy and diversity as new dimensions of distributed user interfaces. In: Workshop on Distributed UIs and Multimodal Interaction (DUI 2014), pp. 27–30. ACM DL (2004)
6. Gerber, P., Volkamer, M., Renaud, K.: Usability versus privacy instead of usable privacy: Google's balancing act between usability and privacy. SIGCAS Comput. Soc. **45**(1), 16–21 (2015)
7. Hassenzahl, M., Platz, A., Burmester, M., Lehner, K.: Hedonic and ergonomic quality aspects determine a software's appeal. In: CHI 2000, pp. 201–208 (2000)
8. Heil, S., Bakaev, M., Gaedke, M.: Measuring and ensuring similarity of user interfaces: the impact of web layout. In: Cellary, W., Mokbel, Mohamed F., Wang, J., Wang, H., Zhou, R., Zhang, Y. (eds.) WISE 2016. LNCS, vol. 10041, pp. 252–260. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48740-3_18
9. International Standard Organization: "ISO 9241-11." Ergonomic requirements for office work with visual display terminals (VDT) – Part 11 Guidance on Usability (1996)

10. Laprie, J., Randell, B.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secur. Comput. **1**(1), 11–33 (2004)
11. Lufthansa: Cross Crew Qualification courses. https://www.lufthansa-flight-training.com/documents/10156/5537743/Cross+Crew+Qualification+Course+CCQ. Accessed 2 Nov 2017
12. Martinie, C., Navarre, D., Palanque, P.: A multi-formalism approach for model-based dynamic distribution of user interfaces of critical interactive systems. Int. J. Hum. Comput. Stud. **72**(1), 77–99 (2014)
13. Martinie, C., Palanque, P., Navarre, D., Winckler, M., Poupart, E.: Model-based training an approach supporting operability of critical interactive systems. In: EICS 2011, pp. 53–62. ACM DL (2011)
14. Martinie, C., Palanque, P., Ragosta, M., Fahssi, R.: Extending procedural task models by systematic explicit integration of objects, knowledge and information. In: ECCE 2013, pp. 23:1–23:10. ACM DL (2013)
15. Navarre, D., Palanque, P., Basnyat, S.: A formal approach for user interaction reconfiguration of safety critical interactive systems. In: SAFECOMP 2008, pp. 373–386 (2008)
16. Petrie, H., Kheir, O.: The relationship between accessibility and usability of websites. In: SIGCHI Conference on Human Factors in Computing Systems (CHI 2007), pp. 397–406. ACM (2007)
17. Reason, J.: Human Error. Cambridge University Press, New York (1990)
18. Sasse, M.A., Karat, C.-M., Maxion R.: Designing and evaluating usable security and privacy technology. In: 5th Symposium on Usable Privacy and Security (SOUPS 2009). ACM (2009)
19. Section 508: The Road to Accessibility. http://www.section508.gov
20. Villanueva, P.G., Tesoriero, R., Gallud, J.A.: Distributing web components in a display ecosystem using Proxywork. In: 27th BCS HCI Conference (BCS-HCI 2013). British Computer Society (2013)
21. Weyers, B., Bowen, J., Dix, A., Palanque, P.: The Handbook of Formal Methods in Human-Computer Interaction. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-51838-1