ENSURING SAFETY AND SECURITY FOR AVIONICS: A CASE STUDY

Youssef Laarouchi[1,2], Yves Deswarte[1,2], David Powell[1,2], Jean Arlat[1,2], Eric De Nadai[3]

[1] *CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France*
[2] *Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France*
[3] *EDYMI2 BP M03020, AIRBUS France, 316 route de Bayonne, F-31060 Toulouse, France*

[1,2] *firstname.lastname@laas.fr*
[3] *firstname.lastname@airbus.com*

ABSTRACT

We present a case study in the avionics context, in which bidirectional information flows exist between critical components and less critical ones. These flows raise security and safety concerns that have to be taken into account to guarantee correct operation of the critical tasks. To allow upwards flows, we propose fault tolerance mechanisms based on diverse operating systems isolated by virtualization.

1. INTRODUCTION

Since the introduction of fly-by-wire flight control systems in 1984 with the Airbus A320, the number of software components embedded in commercial transport aircraft has been increasing every year. These components are designed to ensure the operation of specific tasks on the aircraft. However, all tasks do not have the same criticality. For instance, a task measuring altitude to calculate appropriate flight parameters is much more critical than one providing In-Flight Entertainment (IFE). This is the reason for giving each task a corresponding Design Assurance Level (DAL), as detailed in the DO-178B standard. The most critical tasks (DAL-A) are designed and developed under strict constraints, which make them very expensive, economically speaking. On the contrary, the use of commercial off-the-shelf (COTS) components for less critical tasks is more convenient since they offer extended functionalities at a more reasonable cost, but they do not satisfy the validation requirements of highly critical tasks.

Any communication between these heterogeneous-criticality components has to be completely mastered to ensure that the less critical tasks do not alter, in any way, the operation of more critical ones. A full isolation between each DAL level could be contemplated, but this would be too restrictive since it would not allow any inter-DAL-level communication at all, while such communication may be useful or, indeed, necessary.

The solution currently implemented on recent aircraft such as Airbus A380 allows unidirectional information flows from higher DAL levels to lower levels. In this case, critical software can control or send data to less critical components, but cannot receive any information (including acknowledgements) from them. This form of interaction can be likened to diodes used in electronic circuits.

In this paper, we refer to DAL levels as integrity levels, where integrity is taken to mean the level of confidence that can be placed in a given component [1].

In the ArSec project♦, we have identified several case studies in which information coming from low-integrity components could be used by critical components to fulfill their task, if that could be done with sufficient confidence. In this paper, we present one of these case studies, related to a possible evolution of maintenance operation on future aircraft (Section 2). To guarantee that such communications do not corrupt the whole system and do not make its critical tasks more vulnerable to accidental faults or malicious attacks, we propose a model that allows such communications to be controlled (Section 3). The implementation of our case study with respect to the chosen model is detailed in Section 4.

♦ This study is part of the ArSec project (*Architectures de Sécurités*), a cooperation between LAAS-CNRS and AIRBUS France under AIRSYS convention. This study has been also partially supported by the European Network of Excellence ReSIST (Resilience for Survivability in IST), funded by the European Commission Framework Program 7, Contract Number 026764, http://www.resist-noe.org.

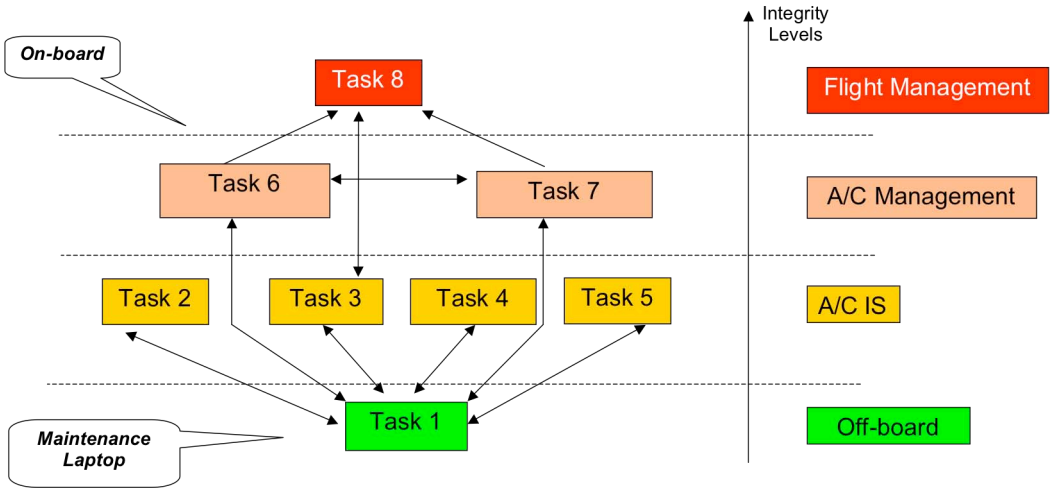

Figure 1. Maintenance operation tasks

2. MAINTENANCE OPERATION CASE STUDY

Maintenance operations present an important economic challenge for airline companies since they define the time needed by ground technicians to authorize take-off of the concerned aircraft. The shorter this time, the more profitable the aircraft for the company. Aircraft maintenance is ensured by operators who analyze the aircraft flight reports and according to maintenance manuals, decide to launch a set of tests on the components detected as potentially faulty. Currently, many maintenance manuals are still not electronic, and even the electronic ones are prohibited from interacting directly with the aircraft in order to prevent any risk of corruption of critical components. The presence of a human operator is the only guarantee that maintenance operations follow a safe procedure that does not corrupt critical systems.

For future aircraft, it is envisaged that the manuals will be stored on a Maintenance Laptop (*ML*), which could be connected to the aircraft through Ethernet or wireless connections. The maintenance operator would use the *ML* and move freely in and around the aircraft. He would inspect directly the faulty components detected by flight reports, follow the procedures indicated on the *ML*, and launch directly from the *ML* the set of tests that he formerly launched from specific embedded devices [2].

In ArSec, we analyzed the information flow introduced by this new aircraft maintenance approach. These flows are identified according to the source task (the task that generates information) and the destination task.

Figure 1 presents a set of task interactions during a typical maintenance operation. The maintenance laptop executes *Task 1*. This task is the least critical one since it is embedded on a portable machine running a COTS operating system, which, from a critical avionics perspective, is the quintessence of untrustworthiness. The other tasks are onboard the aircraft, and are thus validated to the Aircraft Information System level (A/C IS) or higher, according to the criticality of each task. Data flows are then classified into safe flows and unsafe flows.

If we consider two tasks *A* and *B* such that *Task A* has an integrity level greater than *Task B*, then any flow of data from *Task B* towards *Task A* is considered as unsafe. Flows from *Task A* to *Task B* are safe, as are flows between tasks at the same integrity level. In our case study, all flows coming from *Task 1* and going onboard are unsafe, which would violate the “diode” principle presented in Section 1, and thus have to be fully controlled.

Several solutions have been proposed to allow communication between applications having heterogeneous integrity levels, including Biba's model [3], Clark and Wilson's model [4], Integrated Modular Avionics (IMA) [5], and the causal dependencies principle [6].

In ArSec, we consider and extend Totel's model [7], which, by checking all the information flows existing in a system, allows bidirectional communication between software components at different integrity levels. Section 3 presents this model and Section 4 details the considered solution to implement maintenance operation tasks on one physical machine according to the model.

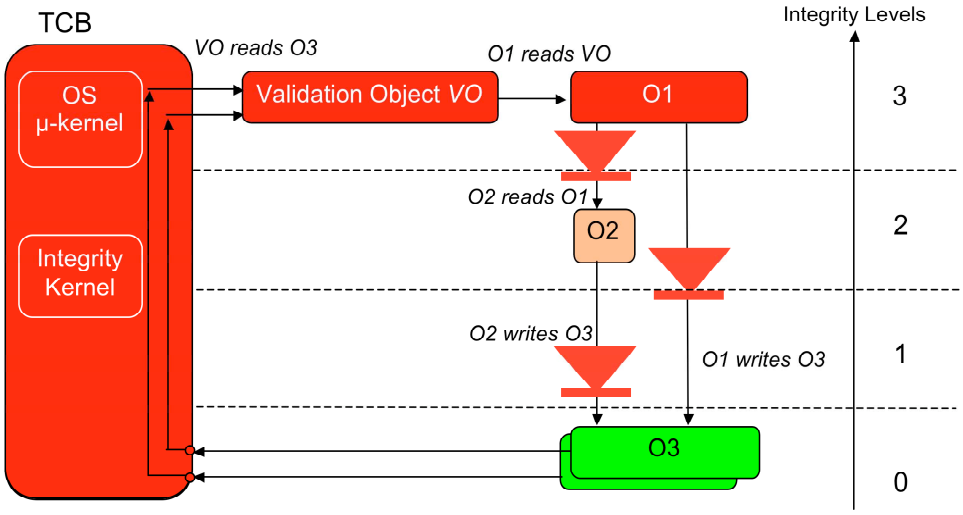

Figure 2. Totel's model

3. TOTEL'S MODEL

Figure 2 presents the global software architecture that has to be deployed to allow bidirectional communication between objects having different integrity levels.

Objects may be software components or complete hardware/software sub-systems. The direction of arrows corresponds to the direction of logical information flow.

As mentioned earlier, downward communication does not adversely affect operation of critical tasks. Thus, downward communication does not need any specific checking and is allowed. However, direct upward communication is forbidden. The only way for a high-integrity-level object to receive information from a low-integrity-level one is to use a specific communication channel. This channel must be ensured by a Trusted Computing Base (*TCB*) that offers a specific entry point for each upward communication flow. To avoid any violation of this principle, the *TCB* must mediate all communication within the system and implement the diode mechanism to prevent any direct upward information flow. Only specifically authorized upward flows are allowed through the TCB, and each of these flows must be directed to a Validation Object (*VO*).

In Figure 2, level 0 object *O3* needs to send information to object *O1*, which is the most critical one in the system. We have to ensure that information coming from the lower levels does not alter the behavior of *O1*.

To ensure this, all information coming from lower levels has to be validated by a Validation Object (*VO*) associated with *O1*. *VO* is the only object in the model allowed to receive data from lower integrity levels. It must have the same integrity level as the object for which it validates information. *VO* implements a fault tolerance mechanism (e.g., a decision algorithm) to produce data validated at the required level based on data coming from less validated objects (e.g., *O3*). The implementation of *VO* is tightly linked to the type of information that it checks. For example, in Figure 2, *VO* can compare or vote on redundant input data provided by two or more instances of object *O3*.

The TCB of [7] consists of the dedicated upward communication channels, a microkernel and an integrity kernel. The microkernel is used to control the resources (e.g., memory, CPU, etc.) shared by the different tasks, to prevent less critical tasks from consuming resources needed by more critical tasks. The integrity kernel implements integrity checking mechanisms that ensure that all information flows (except the dedicated TCB channels) are safe, i.e., respect the safety requirement expressed in Section 2. The TCB needs to interact with the most critical objects of the system, so it has to be validated at the highest integrity level.

Totel's model presents some limitations [8]: it considers only a single physical machine, and the use of a unique TCB makes its validation complex. However, it is the only integrity model that allows bidirectional communication between heterogeneous integrity levels.

In ArSec, we decided to implement laptop-based maintenance according to this model, using diversification and virtualization, as briefly described in the next section.

4. IMPLEMENTING MAINTENANCE OPERATIONS USING VIRTUALIZATION

As described in Section 2, maintenance operations can be launched from a single physical laptop: *ML*, which possibly runs a COTS operating system such as Windows-XP®. Consequently, the maintenance software may run on a potentially corrupted platform, especially if the considered laptop may at some time be connected to the Internet (e.g., is also used to browse the Web or for e-mail, and is thus potentially vulnerable to malware such as worms, viruses, Trojan horses, etc.).

In Totel's model, validation objects are the only entities in the system allowed to receive data from lower-integrity level objects. *VO*s implement fault tolerance mechanisms [9] to raise the integrity level of received information. These mechanisms are essentially based on redundancy and diversification of information sources. The diversification can be topological (by using geographically-separated machines having the same configuration), functional (by using different designs aimed at meeting the same requirement), temporal (by executing the same software at different moments), physical (by using different hardware) and/or translational (by using different compilers).

The use of a single physical machine in the potential presence of malicious faults makes functional diversification the most suitable redundancy to support maintenance operations. In ArSec, the maintenance task is diversified over (at least) two different execution platforms.

The first version is implemented for a Windows-XP platform, the second for Linux. The only way to execute these two implementations simultaneously, on the same physical machine, is to use the virtualization technique. Each version of the maintenance task is executed on a distinct virtual machine running its own guest operating system. The result of the two versions is then transmitted to a validation object, which runs on a third virtual machine (cf. Figure 3).

The figure presents the interaction between the *ML* and the object *O2* (representing a task running in the aircraft). *O2* communicates only with the virtual machine monitor (*VMM*).

In our study, we use Xen [10] as a virtual machine monitor. The Linux and WindowsXP virtual machines are COTS and are not trustworthy, they are considered as DAL-E (i.e., low criticality) software components. The two versions of the maintenance task are installed on these virtual machines. The validation object *VO* has to be validated at least at the level of *O2*, since it communicates information from untrustworthy virtual

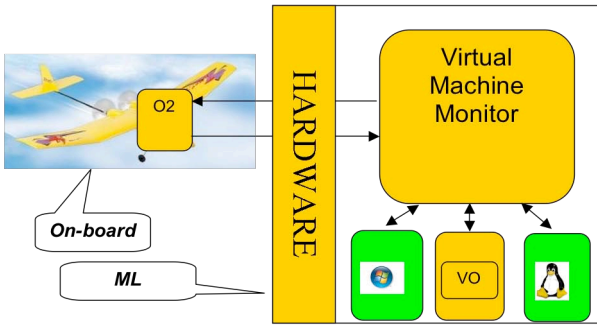

Figure 3. Maintenance task using virtualization

machines to the more critical tasks on the aircraft. Consequently, *VO*, *VMM* and the guest virtual machine on which *VO* runs have to be validated to at least the same level as *O2*.

The use of a single physical machine executing different replicas of the same software raises the problem of human machine interaction. Indeed, the maintenance laptop has only one keyboard, one mouse, one network adapter, one screen, etc. These different devices are virtualized for each operating system. The use of different virtual interfaces would mean that the human operator would have to enter the same inputs into each virtual machine, which would then need to compare the results. It is clear that this interaction can lead to human interaction error.

Consequently, it is important to ensure that the operator has just a single interface, in the following way: the inputs are captured, then forwarded to each virtual machine, the outputs of each virtual machine are captured, compared with the other machines' outputs and then validated by the Validation Object. The latter decides whether the output is correct (so it is displayed) or not (so an error message is raised and the whole machine is stopped). The different levels of comparison and the final prototype architecture are described in [2].

5. CONCLUSION

In this paper, we first presented a case study identified with Airbus, in which functional information flows have to be communicated from low integrity levels to high integrity ones. Such upwards information flows could considerably simplify and facilitate maintenance operations for future aircraft, which would have an important economic impact for airline companies. We have then presented the integrity model that we use to enable such flows: the Totel model.

Finally, we have described some implementation aspects of the maintenance task, using virtualization. The choice of this technique was imposed by the case

study, and features some challenges, specially related to the validation mechanisms and synchronization between diversified tasks, and also the human machine interface, on which we are currently working.

6. REFERENCES

[1] Y. Laarouchi, Y. Deswarte, D. Powell, J. Arlat, and E. de Nadai, "Criticality and Confidence Issues in Avionics," in *12th European Workshop on Dependable Computing (EWDC)*, Toulouse, France: 2009.

[2] Y. Laarouchi, Y. Deswarte, D. Powell, J. Arlat, and E. de Nadai, "Enhancing Dependability in Avionics Using Virtualization," in *EuroSys Workshop on Virtualization Technology for Dependable Systems*, Nuremberg, Germany: 2009.

[3] K.J. Biba, "Integrity Considerations for Secure Computer Systems," *MITRE Co., Technical Report ESD-TR 76-372*, 1977.

[4] D. Clark and D. Wilson, "A Comparison of Commercial and Military Computer Security Policies," in *IEEE Symposium on Security and Privacy*, Oakland: IEEE Computer Society Press, 1987, pp. 184-194.

[5] J. Rushby, "Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance," *NASA report*, CR-1999-209347: 2000.

[6] B. d'Ausbourg, "Implementing Secure Dependencies over a Network by Designing a Distributed Security Subsystem," in *Computer Security — ESORICS 94*, 1994, pp. 247-266.

[7] E. Totel, J. Blanquart, Y. Deswarte, and D. Powell, "Supporting Multiple Levels of Criticality," in *28th Annual International Symposium on Fault-Tolerant Computing*, 1998, pp. 70-79.

[8] Y. Laarouchi, Y. Deswarte, D. Powell, and J. Arlat, "Safety and Security Architectures for Avionics," *Doctoral Consortium (DCSOFT 2008)*, Porto, Portugal: 2008.

[9] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE TDSC*, vol. 1, 2004, pp. 11-33.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *9th ACM Symposium on Operating Systems Principles*, ACM Press, 2003, pp. 164-177.