OPEN SOURCE SOFTWARE IN CRITICAL SYSTEMS

Motivation and challenges

Philippe David[1], Hélène Waeselynck[2] and Yves Crouzet[2]
[1]ESA/ESTEC, Keplerlaan 1, 2200 AG Noordwijk, The Netherlands; [2]LAAS-CNRS, 7 av du Colonel Roche, 31077 TOULOUSE Cedex 4, France

Abstract: This paper summarizes the main conclusions and recommendations from a Working Group on "Open Source Software and Dependability". The Group was launched in the framework of a cooperative structure, a Network for Dependability Engineering, and gathered representatives of ten academic and industrial organizations.

Key words: Open source software, COTS, critical systems, certification, wrapping, validation.

1. INTRODUCTION

While Open Source Software (OSS) is penetrating the software business at large, software-intensive applications having high dependability requirements have been little concerned so far. At a first glance, introducing OSS into such critical systems seems risky. The constraints imposed by adherence to certification standards may be deemed irreconcilable with the open-source development model. Still, the question is being taken seriously in domains like transportation, space, or nuclear energy: gaining acceptance into dependable systems might well be the new challenge for some mature OSS products.

It is no longer true for the development of highly dependable systems to be solely based on dedicated, specific equipments, with cost not being a primary concern. Today, market pressure – together with the increasing complexity of applications – has led companies to consider the use of off-the-shelf components in parts of their systems. Compared to proprietary

commercial off-the-shelf components (COTS)[1], OSS components may offer an interesting alternative. Availability of the source code may obviously be a strong advantage from a dependability perspective. There also are some strategic concerns, like avoiding dependence on monopolistic support, that may turn the balance in favor of OSS.

This is why a Working Group on "Open Source Software and Dependability" was launched in May 2001 by the **Network for Dependability Engineering**[2] (Réseau d'Ingénierie de la Sûreté de fonctionnement – *RIS*). The working group included representatives from ten academic and industrial organizations: Airbus, Astrium, ESA, INRETS, IRISA, LAAS-CNRS, LSV, SNCF, Technicatome, Thales. The results of the Group were concretized by the publication of a book [1] (in French) in September 2003, and by the organization of an open seminar that gathered more than 80 participants.

This paper summarizes the main conclusions and recommendations from the Group. Section 2 discusses the use of OSS as an alternative to proprietary COTS, and compares their respective advantages and drawbacks. Section 3 is concerned with certification. It is studied to what extent the use of OSS may be compatible with certification requirements. Section 4 presents architectural design principles, so as to host OSS in a dependable way. Section 5 discusses OSS validation issues. Section 6 concludes on recommendations to promote the use of OSS in critical applications.

2. OSS AS AN ALTERNATIVE TO COTS

Critical computerized systems can be life-critical (such as a fly-by-wire control system) or money-critical (such as a telecommunication satellite). Whatever the critical domain, the increasing complexity of applications has yielded more and more functions to be implemented by software. Cost constraints, as well as interoperability ones, have favored engineering practices based on software reuse and standardized interfaces. System providers were pushed to focus their software development activity on domain-specific functions, and to integrate standard off-the-shelf components for non-specific ones, such as executive support or communication protocols.

[1] Throughout this paper, the "COTS" acronym will denote *proprietary* commercial software, as opposed to open source software. This is the usual meaning of "COTS" while, strictly speaking, nothing prevents commercial software from being distributed under the terms of an open source license.

[2] http://www.ris.prd.fr

Proprietary commercial software was first considered for integration. Such COTS components are usually purchased as black boxes, with no access to the source code. A support contract is concluded with the supplier, in order to facilitate the integration of the COTS into the system and perform the maintenance actions. Unfortunately, such a solution might not be satisfactory for critical systems.

2.1 Drawbacks of Proprietary COTS

The quality of the technical support is primordial, but can largely vary depending on the supplier. Moreover, critical systems are long-living systems (e.g., from 15 to 40 years of operation for systems developed by members of the Working Group). In order to avoid re-entering a certification process, there is a need to limit the upgrades and "freeze" the components versions for long periods of time (typically five years) with only corrective maintenance actions. The need for stability may conflict with the supplier's commercial strategy, which is to release new versions at a frequent rate. The supplier may have no interest in ensuring high quality support for obsolete versions of the COTS. Also, due to the high competition in the software market, the supplier may have disappeared before the end of the system's operational life. The risk should be considered from the very beginning. If no alternative support can be found, an ultimate solution can be for the integrator to acquire technological expertise on the COTS, which cannot be achieved for black-box components.

Should the system suffer major failures due to the unreliability of the COTS, the responsibility of the supplier would be limited. This is fairly understandable: the COTS was not designed to fulfill ultra-high dependability requirements, and it is the responsibility of the integrator to control the technical risks induced on the critical system. An acute problem is then the low expertise the integrator has on the COTS. Once again, black-box usage may be unacceptable even if part of the dependability case is done with the support of the supplier.

These risks may force integrators to negotiate the purchase of the source code. This is usually obtained at a prohibitive price. When critical systems only represent a marginal part of the supplier's business, the integrator is not in strong position to conduct the negotiation.

Such problems have yielded integrators to consider the use of open source components for which, by definition, there is no restriction to access the source code.

2.2 The OSS Alternative

OSS licenses give the users the right to read, modify and redistribute the source code. OSS is generally associated with a community development model, the so-called "bazaar" [2]. But underlying practices are actually so disparate that it is not possible to refer to a single model. For many successful OSS products, the development and maintenance activities are more structured than the mythical bazaar. As for (proprietary) COTS, there are companies selling support on some OSS. Since the justification for these companies to exist is their deep expertise on the product, not the delivery of the product, the supplied support may even be of higher quality than in the case of classical COTS products.

Basically, the risks induced by the integration of off-the-shelf (OTS) components, whether proprietary or open-source, are similar. But in the latter case, availability of the source code should allow the integrator greater latitude to mitigate and control the risks. The risks include:

- An increasing gap between the frozen system-specific version and the new versions of the component.
- The disappearance of the support company.
- Technical risks due the component's failures.

There are also risks associated with the terms of the license. Here, open source licenses have some specificities that must be understood, such as the contamination effect of the GPL license.

From a technical viewpoint, the main challenge is to produce a dependability case that is accepted by the Certification Authority. For COTS components, the case has to be prepared with the support of the supplier, and still necessitates a strong involvement of the integrator. For OSS components, additional scenarios become possible: a consortium of integrators may share part of the effort required by certification, as soon as they have a common interest into a given open source product.

3. THE CERTIFICATION CHALLENGE

The *RIS* Working Group examined the various certification standards that were relevant to the application domains of its members: avionics systems (DO-178 B [3], ARP 4754 [4]), railway systems (CENELEC 50128 [5]), space systems (ECSS-Q-40A [6]), nuclear systems (IEC 60880 [7]), as well as a generic standard (IEC 61508 [8]). Whatever the standard, the associated requirements and recommendations depend on the criticality level of the software component: the higher the level, the more demanding the standard. The classification into levels, determined from the impact a failure may have

on the system, was found quite homogeneous (see the correspondence we established in Table 1).

Table 1. Software Criticality Levels

	Railway systems	Avionics systems	Space systems	Nuclear systems
No impact	SIL 0	E	/	/
Impact on the system	SIL 1-2	C-D	critical	B-C
Loss of human lives	SIL 3-4	A-B	catastrophic	A

Our analysis is that the use of OSS is not realistic at the highest levels (e.g. SIL 3-4 for railway systems) – unless the software component was purposely developed to meet the certification requirements, and turns out to be distributed under an open source license for some strategic reasons. Since the requirements are very stringent, it is not possible to fulfill them *a posteriori*. Intermediate levels (e.g., SIL 1-2) are quite conceivable, but the integrator will have to spend some effort to build appropriate documentation, and perform additional validation and design rework activities. Some standards only define the objectives to be reached (e.g., DO-178 B) while others provide recommendations on the methods to be used (e.g., CENELEC 50128). Our analysis is that whatever the application domain, the same methods can be used to fulfill comparable objectives. This leaves open the possibility, for consortium of users, to share part of the required effort. Obviously, for the lowest levels (e.g., SIL 0), the use of OSS components should not be a problem (at least from a certification viewpoint).

As mentioned in the ARP 4754 document, the criticality level can be reduced if appropriate architectural features are used. As an example, triple redundancy may be used to implement a level A avionics component based on level C components (note, however, that the dissimilarity and independence of the three components must be established with an A assurance level). In our opinion, the ability to build critical systems from less critical components, through architectural solutions, is the key to allow the introduction of OSS. Once again, it should be possible to identify generic (not domain-specific) solutions for hosting OSS. Two of them, namely system partitioning and component wrapping, are briefly discussed in the next section.

4. HOSTING ARCHITECTURES

The hosting architecture should address different problems:

- The system has to be protected from the OSS failures.

- The OSS functionalities may be too abundant (compared to the needs of the critical system), or they may necessitate some adaptations.
- The architecture should be able to cope with the evolution of components (long term maintenance).

In order to prevent error propagation from unreliable components to critical ones, a classical solution is partitioning. The system is partitioned into subsystems such as the failure of any subsystem is confined into this subsystem. The subsystems have the criticality of their most critical component. *Physical* partitioning corresponds to a strict segregation of subsystems, while *logical* partitioning enable software of different criticalities to coexist on the same execution platform. Logical partitioning involves both spatial (e.g., via memory protection mechanisms) and temporal (e.g., via scheduling policies, watchdogs) isolation. Such techniques, widely used in critical systems, are relevant from the perspective of hosting OSS, as well as COTS, components.

Another protection technique is wrapping. A wrapper encapsulates a software component, enabling to filter and control invocations of the component's services, and to incorporate error detection and recovery mechanisms. Hence, wrapping is appropriate to adapt the OSS interface to the system's needs (e.g., by restricting the set of used functionalities), to limit the impact of component evolution on the architecture (only the wrapping layer has to be reworked when the component is changed), and to add protection mechanisms. Examples of fault tolerance wrappers for real-time executive supports can be found in [9, 10]. Once again, we believe that part of the effort to develop such wrappers could be shared by consortium of users, because there are common needs whatever the application domain. Wrapping can be applied to both OSS and proprietary software components, but the achievable degree of protection is strongly dependent on the access to the source code: sophisticated error detection and recovery mechanisms require observability and controllability of the internal state of the target component.

5. VALIDATION OF OSS

One manifesto of the open-source movement [2] expresses the following idea: "Given enough eyeballs, all bugs are shallow." According to the author, given the large number of users "eager" to contribute, every bug should be quickly located and fixed.

Not surprisingly, this view has generated much controversy (see e.g. [11, 12]) and there are strong arguments showing that the claim is over-optimistic. Note that, in practice, we observe that many successful OSS are

developed and maintained following a more centralized approach, with most of the validation effort relying on an organized board of experts. Anyway, from the perspective of certification, the existence of a large number of "eyeballs" is not sufficient for justifying confidence on the OSS. Hence, the integrator will have to look for more convincing evidence, and to supplement the case by additional validation effort.

Whatever the certification standard, testing plays an import role in producing evidence.

Conformance testing aims at verifying that the OSS conforms to its specification. The test documentation must ensure traceability between functional requirements and test cases. Unfortunately, the test suites provided in open-source distributions are often impossible to exploit, because they have been determined in an *ad hoc* manner, without any concern for traceability analysis. Worse, there is sometimes nothing such as a software requirements document. For OSS products distributed by support companies, it may be the case that testing has been designed more rigorously, based on an elicitation of requirements. But the test suites and associated documentation are generally not included in the distribution, because they are part of the strategic know-how of the company. Fortunately, the problem can be mitigated for OSS products implementing interface standards. As soon as the interface specification is publicly available, consortium of users may share the effort to produce conformance tests that can be exploited for certification purposes. A good example is the Open Group Consortium[3], from which POSIX conformance test suites can be obtained.

Besides conformance testing, we believe that *robustness testing* is an important issue from a dependability viewpoint. Robustness testing checks the OSS behavior in the presence of faults or stressing operational conditions. It may serve several purposes:

- comparing the robustness of candidate products. Given a family of similar products, robustness testing results can contribute to the decision of which one to select for integration in the system.
- characterizing the failure modes of a selected component. The test results may then support subsequent design decisions, like the design of wrappers supplying protection mechanisms.
- Assessing the effectiveness of implemented wrappers (in that case, the tested artifact is the wrapped OSS).

Some results can already be found for a number of executive supports, COTS or OSS: microkernels [13], operating systems [14], midlleware [15].

[3] http://www.opengroup.org

Incidentally, they show that open-source executives are not less robust than proprietary ones.

Depending on the target criticality level, more formal verification techniques should also be considered.

Automated *static analysis* allows properties of the source code to be proven. Examples of properties include compliance with coding standards, or the absence of errors like buffer overflows, out-of-bounds arrays access or race conditions for concurrent tasks. Real-time properties can also be addressed. As an example, in [16], static analysis is used to determine the worst case execution times for an open-source real-time kernel.

Model-checking involves checking that a model, specified under the form of a state automaton, satisfies a certain property (usually specified as a temporal logic formula). Typical use of model-checking is for the verification of protocols. Ideally, model-checking should be performed at the specification level, before the protocol is implemented. However, it has also proven useful to verify existing protocols *a posteriori* [17, 18]. In this case, access to the source code is necessary to build a proper abstraction (i.e., the model fulfills the property only if the original source code also fulfills it).

All these techniques may be used to produce common certification material for an existing OSS.

6. CONCLUSION AND RECOMMENDATIONS

The conclusion of the Group is that OSS products are a strong opportunity for critical systems integrators. Their use in operational systems is quite conceivable, with appropriate architectural solutions to lower the criticality level, and actions to reach the certification standard for that level.

It was clear from the beginning that the introduction of OSS in critical system would require a pro-active approach. The open-source movement has emerged with no particular concern for critical systems, and it is up to us to invent the means to take advantage of this movement. In particular, making OSS products compatible with a certification process cannot be achieved for free. Also, it was felt important by the Group to issue recommendations not only on how to *use* OSS, but also how to *contribute* OSS. If OSS is a strategic concern, then critical system actors should consider entering the "virtuous circle" so as to support the movement. Ideally, measures to promote OSS in critical systems would include:

- Intiating an evaluation center for OSS.

  The center would launch evaluation campaigns for some OSS of interest and make the results publicly available. The center would be a means, for critical system players, to share information on relevant

products such as executive supports or protocols. It would also enable OSS developers to receive feedback on their product.

- Initiating a common case for certification.

  The view expressed in this paper is that part of the material to be produced can be exploited whatever the application domain, as soon as the standards impose similar requirements for the OSS. It would be necessary to initiate a project to demonstrate this idea, with partners from different application fields, both system suppliers and assessors. The project would focus on one OSS of interest, a target criticality level (say, the intermediate levels in Table 1), and would investigate the feasibility of satisfying the certification requirements by means of a common set of methods.

- Demonstrating architectural principles to host OSS

  Hosting architectures will be based on redundancy, partitioning and wrapping. However, these architectural principles are too general and cover a wide spectrum of solutions: there is a need for more precise recommendations. A small subset of solutions should be investigated, prototyped and validated on a common platform. A candidate platform is the Integrated Modular Avionics (IMA), which is characterized by an open architecture using standardized interfaces.

  A contribution to the community would then be the availability of open-source wrappers, implementing protection mechanisms for OSS components.

- Promoting open-source development environments

  In this paper, we have focused on OSS used as components of critical systems. But OSS can also penetrate the development environments used to produce critical software.

  OSS development toolsets would present many advantage, including independence on monopolistic support, easier integration of tools through non-proprietary interfaces, and ability to benefit from technological advances from the academic world. We recommend that projects be launched to investigate viable alternatives to the proprietary development environments currently in use.

These recommendations would require a strong involvement of the critical system industry, as well as political and financial support from public organizations. However, we believe the effort is worth while.

ACKNOWLEDGEMENTS

This paper reports on the results of a collaborative work performed in the framework of the *RIS* network. We wish to thank all our colleagues who contributed to the Working Group on "OSS and Dependability": Béatrice Bérard (LSV), Philippe Coupoux (Technicatome), Yseult Garnier (SNCF), Serge Goiffon (Airbus), Georges Mariano (INRETS), Vincent Nicomette (LAAS-CNRS), Luc Planche (Astrium), Isabelle Puaut (IRISA), and Jean-Michel Tanneau (Thales). We are also grateful to the *RIS* Advisory Board, and particularly its coordinator Jean Arlat, for their constant support and constructive comments throughout the life of the Working Group.

REFERENCES

[1] B. Bérard, P. Coupoux, Y. Crouzet, P. David, Y. Garnier, S. Goiffon, G. Mariano, V. Nicomette, L. Planche, I. Puaut, J-M. Tanneau, and H. Waeselynck, *Logiciel libre et sûreté de fonctionnement – cas des systèmes critiques*, P. David and H. Waeselynck (Eds), Hermes Science, ISBN 2-7462-0727-3, 2003.

[2] E.S. Raymond, "The Cathedral and the Bazaar", *First Monday*, vol. 3, no. 3, March 1998. http://www.firstmonday.dk/issues/issue3_3/raymond/index.html

[3] RTCA / DO178B, "Software Considerations in Airborne Systems and Equipment Certification", 1992.

[4] SAE – ARP 4754, "Certification Considerations for Highly-Integrated or Complex Aircraft Systems", 1995.

[5] CENELEC EN 50128, "Railway applications — Communication, Signalling and Processing Systems — Software for Railway Control and Protection Systems", 2001.

[6] ECSS-Q-40A, "Space Product Assurance – Safety", 2002.

[7] IEC 60880, "Software for Computers in the Safety Systems of Nuclear power stations", 1986.

[8] IEC 61508-3, "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems. Part 3: Software requirements", 1998.

[9] E. Anceaume, G. Cabillic, P. Chevochot, I. Puaut, "A flexible run-time support for distributed dependable hard real-time applications", *Proc. of the 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'99)*, pp. 310-319, St Malo, France, May 1999.

[10] M. Rodriguez-Moreno, J-C. Fabre, J. Arlat, "Wrapping Real-Time Systems from Temporal Logic Specifications", *Proc. 4th European Dependable Computing Conference (EDCC-4)*, A. Bondavalli and P. Thévenod-Fosse (Eds), Springer Verlag, LNCS no. 2485, pp. 253-270, 2002.

[11] N. Bezroukov, "A Second look at the Cathedral and the Bazaar", *First Monday*, vol. 14, no. 12, December 1999. http://www.firstmonday.dk/issues/issue4_12/bezroukov/index.html

[12] J. Viega, "the Myth of Open Source Security", *Earthweb.com*, may 2000. http://www.developer.com/open/article.php/626641

[13] J. Arlat, J-C. Fabre, M. Rodriguez, F. Salles, "Dependability of COTS Microkernel-Based Systems", *IEEE Trans. on Computers*, vol 51, no. 2, pp.138-163, February 2002.

[14] P. Koopman, J. DeVale, "Comparing the Robustness of POSIX Operating Systems", *Proc. 29th Int. Symp. On Fault-Tolerant Computing (FTCS-29)*, Madison, WI, IEEE CS Press, pp. 30-37, 1999.

[15] E. Marsden, J-C. Fabre, "Failure Mode Analysis of CORBA Service Implementations", *Proc. IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware'2001)*, Heidelberg, Germany, 2001.

[16] A. Colin, I. Puaut, "Worst-Case Execution Time Analysis of the RTEMS Real-Time Operating Systems", *Proc. 13th Euromicro Conf. on Real-Time Systems*, Delft, NL, pp. 191-198, June 2001.

[17] K. Havelund, A. Skou, K.G. Larsen, K. Lund, "Formal Modeling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL", *Proc. 18th IEEE Real-time Systems Symposium (RTSS'97)*, San Francisco, CA, pp. 2-13, December 1997.

[18] B. Bérard, L. Fribourg, "Automated Verification of a Parametric Real-time Program: the ABR Conformance Protocol", *Proc. 11th Int. Conf. Computer-Aided Verification (CAV'99)*, Springer Verlag, LNCS no. 1633, pp. 96-107, 1999.