Click here to view PowerPoint presentation; Press Esc to exit

The Utility of Open Source Software in Military Systems

Agustín Izquierdo Esperón, José Prieto Muñoz
GMV, S.A.
C/Isaac Newton 11, PTM, Tres Cantos, 28760 Madrid, Spain

aizquierdo@gmv.es jprieto@gmv.es

Jean Michel Tanneau
THALES R&T
L'Orée de Corbeville, BP 56, 91401, Orsay, France

jean-michel.tanneau@thalesgroup.com

ABSTRACT

The MILOS (Military Systems based on Open-source Software) project was a European research program in the Eurofinder framework, attached to the CEPA 6 and co-financed by the Ministry of Defence of France and Spain. The companies involved were THALES and GMV.

The MILOS project aimed to demonstrate benefits of Open Source Software in large software based military systems, by casting off constraints inherent to traditional proprietary COTS and by taking advantage of new opportunities, which occur thanks to OSS's characteristics.

The goal of MILOS was to analyse the feasibility of the use of Open Source software in military systems. This paper is a brief introduction to Open Source and its possible eventual relationship with both government and military agencies.

1. INTRODUCTION

2. WHAT IS OPEN SOURCE SOFTWARE

A simple definition of 'open source' software, sometimes referred to as 'free' or 'libre software', is that the software's source code is available to the public. However, simply making the source code available does not necessarily mean that a program meets the definition of open source software. In order to classify a program as open source software, its license must comply with some criteria. These criteria, designed to represent the *consensus* view on just what open source software is, are the following:

- Source code

  The program must include the source code, and must allow distribution in source code as well as in compiled form. Where some form of a product is not distributed with source code, there must be a well-publicised means of downloading the source code, without charge, via the Internet. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a pre-processor or translator are not allowed.

Paper presented at the RTO IST Symposium on "Coalition C4ISR Architectures and Information Exchange Capabilities", held in The Hague, The Netherlands, 27-28 September 2004, and published in RTO-MP-IST-042.

- Integrity of the author's source code

It is strongly recommended that *"none of the authors restrict any files, source or binary, from being modified"*. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to have a different name or version number from the original software.

- Derivative works

The license must allow modifications and derived works and must allow them to be distributed under the same terms of the license of the original software.

- 'Free' redistribution

The license may not restrict any party from giving away (or selling) the software.

So, 'closed source' or 'proprietary' software is defined as software that does not meet the criteria stated above. For example, with closed/proprietary software, a typical user is neither allowed to access to the source code nor to modify the source code, and is usually prohibited from redistributing the software. Virtually every type of open source software has some variant of an open source software license associated with it. In fact, the use of open source licensing is an essential element in making open source program 'open source.'

One should also note that, when the word 'free' is used, it does not necessarily mean the same thing as 'zero price' 'free', in relation to open source software, means 'freedom' from constraints.

3. SUMMARY OF WORKS AND OUTPUTS OF THE MILOS PROJECT

The first stream of the performed work was to identify and to analyse stakes of OSS within software based military systems. Then to propose an organisation that enables to benefit from OSS while reducing its related risks. Such analysis and proposals, which followed, have been carried on by comparison with the COTS approach.

The second stream of the work was devoted to apply those findings to the development of an OSS based demonstrator.

These two streams were strongly connected, the first one providing guidelines and process to the second one which, in return has provided fruitful feedback to tune and adapt findings of the theoretical approach.

The first phase dealing with the theoretical study started with the compilation of the requirements of the client/supplier chain, where the OSS world was first understood, with its actors and their motivations. An explanation of how particularities and qualities of OSS components lead to opportunities for the system was given: right features, maintainability, tailoring, favourable licensing and pricing, standard respects, market share (de facto standard), quality/reliability, performance, security, total cost of ownership, scalability. Special attention was paid to how some of these characteristics (mainly adherence to standards and licensing schema) should help to the long-term maintenance issue of military systems (compared to a COTS based solution).

However, three characteristics of OSS were pointed out, which although bringing opportunities, introduce threats for the system: its licensing schema, to be an "external component" and to be in "continuous evolution". A dedicated OSS "evaluation / selection process" and emphasis on "Configuration management" should dramatically mitigate related risks while enabling to fully benefit from OSS. These issues, solutions and related organisational changes are highlighted in the following section of this paper.

As the proposed change to the evaluation/selection process was innovative, it was tested in the context of the development of the "Milos demonstrator". The scope of the development was first refined through a technical requirements specification, where requirements related to the development infrastructure and to the targeted application domain (functional subset of a Command and Control Information System) were highlighted.

Based on the list of product segments as identified in the technical requirements specification, a general OSS market survey was provided. It was used as input to the evaluation of candidates for each segment relevant to the demonstrator, evaluation results were in a Requirements compliance and trade-off report.

The selection and evaluation process dealt with both:

- Technical topics, which are different for each family of tools. Every tool must be compliant to some technical criteria.
- Industrial topics, concerning support, sponsors, documentation… For Open Source, industrial topics play an important role in order to limit risks.

Information for both parts is collected, beginning with the technical topics. Since the list of tools in the market survey report were, more or less, in compliance with the technical criteria (technical requirements specification), all of them were "industrially" investigated.

In both cases, a set of criteria were collected and weighted (regards the relative significance). Later those were used to qualify the tools.

Selection of the components to be integrated in the demonstrator followed results of the evaluation phase with a few of exceptions, which took care of a set of constraints defined for the demonstrator.

A test scenario was released to functionally validate the demonstrator and therefore the suitability of OSS for military systems development. It was also validated through a previously defined validation plan.

To conclude this theoretical work, relationships between the different actors were studied: the OSS community, the technology provider (OSS support provider), the systems' integrator and the customer. This study was done from the business points of view of the technology provider and the integrator roles.

The second phase of the project, the practical application, started with the installation of the selected OSS components with special care to compatibility (interoperability). Then came the "glueing" of the OSS components together with specific developments according to specifications previously drafted. After unit testing and final integration, the demonstrator was evaluated through the operational and the technical scenario previously defined.

4. THE EVALUATION PROCESS

The evaluation is twofold: the technical evaluation (in order to assess technical capacity) and the industrial evaluation (in order to assess confidence in mid-term).

But, prior to any kind of evaluation of an OSS candidate, its licensing schema is an issue that should be taken into account through a legal analysis. According to how the OSS component is used in the system, some licensing schema may impose the redistribution of part (or even the entire) system under the same license of the OSS component (such license is called “viral” – e.g. the GNU GPL). In such a situation, the system’s integrator ‘s intellectual property rights related to those “OSS license-affected” part are altered, he is not allowed to release this part of the system which uses the OSS in a proprietary system. Consequently, the legal analysis step is going to act as a first level filter to eliminate those components whose license doesn’t comply with the intellectual property right strategy of the integrator for the considered system.

The technical evaluation process can be depicted as follows: for each segment of products under consideration, a list of technical criteria (featuring technical capacity and functionality expected for candidates in the segment to be fulfilled) is defined with accurate metrics (how to measure). The criteria are “weighted” to feature their relative importance and to take into account the context of the evaluation (or targeted systems - in Milos, weights were set-up with the command and control information system context in mind). To allow comparison between candidates the valuation of criteria is normalised as well as weights.

Even after having checked the accuracy of a component to the system‘s technical requirements, the decision to use such “external component” is always a critical one, since it creates a dependency link between the system (which is responsibility of the integrators) and this component (which is responsibility of its provider). The objective of the “industrial” evaluation process is to assess and mitigate risks related to this kind of “dependency link” in a mid-term horizon (3 to 5 years; a longer period is meaningless in the rapidly moving software world).

For the industrial evaluation, a way to gain or assess the confidence in an OSS through an investigation of the project life was proposed. The points studied were: its lead maintainers, its developers and communities of users, the usefulness and reactivity of supporting information media (mailing lists, forums), consistency and accuracy of information released by the project (roadmaps, vision), the support of major software / computer industries.

As a conclusion, both the OSS evaluation process and its outputs were considered as major results of Milos, and it was our intend to share such information and to establish co-operation with other industrials, which share the same concerns (i.e. which look for integration of OSS in systems having a long-term maintenance constraint).

- When running the evaluation process, we have investigated more than 40 segments (families of products) covering 330 OSS. The effort needed to run the investigation was high, especially for the industrial-like investigation for which we have simplified the criteria. As a future direction, it could be worth to automate part of this investigation (e.g. to look for utilities that compute some criteria as: the average number of mails per day/month, the mean time between question and answer, growth of traffic, growth of the community, identification of main people that answered questions).
- The investigation output was a snapshot of the OSS market at the time the study was done, to keep alive such information we have proposed to release it to the public with the aim at federating users and experts communities through a dedicated portal. We’ve passed information to the eCots portal (www.ecots.org, software components open directory project) which provides products segmentation and facilities to host a community.

5. THE CONFIGURATION MANAGEMENT

One major point with OSS is its frequent releasing policy, which allows integrating rapidly bugs fixing and enhancements; this is also considered as a crucial feature to keep active the developers and users community.

On the other hand, system's integrators need to freeze all components that are part of the system for a period of time that is compatible with development constraints (integration of external components with the developed applications). Between keeping in touch with the latest edge of the project and stability, a trade-off needs to be taken.

Stability is needed for either the development phase of the systems but also to enable corporate capitalisation of know-how related to those components.

5.1 Stability and the system context

Freezing OSS components doesn't mean that it is no more possible to integrate some corrective patches; the decision to do so depends on the issues the patches correct (e.g. blocking bugs or security holes). It should also be noted that some corrections lead to a new version of the component; this is under the responsibility of the OSS lead maintainer, who decides depending on the severity.

In addition, the integrator may have modified the current version of the OSS to adapt it to the system's specificity; it is highly recommended that such modifications are tracked through patch mechanisms in order to stay compatible with what is done by the community.

Where there is no dedicated organisational entity to manage the OSS (see below), it's responsibility of the system's team to manage versions of the component, its patches as released by the community, its patches as released internally to cope with contextual adaptation and, all the related documentation. This is mandatory to be able to rebuild at any time the actual version of the component that is in use in the system and will make easier, further replacement of this component.

Such a version management of OSS is then integrated in the system configuration management to provide the full figure of what has been used and integrated.

5.2 Stability and the organisational context

According to the frequency of usage of a component within a system, a dedicated unit should be defined in the integrator's organisation. This unit will take charge of supplying, adapting (internal patches), releasing to businesses and deciding updates policy (when and which version of the OSS component to use, whether to apply patches as released by the community or not). This entity will act as interface between the systems development teams and the OSS communities; they may also provide some kind of support to their internal customers. In such a case this entity will manage the actual versions of components as released to systems developers (i.e. the OSS and subsequent patches that may have been applied to build the "internal reference" released to systems developers).

Systems development teams should use this customised version of the OSS as its reference and may adapt it to systems constraints if needed (contextual patches).

6. USE AND INTEGRATION OF OSS (THE TECHNICAL VIEW)

It was not the aim of the project to build a full-fledged operational system. Nevertheless the development of the Milos demonstrator allowed to point out some issues and to verify some advantages of OSS.

A summary of this phase, which emphasises key points related, could be the following:

- The Framework installation

Before installation was broached, an installation manual for each OSS tool to be use was prepared. It greatly helped the framework installation.

No real difficulty was encountered in the operating system installation (made in dual-boot with Windows) but it requires standard skills in Unix administration. Concerning tools, a choice had to be made between RPM or source installation. The second option was preferred, since it is considered better for the system mastering.

A set of tools had a really straightforward installation process (17 out of 28), some only needed configuration (9) and only two were really difficult to install. On tool (Subversion) had to be changed since it did not fulfil our needs, but this tool was not a final release (not mature enough). Mailing lists were of great help during the process.

The main conclusion was that installing the framework was quite easy except for a few tools. It was even easier since many tools were Java-based. Windows users may however have difficulties to find operating system configuration tools since it requires some skills in Unix administration and since they are less integrated. The counterpart is that flexibility in configuration was proved to be really good.

- The components integration

Main difficulties encountered concerned tools versions mismatch management, since many low-level tools are widely integrated in higher level tools, leading to large sets of dependencies. Moreover, each tool needs some prerequisite knowledge to be gathered (integrator experience).

A first phase during which a coherent set of tools, or even tools versions, is identified seemed necessary. All the development tools selected proved to integrate well.

Security aspects were however not considered for the demonstrator and this may lead to difficulties in a scope where tools from very different places are gathered. Anyway, this should be investigated deeper.

- Specific development

Some tools do not come with sufficient documentation (our opinion). The main source for developers support used was mailing lists. It has to be noticed that many questions were already answered for other developers and did not require posting a request. Even beginner's questions are answered.

Concerning development tools, Eclipse was of great help and was robust, powerful, portable and extensible (through plug-ins). Its main drawback is a huge need in resources (memory, CPU).

We consider that OSS software used did not bring difficulties in designing the architecture we planed. Some purely technical difficulties arose, in particular with xsl:fo, which is hard to write and maintain.

We globally consider that the tools selected for development provided good productivity and allowed us to build all the elements we planned.

Performance could be improved, at the cost of an in-deep evaluation of where time is consumed, which was only partly done during the demonstrator test phase, for it corresponds to an important effort.

- Impact on design

Related to impact on the systems design of the introducing of OSS, there is few to say, which is specific to OSS. The major point to take care of, is the fact that it is an "external component" which therefore should be clearly isolated from the rest of the system (e.g. through wrapping techniques) if its API is not standardised; this for the sake of robustness and maintainability.

- Main differences compared to COTS

The main differences noticed were:

- source code availability
- the lack of tools for system design (UML tools, GUI builder)
- a good versioning system (CVS) but restricted to small to medium teams (no connection to a bug tracking system)
- the way to obtain support, which is really different from COTS but efficient for standard development (maybe not for expert modifications).

The office tool (OpenOffice) was judged powerful and easy to use, but it still has the following "drawbacks" (not technical):

- different interface for Windows users which requires adaptation time
- some interoperability loss with usual office tools.

- Future directions

Future directions could concern performance and security aspects.

Also, the following services could be considered:

- directory services
- adaptable deployment
- system configuration and management
- formal electronic mail management

A more ambitious task could consist in defining a complete runtime and development platform, taking requirements of a real system into account, mixing COTS and OSS based on the requirements for each service.

7. CONCLUSIONS

Besides technical considerations, the introduction of OSS is a change that mainly affects the way integrators use to work with COTS; as such, it should be managed within a project by:

- taking care of "brake" factors
- analysing new opportunities and threats of the change

- bringing out limitations of previous way of doing as well as regulations that had been defined to deal with those limitations
- proposing changes that take care of the two previous points and, gaining high management implication
- conducting (leadership, information and/or training sessions) and crystallising the change through new procedures and organisations.