BAE SYSTEMS

Open Architecture vs. Open Source in Defence Systems

By Peter Hammond

BAE Systems Integrated Systems Technology (Insyte)

Unit D1, Marabout Industrial Estate, Dorchester, Dorset DT1 1YA

peter.hammond@baesystems.com

Abstract

Following the recent trend to use COTS hardware for defence purposes, the UK government wishes to exploit open source solutions for defence software systems. There are, however, a number of obstructions to using an open source licence for such systems. This presentation will describe the issues in developing against an open source requirement for MoD projects, and how these challenges can be overcome by using an Open System Architecture approach. This approach can give the customer flexibility in replacing and reusing components, while militarily or commercially sensitive intellectual property is still protected.

Introduction

Defence software systems have historically been largely proprietary, closed source arrangements, where the source code and intellectual property remains with the supplier. Following the recent trend to COTS[1] hardware, the increase in open systems exploitation, and the rise of open source software in the commercial sector, the UK government is interested in exploiting open systems and in particular open source solutions for defence contracts. This is seen as a route to improved reuse and extensibility, with the procurement advantages of not being locked in to a single contractor. It can also provide assurance of long term support for the customer, in that even if the original vendor goes out of business (or just goes out of that particular business) the software can still be maintained by a third party.

I will discuss here the benefits and drawbacks of open source development in the context of defence systems, and compare this with the open system style of development. It is our belief that open system development provides real benefits for the customer, possibly greater than would in fact be realised by open source development, and is the only commercially viable route for many types of defence system.

It is important to emphasise that this paper is concerned with strategies for delivering specific projects, and whether or not to create new open source offerings. This is not related to using or contributing to existing successful open source products. We have used, and in some cases contributed to, several open source products.

What are Open Systems and Open Source

Being “open source” means more than simply letting the customer see the source code, although precisely what it means is somewhat subjective. The Open Source Initiative definition [1] is commonly accepted in the software engineering community, and is the meaning used here. This states that open source software has a licence that permits any recipient of the code to re-distribute, modify or derive from it, subject to certain conditions, chiefly that these rights must be passed on with the derived work. It specifically does not allow any restrictions on who may use the software or for what purpose.

[1] Commercial Off-The-Shelf

There are also many definitions of open systems, or open systems architecture. For example, Federal Standard 1037C, “Telecommunications: Glossary of Telecommunications terms” [2] defines open systems architecture as

> The layered hierarchical structure, configuration, or model of a communications or distributed data processing system that (a) enables system description, design, development, installation, operation, improvement, and maintenance to be performed at a given layer or layers in the hierarchical structure, (b) allows each layer to provide a set of accessible functions that can be controlled and used by the functions in the layer above it, (c) enables each layer to be implemented without affecting the implementation of other layers, and (d) allows the alteration of system performance by the modification of one or more layers without altering the existing equipment, procedures, and protocols at the remaining layers

The US Department of Defense Open System Joint Task Force [3] puts it rather more succinctly:

> A system that employs modular design, uses widely supported and consensus based standards for its key interfaces, and has been subjected to successful validation and verification tests to ensure the openness of its key interfaces.

Most definitions emphasize the importance of open, rather than proprietary, standards used to facilitate communication between replaceable modules.

Open Source Development

The Benefits

To many people, the chief benefit of open source is that it is free of charge. This is not, however, the point of open source, nor is it the principle benefit cited by the major open source advocates. One of the main benefits cited for open source development is the quality improvement that can come from massive peer review when a wide developer community gets involved with a project; this has been summed up as Linus’s Law: “To enough eyes, all bugs are shallow” [4]. Another major benefit, and the one most of interest to the government customer, is that the customer is no longer at the mercy of the supplier; if anyone can see the source code, then anyone can maintain it.

There can also be benefits for the supplier. If the project is one that attracts many developers to contribute, the original supplier gains from this additional input as well as everyone else. It can be shown, using game theory [5], that the value to the original supplier is greater than they would have obtained by “going it alone”, and this value is not diminished by being shared.

The Reality

In practice, it is rare for a project to develop the momentum that leads to these strong network effects. Krishnamurthy [6] found that the median number of developers on mature open source projects was only 4. Figure 1 shows the frequency distribution of the number of developers, taken from the data of table 2 in that paper. Even on highly active projects that are aimed at developers, rather than end users, that vast majority of users do not contribute. For example, in January 2007, one of the five most active projects on Sourceforge was ADempiere Bazaar [7]. It had 18,400 downloads in the preceding two months, over 3000 in two days following a release. In the same period, there were code submissions from just 9 different developers. Figure 2 shows a pareto chart of the number of submissions by each developer; it can clearly be seen that the work is not shared evenly between them. It is particularly unlikely (in general) that any defence-specific project would generate sufficient interest from the wider community, or even from third party contractors. In “The Magic Cauldron” [8], Eric Raymond acknowledges that open source only really makes sense when there are many people interested in participating.

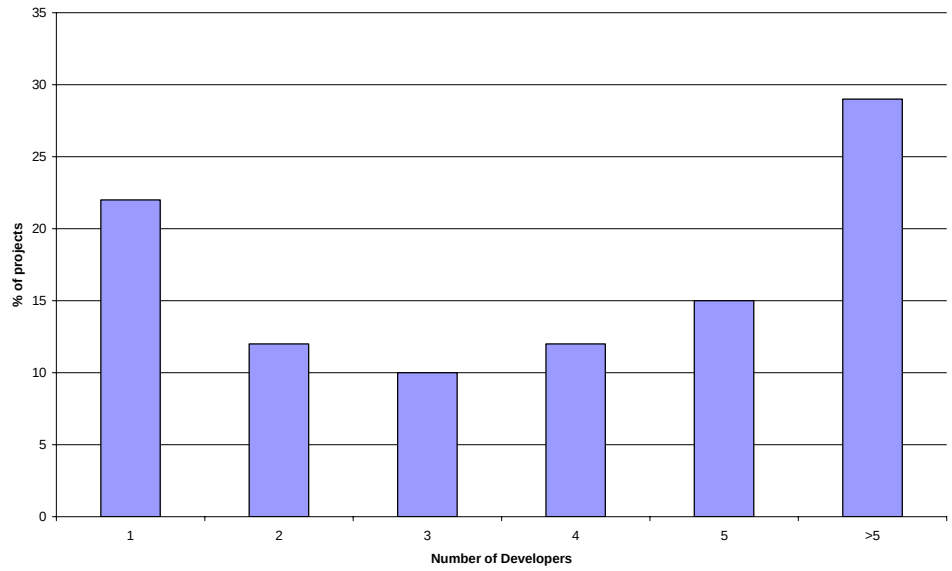

Figure 1: Frequency distribution of number of developers on 100 mature open source projects, based on data from Krishnamurthy [6]

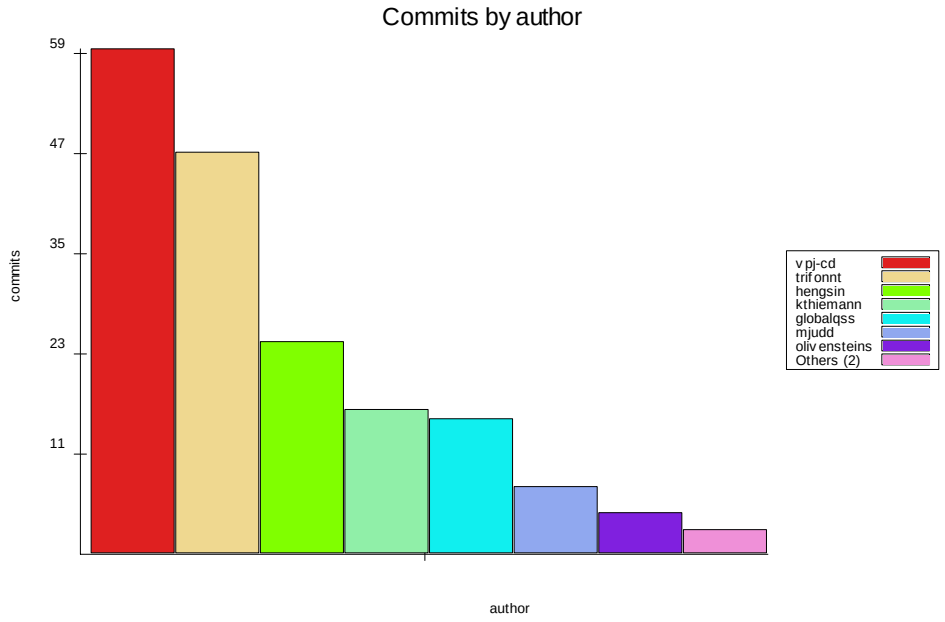

Figure 2: Pareto chart of the number of submission to the ADempiere Bazaar project over a two month period.

There are a number of additional challenges specific to defence systems that hinder development under open source licences:

- It is clearly not desirable to release sensitive algorithms into the public domain. Paradoxically, it is possible to develop security systems as open source; it is a well known principle of cryptography that the security lies in the key, not the algorithm [9]. However, in defence products, often the very existence of the algorithm is itself classified, and certainly the details of many algorithms are highly sensitive.

- The use of third-party open source components may compromise high integrity or safety related systems. Rightly or wrongly, open source software is perceived to have trouble in satisfying regulatory standards for high integrity systems, although it is possible and has been done [10].
- Many defence systems have long support phases built into the contract. There are clearly some interesting contractual complications if the source code that is being maintained by one contractor might be changed by another. Who is responsible for re-validating the system? Who is responsible for fixing bugs that are discovered later? It is easy to say, “the contractor in whose code the bug is found”, but it may not be easy to decide who that is. For example, if a defect is found because one function, unchanged from the original build, is passing input that a modified function is not accepting, it may not be clear where the actual fault lies. Has the new code unjustifiably constrained the input, or has the old code always been just lucky?
- Often systems are built on top of existing subsystems, in which the supplier has invested its own funds in order to gain competitive edge and as such is unwilling to release it to competitors.
- As with any development, there has to be a reasonable profit in it for the contractor. Although the software may be free in the sense of open to modification and derivation, open source does not imply free of charge. In particular the longer term cost and management implications of open source systems are not clear yet, particularly where the community is fairly small.

Open System Development

The Benefits

Open systems have considerable appeal to the defence customer, as can be seen from the effort put into initiatives such as the US Department of Defense Open Systems Joint Task Force (OSJTF), US Navy’s Open Architecture Computing Environment (OACE) [11], and the UK Royal Navy’s Modular Open System Architecture (MOSA). The open architecture approach is seen to provide many of the customer benefits that are sought from open source. It allows the customer to:

- Avoid lock-in to a single vendor.
- Replace or upgrade a component as new techniques and technologies become available;
- Re-use facilities to build new applications on existing components;
- Incorporate products from other suppliers, such as small, niche suppliers who may not have the resources or inclination to build an entire system.

Enabling this last point we see as quite important. A small start-up company may be building its entire business on a single technology, and may be highly unlikely to give away their “crown jewels” in an open-source offering. However, this highly specialised, advanced knowledge is exactly what the customer is most likely to want to incorporate and is exactly what this solution enables. Of course, not all open source licences or deployment models would force a third party to open their sources in order to incorporate their work into a system, but the only way to retain closed source without taking on the systems integration and support work is to write modules against an API, which brings us back to the open system situation.

For the supplier there are also benefits:

- Existing intellectual property is protected. Since, as discussed above, it is unlikely that the vendor will gain anything from opening their source, it is not unreasonable to wish to protect the IPR assets.
- Security and support concerns are addressed by retaining control over the source code.
- Sub-systems can be open-sourced where it is in the customer’s and supplier’s interests. This is most likely to be in areas that have application beyond defence, and contain well-known algorithms, so that Linus’s Law is likely to apply.
- Reusable components are available for the supplier’s own future products.

Although it has not yet been put to the test, there is some evidence that in practice an open architecture solution will deliver greater flexibility and reuse opportunities for the defence customer than an open source arrangement. It is considerably easier to understand the interfaces and framework of an open architecture than to understand the entire code base of the product. It would take a significant effort for a third-party contractor to gain sufficient understanding of the code base to be able to correctly maintain it.

The Reality

In general, the final product of an open architecture development will probably be more expensive than a traditional closed-source, closed-system alternative because getting truly open interfaces requires stakeholder input for peer review. It is well understood that merely publishing a proprietary API does not make an open system, see for example the debate surrounding the standardisation of Open Office XML [12]. It is not clear that third parties will be willing to provide this effort for free, so the customer could be left having to foot the bill for working groups and reviews.

There is also considerably more complexity in the system, and therefore greater effort and cost in the initial development. The art of designing an open, extensible system using interfaces is to predict what features will be needed by future developers and make sure they are available, or at least can be added easily. Partly this is just the usual object-oriented analysis skills of abstraction and encapsulation, but it goes much further. In any normal development, it would be termed "speculative generality", and be considered a defect to be removed in order to simplify the system [13]. In addition there is the need to document the interfaces and supply developer kits to allow validation of new or replacement components.

A small open system architecture has much in common with a plug-in framework [14]; indeed the latter can be seen as an instance of the former. However, in the cases that we are primarily concerned with here, there will be significant differences arising from different forces on the problem. An open system need not be based on a recognised middleware, but it usually will be. Larger systems will generally be deployed over several processors with several processes on each, and in this case it is clearly beneficial to re-use proven solutions for communication, resilience etc. There are also benefits from using a middleware in a small system:

- *Language heterogeneity*: Many systems in this area, even relatively small ones, have modules developed in several languages in them. It is common to be integrating numerical models written in FORTRAN with legacy components in Ada, C and C++, and put a Java or C# HCI[2] on the top. A middleware takes care of connecting languages with disparate ABIs[3]. Conversely, plug-in systems are usually suited to single-language programming.
- *Discovery and Binding*: Finding and starting each component may seem simple, but is often more difficult than it looks. Furthermore, there are cases where it will be necessary to have several instances of a given interface to satisfy the requirements of a system. For example, sonar propagation loss models have a limited domain of applicability, and so to give an acceptable estimate sonar performance across the whole operational range of a ship will require several different models. A middleware will provide facilities for registering and brokering the various implementations as required by the application (for example CORBA's trading service, or COM's registry). A plug-in will provide its own registry, but this can be seen as a barrier to adoption if it appears to be proprietary.
- *Scalability*: Part of the purpose of an open system is to permit reuse of the components in configurations that were not originally intended. An open, middleware-based framework will provide new clients with an easy way into the system, and will provide some level of support for cross-process, cross-host and multi-user scalability. Applications that start out life as stand-alone systems are often later incorporated as subsystems in larger systems, e.g. combat management systems. While a middleware does not magically provide a robust, scalable multi-user solution around ropey code, it does provide a basic level of scalability around reasonably portable code.

---

[2] Human Computer Interface

[3] Application Binary Interface

While these are all significant benefits, they are also not without cost. When developing against a middleware, the designer has two choices; either they can whole-heartedly embrace the idiom of the middleware, and allow its type system and conventions to permeate almost the whole code base; or, they can attempt to develop the main bulk of the code in a neutral manner, and provide a thin bridge layer that isolates the middleware dependency. The former case is easy to do, but may leave the code is a sorry state when that technology is no longer flavour of the month. It may also impede testability, depending on the middleware. On the other hand, a bridge layer means more to write and more to test.

If the interfaces are designed with cross-process operation in mind, the methods will be different from those designed for in-process calls, as the marshalling overhead requires that they be larger-grained. This may impact on certain design issues; for example, how can an arbitrary component affect things like menu enabling, without crippling system performance with remote procedure calls every time the mouse moves?

There are technical challenges to be overcome to allow disparate components to interact seamlessly, particularly where a single, task-oriented HCI is required. This is a more difficult challenge than the command-oriented HCIs more commonly found in plug-in systems, and is compounded by the possibility of the system acting as a subsystem to a remote HCI, or even operating in a headless mode. Similar strategies can be employed, but as with other methods, they will need to reflect the appropriate granularity,

Conclusions

Open source solutions have many benefits for appropriate projects, but are not without their pitfalls. An open systems approach brings its own costs, which must be carefully considered. However, when combined with judicious use of open source solutions, such systems can produce benefits to both customer and supplier in many cases where pure open source solutions will not deliver.

References

[1] http://www.opensource.org/docs/definition.php

[2] http://www.its.bldrdoc.gov/fs-1037/

[3] http://www.acq.osd.mil/osjtf/whatisos.html

[4] en.wikipedia.org/wiki/linus's_law

[5] C Y Baldwin & K Clarke, "The Architecture of Cooperation: How Code Architecture Mitigates Free Riding in the Open Source Development Model", Harvard Business School Working Paper Series, No 03-209

[6] S Krishnamurthy, "Cave or Community: An Empirical Examination of 100 Mature OpenSource Projects", First Monday **7** [6] 2002, http://firstmonday.org/issues/issue7_6/ktrishnamurthy/index.html.

[7] http://sourceforge.net/projects/adempiere/

[8] www.catb.org/~esr/writings/magic_cauldron; also E Raymond, "The Cathedral and the Bazaar", O'Reilly, CA, 2001.

[9] B Schneider, "Secrets and Lies: Digital Security in a Networked World", p.91, Wiley, 2000.

[10] E Fergus, "A Regulatory View of Software Reuse" *Proc IEE Seminar on COTS and SOUP*, London, Oct 2004.

[11] www.nswc.navy.mil/wwwDL/B/OACE

[12] A Griffiths, "The Power of Inertia", *Overload* [77] 2-3, Feb 2007.

[13] M Fowler, "Refactoring: Improving the Design of Existing Code", Addison-Wesley, 1999.

[14] Klaus Marquardt, "Patterns for Plug-ins", *in Pattern Languages of Program Design 5*, Dragos Manolescu, Markus Voelter and James Noble (eds), Addison-Wesley 2007.

Acknowledgements

Based on material first published at Undersea Defence Technology Europe 2006, reproduced by kind permission of Nexus Media Ltd.

Thanks are also due to Alan Minister and Simon Schafer of BAE Systems for reviewing and refining the manuscript.