WRAPPING THE COTS DILEMMA

Ian White
Defence Evaluation and Research Agency (DERA)
Portsdown Hill Road, Fareham, Hants, PO17 5AD
ENGLAND
iwhite@dera.gov.uk

Abstract

This paper first reviews the problems of using COTS, notably product assurance, vulnerability and product continuity. The concept of *wrapping* is then introduced. Conceptually, wrapping is a process to mitigate COTS limitations, and may be applied to any of the acquisition, design and implementation phases of a system. It is a fundamental assumption that COTS items being wrapped are not themselves amenable to any significant changes in their design or function.

1. INTRODUCTION

The title of the paper alludes to the common process of 'wrapping' inadequacies of COTS between added-value services that supplement the basic performance of COTS. Sometimes the wrapping is to add functionality; sometimes to limit poor functionality; sometimes the wrapping is cosmetic.

The use of commercial off the shelf (COTS)[1] components as the core elements of military information systems [command control and communications] is now widespread, because of their availability, low cost, and generally high levels of functionality. The military also increasingly expect a common look and feel between IT in the home, barracks and battlefield. However COTS elements are produced for the civil marketplace, and evolve in the context of a fine balance across commercial issues of:

- cost-competitiveness
- customer expectation of quality
- customer tolerance to shortfalls in quality
- lifetime in the marketplace
- commercial through-life support needs.
- time to market
- mechanisms for maximizing a future market share.

This balance of features is substantially different from those for traditional defence procurement, with huge advances in technology, and brutal conditions of the commercial marketplace, have resulting in the unprecedented growth in the capability and lowering cost of IT products. There are, from the military perspective, many inherent instabilities and inadequacies in this marketplace.

The military need to exploit COTS virtues and forgive COTS sins. The dilemma is that good comes with some degree of evil. The good things about COTS are taken for granted here. For insight into the future there is plenty of literature. For the author's views on future military communications see ref. [1].

2. COTS ASSURANCE - THE DILEMMA

If the military are to use COTS, and there is really no economically viable alternative in many situations, the military needs and commercial qualities must be reconciled. The heart of this issue is Assurance: to answer reliably the questions, what -

- does the product do?: is it properly specified?
- guarantees are given of its performance?
- is its reliability in performing this task?
  - as stand alone?
  - in an inter-operating military environment (both military and COTS hw and sw)?
- undeclared features does the product have (especially to 'illegal' inputs)?
- is the stability of the product in the marketplace?
  - its continuity of product?
  - the continuity of its function in replacement products?
- are the implications of new functionality on previous assurances

2.2 Manufacturers' Guarantees

Hardware guarantees are stronger than software guarantees, but neither is very strong from buyers' assurance viewpoint.

[1] To save space, in this paper COTS is variously used as a noun, (e.g. the use of COTS in military systems) or as an adjective (c.g. the use of COTS components).

Paper presented at the RTO IST Symposium on "Commercial Off-the-Shelf Products in Defence Applications "The Ruthless Pursuit of COTS"", held in Brussels, Belgium, 3-5 April 2000, and published in RTO MP-48.

Hardware

Guarantees typically provide for remedy of physical failures, including parts and labour, for the guarantee period. There is no specific *fitness for purpose* guarantee, although PC manufacturers make general compliance statements regarding preferred software systems. Whilst they give guarantees against physical failure, they give no guarantee against design defects, although they are often legally bound by national 'fitness for purpose' trading legislation, and health and safety considerations in this respect. For the military user with a long-term interest in function, design defects are the more important aspect.

Software

Main stream COTS software is not guaranteed by its manufacturers against functional failures; nor do manufacturers usually undertake to indemnify against software failures nor to correct all software failures. Examples of hardware and software guarantees from mainstream manufacturers indicate the problem faced by military systems users:

HARDWARE

Chrysler 3/36 Warranty: *"The basic warranty covers every Chrysler supplied part of your vehicle, except its tires . . . tires are covered by separate warranty"*

"The 'Basic Warranty' covers the cost of all parts and labour needed to repair any item of your vehicle . . . defective in material, workmanship of factory preparation. You pay nothing for these repairs"

Typical computer hardware warranties guarantee the hardware product against defects in materials and workmanship for a period of one year from the date of original purchase.

SOFTWARE

Software warranties are far poorer, typically disclaiming any liability for their use, nor giving any specific undertaking to correct faults that are discovered.

Why are guarantees so poor, especially for software? In the next section the expectations for software and hardware assurance of function are examined.

2.3 Software Assurance

The problem

The problem is that software now is like architecture in medieval times.

"If I were to build a bridge, I would, by using classical stress analysis, have figures for strength, elasticity, and a variety of other predictive capabilities to tell me how to build it. If my bridge design was wrong, it would probably, even during building, become evident that something was wrong.

If I wish to build a large software system - the bridge from the concept to the action - I have very little in the way of theory to do it. Building a software intensive real-time system is a task for which we have few exact tools and no fully effective means for predicting its performance." [2].

Closed Source Software

Many common software products, for example office aids, operating systems, and compilers, are complex programmes typically of several million lines of executable code. Current software evaluation technology is not able to formally test such software for the conformance of its functionality against specifications, and often such detailed specifications are not available. Nor are specification methods sufficiently strong to ensure that specifications are themselves either complete or consistent. The primary COTS software testing process is:

<*beta* test using knowledgeable users, and revise>
then <sell and take customer feedback>
then <prioritize faults against cost/marketplace need>
then <revise, and issue marketplace patches[2]>.

It is commonplace for such test programmes to uncover literally thousands of faults. This is the number of known faults corrected, not the total list! The manufacturers' versions of the latter lists are not normally published, although WWW listings by user groups are common. Furthermore, virtually all commercial software products are delivered as compiled code tied to a proprietary operating system. The source code is not available; it is subject to neither third party peer amendment nor review.

It appears that the fault rate of large software programs:

- is never amenable to any analysis to determine its extent
- is a linear function of the number of skilled users who apply it.

The significance of faults is difficult to quantify, since a minor fault for one user, may have catastrophic consequences for another.[3] What is evident with large COTS software products is that system failures caused

---

[2] i.e. short blocks of replacement code for aspects of the SW that did not work correctly.

[3] The Excel bug in which one specific number was incorrectly represented internally is an example (i.e. 1.40737488355328, = 0.64; several related numbers also fail), Risks digest [3], Vol. 15, no 39.

by various known and unknown fault states are commonplace. A source code bug rate of 4 bugs /Kloc[4] is a commonly cited number [3], however this would imply that popular PC operating systems, which range from about 8 to 64million lines of executable code initially have circa 32,000 to 256,000 faults, which is not realistic. Most faults are excised in development and beta testing. The point is that even with fault rates orders lower than this, the residual fault rate is not trivial.

This commercial development model poses serious questions for users requiring functionality with high integrity. Whilst a judgment may be made on integrity after some initial bedding-out time, it is not in itself a guarantee of performance. The Windows 3.11® product is old but reasonably stable. The problem of this strategy is that such products are not supported for more than a few years, and W3.11® is now entering this unsupported threshold[5]. Unfortunately, manufacturers give no guarantee that the later products will be as reliable as the older ones. Generally they are not, because they are less well tested, and have additional functionality, less well tried, and sometimes unwanted.

The arena of reliable software is still a meagre green patch in the wide range of commercial software. Whilst there have been significant advances in the development of formal, and quasi-formal methods, their application still remains limited, mainly to relatively small programs, for overtly safety critical functions such as avionics fly-by wire functions.

Open Source Software (OSS)

The use of open source software, developed collectively by 'hackers'[6], allows a review process not just by users, but by programmers and users, who all have access to the source code. This level of scrutiny produces some excellent code, which is potentially of a substantially higher quality than closed source software. Military procurers have traditionally been rather wary of the culture of open software programmers and users. Ironically it could prove to be a more reliable source than proprietary products. Strong interest is now being shown in the open software UNIX ® operating system *LINUX* ®, which performs well, is well supported (including by some commercial organizations), and is gaining ground due to a strong degree of dissatisfaction with proprietary closed source products in some sectors of the operating systems marketplace. Perhaps even more remarkable is the success of *Apache*, which is the widely used Web server software. This is a function requiring very high reliability, which the OSS process has clearly achieved.

2.4 Hardware Design Assurance

Chip Design

The design of complex integrated circuit chips is also extensively supported by software processes. Because the implications of a failure in the chip design are severe, with one chip design being manufactured in millions, the manufacturers take considerable trouble to ensure that the functionality is sound. This includes a wide range of emulation testing. Notwithstanding this, errors do occur, a notable example being Intel with its early Pentium ® chip, which had a arithmetic processor error. The elementary functions of a computer chip are grounded in mathematical processes and operations, and the functionality of processors is a more constrained domain that the full spectrum of COTS software. Because of these factors chips, from both design integrity and physical reliability viewpoints, are normally orders better than software, although faults do still occur.[7] This allows us to conclude that:

**For most military systems,
the degree of assurance in chip designs from
major manufacturers is adequate.**

2.5 Vulnerability

The problem

The limitations of testing and completeness of function of COTS systems has been discussed. Associated with these weaknesses is a huge user and player community on the WWW, that disseminates information on system weaknesses, their cures, and of course the malicious exploitation of known weaknesses for disruptive purposes. Notable weaknesses of COTS, cited in [4], are:

- easier for non-specialists to make intrusions
- low-cost solutions ignore stronger protection issues - security, integrity and EMC.
- COTS are more vulnerable to viruses, Trojan Horses, Easter Eggs etc. due to the wide

4 Kloc = 1000 lines of code

5 It is many manufacturers' stated policy to only support the current and the previous release of software products.

6 Not the colloquial meaning, but referring to software aficionados who devote their lives to developing open software.

7 The command f0 0f c7 c8 causes the Pentium processor to halt; recovery requires power-down/power-up (Risks Digest, vol. 19/45).

dissemination of information on these over the Web.

- conformance is difficult to establish with systems needing frequent patches to remedy defects.
- low COTS price, and rapid 'time to install' creates procurement pressures to adopt COTS, even if the full extend of vulnerabilities is not clear.
- maintenance for COTS creates some potential problems in battlefield situations.
- unanticipated inter-system interactions
- accidental, or deliberate intrusions or erroneous functions.

A fascinating and extensive catalogue of these are presented in the *Risks Digest* web pages produced by the ACM [4].

Vulnerability Tests

Tests on COTS products can be 'black box' or 'white box'. The latter is where all details of the COTS item are available (software listings, specifications, performance results etc.). Clearly for COTS a black box approach is more realistic, although few black box tests are available for COTS testing.

Enhancing Robustness

Information warfare (IW) protections include increasing system diversity to provide resilience, and as a way of making any focused analysis of system weaknesses more difficult. This is of course counter to the main stream of COTS development where at any one time there is normally a dominance of a few or even one product for key functions.

3. WRAPPING THE DILEMMA

3.1 Principle

If the military are to use main stream products how can they do better than supinely accept the poor quality assurances, and vulnerabilities of commercial software? We cannot give any strong dictats to manufacturers, and so must somehow 'wrap the dilemma'. The wrapper lets the good things through, and traps the bad things. That is the principle; the practice is however very limited.

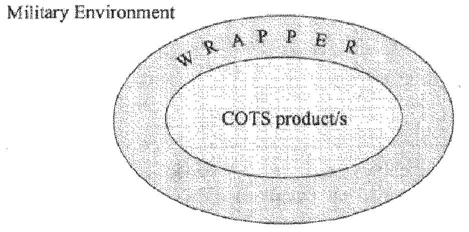

FIGURE 1 -- Wrapping COTS Functionality

The diagram illustrates the idea; the questions are:

- what needs wrapping and why?
- who produces the wrapper?
  Wrappers fulfil two principal functions
- They trap poor or dangerous performance features (i.e. provide better assurance)
- They specifically enhance existing performance
  A wrapper can be considered to be any, or all, of:
- An additional acquisition/procurement test
- Additional software
- Additional hardware.

3.1 Wrapping the Acquisition Processes

Product validation

It is possible in principle for the military buyer of COTS to test it himself to derive his own assurances of performance. This process is both difficult, and liable to be invalidated by the frequent changes made by manufacturers in their COTS products. Evaluation techniques include reverse engineering of closed source software, and using open source software, subject to certain tests. There are very few validation test suites for COTS systems. The primary source for such tests are national conformance testing centres, whose tests are primarily aimed at establishing standards compliance of systems interfaces.

Reverse Engineering

One possibility with some products is to reverse engineer from compiled code to evaluate the overall systematic functionality of the product. Although this is possible in principle, it has little general application because:

- for most programs it cannot be done economically[8]
- for large programs it cannot be done at all
- even if feasible, the scale of the analysis would require extensive resources to apply the tests
- any changes made as a result of such tests would
  - not be subject to any form of fault support from the manufacturer
  - probably be a breach of the manufacturer's copyright.

An alternative assurance strategy is to negotiate confidential access to the source code. For major

[8] One of the main targets for reverse engineering are computer viruses, which are normally quite small programs. Even these can prove difficult to analyse in this way.

COTS products most manufacturers see no commercial benefit in this, when set against the risks of their code confidentiality becoming compromised. Because major products are developed by large teams, at substantial expense, external assurance audits will require similarly large skilled teams. Accordingly a careful cost/benefits analysis for such assurance would have to be made. This leads to the second strong conclusion that:

In general reverse engineering is not practicable.

Exploiting Embedded Systems Software

Software for embedded systems is produced to a generally higher standard of reliability than for most software applications. Furthermore its design gives much more attention to economy of storage and processor demand than does general user software. One illustration of this is embedding *Windows*. Microsoft's *Windows CE* (*WinCE* ®) is a compact operating system, with some attractive 'military' features.

"While the industry trend (in OS's) is for successive OS releases to require even more hardware resources in terms of processing power, RAM, and disc space, WinCE has been written from the 'ground up' to operate with only the most basic hardware requirements"

Traditionally, Microsoft OS are supplied as a core kernel associated with a large number of supporting services, such as file systems and network support, all in a monolithic chunk. All these supporting services need to be stored on disc and loaded into RAM whether or not they are required by the applications running. The functionality available can be modified by adding or removing drivers for hardware options, but the kernel or removal of a service from say Windows NT®, would cause system malfunction with potentially disastrous consequences. Furthermore this would be considered a breach of the licensing agreement between the end user and Microsoft!" [5]

The virtues of a system built over a monolithic operating system are numerous and include:

- functionality limited to only that which is needed
- economy of required platform functionality (RAM, disc processor speed)
- far easier to configuration manage
- easier to undertake user driven test and evaluation of reliability/integrity
- deeper understanding of function available to user
- easier to assess vulnerability
- easier to apply formal or quasi-formal integrity/consistency tests to overall function.

The drawbacks are:

- less readily available range of functionality
- more expensive to develop
- may be difficult to expand to match the full range of the monolithic OS.

There are no standards here. The *EPOC*® [9] embedded operating system from Symbian (a joint venture company of Psion, Nokia, Motorola and Ericssen) is also a strong contender in this marketplace. The Symbian consortium is now attracting a very substantial international company following, and is seen as a major competitor to Microsoft's *WinCE* ® [6].

Safety Critical Requirements

If software must fulfill a safety critical function, then it must be structured for this task, and also run on high reliability hardware platform/s. We can take some ideas from fault-tolerant systems, for example running 'equivalent' software on different machines to provide parallel answers to important questions. In some cases running a hot standby can protect against hardware or software failures. These options require very high integrity choice mechanisms such as <majority voting on three processes> evaluations, or <failure detect - switch to standby>. The former requires that the same process is implemented (coded) in different ways to achieve some degree of independent failure mode. With COTS this is not generally possible, although some investigations have been made on the use of Unix systems from this perspective. These techniques are approximately three or two times respectively more expensive than stand-alone, and similarly more expensive to maintain. They may not be a cost effective solution for many defence support roles, but if COTS must be used for defensive critical roles, including much of C4I, then these options need to be considered much more seriously.

Unfortunately very little software written in the COTS marketplace is evaluated against these sorts of criteria. Nor are such results published as part of a assurance certification. This leads to the strong conclusion that:

Safety critical (SC) software cannot be provided from mainstream COTS. However more reliable software can be made using SC principles.

3.4 Architecture and Standards as Wrappers

[9] The name *EPOC* derives from the abbreviation of epoch, as this was regarded as being a new epoch in Portable Operating Systems.

"The difference between doctors and architects, is that doctors bury their mistakes"
Edward Lutyens (British architect).

Architecture mandates are the wrappers around the system design process. It is a cherished belief in military planning and procurement organisations that they can collect standards into a compendium, and even the relationships between standards, e.g. in profiles or structured APIs, and order the world to do their bidding. However to be effective an Architecture must meet the following conditions:

1. It must advocate a sufficiently precise set of standards, and related implementation processes
2. There must be a general community (industry, procurers, users) acceptance that these mandated elements are reasonable
3. It must be enforceable through indirect and evolutionary mechanisms as much as by policing.
4. It must be kept up to date
5. the procurement process must be matched to the timescales of the elements advocated within the architecture.

Historically many aspects of these mandates fail, often not through inadequacies in the standards themselves. The pace of COTS development, as part of this scenery, renders such mandates suspect. Using COTS has severe implications for those who seek to mandate procurement standards in IT. A fundamental reason is the standards generation process. To determine, write, and apply a new standard requires much discussion between peer experts, practical evaluations, refinements, and much deliberation over the writing of the standard to make it as clear and unambiguous as possible. This is inevitably a long process. There must be implementations those standards, which may be inimical to some companies' commercial objectives.

An Architecture mandate seeks to aggregate such standards, and in effect becomes a meta standard for a chosen set of systems (e.g. defence IT systems). It is axiomatic that to undertake this task requires a range of expertise that is both technically wide and deep. Defence architecture teams are never greater than a few people strong, they all have some technical weaknesses, and as the objective is an aggregation of standards, a full understanding of their interactive use is needed. In IT, this process is further aggravated by the rapid rate of change of products, usage styles and the continuous emergence of de-facto standards. This will always raise questions about the rational of any defence architecture mandate. Hence,

Architecture mandates do not address the problems of using COTS in military systems.

3.5 Wrapping COTS Products

Detecting faults and undefined states

Apart form seeking assurances that the product does what it should, there are also many weaknesses in products when they are used in ways that are not within the specification. Such 'out of range' states may be entered by accident, or by malicious design. COTS products are different from bespoke designs. Some manufacturers produce quite detailed specifications, others do not; some even charge for additional information describing the product. Most consumer software for example comes with no written documentation, other than installation instructions, licensing dictats, and installed 'help' facilities, that are often poor. The handbooks describing the product are an additional secondary market. No guarantee given that such descriptions are complete or fully accurate.

In addition, products often come with undeclared features (Easter eggs), that are used by the developers, and which the vendor may not wish the customer to access. These may be left for maintenance purposes, or as a deliberate act of configuration control ("this version passed acceptance tests just in time so don't change anything else!"). Such additions may even be gratuitous, as with the flying game in Microsoft *Excel* ® [10].

If we are to use such products in defence applications we need to know a lot about their behaviour. Apart from faults where the application does not do what it should, for a legal input parameter set, there are always many inputs sets which are illegal, and for which the system state is undefined. Note that manufacturers normally do not consider such results to be faults. Such states often cause the system to crash, so for any high reliability system these states need to be trapped. The scope for users, and hosted applications to invoke such values is evidently high.

We have already discussed design type checks on the procurement of systems. A better approach is to embed such COTS elements within some form of wrapper. The technology of wrapping is still developing, and is one where the military R&T need a greater and more strategically determined focus. The standard way of testing a process/equipment is by black box testing, where it is accepted that its detailed inner workings are not known. Performance is based on the interpretation of the relationships between inputs an outputs.

Black Box Testing - CMU's Ballista

A common problem with many operating systems is that parameter entries are not properly delimited. The extent of this problem is nicely illustrated by the work done by Carnegie Mellon University (CMU) on the *Ballista* Project [7], [8]. This project exploits the fact that many system crashes/lock-ups are caused by illegal values of module parameters. Accordingly they determine a parameter range for each operating system (OS) module (both legal and illegal values) and then test these, both individually and in nested iterations, with combinations of these parameter values.

It is part of the *Ballista* philosophy that the code for the module under test (MuT) is not available. Each module is characterised only in terms of parameter and data types required. Failure is defined informally according to user accepted crash/lock-up conditions. No special test harnesses/scaffolding is needed. *"The set of test cases used to test a MuT is completely determined by the data types of the parameter list of the MuT and does not depend upon a behavioral specification."[8].*

Ballista has been used to evaluate *POSIX* OSs by testing 233 different *POSIX* calls. Input parameter values applied over all value combinations over various *POSIX* OS systems showed an average failure rate of about 15%. (i.e. failure count/{no. of input parameter combinations attempted}). The lowest rate was 10% for AIX 4.1, the highest 22.7% for QNX4.24.

Knowledge of these failure states can be used to define a wrapper that traps known illegal values, or even legal values, which cause failure. If the test results used for figure one are re-evaluated with all non-exception (legal value) tests removed, the resulting normalised failure rate is nearer 30%.

Using the diversity of these different OSs to provide a more robust response has also been investigated but shows little advantage unless a high level (many OSs) are used.

A wrapper family based on these results is currently under development by CMU. Although there may be speed penalties imposed by the use of wrappers, the reliability of the operating system can be dramatically increased.

[10] To access this game on *Excel-97* : open a new worksheet and press F5; in dialogue box type X97:L97; click OK; press tab; Press Control-Sh AND click *Chart Wizard.* Excel's *Mountain World* is revealed, and upon one of its peaks is a list of credits for usability testing! The game in Excel 95 is more elaborate. In *Powerpoint-97®* select *Help/about Microsoft Excel*; click the *Powerpoint®* icon in the dialog box; a list of the programmers appears.

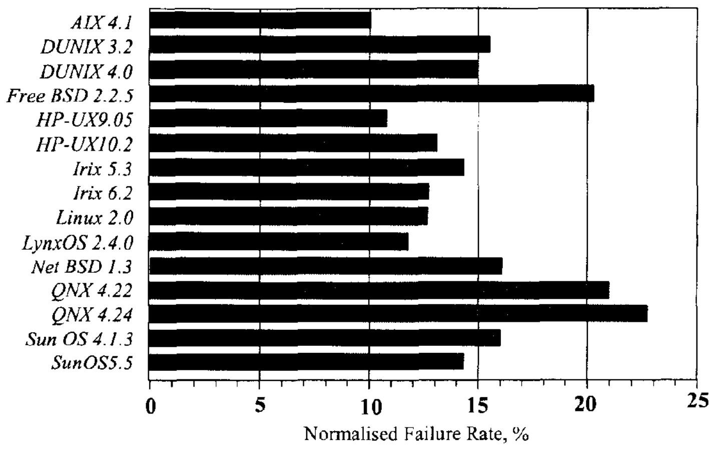

Figure 2 -- Ballista Testing: Posix System OS 'failure' rates[11]

It is salutary to note that as many manufacturers argue that such exception inputs are not faults, but user mistakes, they are not at fault. The corollary to this is that manufacturers may not include such states in their bugs databases, and may make little or no attempt to correct them.

3.6 Adding Functionality

Principle

Often in military systems the basic functionality provided by COTS systems is inadequate, or even non-existent. It is often possible to wrap a to COTS product, or service, with additional equipment to provide the additions required.

Crypto

The commonest military example of adding functionality is that of cryptographic devices. For example several ISDN cryptos are available that wrap the ISDN 2B+D channel, applying serial cryptography to the 2B stream, and applying signaling filtering to the D channel to prohibit data transmission and to constrain the range of signaling allowable.

The NATO KG81 applies a similar encryption to E1 (2Mbits/sec) trunks. Cryptographic devices of this type are both logical and physical wrappers.

Software insertion

Software wrapping, for example between a communications protocol and the API poses more significant problems, because the logical

[11] (from ref [8], courtesy Professor Philip Koopman, CMU Ballista Programme).

boundaries of software are less well defined, and more readily covertly subverted, there is far less confidence in the minds of security accrediting authorities about the use of software insertion mechanisms. This can be mitigated by related physical separations, for example dual processors, with accredited inter-links. Unfortunately this suspicion is not reflected in civil manufacturers' designs, so that security wrappers are often GOTS items, produced and distributed under specialist controlled conditions.

Physical Robustness

Another common need for additional functionality is to provide physical robustness, by additional packaging of equipment and by screening to mitigate the effects of radiation and Tempest. This is perhaps the original wrapper.

Enhancing performance

In some military applications the performance of COTS systems may be broadly what is needed, but with poor performance. Current IP routers, and ATM switches have poor performance in the present of data transmission channels that have poor error rate performance. This latter feature is a common characteristic of tactical military communications. To enable COTS products to be used frame and cell hardening is needed, whereby the link error rate is improved by error correction coding. This allows the commercial switches to be used, without any modification. There are now several products on the open marketplace that perform ATM Cell Hardening, including Comsat, SRC, Thompson and Marconi.

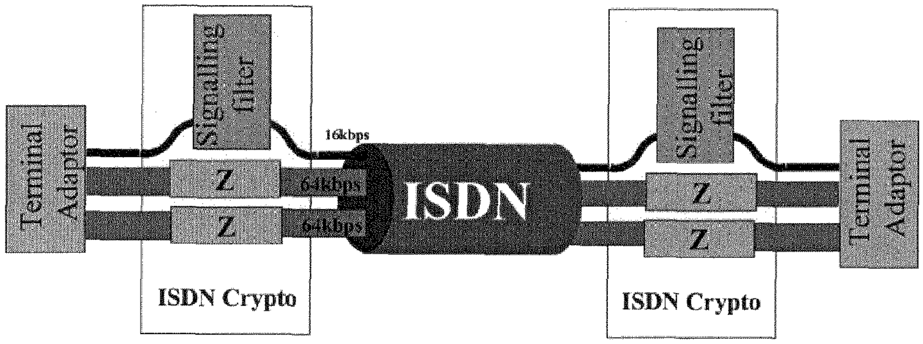

Figure 3 -- ISDN Crypto 'Wrapper' for 2*64 + 16 data channels

3.7 Interoperability Standards

Wrapping Military Legacy

Interoperability Standards exist in both the civil sector and NATO. An inverted strategy can by taken within military systems by wrapping them in a civil standard interface. This is being done with some systems adopting the various ISDN interfaces for military systems interfaces. For network management interfaces the process of wrapping legacy capabilities with common object interfaces is also being pursued.

There are a number of exciting developments in the civil sector on interoperability, all of which offer wrapping opportunities for military IT systems. Common examples are the use of ISDN BRA and PRA interfaces in military systems, the use of HTML web page based interfaces between systems (e.g. for network management), and more promisingly the use of CORBA techniques for integrating a wide range of legacy and proprietary systems.

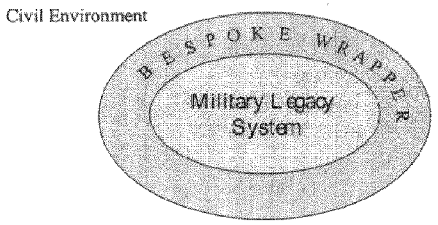

Figure 4: Wrapping Military Legacy, to Civilize it

Interoperability Opportunities

An interesting idea within the interoperability domain has been to develop a protocol language which allows 'on the fly' translation between protocols. This idea now has a real instantiation with the introduction by Sun Microsystems of the JINI ® concept. Here different computer devices are able to tell other elements in a computer network just what they are, and what they can do, using Java as the basis for this dialogue. A further development is to apply this idea not just to computer system interactions, but to a wider range of electrical elements. Wireless interconnection of these elements is within a programme called *Bluetooth* [9]. It provides wireless communications between element at ranges up to 10m, using an unlicensed region of the radio spectrum. In the office world this would link not just computers, but a very wide range of equipment, fax machines, telephones, shredders etc. In the home this linkage would be to all electrical equipment, the television, cooker, fridge, toaster, etc to advise the owner of the state of his home. It is the management infrastructure for the intelligent environment.

System management capabilities, deriving from the wide ranging work on enterprise resource planning/management, and the Telecommunication Management Forum's work on process definition, are further examples of opportunities for establishing usable military interfaces.

4. STRATEGIES FOR SELECTING COTS

4.1 Military-Civil Convergence

Mobile data services

What starts life as a 'Military Feature' in time often becomes a civil one. The need is for *'the office in a pocket'*, leading to increased data capacities for mobiles as well as more standardised forms of compact file structures for office utilities, for example 'lean' web page formats.

Security

The internet serves several hundred million subscribers, mostly without significant security features. This is a great untapped marketplace, since much of the internet will ultimately require a range of security services for its transactions, notably e-commerce both business to customer (B2C) and business to business (B2B). Already the ubiquitous virus has led to vulnerability protections becoming widespread.

Security mechanisms are now widely understood in the open literature and the knowledge is worldwide. Even unbreakable codes are readily within the reach of even modestly sophisticated users. The result of this strong drive towards powerful security will lead to a convergence of the civil and military security communities.

4.2 Which Products?

Four Key Questions

Clearly COTS products should be chosen because they perform a function that the military needs. The questions to be asked are about assurance and vulnerability, thus:

- Can the product be tested for (what it should do, and for what it should not)?
- Can it be replaced?
- Will it inter-operate with other systems/elements?
- Can it be wrapped without any internal modification?

It is a new challenge for military R&D to determine what research is needed to understand and best exploit the COTS marketplace. Since the marketplace overall is immense, an in-depth coverage is simply not feasible. So what should be studied? The focus must derive from both military functionality and procurement concerns such as those listed above.

Systems Integration

Many of the preceding issues have been concerned with understanding the function and performance of specific COTS products. To create systems comprising these elements requires:

- a design process
- systems integration and testing
- through-life support.

Each of these processes is different from those of old, where the design, even of much of the hardware, was bespoke. Specific performance limits, the implications of the changing COTS products in the time between design and integration, require that the process is much more fluid. Paul suggests that the dynamic pace of both requirements and technology invalidates many of the current design paradigms. There is a need for a developed concept of 'living systems', and design and support mechanisms to relate to this idea, rather than the idea of the system as a gestation aiming at a future 'fixed point'. He argues that the concept of a target 'fixed point' that the designers are trying to achieve is an obsolete, and dangerous concept within IT systems [10]. Certainly there is much to be said for integration being considered as in integral part of the design process, and the system itself being structured to permit its' evolutionary development.

A useful 'lessons learned' summary of COTS integration experience is given by Fox and Lantner [11]. A summary of some of their points is:

- COTS design results naturally in accelerated development, which precipitates an early start of integration testing.
- To be able to undertake integration testing test harnesses of various sorts are needed to simulate various aspects of the operational environment. The development of these can be a serious cost and time constraint.
- Maintenance on identified problems is by the COTS vendor, and may not meet the integrators needs. Problem investigation and identification and major parts of the integration task.
- the systems integrator must understand how products are configured (configuration files for specific products need to be rigorously controlled).
- Through-life support for COTS requires planning for replacement elements, at frequent points in the life-cycle, and this requires very strong configuration control.

5. ACKNOWLEDGEMENTS

This paper builds on earlier NATO IST contributions on the use of COTS by John Laws, Randall Schumaker, Major W. Tack, and various discussion and papers from within the UK's Defence Evaluation and Research Agency (DERA).

6. REFERENCES

[1] White, I, *Future Military Communications - A Preview*, unpublished paper.

[2] White, I., *Science in C3, Friend of Foe*, Joint Director of Laboratories Conference on C4I, Monteray, Cal. USA, 1994.

[3] Smith, C., *Win2000, 63000 Bugs*, Risks Digest, ACM publication; 19 July 1994, Vol. 20, no 80 (see also 81 and 82): all volumes available on the Web at: http://catless.ncl.ac.uk/Risks

[4] Tack, W., Maj., *C4I Vulnerability Using COTS*, IST Panel; 'Issues' Paper, Oct 1998.

[5] Watts, C., *Embedding Windows*, IEE Review, November 1998.

[6] *Overview of EPOC*, http://www.symbian.com/tech/papers/overview/overview.html

[7] *The Ballista Project*, Carnegie-Mellon university, http://www.cs.cmu.edu/afs/cs/project/edrc-ballista/www/

[8] Koopman, P., and De Vale, J. *Comparing the Robustness of POSIX Operating Systems*, 29th Annual International symposium on Fault Tolerant Computing, 13 - 18 June 1999, Madison Wisconsin, USA.

[9] *Overview of Bluetooth*, http://www.mobilebluetooth.com/

[10] Paul, R., *Why Users Cannot Get What They Want*, ACM SIGIOS Bulletin, Dec 1993, Vol. 14, No 2, pp 8 - 12. (also on University of Brunel (UK) web site).

[11] Fox, G. and Lantner, K. W., *A Software Development Process for COTS Based Information System Infrastructure*, http://www.stsc.hill.af.mil/crosstalk/1998/apr/process.html