Modular Open Systems Architecture

Peter Henderson

14th April 2009

This paper, along with a bibliography, is available from **http://openpdq.com/OpenSystems**
.

The author can be contacted at **mailto:p.henderson@ecs.soton.ac.uk**

1 Overview

The usual definition of an Open System is that it is built from components that have been designed to be interchangeable. The interfaces to these interchangeable components are clearly defined and openly published. This enables anyone to produce a new component that enhances the capabilities of the system into which it is plugged.

The business model that supports this enterprise is one of mutual benefit, where the supplier of the Open System and the supplier of the Plug-In enhance each other's market position.

Open System - An Open System is a modular construction that has been designed in such a way that its modules have precisely defined and publicly owned interfaces that allow independent suppliers to provide improved capability by providing plug-compatible, innovative modules.

This paper discusses what it means for a system to be Open. It discusses the Costs and the Benefits. It gives a plausible argument that the benefits can be realised if the Principles of Open Systems are followed, in particuar that the system should be based on a clearly defined Open System Architecture.

It is not essential that all the components of an Open System are open, indeed it would be impossible to realise this ideal in most commercial scenarios. The openness refers to the interfaces to components. These must be sufficiently fully-general and sufficiently well-defined that producing plug-compatible components is both practical and economically sensible.

The architecture of the system as a whole must be based on a model of the potential for evolution and innovation that would make producing new system components attractive to an independent organisation.

The paper also discusses the relationship between Open Systems and Open Source Software, which are not the same thing, although the ideas and the principles are clearly related. In particular, Open Systems Principles are not specific to just software.

The paper attempts to establish the principles of Open Systems and to introduce methods of Open System Architecture, which are fundamental to the development of open systems. An extensive bibliography is provided.

2 Introduction

We are particularly concerned with systems that are very large, such as those installed by national goverments, where large teams of engineers are involved in building the system over a very long lifetime. The requirements that the system is expected to meet will evolve though the whole-life of the system - during both its development and its life in service. Consequently, the system will need to be designed in a way that allows it to evolve through life.

A well-designed Open System has the property of supporting innovation and rapid evolution, because of the ability of independent suppliers to produce new functionality concurrently.

Whether we are considering a software system, a physical system, an electromechanical system or a purely electronic system, certain concepts are universal. For example, the system of interest will be modular. Hence we tend to think of modularity and openness as being part and parcel of the same thing. They are not. A system can be modular without being open. Here, we are primarily concerned with the property of openness and take modularity as given.

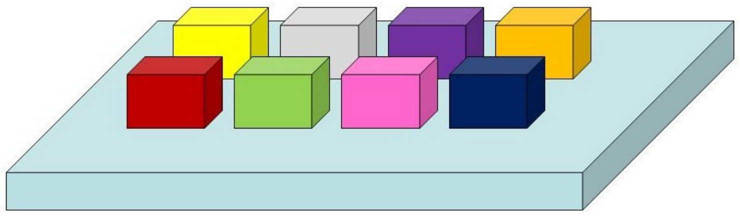

Figure 1. An Open System. Components plugged into an Open infrastructure.

Figure 1 is a schematic of a system that is often used to discuss to concepts behind Open Systems. It shows eight components (or modules) plugged into a common infrastructure. The basic idea is that this infrastructure provides the means whereby the pluggable components are able to communicate with each other. Variants of the system can be specialised by choosing alternative components to plug into the infrastructure.

Familiar examples of such a system would include the contemporary arrangements of hi-fi systems that many of us have, where the (rather minimal) infrastructure comprises the various standards whereby audio devices can be matched. Familiar from computing systems would be the contemporary operating system which is a complex infrastructure that allows applications programs to communicate with each other, not least through shared resources such as the file store.

More specifically, for the kind of large, software-intensive systems that governments install, it is the middleware in the infrastructure that provides the open platform into which application-specific components are plugged. Realising openness at this uppermost, application layer is as fundamental to the success of Open Systems as the use of COTS in the infrastructure itself.

The key aspect of this modular structure is that the architecture has been chosen so that the replaceable components can be readily replaced in order to create innovative new evolutions of the system. This is not a trivial design exercise. It is essential that the components are chosen in such a way that considerable variation is supported, so as to maximise the opportunity for innovation. This requires substantial effort on defining and maintaining the architecture, not least the level of granularity that is chosen for the replaceable components. We will cover these aspects later, when we describe the principles of Open Systems and methods of Architecture Description.

System Architecture - The architecture of a system is (an expression of) its gross structure at a high level of abstraction. For large systems, the architecture and its description, will be modular. This allows the designers to convince themselves and other stakeholders, by modular reasoning, that the solution represented by the architecture is capable of achieving its objectives, including through-life development delivering incremental improvements, to cost.

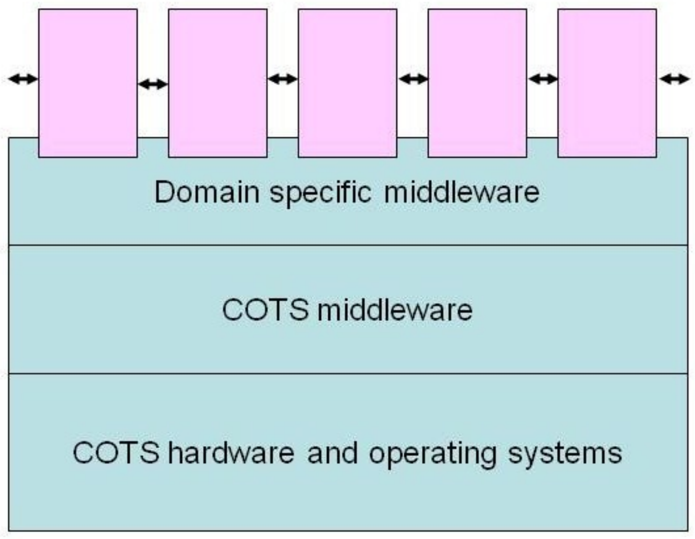
Domain specific middleware
COTS middleware
COTS hardware and operating systems

Figure 2. An Open Software System showing COTS infrastrucure and Domain Specific Applications.

Open System Architecture - A System Architecture which has been organised as a collection of collaborating components with well-defined and publicly available open interfaces, such that the potential for a wide range of prospective evolutions of the base product are designed-in from the start. Realistically, Open Systems must embrace legacy components, so the means of developing architectures that support opened infrastructures and opened interfaces to legacy components is fundamental.

Figure 2 shows a more detailed view of the modular architecture that we envisage for software-intensive Open Systems. As technology matures, so the architecture adapts. More and more of it becomes part of the standard infrastructure that everyone builds on, and COTS components become available from which to build the infrastructure. This sedimentation of technology is a natural evolutionary process that our Open System vision needs to embrace.

At some level however, at some point in time, the infrastructure and the components become domain specific and few if any application-level COTS components are available from which to build solutions. Typically, what is available is lots of legacy components from which to build. It is this area that Open Systems comes into play. When developing a solution that will have a long life and will be economical to evolve through-life, the designers must develop an architecture where the components and domain specific infrastructure have been chosen to allow both legacy and new components to be integrated into effective and economical solutions.

Standards, Specifications - In common parlance, we use the term Standards in two ways. First it is commonly (and correctly) used for International Standards that are under the auspices of official bodies such as ANSI. Secondly, and rather informally, we commonly use the term for the kind of consensus-based specifications that are agreed by industry-collaboration bodies such as OASIS, W3C etc. The latter are as important, or possible more important, to the development of Open Systems as the former. A better term for the second category is Specification.

Open Systems Governance - Bodies such as OASIS, OMG and W3C govern the evolution of Open Systems. They do this by publishing consensus-based Specifications which have the property that companies can reasonably trust that versions of the Specifications will be adhered to by other suppliers and can base a business-case on delivering to those Specifications. These bodies are basically developing specifications for middleware. Application specific bodies also exist (e.g AUTOSAR, OGF) which specialise in domain-specific standards, also essential to the evolution of Open Systems.

3 The Benefits

Many of the benefits of Open Systems are already realised in the commercial sector, where standards for such things as databases, documents and messaging have enabled the rapid evolution of enterprise computing and of the web. This is independent of, but substantially supported by, the Open Source community who, as well as adhering to commercial standards, publish the entire corpus of their work for anyone to use under, usually very generous, open licences.

The advantages of Open Source include the belief that one is not dependent on any one supplier as a sole-source, since there are no secrets and in theory one could commission anyone to maintain an Open Source product on which your business has come to rely.

The Open Systems community is a little different, because proprietary components are supplied into solutions in this area. It is just the interfaces to those components that are open. In order to be equally confident of supply, a business needs to be confident that they could commission enhanced functionality from anyone who could economically supply add-ons to the proprietary components.

Notwithstanding this distiction between Open Systems and Open Source, many of the benefits of Open Source can be realised by Open Systems. The benefits are not only to the customer for the solution, but also to the suppliers of its components, who will realise both increased markets and reduced costs. The benefits are summarised in the following table and elaborated on in the following paragraphs.

Reduced Whole Life Cost
Reduced Schedule for each Iterate
Reduced Incumbency, Wider Supplier Base
Greater Potential for Innovation
Greater Resilience
More Rapid Capability Insertion
Potential for Evolution from Legacy
Opportunities for Test and Acceptance Cost Reduction
Potential for Improved Interoperabiity
Potential to Exploit COTS
Commonality across currently Separate Domains
Most Practical Approach to Systems of Systems

Figure 3. Twelve Benefits of Open Systems

Reduced Whole Life Cost. The reasons why we might realise reduced costs for a large system throughout a long life of evolutionary change are many and various. The overriding reason is that, because the Architecture and the Interface Specifications are agreed by consensus, the boundaries that these agreements establish maintain a high degree of consistency in the solution and a looseness of coupling in the solution, that in turn enables the relatively straightforward insertion of enhanced functionality.

Reduced Schedule for each Iterate. The life of an Open System (indeed, any large system) is one of frequent evolutionary change. Not only is the cost of each iterate reduced, but the time to delivery is also reduced. Indeed, some of the suppliers' benefit is in the reduced cost. Again, it is the imposed looseness of coupling that is responsible for this reduced schedule.

Reduced Incumbency, Wider Supplier Base. With propietary solutions, the customer becomes dependent on an incumbent supplier. With Open Systems the customer can anticipate buying essential business evolution from independent suppliers. One or more of those suppliers will supply components into the infrastructure layers. More important for the customer is the components supplied into the application layer, where their business differentiation lies and where their potential for increased competitiveness is realised. It is essential for the customer that these application layer modules are not single source.

Greater Potential for Innovation. Because Open Systems bring many more people into roles where they can contribute to evolution of a solution (not least because many more suppliers are engaged) there is much greater prospect for innovation in that new ideas have a realistic prospect of being prototyped and market-tested. This is certainly true of Open Source, where the many-eyes aspect of that domain contributes to so many of its quality attributes.

Greater Resilience. There are also many reasons why an open solution is more resilient. One is that reusable components have parts to play in more than one solution and so they mature more quickly. Another is that, because open systems are a reliable way of evolving from legacy systems, they can inherit the reliability of those legacy components that are retained. But mostly it is the consistency and concreteness of the Architecture and its encouragement of loose coupling, that leads to solutions with greater resilience.

More Rapid Capability Insertion. For the customer the big prize is often the ability to realise business evolution quickly, in order to take maximum benefit from their market position. An Open System has to be designed with evolution at its heart. The standards that the interfaces implement have to be able to evolve independently. The consequence is that significant enhancements in delivered capability can be realised by upgrading only one or two components, leaving the others unchanged, and with full backward compatibility.

Potential for Evolution from Legacy. Open Systems are achieved by refactoring legacy systems into reusable components, which are given open interfaces and reestablished on an open infrastructure. This opening of legacy systems maintains the investment in existing business critical components while releasing the potential for change afforded by Open Systems. Where a legacy system is proprietary, the supplier has to be convinced that opening makes business sense.

Opportunities for Test and Acceptance Cost Reduction. Both modularity and openness contribute to significant potential to realise reduced costs in Test and Acceptance. Modularity is essential if modular reasoning is going to be used, which allows some if not all of the evidence that a solution is fit for purpose to be reused when a new case has to be made for an enhanced system. Openness maintains the looseness of coupling and relative independence of the components which, to a large extent, ensures that regression testing will be uneventful and hence of considerably reduced cost. Of course, neither modularity nor openness obviate the need for regression testing.

Potential for Improved Interoperabiity. Large systems have significant external as well as internal communications requirements. The use of open standards on external communications is what increases the potential for improved interoperability. The open standards do not refer only to the protocols but also the message contents. Where these interoperability requirements are domain specific, there is the need for governance of the standards that dictate the application level correspondence.

Potential to Exploit COTS. This almost goes without saying. Of course one uses COTS in open systems, but these are normally only available in the infrastructure layers and for generic application level components (document handling, for example). In general, infrastructures should be built exclusively from COTS. One could actually reverse this statement and say that the infrastructure is the sum total of the COTS used in the system. The COTS supply the environment in which the domain-specific applications are deployed. That's also why it is the application layer where all the action is, in Open Systems.

Commonality across currently Separate Domains. The economies of scale that can be realised by deploying the same components or the same subsystems across separate domains, such as separate parts of an organisation, are one of the main reasons for building Open Systems and deploying reusable components. It means that requirements capture may need to be done across a broader part of the organisation than just the part that is being evolved. Considering a broader domain affords the opportunity to increase commonality across an organisation.

Most Practical Approach to Systems of Systems. Systems are always deployed into existing systems, whether they are open or not. Conversely, the components of a System of Systems are systems themselves, but considered as black boxes by the Systems of System architect.These components however present interfaces to the System of Systems and these interfaces will be the more versatile if they are open and the System of Systems realises the benefits of Open Systems enjoyed by the deployed components

There may be more benefits than I have listed here, and certainly the above are neither in rank order nor mutually exclusive. Of course, there is a downside to Open Systems and that is what we cover next

4 The Blockers

Not all systems are Open Systems. The reasons for this are also many and various. Here we address some of those that we have encountered.

Initial Cost
Suppliers Perception of their Future Business
Inherent Complexity of Domain
Specific Domain Issues (Safety, Security etc.)
Interoperability with Legacy Installation

Figure 4. Five Blockers of Open Systems

None of these blockers are essentially cases against Open Systems. Indeed, in many of these cases one can argue that the perception of a case against Open Systems is just plain wrong and that the best way to overcome the blockage is to build the solution as an Open System from the start. But this is a perceived risk that many organisations are afraid to take. Either they lack courage, or simply, understandably, take a short-term view in that the prospect for early income from a project overrides the potential for posibly larger future profits.

Initial Cost - To make a system open, either from scratch or by refactoring a legacy system, will have an initial cost. There is additional work to do in establishing an architecture that is suited to evolution and innovation in the application domain. There is additional work to do in giving precise specification to the interfaces and in establishing the governance that will maintain the consistency and managed evolution of these specifications. This initial cost has to be recovered from the through-life savings (including opportunity costs) that can be made. An unwillingness to incur this initial overhead increases the likelyhood that an initial solution will not be open.

Suppliers Perception of their Future Business - Companies that already supply into a market need an incentive to make their systems open. They need to see that it will enhance their own market position to allow others to share what might be considered proprietary IP. It can be difficult to establish a plan, with acceptable risk, that allows others to compete as well as collaborate and innovate. Yet it is precisely this willingness to relinquish control that has seen Open Systems (Open Source, in particular) be successful in many markets. The lack of incentive to take this risk can often block the introduction of Open Systems into new markets.

Inherent Complexity of Domain - Some application domains are inherently complex, especially those domains that are either very large or very new. This complexity establishes a hurdle that leads designers into tightly-coupled solutions that concentrate on delivering functionality, over adequate considerararation of whole-life evolution. The extra resource that would be require to make an Open System in this situation is not available, because the risk of failing to deliver an initial solution is thought to be too great.

Specific Domain Issues (Safety, Security etc.) - One significant aspect of additional complexity is that some domains have overarching NFRs, such as real-time performance, safety, security etc. These NFRs drive the design, rather than the consideration of through-life cost. Often, such NFRs are considered non-modular and so the solution that is arrived at doesn't have a modular structure that lends itself to loose coupling. Yet, such systems will still have the need for innovation and through-life evolution that will benefit from an open solution. Architects need to find solutions to these NFR dominated domains that are open and can reaise the benefits of Open Systems, not least the modular reasoning that supports accreditation and certification in these domains.

Interoperability with Legacy Installation - We have major investments in legacy applications which must form part of any future solution. Consequently, there is a tendancy towards interfacing with these legacy applications in the way that these applications were designed to work, often using proprietary or unusual protocols. This need to interoperate can dominate the decisions as to which technologies and which protocols to use and these my not be open or openable. Again, the extra investment (and risk) needed to establish an Open System around legacy components is an act of faith on the part of the architect.

There will be other blockers and indeed, in a commercial sense, one would only make a solution open if there was a good business case for doing so. We will discuss later the various business models that have incentivised the pursuit of Open Systems in many domains (not least Open Source) but first we will look at the principles that an architect wanting to establish an open solution must follow and the methods that they might use to do this.

5 The Principles

In order to establish an Open Architecture, architects will adopt and follow certain principles. We list some of those here.

Figure 5 shows a typical Application Architecture. This is the arrangement of plug-in components that form the application capability. It is effectively the interoperation of the top-level components shown in Figure 2, embedded in the lower layers of infrastructure. In practice, of course, a similar pattern of components will be considered at each level, with that arrangement providing capability to the level above. But, in practice, the sedimentary lower layers, built largely from COTS, will already be open, so it is at the uppermost, application level that the architects must work to achieve overall openness.

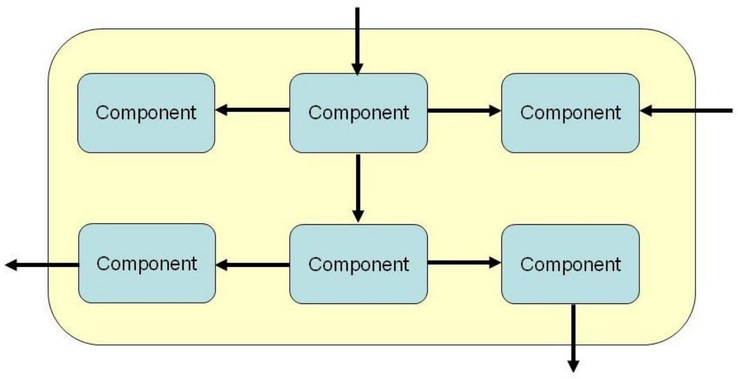

Figure 5. An Application Architecture showing application specific interfaces.

Architecture at this level is a difficult creative process and consequently on that is iterative in nature. An architecture team need a shared model around which to develop these interations. Capturing this model is the subject of the next section. There are generic principles of Open Systems that the architects must follow in order to achieve a suitable level of openness. These are listed below and elaborated upon in the ensuing paragraphs.

Be designed for Modularity, Reuse and Vision
Have fully Specified and Precise Interfaces
Be constrained by Application Specific Governance
Achieve Loose Coupling
Have a granularity that enhances Evolution potential
Have a Recursive Architecture
Have Generalised Components, for Commonality across Domains
Exploit Legacy Assets

Figure 6. Eight Principles of Open Systems

Each of the principles listed in Figure 6, and elaborated below, should be read with the prefix - An Open System will ... These principles are not in any significant order, nor are they complete, but we do think they are important and can form the core of guidance for Open System Architecture.

Be designed for Modularity, Reuse and Vision - An Open System will be a modular construction where the architects have specifically chosen the granularity of the decomposition and the capabilities of the components to realise an evolving family of solutions towards an eventual Vision system. Establishing this Vision system and choosing the granularity are key aspects of creating a long-life architecture with potential for innovative enhancement. This **is** the Architecture.

Have fully Specified and Precise Interfaces - The nature of openness requires that some things are fixed, at least in the sense that their evolution is understood and managed. It is the Interfaces to the components that must have this property, in order to ensure that suppliers can have confidence in the market for their innovative components. The architects must establish the means whereby these interfaces are determined and specified. A precise specification is required, which would include an abstraction from the (service-oriented) behaviour of the components as well as the syntax of how their capability is invoked. **These** are the Specifications.

Be constrained by Application Specific Governance - The Architecture and the Specifications will be documented and the evolution and authenticity of these documents will need to be managed by a body that can be trusted to provide a public service. Establishing this governance is as much a part of creating an Open System as the more technical design and specification tasks.

Achieve Loose Coupling - To realistically maintain its openness, the Open System must have the property that making evolutionary and innovative enhancements in capability has impact on a minimum number of components. Loose-coupling of components is required to afford this requirement. One way in which loose-coupling can be tested is by showing that components can be removed with only small impact on the capability of the system as a whole. Similarly, enhanced components can replace existing components, again with minimal impact on the general functionality of the system.

Have a granularity that enhances Evolution potential - Consideration of the level of granularity needs to take into account prospects for evolution. Some components, providing generic services, may be very large (e.g capability servers) while others providing specific detailed functionality may be very small (e.g. algorithms). Architecture Patterns will provide the means for designing this parametric form of modularity.

Have a Recursive Architecture - This may sound a little technical, but it is essential. A recursive architecture is one where the interfaces to one component look sufficiently like the interfaces to another, that the one can replace the other. This is particulary true when a collection of components (a subsystem) can be treated as if it was a single component, for the purposes of plugging it into an existing assembly. For example, a single server might be replaced by a cluster of servers. Making the interfaces sufficiently generic that this particular aspect of mix and match is possible, is where the potential for innovation lies, and where many of the benefits of Open Systems will be realised.

Have Generalised Components, for Commonality across Domains - In addition to making the components repaceable for the purposes of enhancement of a single solutuion, the architects should endeavor to make them sufficiently general that they can be exploited across many domains. They can become the COTS of the future.

Exploit Legacy Assets - Far from ignoring existing solutions, the architect for an Open System should look to exploiting legacy components as much as possible. This will mean that legacy infrastructures will need to be openned and reusable components identified. These components will need to be fitted with adaptors to bring them into the new Open Architecture, but that should not be considered a secondary or unimportant task. The legacy solutions are the principal source of leverage. The issue for the architect is how to avoid the trap of being enticed by proprietary protocols and how to effectively replace these with open ones.

6 The Methods

Architects developing Open System Architectures need methods and tools to collaborate and communicate an evolving design and to maintain that capability throughout the life of the Open System. This is where the notions of Specifications, Standards and Governance have their role. The methods and tools are deployed in order to ensure that this through life requirement is met.

Establish Vision System
Devise Architecture Description
Devise Application Components
Check Completeness and Consistency
Use Modular Reasoning
Map Legacy Assets
Establish Governance Methods

Figure 6. Methods used for Open Systems Architecture development

The tasks listed in Figure 6 are elaborated below, with some indication of the methods to be deployed and the types of tools to be used in accomplishing these tasks. Central to the architects' task is the need to maintain up-to-date and consistent shared models of the solution. These shared models are documents, so the adoption of standard notations and types of document is fundamental. Consequently, that is a theme that permeates these methods.

Establish Vision System - Capturing the Vision System and interim goals as a series of plausible Capability Builds is the objective of this task. Some type of Gannt chart should be used to illustrate the timeline but this needs to be accompanied by sufficiently detailed descriptions of the builds that stakeholders can be convinced that the direction is the right one. The language of description for these increasingly elaborate targets is the language of capability requirements, although that should be augmented by some details of the openness to be achieved, not least by showing the anticipated evolution of components within an Open Architecture. The first interim target is then the subject of the first attempt at developing an Open System.

Devise Architecture Description This is where the architects would be expected to choose, refine and then deploy ans Architecture Description Language. It is likely, for large systems of the type envisaged here, that the ADL will be based on standard notations such as UML or SysML, adopted within a framework that enumerates the document types to be produced and documentation standards to be adopted. The use of Architecture modelling tools should be determined, in particular how these are going to be used generate and validate the developing documentation. The use of XML as a documentation as well as interchange standard should be considered, not least because many architecting tools support that and this is a rapidly developing field of eandeavour.

Devise Application Components - The principal focus of devising an Open System is to determine the set on components that are going to procured and to give precise and complete descriptions to their interfaces at a suitable level of abstraction. This is where it is not sufficient to have chosen and ADL but it is essential to specialise that ADL (i.e. choose from among it many concepts) so that the boundaries of these components are clearly delineated and the interfaces clearly enumerated. The architect must seek generality. One way to show that this has been achieved is to demonstrate the deployment of components in different combinations or arrangements to achieve different capabilities. These deployment/build scenarios should be used to test the openness of the components. The interfaces should be specified to a sufficient level of detail that plausible arguments as to their efficacy can be given.

Check Completeness and Consistency - Another role for (a different type of) scenario is one of checking the specification of components for completeness and consistency. The architecture team should always be checking. They should not expect the design to be consistent at early stages, any more than they would expect it to be complete, but they should be aware of where the inconsistencies and incompletenesses are are always working towards eliminating them. This formal validation of the design can be supported, to some extent, by contemporary architecture tools, but the architects need to be judicious in their choice of such tools, which can be very resource-hungry.

Use Modular Reasoning - Open Systems add to the potential for exploiting modularity in the context of using modular reasoning for creating cases required for testing, accreditation and certification. The additional potential, beyond modularity, is related to the fixedness of (generations of) interface specifications and of the prospect for reasoning about (implementations of) these interfaces independently. The tools exist for constructing claim-argument-evidence based, sound arguments that a component satisfies its obligations and that assemblies of components satisfy theirs. The architecture of the Open System can be organised to support this prospect, by ensuring that the needs of modular reasoning are addressed thoughout.

Map Legacy Assets - Large systems are created by developing legacy assets. Thus it is essentila that the architects Reverse Engineer existing systems in order that they can develop an architecture which will embrace these assets as components in the poropsed solution. The methods for reverse engineering an architecture are simply to use the architecture description techniques adopted for the new system as a means of capturing a plausible description of the legacy asset. This act of Asset Mapping amounts to a rational reconstruction of the existing system and affoirds the means whereby, ownership permitting, it can be openned for harvesting of components.

Establish Governance Methods - The architects have a significant role in the scheme set up to develop the specifications. They will determine the structure of committees, because that will reflect the architecture. They will determine the documents that these committees will process and eventuallly sign-off. The architecture description techniques that they choose to use will have toi be chosen in such a way that the governance process is effective and (cost) effective.

This report concentrates only on the architecture process and does not venture into the implementation or acceptance processes. Nevertheless, it is essential, as the above comments suggest, that the architects design something that is open, whose openness is testable and which can be built and tested in a cost-effective manner. The choice of methods in the areas outlined above should always bear in mind the realiity of the implementation process.

7 The Business Models

Resistance to Open Systems from companies is often related to the fact that their business model is based around their exploitation of IP and opening that, or part of it, to public scrutiny could have a damaging effect on their income. However, there are also many examples of organisations that have developed business models around Open Systems that either increase their income or reduce their costs, or both, with a positive effect on the bottom line. For an organisation to choose to open part, or all, of their IP to public scrutiny requires that they develop a business model that makes this an acceptable risk

The following business models are based partly on Open Source (where the models have been well documented) and partly on the Open Systems that are found in contemporary infrastructures. Not all of these are unique to Open Systems, but our comments will concentate on the openness dimension.

Sell more Infrastructure because others supply innovative plug-ins
Give away the platform, sell the applications
Sell more solutions because of the potential for innovation
Reduce costs because of potential for reuse and outsourcing
Reduce costs because of potential for commonality
Give away system, sell support

Figure 7. Business Processes

By and Large, these business models are self-explanatory. In the following I will restrict myself to a few brief comments, with the intention of expanding on these in a separate paper.

Sell more Infrastructure because others supply innovative plug-ins - For example, this is what encourages the supplier of a proprietary operating system to maintain the openness of their infrastructure, because other suppliers supply innovative components that enhances the appeal of the operating system. The infrastructure and component supplier are in a symbiotic relationship, where each enhances the income of the other. This only works because of the openness of the interfaces. Although the governance is not open, its essential propery of fixedness is guaranteed by the market.

Give away the platform, sell the applications (or the service) - In an extreme case of separating the appliactions from the infrastructure, you can afford to give the infrastructure away if customers pay for their use of it (e.g. mobile phones). This only works because openness is assured.

Sell more solutions because of the potential for innovation - The customer is persuaded to buy because this is an Open System and they can expect more innovation, more rapid evolution and competitive costs because there is a market in components. The supplier must create a business model that sees a suitable bottom line from a combinatuion of more sales (of a lesser capability) and reduced costs.

Reduce costs because of potential for reuse and outsourcing - Components can be more economically adopted if they already exist or if they can be procured from outside organisations.

Reduce costs because of potential for commonality - The developer of components can more readily realise their investment if the same component can be used across many solutions. This is a key attribute of Open Systems.

Give away system, sell support - The familiar business model from Open Source, where customers are prepared to pay for the skills required to develop solutions based on the Open Architecture, but value the independence of sourcing that they anticipate accompanies the fact that they (or their alternative supplier) can have access to the source code of the solution. Much the same business model appplies to Open Systems which include proprietary components, when those components are built to standard interfaces and can potentially be sourced from multiple suppliers.

There are, of course, as many business models as their are businesses. Open Systems present an opportunity for suppliers to meet the needs of customers with rapidly evolving business needs, where the risks that the customer anticipates in their business are mitigated by the prospect of rapid deployment of new capability and where the prospects for the supplier are enhanced through increased demand and reduced costs.

8 Conclusions

The purpose of this note has been to begin to document the concepts of Open Systems and Open Architectures.

We have enumerated the benefits and the blockers to Open Systems. We have described some of the principles and introduced some of the methods that an architecture team would adopt when developing an Open Architecture. We have argued that Open Systems will lead to more rapid evolution, to more resilient solutions and to economies of scale and schedule.

The methods outlined here need further elaboration and that will be the subject of a subsequent paper on Architecture Methods.

An extensive bibliography accompanies this paper. It is annotated and will be kept up-to-date. It is available, along with this paper, at **http://openpdq.com/OpenSystems**