Embedded Electronic Systems and Software Architecture

Most Experts in Embedded Electronic Systems recognise that using an identifiable and maintainable software architecture has significant benefits in development time, cost and operational reliability. In fact most of the standardisation effort within specific industry segments has achieved alignment of the different software architectures to create a standardised (common) software architecture or at least a set of standardised interfaces between architectural components, but done little to improve the overall product (system).

Although beneficial, the reality is that the ‘software architecture’ is a small and relatively insignificant part of the embedded system puzzle. Software Architecture is NOT the only component of Electronic product design.

These initiatives, although welcomed in raising the overall standard of the software components, should not be seen as the key answer to better products. First we need the recognition that better products (systems) come from better systems engineering, not just from better defined, developed and controlled components (albeit that at the pinnacle of engineering these all become important).

In an attempt to argue the statement made in the title, two distinct comparisons are given, one outside the IT industry and one within. These attempt to highlight the criticality of System Architecture, rather than Software Architecture, as a topic by highlighting the latter’s limited scope.

Analogy 1 – The design and development process
Even seasoned electronics engineers and system designers will probably recognise the numerous skills and trades employed on civil construction projects and also be aware of the consequential costs of a mishap. For any construction project, from simple maintenance to whole new building and at every level of expertise from a hapless DIYer to a seasoned tradesman…the analogies to electronics and software can be readily drawn

Would such an engineer allow his new house to be constructed from a standardised set of foundations up to, say, the floor slab (with no emphasis on the quality of that implementation) and yet allow the rest of the house design and implementation to be unconstrained save only a set of brick-laying rules (which were given as guidance rather than absolutes)? Or even an (inadequate) sketch of what the key structural interfaces should be?

Analogy 2 – The standardised architecture in reality
The ubiquitous PC, common hardware, universal operating system, well defined API, network for larger systems and many and varied applications from different sources. Software standardisation efforts aim to address quality improvement (or at least mitigation) with increasing complexity of application and the ability to drive cost by multi-party sourcing. Much of the standardisation is at electronic interface level and less rigorous at software level.

In both analogies, what the author tries to make apparent, is that in isolation, the software architecture is only a component of electronic system design, not a silver bullet that defines entire product designs! We need to engage on the remainder of the architectural discipline at a more system level, if we are to succeed.

Analogy 1 – The Construction Industry

The following analogies are almost certainly incomplete, but are deliberately used to demonstrate an attribute of the system, rather than be a perfect match. For instance – not many buildings are built to be ‘portable’ i.e. moved from plot to plot (with the exception of mobile homes) but it is perfectly feasible for an architect to achieve this capability. However, most modern building plans are expected to realise a significant amount of re-use (usually as cosmetic variants or mirror images say), which the architect has to encompass.

For the particular example, the component design of a new embedded controller forming the totality of an electronic control system (a single box solution) is likened to a new building project. Obviously bigger systems can be designed, which may encompass many ‘components’, in either industry and are readily envisioned.

An interesting observation is that a new build house has no verification or validation testing (barring inspection/audits during implementation). Success is built on confidence of previous designs and processes and a mature understanding of the engineering (or design) margin. A Warranty against (catastrophic) failure is only given for a limited period.

An 'Architect' (in a domestic building scope)

In construction industry terms an Architect realises the entire 'design' of a building on behalf of a customer, from the customer's intended use, through materials, aesthetics, local legislation and codes of practice, costs etc. The architect would consider detail from outside appearance (how the building suits its surroundings and environment), how it impacts those surroundings (drain on services, roads to air current movements) to internal décor and lighting detail.

Generally he also programme manages a whole series of experts on the detailed engineering of the subsystems of the design, covering such topics as structural engineering, and inevitably the build process itself (suppliers, subcontractors, local authorities, timescales etc).

For a normal domestic dwelling, the architect would typically be a single individual.

A domestic dwelling is a complex system, with many entire subsystems that have to interface and complement one another. In order for this to be achieved the architect must outline the design, partition the work and solicit responses from the chosen suppliers, facilitate the discussions between all parties and negotiate and document a detailed definition of the final design.

Then, throughout the execution of the development he has to ensure that everyone follows the plan and, where deviation becomes necessary because of unseen circumstances, find a compromise that is acceptable to all.

In this role he is not an expert in every discipline, but he is knowledgeable enough to ask the pertinent questions, or has sufficient trusted partnership relationships to be confident in the final design. Experimental design carries more risk as it pushes the boundaries of the body of expertise.

Local authorities may not only define standardised interfaces to services to the building, but may also demand certain materials definition or aesthetics, standards of workmanship etc and have a series of inspectors, without whose inspections and approvals (at critical design phases) the construction may not be completed (no authorisation).

The System Architect (in Embedded Electronic System scope)

To be consistent with the construction industry he

- *Should* realise the entire 'design' of a system on behalf of a customer, from the customer's intended use, the materials, aesthetics, local legislation and codes of practice, costs etc.

- *would also* programme manage a whole series of experts on the detailed engineering of the subsystems of the design, covering such topics as electrical or mechanical engineering and of course software, and inevitably the build process itself (suppliers, subcontractors, local authorities (country of sale), timescales etc).

The System Architect is typically the product (system) champion for all of the system engineering activities required for the product (system), and as such is a mature Systems Engineer in outlook and competence.

The typical experience in engineering companies however is that invariably the Hardware and Software competences have to interpret a System Engineer's view of the System design (usually without access to the rationale that led to the design decisions or how they achieve the customer's requirement), partitioning the hardware and software from experience and cost, rather than driven by design or performance expectations. In so doing, these Hardware and Software engineers are one step removed from the customer's and designer's requirements.

The designs are largely tackled as 'custom' solutions, admittedly using previous experience, but with less than an optimal eye on re-use at a design level (re-use at a component level does exist - c.f. different plans, reclaimed bricks!). Frequently the solution proposed to the customer by the system designer offers little opportunity to re-package existing components for appropriate cost, timescale and reliability gains.

The design is then scheduled for implementation, with little feedback to ensure that the design is at a minimum 'adequate', save the 'completeness' check – i.e. it satisfies all interfaces and appears to do the job. No critical review, no oversight by the original system designer, who in general does not have sufficient knowledge of the discipline to ask insightful questions, and usually no trust of the parties involved!

The programme manager frequently has no design expertise and no knowledge of the disciplines or their interaction – his role is generally that of a progress chaser and customer interface (with regards development execution and delivery)

For a simple single embedded control system, the system designer, hardware lead, software lead and programme mangers are all different people with little appreciation of each others roles and/or complexities of the disciplines.

An embedded control system may well be a simple system, with a few subsystems that have to interface and complement one another. The architect might outline the design, with no view on its execution and so not take responsibility for partitioning the work with the (captive) suppliers. It is unlikely he would have sufficient knowledge to facilitate any discussions between the competences so it is unlikely that a coherent document of a detailed definition of the final design will ever exist.

Then, throughout the execution of the development the programme manager must ensure that everyone follows the plan (which is poorly defined technically) and, where deviation becomes necessary because of unseen circumstances, find a compromise that is acceptable to all, from a position of no inclusion at the original design and little knowledge of the skills disciplines involved, the customer's requirement or the designer's intent.

In this role he is not an expert in every discipline, he is not knowledgeable enough to ask the pertinent questions, and distrusts partnership relationships so has little confidence in the final design. With poor understanding, experimental design can be incorporated by enthusiastic engineers and the risk to the project escalates.

The standardised interfaces to services are frequently minimal and generally only imposed by a specific customer, and leave open the requirements on materials definition or aesthetics, standards of workmanship etc. Any inspectors are treated as pariahs, are allowed little overview and whose inspections and approvals (at critical design phases) carry no authority to alter the course of development.

Analogy 2 – The PC world

For the particular example, the component design of a new embedded controller forming the totality of an electronic control system is likened to a single PC with a set of applications running on it. Obviously bigger systems can be designed, which may encompass many 'components', in either industry and are readily envisioned.

The Personal Computer probably has the most diversity of application developers/designers and has long-since trodden this route – a few competing standardised architectures (foundations consisting of variant hardware and software designs, implementing their own standardised API – e.g. Linux / Windows) still exist, most of these differ in name and detailed execution rather than approach, but that is sufficient for incompatibility.

In general PC applications do not interoperate with one another – unless they are from the same design stable – and even then, with limited data transfer interfaces. Change is usually burdened in pursuit of improved performance and is therefore rarely backward compatible.

Different applications are still re-engineered for a number of different platforms. In fact a number of 'virtualisation' techniques and entire industries have evolved to fill this 'non-standard' gap. As the number of platforms increases so do the diversity of techniques for handling them as 'standard' devices.

At a node level:

In a Windows API the definition is fixed (static) and broad. No interfaces are 'optimised out', no 'assembly process' has to be executed.

For a typical Electronic Control Unit – a configuration solution (usually pre-compilation, resolves interfaces for connected components of that application with that particular Hardware and Device Drivers. No redundant interfaces at either Application or from the Device Drivers should exist.

At a System level:

In a Windows API the definition is fixed (static) and broad. No interfaces are 'optimised out', no 'assembly process' has to be executed. The 'sockets' layer resolves routing of all services to local or remote nodes dynamically.

For an Electronic Control Unit the connectivity and routing is often a design time association, in the pursuit of performance and the assurance of predictability from static resource allocation.

The Development traits of PC Hardware and PC applications

- Hardware platforms that are grossly over-specified for any individual task
- Hardware platforms that are 'cheap' only through volume / competition
- Software applications that are 'bloated' for any individual task (services defined and employed even though they are unused in that application)
- Software applications that place no warranty on their use
- Software applications that (may) co-exist, but do not co-operate
- Software operating systems which fail to isolate individual applications from one another (performance, errant behaviour, security)
- Software systems which are impossible to test for all possible derivative environments (hardware or other applications)
- Software systems that are unstable and/or unreliable because of the control of interactions
- Dynamic behaviour that is 'co-operative' but not deterministic

What is "Software" Architecture?

Generally it is considered as the High-level structure of a "software" system. Its important properties are:

- A high-level abstraction that allows the "software" system to be viewed as a whole
- A framework that supports the functionality of the system (including dynamic behaviour)
- A structure that accommodates security, flexibility, extensibility as well as current functionality – at a reasonable cost of change
- That it hides implementation detail

But this should be true of System and its subservient Electronics and Software!

Although the above definition is defined for software, it should be equally applied to System and Electronic Hardware (although it would appear all too infrequently in these domains!). It can be applied recursively to each stage of the architectural view, starting abstract from the physical components in a functional description.

Application Architecture is the consideration of the parts that make up a system: how many there are, how big they are, how they divide the responsibility, how they collaborate and how they communicate. All this must be viewed at a high enough level of abstraction that it may be applied to more than one system. Once understood across the many systems, so the focus can be brought on the individual system, each in turn.

What does Software Architecture achieve?

- Decides how features fit together (all internal joints or assembly points – traditionally through defined 'connectors', often seen as 'contracts' between components)
- Decides on the quality (trade-off) of the features and/or assembly process (Cohesion versus Coherence)

- Defines the interface mechanisms (Communication and Control) necessary to achieve system behaviour
- Supports isolation of subsystems such that failures are contained / protect the remainder of the system
- Defines the limitations of capability/flexibility (parameter, range, precision and types used during calculation and/or interfaces)
- Defines the re-usability/extensibility of the components (at a component level)
- Support the testability of the components and of some subsystem assemblies
- Increases the first-time implementation cost in support of the future
- Potentially increases the overhead and cost of the implementation to save on future engineering cost
- Potentially improves the whole-life cost by enabling re-use and reducing risk
- Improves the reliability/availability/safety/security of the system

What does Software Architecture NOT achieve?

- Define a System Architecture
- Define a Hardware Architecture
- Decide what the product does (component design/algorithm design or implementation detail)
- Decide that the product matches its intended purpose
- Decide on the product performance (although a bad software architecture can hinder this!)
- Define the product cost (Piece part of total engineering)
- Differentiation of product for different System realisations

Example 1

Architecture decisions may lead you to define an internal supporting framework. It can decide on data and control interfaces, but it relies on the system design to tell you what parameter size and what loading (capacity/frequency) is placed on them.

Good (software) architecture may protect the remainder of the system if the system is poorly designed and over-stress causes components to fail. The limitations of the architecture and implementation need to be fed back to the system design to ensure design re-use understands the limitations of use for future system re-use.

Example 2

Architecturally, many users may want access to the same information from remote nodes, but the provision of the information of interest needs to be managed to be consistent and non-conflicting (ideally from a single source). Local variations may wish to have this information presented for use in slightly different applications, or to support local decisions, or computation, which may in itself be fed back to other users of the network. Design decisions have to be made versus availability, traffic capacity, age of data, filtering, resolution of conflicts for the dynamic of the system as well as the static design.

The Software Architecture does not live in isolation, but must be co-designed and coherent with the System architecture and Hardware architecture of all nodes on which the software is expected to reside or be executed (which may be many instances). Software limitations (of the architecture and eventually the performance or functionality of its implementation) need to be fed back to the system design to ensure design re-use understands the limitations of use for future system re-use.

Systems Engineering

The Systems Engineers (under the instruction, guidance and championing of the System Architect) must take the customer’s product vision and convert it into well-defined specifications of components that are able to be engineered. Those components, once assembled, must be assured to deliver the system performance, functional and other attributes that the customer desired.

Ideally the Systems Engineers will have achieved this through many trade studies, selected technologies that are fit for purpose, give an optimal use of resources and achieve many other non-functional goals such as safety, security, reliability, robustness that the customer will value to ensure future business engagements.

The System Engineer must not only define the component specifications (which indeed may be sub-systems in their own right) must be sufficiently well-specified in the key attributes that affect system performance, he must also ensure that they are able to be integrated and assembled in an appropriate manner. (E.g. Imagine defining a bridge that was validly defined but could not constructed across a tall ravine, as no mechanism existed for its in-place construction.)

He must ensure that the systems of operation, maintenance and ultimately disposal are addressed as necessary. He must also ensure that the evidence exists that all the components meet their component specifications such that the integrity of the system matches his model, or calculations.

In large systems, of significant complexity, usually by virtue of numbers of components, or component interactions, he must assure himself that any emergent property (i.e. behaviour that results due to unforeseen properties of the components, or the way the system is used) is acceptable or appropriately managed or mitigated.

As the ‘front-man’ to the customer, the system engineer must define a system to do what the customer envisioned (which may be different from what he defined)... and help him accept that the final assemblage delivers that vision.

Mechanical Systems choreographed by Software

Much of our modern world consists of physical sensors and mechanical actuators, whose control and precision is achieved through embedded control systems of significant complexity. That these systems are then connected, by example, via communication networks and power conversion systems, with independent controls of routing, and power generation strategies, all seeking different optimisation goals, makes many of our modern day systems inherently complex.

That we deliver these by substantial software systems with increasingly standardised architectures is a key component in our comprehension of the mechanisms, but is not the key enabler to delivering better systems.

We need to improve our System Engineering skills, to deliver optimal system solutions.

Most modern day development have significant disconnects between the Systems Engineering of the entire system (for a control system that would include all the mechanical and physical components as well as the choreography of those components through some control algorithm) and the realisation of the components (or sub-systems) to be engineered.

Our next big challenge is to step seamlessly from the description of the problem, as seen by the System Engineers, through the prescription of the solution by various system components, through the implementation, verification and integration of those components, to the validation of the system solution to the problems for which the system was envisaged.

This new world will need much better communication between the component engineering and system engineering, to allow the levels of recursion, trade studies and confidence in solution technologies, to ensure development of better future software enabled systems.

A standardised (and therefore rigid) software architectural solution may no longer be an appropriate place to start.