Systems Architecture: The Empirical Way — Abstract Architectures to 'Optimal' Systems

Democritus and Plato Re-engage Digitally 2,400 years on!

Graham R. Hellestrand
VaST Systems Technology Corporation
1250 Oakmead Pkwy, Suite 310
Sunnyvale, CA 94070
+1-408-328-0949
g.hellestrand@vastsystems.com

ABSTRACT

It is a profound dislocation to have reality replaced by models – that revolution is won, that blood spilt. The first radical changes in global companies that I have witnessed, attributable to the systems engineering and architecture Virtual System Prototype inflection point, occurred in 2004. In these instances, the whole assumed order of architecture in the engineering process was tipped upside-down by a single set of quantitatively derived results that are turning out to be critical to each company's ability to compete and win in its fiercely competitive market. These changes have accelerated in 2005. It is the most exciting time for us all to be part of the wholesale transformation of one of the fundamental engines of the last 50 years of economic growth – Embedded Software-Electronics Control Systems (Embedded SECS) design. The transformation is driven by necessity; the by-products are competitive advantage, speed of execution, quality, productivity, and ability to rapidly adapt to market and business conditions. There is no company, leader or laggard, who can afford to ignore this evidence and hope to survive. This paper addresses the quantitative development of embedded systems architectures - software, hardware, mechanical.

Categories and Subject Descriptors

B8 [**Hardware - Performance & Reliability**]: Performance analysis and design aids; C0 **[Computer System Organization]** General – *System architectures, System specification methodology, Hardware/Software interfaces, Modeling of computer architecture;* C2 **[Computer Communication Networks]** Network architecture & design – *distributed networks*; C3 **[Special Purpose and Application Based Systems]** Real-time and embedded system; C4 **[Performance of Systems]** Measurement and modeling techniques; G3 **[Probability and Statistics]** Experimental design, Multivariate statistics.

General Terms

Design, Experimentation, Measurement, Performance.

Keywords

Quantitative systems architecture, system design process, mapping system architectures, empirical system design, event-based objective function, factor and concept based explanatory function, event data driven optimization, concept driven optimization, mesomorphic architecture, epimorphic architecture.

1. Background and Motivation

An empirical approach to composing optimal architectures for complex embedded systems is relatively rare. The use of physical systems to determine early architectural optimizations is clearly a non-sequitur. The complexity of processor centric, electronic systems that control modern gadgets (such as, cell phones, automobiles, base stations, consumer products) requires a systematic, as well as intuitive, approach to determining an optimal fit for the intended product. And in today's environment, where a company's engineering process is being used as a competitive advantage to dominate competition, the luxury of optimality, across cost, performance, power consumption, quality and time to market (TTM), has turned to a necessity.

Historically, intuitive engineering architecture attempted to optimize systems using dimensions of performance (major) and cost (more or less minor). Since control system architects rarely understood, or had access to, the software that would run on their architectures, they produced conservative, often grossly over-engineered inventions that, when assessed using the now mandatory dimensions – major: cost and performance, average and peak power, TTM, quality; minor: programmability, manufacturability – were usually poor fits for the service required. The failure was due to lack of (i) necessary and sufficient data to determine optimality and, concomitantly, (ii) an empirical process that enabled quantitative decisions to drive the architecture specification process.

The ability to support data-driven decision making early in the engineering process has been an underlying driver for building accurate models of an intended system. Then to perform many hundreds of experiments on successive models to determine an optimal system architecture, measured across agreed dimensions and for a particular product fit, required not just accurate models but also high performance models. The experiments that need to be performed require accurate models of successive physical systems (called Virtual Prototypes (VP)) that can run whole systems of software (operating systems, drivers, communications stacks, middleware, and applications), together with other inputs and outputs (communications traffic, interrupts, video and audio) – the whole *system enchilada* is called a Virtual System Prototype (VSP). Data is typically collected from many probes inserted into the hardware models (electronic, mechanical, RF) and the actual software during the experiments. It is not unusual for one such experiment to run 100 billion instructions – less than 1 hour of simulation time using a high performance, timing accurate VSP. The remaining needs to enfranchise efficient experimentation are (a) availability of models to enable the construction of VSPs and (b) the tools to rapidly build the models and to construct the VSPs.

An historical anecdote, even though VSPs – by definition high performance and timing accurate - are the foundations of empirical electronic systems architecture, their initial economically justified deployment has been for the development of software for real-time, critical control systems. As VSPs are deployed to determine architectural specification, we are seeing the nature of the architecture of electronic systems changing – the architecture of the software and the platform are the twin 1st orders of focus and processor microarchitecture no longer dominates systems, often having a 2nd or 3rd order influence. Architects are not likely to be just electronics or micro-architecture engineers or computer scientists, but those engineers creating the middle, straddling the oft-thought binary software-hardware line of engineering competence. It is extraordinary to be a participant in this pervasive change, when quantitative experimentation is instituted as the key to building competent architectures, the results are usually unexpected, startling, and will rapidly transform the engineering processes and structure of the company.

The paper will concentrate on the empirical process underlying data-driven architectural decision making and the capabilities enabled when the 'optimal' architecture becomes the executable golden reference model driving the remainder of the system engineering process – in particular, the software development and the hardware design sub-processes. We will use the genericized industrial examples to motivate the discussion.

2. The Philosophy of Architecting Systems

Like the two contemporary philosophy adversaries in Greece 2,400 years ago, Democritus and Plato [1], today system architects are divided into two camps – the empiricists and the ratiocinators. The empiricists believe that fundamental truths are derived from observation within a quark-like atomic framework, the ratiocinators that fundamental truths are intuitively apprehended. This paper is firmly in Democritus's camp and addresses the quantitative basis of decision making as fundamental to architecting 'optimal' systems. But Plato still has his voice!

2.1 System Architecture as Ratiocination (Plato)

There are common elements in the empirical and ratiocination frames of operation. Both use a basis of knowledge and both use reasoning. The stoic's view of ratiocination was that seminal reasons were the impetus for animate motion [2]. In other words, the nature of a whole cannot be discovered by dividing it into its component parts and studying each part by itself, that there is a qualitative, unquantifiable aspect to observation.

Strong reflections of this philosophy are in evidence today with the architecting of systems being largely intuitive, qualitative activities. The tell-tale signs of this are the typical tools and methods used by system architects, sadly even of advanced technology products - spread sheets, adoption of under-examined legacy structures and requirements, non-quantitative decision making criteria, qualitative Socratic argument, and drawing conclusions by deduction and induction derived from these activities.

This approach guarantees incremental systems architecture driven from legacy systems. And has little probability of arriving at *optimum* solutions.

2.2 System Architecture as Experimental Science (Democritus)

Atomism is an analytic doctrine that teaches that systems can be discovered by dividing them into component parts and individually studying each part. And has as a premise that all observable changes are caused by actions of the individual parts. The indivisibility of the fundamental parts relates only to the syntactic and semantic structures used to specify and build systems. Then all systems are aggregates of juxtapositions of fundamental parts. In reverse, all systems are subject to quantitative analysis. Reasoning has to do with the meaning of the observations.

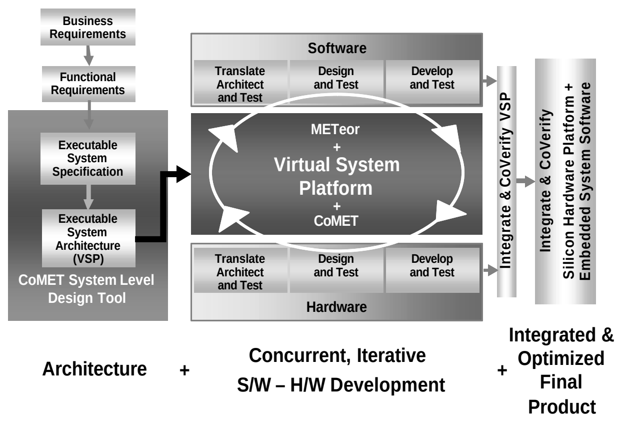
Business Requirements
Functional Requirements
Executable System Specification
Executable System Architecture (VSP)
CoMET System Level Design Tool
Software
Translate Architect and Test
Design and Test
Develop and Test
METeor + Virtual System Platform + CoMET
Translate Architect and Test
Design and Test
Develop and Test
Hardware
Integrate & CoVerify VSP
Integrate & CoVerify Silicon Hardware Platform + Embedded System Software
Architecture
+
Concurrent, Iterative S/W – H/W Development
+
Integrated & Optimized Final Product

Figure 1. Software-Electronics Control System (SECS) Engineering process

This approach, if it systematically traverses the quantitative factors defining the space of likely architectures of the product underdevelopment, and undertaken using hypothesis refutation from Science, cannot guarantee an optimal solution, but will give quantitative reasons for rejecting, possibly, thousands of candidate architectures that were poorer than the selected architecture and, as in Science, provide peer groups with adequate data to further critique the architecture group's recommended candidate for themselves. This is good science and engineering. This philosophy is actualized by employing modern *analysis* tools and methodologies – using VSPs as experimental vehicles, multivariate statistics to help drive decision making, and design of experiments to drive the experimental methodology. However, even with all of the power of factor analysis and structural equation modeling [3], making sense of the data remains in the domain of ratiocination –

but about the data best supporting or refuting the candidate explanations. [4] gives an interesting and enlightening introduction to the linkage between the theoretical/hypothetical and empirical facets of science (& engineering) from the perspective of the more recent practitioners of the philosophy of science.

3. The Empirical Engineering Process

A modern processor centric, SECS engineering process is depicted in Figure 1 and is an evolution of the process presented in [5]. Apart from the Business Requirements to Functional Requirements flow, where Functional Requirements are typically derived *manu et* mente using Use Cases operating on the Business Requirements, the rest of the activities in the engineering process are increasingly being characterized and driven by empirical methodologies and is the focus for increasing automation. The Executable System Architecture (VSP), that results from a mapping of the Executable System Specification, drives both software development and hardware design - two subprocesses that are completely overlapped in time. The final stages are integration and full verification of the VSP, followed by integration of the same software as verified in the integrated VSP on the silicon platform followed by full verification of the silicon system. The concurrent software-hardware design and development, followed by the VSP and silicon integration and test steps of the process are relatively well known and are not discussed.

3.1 The System Specification and Architecture Derivation process

A less iconic view of the left-side subprocess of Figure 1 is shown in Figure 2 - derived from [6].

The path from an Executable System Specification to an Executable System Architecture is not straight, as can be seen in the coloured section of Figure 2. The starting point to derive an Executable System Specification is the Function Requirement. A Use Case analysis naturally, although not yet automatically, leads to a Data Flow Diagram representation of the Executable System Specification (see Fig. 3A below). There is also a two stage mapping process that results in a Virtual System Prototype and a final mapping process that results in an *optimal* VSP.

The upper part of Figure 2 shows essentially a manual subprocess. The lower part - the subprocess for mapping Executable System Specifications to Executable System Architectures – is the focus of the statistical analysis that drives the empirical approach to architecture. There is no well-accepted flow at the architecture level and it is an objective to characterize a standard flow together with the statistical machinery to enable the entire decision making to be quantitatively driven.

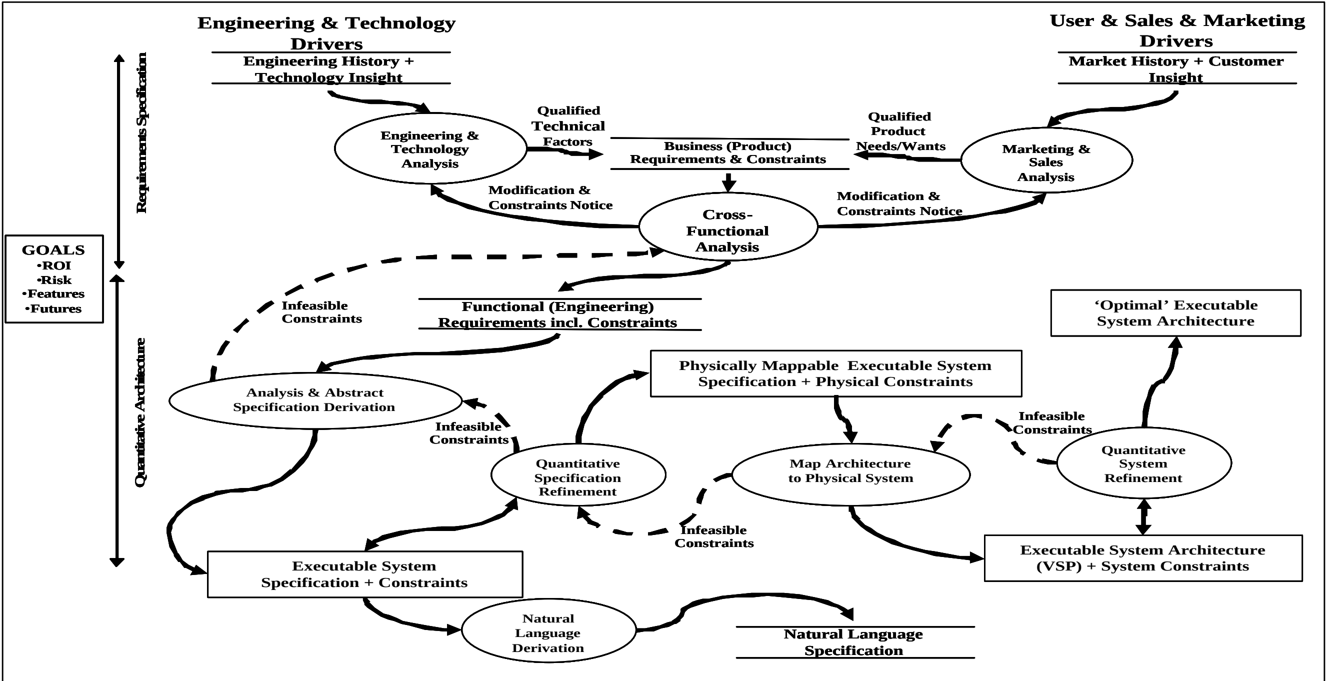
Engineering & Technology Drivers
Engineering History + Technology Insight
User & Sales & Marketing Drivers
Market History + Customer Insight
Requirements Specification
Engineering & Technology Analysis
Qualified Technical Factors
Business (Product) Requirements & Constraints
Qualified Product Needs/Wants
Marketing & Sales Analysis
Modification & Constraints Notice
Cross-Functional Analysis
Modification & Constraints Notice
GOALS
•ROI
•Risk
•Features
•Futures
Infeasible Constraints
Functional (Engineering) Requirements incl. Constraints
'Optimal' Executable System Architecture
Quantitative Architecture
Physically Mappable Executable System Specification + Physical Constraints
Analysis & Abstract Specification Derivation
Infeasible Constraints
Infeasible Constraints
Quantitative Specification Refinement
Map Architecture to Physical System
Quantitative System Refinement
Infeasible Constraints
Executable System Architecture (VSP) + System Constraints
Executable System Specification + Constraints
Natural Language Derivation
Natural Language Specification

Figure 2. Architecture Specification and Derivation Process

The next step is to determine what Executable System Specifications and Architectures are.

3.2 Source Specifications and Target Architectures (VSP)

To constrain the domain of the task of producing optimal systems from abstract specifications, specific models are used to describe Executable System Specifications and Executable System Architectures / Virtual System Prototypes. These are described below.

3.2.1 Executable System Specification

The Executable System Specification, derived by refinement from the Functional Requirements is an abstract, executable model that defines the hierarchical system structure recursively, in terms of tasks (modules with ports) and their intercommunications, along with (i) task behaviour and timing within tasks and (ii) the cause-effect and timing relationship of communications between tasks. A natural language document describing structure can be derived from the executable model but the behavioural description lies with the executional characteristics of the model that are, necessarily, defined in the Functional Requirements. The operational behaviour of the tasks is likely to be described in a number of heterogeneous notations such as MatLab/Simulink, UML, C/C++, functions, etc. depending on the nature of the functions to be computed, such as solving a set of differential equations, iterating through a set of a control algorithm, or computing a filter function. A typical executable system specification appears in Figure 3A.

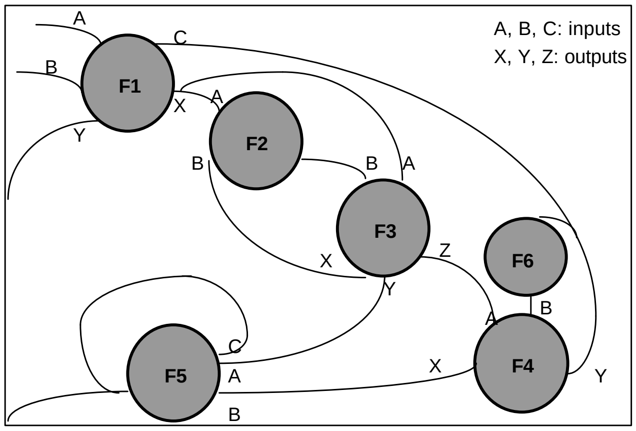

Figure 3A: Data-Flow Description (DFD) of an Executable System Specification

DFD A B :: [F5 [*first*+1_1, *first*$_1$ F4 [*first*$_1$, F6 F4[Y]]
] F3 [*first*, F2 F3[X] *first*], *first*+1_1] F1 A B F4[Y]

Figure 3B: An Executable Functional Description of Fig. 3A

There is a high degree of concurrency in such systems and the degree of simultaneity is a factor of both task behaviour and timing, and communications and the cause-effect relationships carried by each communication. DFDs, at this stage, are abstract and carry little realization information apart from that implicit in the timing constraints associated with task behaviour and communications. For convenient mathematical treatment, the DFD can be reduced to at least one common specification notation – functions and forms [7],[8]. Without traversing the gory details of the notation, the diagram in Fig 3A reduces to the function description in Fig 3B that is well specified with a timing semantics [9] and tractable for execution and mathematical manipulation and reasoning [10].

3.2.2 Executable System Architectures (VSP)

During translation from abstraction to realization, abstract tasks in Figure 3A (functions in Figure 3B) are able to be mapped to software tasks or hardware modules, having the same semantics and communications capability, and the communications are able to be mapped onto hardware communication channels and/or software communications structures (parameters in function calls, inter-process messages, etc.). It is not true that all abstract (or software/algorithmic) constructs map sensibly to hardware implementations – for instance: recursion, dynamic process/data creation and destruction, self-modifying code.

Figure 4 shows a typical virtual prototype (hardware) target for the mapping of an abstract cell phone system. In an ideal mapping process, like that described in the coloured subprocess in Figure 2 perhaps, the VP in Figure 4 would be one of many targets involved in empirically determining an optimal VSP.

In a mapping from an abstract system into a VSP, the goodness of the mapped VSP can be judged against the objective functions defined for the abstract system by measuring the potentially thousands of responses of the VSP while executing the mapped software and subject to the system inputs, and then analyzing them using multivariate statistics. A somewhat ad hoc process, but, nonetheless, scientific and empirical when guided by hypothesis construction and refutation. The potential number of experiments is clearly enormous and again the statistics of the design and analysis of experiments, together with the multivariate statistics is helpful in managing this problem. An alternative approach is to use conceptual models capturing the intent of the Executable System Specification and constructed as part of the statistical analysis, together with design of experiment methodology, to drive the mapping and then perform confirmatory experiments on the results.

The remainder of this paper focuses on empirical mechanisms for producing *optimal* Executable System Architectures from Executable System Specifications.

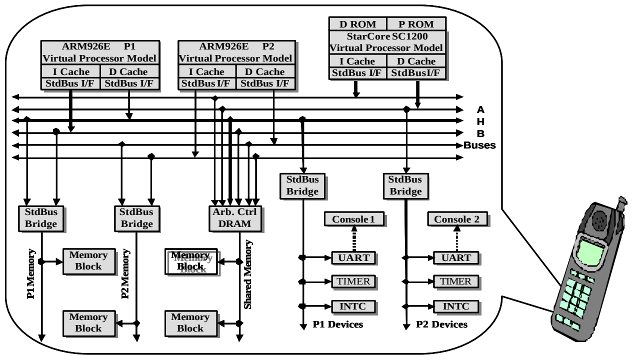

Figure 4: A Typical Virtual System Prototype for Mobility

4. Foundations of the Empirical Process

The empirical process in systems design has twin objectives:

i. The optimization of a system for a particular usage; and

ii. The optimization of the design process, with specialization for producing different systems.

Our major objective in this paper is the optimization of systems at the platform level, given a specific objective function (such as, maximize performance, minimize power). The scientific method is about rejecting hypotheses using a rational, data driven decision making process. One of the challenges in making decisions in this engineering domain is the complexity of modern super systems [11] and identifying patterns in, and making sense of, the potentially billions of pieces of data collected from hundreds of unique sources of measurement of platform activity and latency, available from the silicon and simulation. To address the latter problem we need to enlist the capability of multivariate statistics to reduce raw data to measures of relevant and evaluative factors and, even more desirably, elucidating and reductive (abstract) concepts.

On the optimization side, there are many ways to construct objective functions. The classical way is to track event frequencies and/or latencies and to construct ad hoc functions based on functionally related events, such as CPU events, bus and bus-bridge events, memory events, device events, etc. A more systematic way is to use the multivariate statistics to help formulate dependence relations based on abstract concepts and more concrete factors derived from the interpretation of highly correlated events measured during simulation or silicon activity. Both approaches are described below.

$$\boxed{\begin{aligned} F_{VSP}(| \; & f_{CPU}(\Theta_{cc=0..cn} \circ f_{CPU_{cc}}(\Theta_{CEvType=1..cet} \circ g_{CPU_{cc,CEvType}}(\Theta_{CEvCnt=scecn..tcecn} \circ Event_{CPU_{cc,CEvType,CEvCnt}})), \\ & f_{Bus}(\Theta_{bc=0..bcn} \circ f_{Bus_{bc}}(\Theta_{BEvType=1..bet} \circ g_{Bus_{bc,BEvType}}(\Theta_{BEvCnt=sbecn..tbecn} \circ Event_{Bus_{bc,BEvType,BEvCnt}})), \\ & f_{BusBridge}(\Theta_{bbc=0..bbcn} \circ f_{BBus_{bbc}}(\Theta_{BBEvType=1..bbet} \circ g_{BBus_{bbc,BBEvType}}(\Theta_{BBEvCnt=sbbecn..tbbecn} \circ Event_{BBus_{bc,BBEvType,BBEvCnt}})), \\ & f_{Mem}(\Theta_{mc=0..mcn} \circ f_{Mem_{mc}}(\Theta_{MEvType=1..met} \circ g_{Mem_{mc,MEvType}}(\Theta_{MEvCnt=smecn..tmecn} \circ Event_{Mem_{mc,MEvType,MEvCnt}})), \\ & f_{Dev}(\Theta_{dc=0..cn} \circ f_{Dev_{dc}}(\Theta_{DEvType=1..\mathrm{det}} \circ g_{Dev_{dc,DEvType}}(\Theta_{DEvCnt=sdecn..tdecn} \circ Event_{Dev_{dc,DEvType,DEvCnt}}))) \end{aligned}}$$

$$where \quad f_{CPU_k}(\Theta_{EvType=1..et} \circ g_{CPUk,EvType}(\ldots)) = f_{CPU_k}(g_{CPU_k,1}(\ldots), g_{CPU_k,2}(\ldots), \ldots, g_{CPU_k,et}(\ldots))$$

4.1 Event-Based, Objective Functions

In an event driven simulation environment, an objective function can be expressed as a function whose parameters are functions each characterizing contributions to the objective function of one of the components constituting the system, viz. CPUs, buses, bus bridges, memories and peripheral devices. The parameter functions themselves have parameters that are functions of simulation event types sourced from the various event activities that occur in a VSP during simulation. In general, an objective function will have the following form:

Fundamentally, an objective function is a function of functions of functions of events.

A simple way to visualize and compute an objective function is to build an interpretation table, as in Table 1, below.

These tables are large and even though the *Event Bindings* are simple to implement, typically a pointer to a function and a history buffer of events, the extraction of data from register transfer (RT) models or representative samples of the silicon is difficult, time consuming and subject to experimental errors. The inability to map an event in a behavioural model to an observable data point in the silicon or RT model further confounds the building of accurate tables.

Events, in event-based simulation, are associated with aggregate underlying behaviour of the circuits that the semantics of events are intended to describe. The number sources of Event Types generated by all types of components in a complex platform may number in the hundreds; the number sources of Events Instances

(events caused by the simulation of instances of typed components instantiated in a platform) numbers in the thousands, possibly many thousands. Since component instances, even though morphologically similar or identical, may have different electronic instantiations and be affected by local circuit connectivity, the objective function table may remain large. To set appropriate Event Bindings for entries in the Objective Function Table, the knowledge and skills of the silicon vendors are required. Even where there is a clear path from the event-based, behavioural description to physical behaviour, the information required for the Interpretation Table – typically constant functions but sometimes more complex – is closely guarded. Having set the binding functions, confirmatory experiments need to be undertaken to validate the settings against either an RTL reference simulation or data sourced from the silicon, preferably the latter. This process is surprisingly complex and requires the same statistical approach as is required for scientifically driven empirical investigation – this is described below.

Since the precise physical association of an event with some circuit implementation that it models is not necessarily obvious and, in many cases, unlikely to be independent of circuit effects associated with other events. The ascription of local physical semantics to an event is more in the nature of a verisimilitude. The higher the level of event-based, behavioural abstraction, in general, the weaker is the physical (or structural) connection to implementation. This is a good thing for high performance modeling and, with forethought, does not compromise timing accuracy at an agreed level of granularity – say clock-cycle level. However, it does mean that more sophisticated mechanisms are required to efficiently predict aggregate underlying hardware concepts, such as power. In addition, various event types and instances may correlate highly across a variety of behaviours - for instance, cache misses with processor initiated bus traffic, data path stalling and power consumption. The elimination of dependence between event instance sources, that are the potential *independent* variables in a statistical analysis, should have the effect of condensing the number of variables required to explain a behaviour, or to be used in the prediction of behaviour as well as reduce the propensity for over estimation using data extracted from these models.

The first exploratory endeavour will use Factor Analysis; the predictive endeavour will use Structural Equation Modeling; and finally, systematic experimentation across various candidate systems to determine the *optimal* system will use Design of Experiment methodology.

4.2 Factor and Concept Based Explanatory Functions

The intent is to use the multivariate statistical technique called Factor Analysis to reduce the number of variables needed to explain the patterns in the collected simulation data. Factor analysis groups highly correlated Event Instance sources (variables) so that they can be treated as single entities – called Factors. The factors are subsequently used in another statistical technique called Structural Equation Modeling to test whether a constructed set of equations (relationships) whose dependent variables are known as Concepts and whose independent variables are factors and concepts provides a statistically adequate description (statistical model) of the observed data gathered from the VSP event sources.

4.2.1 Factor Analysis

Analysis of event-based simulation suffers from a surfeit of events to employ as *independent* variables. Without knowing the relationship amongst events, it is difficult to determine the extent of over- and under-estimation of desired properties.

The objective of this work is to simultaneously consider all variables (events) from a simulation run and determine how each is related to all others. The essence is *each of the observed variables (events) is a dependent variable that is a function of a linear combination of the original variables (called Factors). The outcome is the ability to form factors that maximize their explanation of the entire variable set, resulting in a much smaller set of factors than the number of the original variables.* [3].

For example, a simulation of a multi-core platform executing billions of software instructions produces prodigious amounts of data from potentially thousands of event types and event instances (variables). It is useful to be able to describe the dynamic operation of the software and hardware constituting the platform in terms of a much smaller, general set of explanatory factors. A typical set of factors might be: performance, responsiveness, reserve capacity, power consumption and cost, where each factor is a function of functions of events. The effect of determining factors is to simplify the explanation process of the data in terms that are useful. There is obvious high structure (correlation) in event data (for example cache miss, pending buffer access, bus transaction, memory access) and this is the reason for the appropriateness of factor analysis in seeking simplifying explanations of the patterns in the data. High correlations amongst a group of variables indicates that an exemplar variable may be used in place of the rest of the elements of the set, thereby reducing the number of variables used in further analysis. Similarly, for the grouping of highly correlated variables a *factor* score can be computed, and an explanatory name given to the factor that can be substituted for the original variables in further analysis.

Table 1: Type, Instance, Component & Event Bindings

Component Types Binding	Component Instance Binding	Component Event Binding	Objective Function Event Binding
f_{CPU}	$f_{ARM\ 1156\,T\,2\,F}$	$f_{ARM\,1156T\,2\,F_{ICache_{LineEvict}}}$ $f_{ARM\,1156T\,2\,F_{\mathrm{Re}\,g0_{Access}}}$ …… $f_{ARM\,1156T\,2\,F_{ADD}}$	• Constant function • Function of a set of events associated with an event type (eg cache hit) in a component instance of a known component type
	$f_{SC\,1200}$	$f_{SC1200_{TCM_{Write}}}$ $f_{SC120_{Branch_{Taken}}}$ …… $f_{SC1200_{LDD}}$	• Constant function • Function of a set of events associated with an event type (eg Branch Taken) in a component instance of a known component type

Factor analysis programs identify factors in order of significance. It is the responsibility of the analyst to determine what factors to retain (dependent on significance) and interpret/explain what the factor represents in practice. In statistical terms, the correlation of each variable (event instance source) with the factor (known as the factor loadings) indicates the degree to which the variable is representative of the factor.

4.2.1.1 Example of Ratiocinated Factors Explaining Data from Simulation Experiments

Events arising in a simulation may be measured in several ways:

- Count per stimulus test over a selected set of stimulus tests (for example: the EEMBC [12] suite of software for various embedded platform benchmarks)
- Count per interval of time over a (set of) stimulus test(s)
- Duration of an indicator event (such as pipeline stall) triggered by a starter event (such as a D-Cache miss)

Highly correlated variables (Event Instances), having different measures, have been tabulated and then explained in terms of factor names in Table 2, below.

4.2.2 Factor Based Objective Functions - Structural Equation Modeling

Structural equation modeling (SEM) is an extension of multivariate factor analysis. The technique enables multiple relationships between dependent (concepts) and independent variables (factors) to be examined simultaneously. This is a confirmatory, rather than an exploratory technique (cf factor analysis), in which a prescribed set of relationships (equations) can be tested to determine if they are significant. For example, a particular set of equations describing power and performance can be tested against the simulation data to determine whether it holds true at a selected significance level (usually 95% for engineering and science analyses). An interesting issue here is that the entire set of equations (dependent/concepts and independent/factors variables) used to describe the patterns in the underlying data are constructed using ratiocination - Plato and Democritus lying in the same bed! The techniques, even though empirical, are subject to the vagaries of human naming, imputation and omission. The mechanism for mitigating such highly probable biases is to construct alternative, competing sets of equations and subject these to the same confirmatory analysis. And finally, comparing the models to determine the best fit to the data.

The equation set is derived from a set of cause-effect relationships, constructed purely by ratiocination, in which the factors derived in Factor Analysis are independent variables (causes) that precede and produce effects on (unobservable) Concepts. Concepts and causes (factors) can be used in further cause-effect relationships (recursion is impermissible). The focus of SEM is on the pattern of relationships across event sources, not on the event sources themselves.

$$SEM\,model : \{Concept_k\}_{k=1..n}$$

$$where \quad |$$

$$Concept_k = d_{f_k} + \sum_{fac=1..facn} k_{fac} \bullet Factor_{k,fac} + d_{c_k} + \sum_{fcc=1..fccn} k_{fcc} \bullet Concept_{k,fcc}$$

$$Factor_{fc} = d_{fc} + \sum_{gev=1..gevcn} k_{gev} \bullet g_{gev}(\Theta_{EvRange=s..t} \circ Event_{g_{gev,EvRange}})$$

Since ratiocination controls the conceptualization of the equation formulation, it is warranted to return to the factor analysis results and determine what variables should be regarded as real indicators of each factor and to ascribe a loading (correlation) to those variables. The attempt in this part of the process is to make variables indicators for one factor only, unless ratiocination argues to the contrary. This has the effect of reducing the number of terms in the linear equation describing a factor. The same process can be applied to reduce the number and to modify the loadings of factors on concepts and concepts on other concepts in the set of equations. It also provides the key variables, factors and concepts (all of which are causal variables) that can be manipulated during experimentation to produce desired effects on an objective function. The use of the modified set of causal variables in the equation set requires an estimation of the reliability of each casual variable as an indicator of its dependent variable.

Table 2. factor Semantics

Event Correlate Group	Correlated Variables (Model Attributes)	Potential Factors (Evaluative Dimension)	Measurement Methodology
1	CPU$_A$ D-Cache miss	CPU$_A$ D-Memory Read/ Write Utilization	Count per stimulus test or time interval
	CPU$_A$ Pipeline Stall	CPU$_A$ D-Memory Read Latency	Duration of D-Mem Read
	CPU$_A$ D-Bus Read		
	CPU$_A$ D-Memory Read		
2	DEV$_{Int}$ Interrupt Request	Dev$_A$ Interrupt Service Latency	Duration of Dev$_A$ request

A Concept is an abstract entity that can be described as an effect produced, functionally and temporally, by a set of causes (variables) and other concepts. Concepts are neither directly nor perfectly measured but are approximately measured by their causal variables. Some

Concepts used to describe complex electronic platforms are given below in terms of partial causal Factors. Concepts need to be uncorrelated. However, it is clear that concepts may be correlated with factors that are variables in another concept. An example is given below:

i. **Concept: Performance**
   - Constituent Factors: Device Interrupt Service Latency, Memory Access Latency, Non OS instructions executed, messages/packets processed from external networks, multi-media operations performed

ii. **Concept: Power consumption - Average**
   - Constituent Factors: technology, clock speeds, Memory Utilization, bus transactions, memory accesses, pipeline stalls, external platform port activity

iii. **Concept: Mobility Fit**
   - Constituent Concepts and Factors: Performance, Power, Cost, TTM

Now having a set of equations that describe, as a best ratiocination from theory and observation, the structure of the originally observed data, the model can be tested for its ability to produce unique estimates, goodness of fit to the real data, and reliability of the model. Statistically, reliability of a variable, for instance, means the proportion of the Event source (variable) that is free from random error. Reliability of observed variables affects the reliability of factors for which they are indicators which in turn affects the reliability of concepts and from there the whole model. If the model fails in any of these attributes, the model will need to be respecified and tested again. On acceptance of the model, each of its constructs can be tested to strengthen or weaken the overall confidence in the fit of the model. A simple reliability indicator is, if loadings of individual constructs are found to be statistically insignificant, the model needs respecification.

Summarizing:

To continue the Mobility Fit example, the SEM is in Equation 1.

5. Experimenting with System Architectures

The empirical analysis so far has focused on explaining the operation of

$$C_{Performance} = d_{pf_1} \bullet F_{DevIntSerLatency} + d_{pf_2} \bullet F_{MemAccessLatency} + d_{pf_3} \bullet F_{NonOSInstCnt} + d_{pf_4} \bullet \ldots$$

$$C_{Power} = d_{pw_1} \bullet F_{MemUtilization} + d_{pw_1} \bullet F_{ClkCnt} + d_{pw_1} \bullet F_{BusXactCnt} + d_{pw_1} \bullet F_{PipeStallCnt} + d_{pw_1} \bullet \ldots$$

$$C_{MobilityFit} = d_{mf_1} \bullet F_{TM} + d_{mf_1} \bullet C_{Performance} + d_{mf_1} \bullet C_{Power}$$

Equation 1: Structural Equation Model for Mobility Fit

complex system in terms of increasingly abstract factors and concepts. The characterization power of the analysis relies on its ability to uncover unobservable (latent) variables which raises the focus of understanding to a more abstract conceptual level and enables the system to be considered as a whole (the SEM characterization), without regard to hardware, software, mechanical or RF influences. However, the primary event sources are just the data gathered from hardware, software and the physical elements of the system.

The focus now changes to consider the use factors and concepts to drive a systematic approach to experimentation in order to optimize some objective function, presumable with parameters that are the concepts and factors being varied to achieve that optimum. This is the domain of Design and Analysis of Experiment methodology which is described in this Section together with its relationship to factor analysis and SEM technology.

5.1 Design and Analysis of Experiments (DAE) Methodology

This methodology provides a systematic way of designing, performing and analyzing experiments (viz. test or series of tests) *in which intentional changes are made to the input variables (factors and concepts) of a process or system so that the reasons for changes observed in the output (response) of the system may be observed and identified*[13].

The following is an outline of the experimental design process.

i. Formulate the goal (same as for factor analysis). An example of a narrow goal: Minimize power consumption of a software algorithm running on platform A.
ii. Specify the objectives that will satisfy the goal. The goal is effectively an SEM set with outcomes affected by variables, factors and concepts that have casual relationships to the outcomes.
    - Examples:
        + Identify the measurable, continuous factors, including interactions between factors, that affect mean power and variability in power,
        + Determine the optimal setting of factors and the interactions between factors that give minimum power with minimum variability.
iii. Choose an appropriate experimental protocol (essentially confirming the SEM model), essentially: Sample size and Protocol: Iterative experimentation, randomizing the experimental trial order, Replication of experiment to reduce the effect of noise (variability).
iv. Analyze the outcomes. There are three: determine the variables that affect the mean performance, variables that affect performance variability, and variable levels that achieve optimum performance. Then determine whether further improvements are possible.

The system under test is a VSP running various software programs – these together with *external stimuli* form the operating conditions driving the system to produce measurable responses. Programmed systems are peculiar in that the software is an intrinsic part of the VSP (by definition) and it also is part of the data set over which experiments are measured. System software - including operating systems, device drivers, communications stacks, and applications - becomes so complex that, in a given system (including a VSP) subject to unpredictable external events (such as the arrival of a video packet during a DMA event that is holding the data-bus of the DSP processor in a triple processor system), the whole system has unpredictable but bounded responses. Hence the requirement for measurement and analysis to determine an SEM explanation of the observed data, and the use of the SEM model, inter alia, in determining better bounds for system behaviours.

In the DAE methodology, the variables that can be used to produce system responses are many and varied. For example: cache organization of disk block accesses in the disk controller, bus bandwidths, database schema for storing information in cell phone middleware, I&D-cache size of each processor, algorithm for audio echo cancellation, memory latencies, etc. The ability to measure responses in a physical system is somewhat limited by the technology used to build them, viz. if in silicon then observability and controllability are poor, if in FPGA there is more but still limited access. In contrast a VSP has intrinsic observability and controllability and since it is a faithful facsimile of the physical system (at an agreed level of timing granularity) it is ideal for experimentation purposes. The issue now becomes what to measure from the thousands of measuring (event) sources available and what to ignore. The factor analysis and SEM have already addressed part of this problem, but it still requires many experiments and iterations through the flow to determine for the concepts deemed important, what other concepts and factors causally relate to these, and then what are the loading of the variables on the factors used in the SEM set.

The DAE methodology relies on the ability to vary variables (called manifest variables if directly manipulatable) in the system and observe the results. It then prioritizes the results in terms of the variables (called latent variable if they are factors and concepts) that have the most to least importance in producing the observed effects. The effect of many variables changing simultaneously (as in SEM) is a feature of DAE, since it is the interactions between variables that often have the significant effects on the observations. The statistical machinery to do this requires the calculation of the main and interaction effects of variables/factors, then determine which effects are significant. An assumption here is that the underlying data comes from a normal distribution and there are appropriate tests for determining whether this hypothesis holds. The response of various concepts (and an overall objective function) can be plotted versus the significant variables. If there are no interaction effects between variables, the response surface will be linear wrt the variables. Interaction effects produce higher-order polynomial response surfaces. Lines of the same response value are known as contours and traversing a contour will enable a determination of the most efficient set of variables to produce that response (say video throughput dependent on variables cache sizes, memory hierarchy latencies, clock speed, MPEG2 algorithm and frame rate). Maximizing or minimizing a response looks for stationary points (maximum, minimum and saddle-points) on the response surface.

Regression models for predicting response, over the variations of the significant variables, can be built. The predicted response from these models can be compared with the actual responses to directly measure from the system, by setting the variables appropriately. In this way a confidence interval for the mean response can be calculated and the regression model assessed as competent of incompetent. Incompetent models require reformulation the response predictors and may fall back through SEM and factor analysis to remedy the problems.

The final issues to discuss is what experiments are needed to efficiently gather sufficient data to construct a competent response surface, given the SEM analysis and the response function formulations. For an experiment involving *v* variables that will determine a response function, where each variable can be set at a number of levels (values) (say l) and that number is common across all variables, the number of unique experiments required to cover the entire variable space is l^v. This number can be reduced dramatically using factional factorial designs determined by the important main effects and their interactions. Having determined the set of experiments to be performed, the order of the experiments will need to be randomly determined if the simulation model has inbuilt randomized effects, such as noise on interconnects and small changes in latencies.

Two forms of platform architecture experimentation are briefly discussed. The first takes a structure already characterized and changes attributes of modules constituting the platform, such as cache size, memory latency, and bus width. This is called mesomorphic (middle structure) experimentation and it essentially is for optimizing objective functions of architectures that can be tuned parametrically. The second, called epimorphic architecture experimentation, takes the radical, conceptual view enabled by SEM analysis, and asks how can the overall structure of the system (for instance, reducing the number of processors from four to two) be modified to meet some objective.

5.2 Mesomorphic Architecture Experimentation

This is essentially architecture optimization by module tuning within a well characterized platform. Since part of the SEM methodology is to maintain the independence of (that is, to avoid correlation amongst) concepts, they become good candidates to drive objective functions and they should always be significant factors. In practice, using concepts such as performance and power in an objective function for optimizing processor based, electronic systems does confound the objective function since event source data, such as $\text{cache}_{\text{miss}}$, is theoretically (and practically) an indicator for both concepts.

The objective is to run a set of experiments that will produce response surfaces for response concepts expressed in the SEM equation set. For example, an objective function for mobile devices might find the response surface that maximizes performance and minimizes power consumption. Each response concept will have factors and concepts loading on it and each factor will have manifest variables loading on it. We already know that each variable, factor and concept is significant for some concept and/or the objective function.

Assume that the following manifest variables are the significant inputs to our experiments. To simplify the example, two levels of value have been chosen for each variable. The Table below summarizes this information.

There are 7 variables with 2 levels each requiring $2^{7} = 128$ experiments to fully characterize the VSP by computing, for each variable, its main and interaction effects. If main effects and 2nd order interactions are deemed only to be significant, the experiments can be run on a fraction of the full factorial design – in 16 experiments. If we consider aliasing some main order and 2nd order effects, then 8 experiments may be adequate. Randomization of experiment order and replication have no effect on increasing statistical significance of results from simulation runs, unless random variability has been injected into the model to increase its realistic behaviour.

Once the experiments have been performed, the expected outcome is that optimal settings of the variables, to produce the desired concept and objective function outcomes, will have been determined. This work is mechanical. Demonstrating that the SEM model and the DAE model correspond with an acceptable level of significance is regarded as confirmation of the models.

Manifest Variable	**2-Level Values**	**Coded As**
L1 Cache Sizes (I&D)	Small, Large	-1, +1
MPEG 2 Algorithm	Low Performance, High Performance	-1, +1
Buses (I&D)	Non-multiplexed, Multiplexed	-1, +1
Clock Frequency	Low, High	-1, +1
FPU	Not-exists, Exists	-1, +1

5.3 Epimorphic Architecture Experimentation

Quite radical changes in VSP architecture may be envisaged, such as: reducing the number of processors, reconfiguration the memory hierarchy, changing operating systems, etc. Even so, the overall system must be fit for its designated task of control in a product; this means that the response concepts will be maintained. If we reflect back to mapping abstract architectures to VSPs, the pure (SEM) concepts should be developed for the highest level of abstraction and then ideally be invariant across the set of mappings of the abstract system to various VSPs.

The objective in epimorphic architecture experimentation is to determine whether there are alternate VSP structures that will have more desirable responses when characterized using SEM models that maintain the upper-level concepts but, through factor re-analysis and SEM model rebuilding, may have quite different loadings of variables on factors and factors on concepts. The comparison will be over response surfaces produced by each SEM model when subjected to the experimental regimes outlined in Section 5.1.

The DAE part of epimorphic architecture experimentation is somewhat different. The levels in some manifest variables will select various VSPs, along with other variables selecting parametric settings within the chosen VSP, and the response will be across all VSPs and all parameters selectable with the variables and factors that load on the invariant concepts in the set of SEM equation sets characterizing the VSPs.

This is an exciting outcome, The mapping of an abstract system to a small set of VSP targets, that are selected systematically by the statistics underlying DAE, enables the DAE equations to be used predictively rather than just confirmatively (as with mesomorphic architecture). The individual SEM equations describe how well a set of abstract concepts can be realized using a parameterized VSP. The DAE driven process produces a set of regression equations across and within target VSPs that can be used to select suitable candidates from a broad range of VSPs to be targets for the mappers of abstract systems such as those described in Figures 3.

6. Evidence for the Empirical Framework

Since the multivariate approach to experimentation is relatively new to engineering, the available data concerning SECS experimentation involves optimizing objective functions using intuitive experimentation, driven by hypothesis formation and refutation [14]. The objective functions are formulated as described in Section 4 – that is, event driven responses without formal factor or concept explanations of the observed behaviour. However, even this level of experimentation has yielded impressive results.

The case studies below are both composites constructed from similar projects across 2 companies, in each case.

6.1 New Architecture – Empirical Intervention

A typically (intuitively) designed commercial controller incorporated 6 signal processors,1 general purpose processor, 2 matrix switches, arbitrated interconnection buses and bus bridges, complex memory hierarchies with multiple DMA controllers, and many peripheral devices. Several of the central interconnects were multiplexed. It was unknown whether the controller could meet throughput, response time or processing headroom specifications.

In the normal engineering process, an iteratively analyzed and refined version of the intuitive design would be reduced to an implementation (probably silicon) and the architecture subjected to trial post realization. Historically, complex, intuitively designed controllers are respun 2-5+ times in order to *iron*-out the architectural, as well as, the detailed hardware design and hardware-software interaction decisions. A pretty inefficient and very costly process. The estimated cost per architectural respin is a couple of million dollars, split between the cost of a new mask set and the engineering needed to rectify problems discovered so late in the design process.

The new architectural design was a central plank in each company's strategic new generation offering to its customers and prospects in a highly competitive, global market. A competitive differentiator would be the early availability of a system to customers and prospects so that software, to be sourced from the company, customers and 3rd party suppliers could be developed early. Accelerating the development of software is a major competitive factor, since it dominates the engineering effort and the critical development path for both a company and its customers. It is a universal hope that if a prospect begins serious software development using your VSP, they would be likely to commit to buying your silicon for the platform.

The original design was modeled as a VSP in mid 2004 with high performance, timing accurate models which were heavily instrumented. The experimentation began with representative, trial software loads running on the processors. Several major architectural problems were found rapidly: inadequate response times, insufficient traffic throughput and insufficient processing headroom, amongst others. The VSP architecture was extensively and iteratively reworked with the each iteration candidate being subjected to heavy experimentation and measurement using the required software and input stimuli. The rearchitecting took place within a six month time frame. The final architecture was radically departure from that intuitively specified – 3 DSPs, 1 general purpose processor (GPP), unmultiplexed buses, simpler memory hierarchies and interconnect fabric – and met the throughput, response time, and processing headroom specifications. This was all done prior to commitment to silicon. The VSP is currently being offered to customers and prospects to enable them to start on software porting and development, 9-12 months before availability of silicon – representing a TTM reduction near to 50%.

Even though the result is very good, there is no way of determining, by using informal techniques, whether the platform is optimum for its intended purpose. The next step is to engage the more formal techniques outlined in this paper to try and achieve an optimal architecture.

6.2 Iterating Architectures Quantitatively using VSPs

In 2003/2004, a VSP of a closely-coupled, triple core architecture having 1x DSP, 2x GPP, a multilayered interconnect fabric, many bus bridges and peripheral devices, was built and delivered to a customer who then completed the VSP by adding all required peripheral devices and tuning the architecture. The VSP paralleled an existing design for which silicon was already being produced, and was used for developing tests suites, porting software, and to check the adequacy of the architecture for its intended purpose. The VSP was delivered to VaST's customer's customer 9 months in advance of the silicon.

Twelve months later the next iterative architecture development was due. The functional specification required additional computing cycles for executing additional software, and several modified and new peripheral devices. Two lines of investigation were initiated, one dealing with processor microarchitecture modification to increase the CPU clock frequency and the other to use the VSP to experiment with the overall platform architecture and determine what improvements were available from that source.

The microarchitecture investigation yielded the requirement for microarchitecture modifications to accommodate increased clock frequencies – a task that was engineering intensive and would result in an overall 10% increase in platform performance. The platform architecture experimentation yielded data indicating a number of straightforward changes, dealing, amongst other things, with memory hierarchy and latency, that resulted in a more than 50% improvement in performance.

The results were somewhat surprising to the investigators, since a triple processor platform is regarded as processor centric. However, what the data is saying is that microarchitecture improvements produce 2nd order effects when placed in the context of systems platforms. This finding is consistent with the industry data that the results of heroic engineering efforts at the microarchitecture level are completely swamped by the twin effects of overall increased performance due to the underlying semiconductor technology (for example, moving from 130nm to 90nm technology) and the optimization of the platform architecture itself determined by quantitative experimentation with the mundane attributes of architectures, such as memory hierarchies and latencies (including cache), bus – bus bridge hierarchies and aggregate bandwidth and latencies; etc.

Once again a surprising amount of optimization was yielded from basic quantitative investigation. Once again, the ad hoc experimentation did not look at the multi-factorial overall platform optimization issue.

7. Summary, Ongoing and Future Work

It is clear that such experimentation techniques should always be required part of any architect's tool kit and job responsibility, ideally prior to the commitment to an architecture, rather than as a post facto clean-up endeavour.

Another aspect of architecture that will yield 1st order effects on the overall system is the formal experimentation with architecture, structure and algorithmic efficiency of the most complex component of the system – software. It will be very surprising if the systematic experimentation with software, and the improved fit between the hardware architecture and the software architecture, does not yield optimizations that will eclipse the effect of the platform and microarchitectures determined separately.

A framework has been presented for ad hoc (event data driven) and systematic (concept driven) experimentation. Most engineering experimentation, when it is done at all, is of the former type where a progressive set of intuitive experiments is performed, measures taken and further experiments formulated based on evidence from the prior experiments. As can be seen from the experiences related in Section 6, when performed early in the engineering cycle this unstructured approach can have a profound effect – even structural, but it relies on expert intuition and that, however profound, is subjective. When these many hundreds of experiments are each structured to refute, with some level of significance, well thought through hypotheses, a degree of structure enters and the process accretes good attributes of the scientific method.

The more structured approach, based on statistics that elevates the focus of system level considerations to more conceptual than operational concerns, followed by the systematic design of experiments offers a new way to efficiently approach the task of reducing the experimental effort when conceptualizing new architectures optimized for some purpose. The alternative is to perform thousands of ad hoc experiments driven by intuition and success and failure – the school of architecture of hard knocks. As yet there is little hard engineering result from the use of the SEM based approach. However, we are working with customers, especially in the early architectural phases, to see how the more structured approach affects their engineering product, as well as the efficacy of their engineering process. With several global companies, across several market sectors, involved in the effort, it is exciting times for empirical architecture.

Future work involves a pragmatic assessment of the effectiveness of the empirical approach to the more conceptual areas of architecture development. This will require several projects (typical duration 12-18 months) to go through the process to gather sufficient data to be able to report failure, success or progress. The second area of future work is to apply the same statistically-driven quantitative processes to the engineering process itself. The recognition that the DFD process of Figure 2, which describes the early stages of the engineering process, is structurally the same as the DFD of Figure 3A, which describes part of an abstract system architecture, makes this an exciting and promising field of investigation.

8. ACKNOWLEDGEMENTS

I would like to thank Mahdi Seddighnezhad for his thoughtful comments and careful proof reading of this paper.

9. REFERENCES

[1] Encyclopaedia Britannica (2005). Democritus, Plato, atomism. Ultimate Reference Suite DVD. www.britannica.com

[2] Encyclopaedia Britannica (2005). Stoicism. Ultimate Reference Suite DVD. www.britannica.com

[3] Hair, J.F., Anderson, R.E, Tatham, R.L and Black, W.C. Multivariate Date Analysis with Readings. 4th Ed. Prentice-Hall International, Inc., NJ (1995).

[4] Hughes, M.A., Price, R.L and Marrs, D.W. Linking Theory Construction and Theory Testing: Models with Multiple Indicators of Latent Variables. Academy of Management Review, 11, 1 (1986), 128-144.

[5] Hellestrand, G.R. Rapid Design of Software-Rich Chips: Executable Specification → Realization. White Paper. VaST Systems Technology Corp., Oct 2002.

[6] Hellestrand, G.R. The Revolution in Systems Engineering. IEEE Spectrum, 36, 9 (1999), 43-51.

[7] Backus, J. Can Programming be Liberated from the Von Neumann Style? A Functional Style and Its Algebra of Programs. Comms. of the ACM, 21, 8 (Aug 1978), 613-641.

[8] Hellestrand, G.R. The Unified Specification of Mixed Technology Systems, Keynote Paper, Proc. Conf. Synthesis and Simulation of Mixed Technology Systems (SASIMI'95), Nara, Japan, August 1995.

[9] Hellestrand, G.R. Events, Causality, Uncertainty and Control. Proc. 2nd IEEE Asia Pacific Conference on Hardware Description Languages, 221-227, Toyohashi, Japan, October 1994.

[10] Cheung T.K.-Y., Hellestrand G.R. Form: A Functional System Specification Notation, The Fourth Asia-Pacific Conference on Hardware Description Languages, APCHDL'97, HsinChu, Taiwan, 18-20, August 1997, 10-15.

[11] Hellestrand, G.R. The Engineering of Supersystems. IEEE Computer, 38, 1(Jan 2005), 103-105.

[12] EEMBC: Embedded Microprocessor Benchmark Consortium. www.eembc.org

[13] Montgomery, D.C. Design and Analysis of Experiments. 5th Ed. John Wiley & Sons, NY, 2001.

[14] Winters, F.J., Mielenze, C. and Hellestrand, G.R. Design Process Changes Enabling Rapid Development. Proc. Convergence 2004 P-387, Oct 2004, 613-624, Society of Automotive Engineers, Warrendale, PA.