The Design Optimisation of Complex Systems - A Systems-theoretic Approach

Tony Shell, DarkLake Synectics, info@darklake-synectics.co.uk

Abstract. By building on earlier established work, a non-recursive approach to design optimisation is proposed. It provides for the direct synthesis of optimal system design solutions of complex system implementations with multiple components, embodying different technologies, and with multiple levels of design hierarchy.

The derivation and decomposition of comparative evaluation criteria and trade-off functions, using 'mathematical orders' over the space of candidate systems, is explored in detail. An extended form of the Subsystem Trade-off Functional Equation (i.e. nk-STFE for n components and k evaluation criteria) is developed, and its application to optimal design of complex systems architectures is presented. In particular, the effect of overall constraints on system implementation - in terms of *combinatorial constraints* on component choice, and the manner in which this is effectively addressed by application of the nk-STFE formulation - is discussed. The representation of this process of constrained, complex system design as an NP-complete class of problem is reviewed, and some of the general techniques for making system design a realistically practical endeavour is presented in formal, systems theoretic terms.

The development of formal (systems-theoretic) constructs, theorems, and theorem proofs, are provided where necessary.

INTRODUCTION

It is proposed that the fundamental goal of 'good' systems engineering is to specify and develop a 'best' (or, at least, optimal) system solution that will demonstrably satisfy the real needs of all project stakeholders - and to ensure that the identification, design and development of this best solution is undertaken in a logical, quantifiable, documentable and reliable manner.

The main problems faced by the system designer in developing optimal solutions to a complex systems problem can be summarised as follows: (a) establishing the scope, the identity, and the source/ownership of the optimisation criteria; (b) formulating an agreed trade-off requirement; (c) dealing with complex interconnected systems architectures; and (d) selecting an appropriate optimisation strategy with respect to the particular design problem.

An established theory of systems design, using formal constructs and set-theory notation, is used as the basis for the development, and presentation of ideas throughout this paper. In particular, an approach to design optimisation for complex systems implementations is investigated. This work is seen as being complementary to other fields of research into methods of optimisation for complex systems problems (such as the 'directed search' algorithms and the design of experiment (DoE) techniques). It is part of a more general effort into developing the practical aspects of mathematical systems theory.

Current systems engineering standards (process models) and 'best practice instructions' have, in general, little (and sometimes nothing) to say with respect to optimal system design. However, it is argued that with intensifying market competition, increasingly scarce natural resources, and growing levels of pollution, system optimisation has to be a primary objective.

The term *non-recursive* is used specifically to describe a systems-theoretic approach to design optimisation. The term is chosen with some care - it is used in the literal context of not having to *re-do* engineering design or implementation tasks as a consequence of recognising (and accepting) *post priori* that a solution is deficient (sub-optimal). Many of the SE and S/W process models (e.g. the 'spiral' approach) are focused on *iterative* or *incremental* design and are concerned with the design and implementation of a particular system (architectural) solution - they have nothing to do with systems optimisation, which necessarily has to deal with multiple candidate system solutions. It is important to distinguish between these separate (and very different) design issues.

For the 'directed search' optimisation techniques (e.g. evolutionary/adaptive computing) it is very important that the 'rules' for assessing design preferences between candidate system solutions are established and rationalised. Sadly it is often the case that systems optimisation is only attempted as a highly recursive and experiential post-deployment activity (driven by customer and/or competitive market responses, or by system 'failures'). The purpose of this paper is therefore to examine the *practicalities* of an effective a-priori application of system-theoretic method to the non-recursive optimisation of 'real world' complex systems problems.

This approach to system optimisation requires that all candidate complex system designs can be mathematically modelled with regard to all of the various evaluation criteria. By this means, we are able to have the capability of directly synthesising optimal system solutions - even for highly complex systems implementations. In particular, we are interested in employing the basic tenets of General Systems Theory (GST) such that a more formal approach can be taken with regard to system modelling, and hence to theory-based predictions of complex system properties.

An established theory of systems design, using formal constructs and set-theory notation, is used as the basis for the development and presentation of ideas throughout this paper (Wymore 1993).

THE SYSTEMS-THEORETIC SYSTEM DESIGN METHOD

Systems engineering is a scientific endeavour in its own right, with an established theoretical basis, and with clear principles and objectives. It is fundamentally concerned with providing practical (engineered) solutions to 'real world' problems. It also requires of the practitioner the effective application of imagination and creativity within the broad context of general system design.

It is therefore felt to be well worth the effort to apply a proper 'scientific' systems methodology to this endeavour, such that a search for a 'best' (or, at least, optimal) system solution can be conducted according to a precise and rigorously defined rationale.

This paper employs an established theoretic approach for systems design so as to bring rigor, consistency, and clarity to the arguments. Mathematically based systems theory is a well established and proven approach to systems theoretic design - the viability of which has already been extensively reviewed (for example see: Klir 1996, J. Fertig and R. Zapata 1977).

The process of identifying the space of implementable candidate system design solutions is traditionally defined in terms of the identification of those systems that satisfy the technology requirement (that are within the 'buildability space') that can also exhibit a mode of behaviour that satisfies the input/output requirement (i.e. system design solutions that are within the 'functionality space').

For engineering design and development work to proceed effectively it is vitally important that an early consensus is agreed amongst the various system stakeholders as to the precise meaning of a 'best' (or optimal) system solution. This consensus needs to be formulated, and recorded, within the specification of system requirements.

The system requirement may (in part) be specified in terms of an input/output requirement (IOR), an 'embodiment' requirement (i.e. the technology requirement, TYR), and a trade-off requirement (TR). An established systems theory (Wymore 1993) provides for a mathematically rigorous formulation of this particular approach.

The input/output requirement (IOR) involves specifying for every identifiable system input trajectory a set of eligible output trajectories - such that a system solution can be sought which, when presented with an input trajectory will produce an acceptable output trajectory. This effectively describes the system's required function, or 'purpose'.

The technology requirement (TYR) is used to specify the 'closure' of the buildability space of design solutions - in terms of the acceptable physical embodiments of component choices, of design architectures, and of overall system implementation constraints.

The process of identifying the implementability space of candidate system design solutions (i.e. the set S of implementable systems) is defined in terms of the identification of those systems that satisfy the technology requirement (that are within the buildability space) and which can also exhibit a mode of behaviour that satisfies the input/output requirement (i.e. system design solutions that are within the functionality space).

The trade-off requirement (TR) is defined as an order over the set of candidate system solutions (S) such that a means of expressing a preference between any two candidate systems can be specified. They are frequently derived using real-valued functions of the performance and cost figures of merit for the system solutions. The formulation, and use, of these particular specifications of requirement is discussed, in detail, in later sections of this paper.

It is unfortunate that many of the current 'system methods' have a somewhat limited view of systems engineering. There is often a parochial tendency to favour a particular 'technology-centric' viewpoint (for example, to adopt the modelling paradigms of ECBS or IT technologies) and to focus on particular system implementations. There is little concern with regard to the fundamental issues of system implementation, complexity, design decomposition, emergent system behaviours, and optimal systems design.

Descriptions of the systems-theoretic mathematical constructs, their derivations, and the relevant theorems and proofs, that form the fundamental basis of this approach to system design, can be found within the referenced systems engineering text (i.e. Wymore 1993).

APPLICATION TO COMPLEX SYSTEMS PROBLEMS

Systems engineering is principally concerned with the production of *complex* implementable design solutions. The complexity of a system design can be formally captured through the use of a system coupling recipe - where complex systems architecture is described in terms of a list of named system components together with a set of all the output-to-input component interconnections 'pairs' within the system. The 'components' are required to be formally identifiable as individual systems (non-trivial, open), with their own behavioural characteristics. It is in this context that the concept of complexity is introduced - in terms of the interactions of the behavioural modes of the components, such that an overall system mode(s) of behaviour is exhibited (system mode hologenicity). In general, we are dealing with complex (multiple components) systems with (usually) complicated (multiple modes) behaviours (Shell 2001)

There are many different measures of 'complexity' - for example Professor Seth Lloyd [LLoyd 1996] identifies over forty! The choice of definition depends on the context. In this case the context is the systems-theoretic description of system components, system behaviour, and functional implementation. A 'measure' of system complexity can therefore be formally expressed in 'organizational' terms as the size of the ordered list (vector) of system components - a component count (Shell 1999).

This concept of system 'complexity' is important in terms of the subsequent decomposition of optimisation trade-off requirements, and the identification of optimal component designs - as addressed in detail in later sections of this paper.

COMPARING CANDIDATE DESIGN SOLUTIONS

For some specified problem, we describe the totality of the candidate 'implementable' system solutions as a set of system designs, S - where S is the set of system designs that provide the required functionality and satisfy the technology requirements. Assuming that S is not empty, and that there is more than one candidate solution, then the task of the system designer is to be able to express a comparative order over S - and hence determine the optimum design solution on the basis of some quantifiable assessment of systems 'preference'.

System Order. As elements of the overall system requirement, the functions jR, $j \in$ IJS[1,k], $k \in$ IJS^{+} , are all mathematical 'order' functions defined as particular members of the set of functions: FNS(S^2,{0,1}). These orders therefore map from pairs of candidate system designs (i.e. $(a,b) \in$ S^2), onto {0,1} so as to indicate the relationships between systems - that is to say, to assign a value of 1 where system $a \leq$ system b; or otherwise assign a value of 0. System order is, by defnition, and as a minimum to satisfy the purpose of system design, both *transitive* (i.e. where $a \leq b$ and $b \leq c$, then $a \leq c$) and *reflexive* - in formal mathematical terms a 'partial order' (for a more detailed explanation of mathematical order, see Wymore 1993, Ganter & Wille 1999, or Johnson 1998).

System Comparison Categories. A (mathematical) system order function can be used to describe different forms of relationship between the various candidate systems. There are three fundamental relationships: 'equivalence', 'less than', and 'not comparable'. These relationships are defined as follows:

Given $a \in S$ and $b \in S$, if a and b are 'equivalent' then this is denoted and defined as follows:

$$a \equiv b \Leftrightarrow a \leq b \wedge b \leq a$$

If a is 'less than' b, this is denoted by:

$$a < b \Leftrightarrow a \leq b \wedge \sim(b \leq a)$$

and if a and b are 'not comparable', then this is denoted by:

$$a \textit{ ntc } b \Leftrightarrow \sim(a \leq b) \wedge \sim(b \leq a)$$

An example of these system order relationships is shown in figure 1 (for, in this case, five systems) in the form of a *line diagram*.

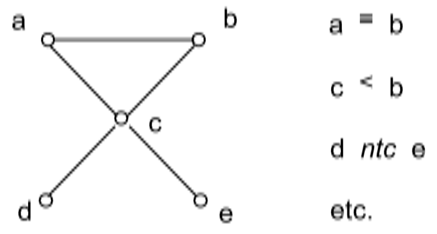

Figure 1 System Order Relations Example

In this type of diagram connecting lines are used to show the relationship between systems, with an order preference being shown as increasing in the vertical direction (and is similar to a Hasse diagram). A full description of the method and notation behind this graphical representation is given in the reference sources (Ganter & Wille 1999: chpt. 0, Wymore 1993: A1.310).

Determination Of System Order. However we need to ask what is meant by a system being (for example) 'less than' another system? How can the concept of a comparison between two systems be defined such that 'system order' of the form $a \leq b$ can have real meaning?

The mathematical assignment of system relative order needs to be based on some quantifiable criteria in terms of attributes of the candidate system solutions. The use of various 'figures of merit' functions (jFoM) can be employed for this purpose and will generically be of the form:

$$jFoM \in FNS(S,RLS),\ j \in IJS[1,k],\ k \in IJS^{+}$$

where S is the set of all candidate system solutions, and RLS is the set of all real numbers.

Each system figure of merit value (i.e. jFoM(x), where $x \in S$) will therefore be enumerated in terms of the specified system criteria - for example: accuracy, capacity, development cost, operating cost, development risk, etc. These are usually expressed over the intervals of real numbers, and normalised to give a 'utility' value in the range RLS[0,+1.0].

These figures of merit can therefore be used to provide a measure of the merit of each candidate system, for each particular evaluation criteria.

A system order can now be defined in terms of real-valued functions representing comparisons that are measures of system 'merit'. This real-valued function order is therefore required to be defined such that it is an 'order' in the true sense - such that $RFO(S, FoM) \in FNS(S2, \{0,1\})$, and can be expressed in terms of the set of candidate system designs (S) and each figure of merit function (jFoM) as follows:

$$RFO(S, FoM) = \{ ((a, b), y) : (a, b) \in S{\cdot}2;\ y \in \{0, 1\};\ y = 1 \Leftrightarrow FoM(a) \leq FoM(b) \}$$

- and hence, for the j^th order, we have an order jR over S:

$$jR = RFO(S, jFoM)$$

The ideas developed in this paper shall, throughout, assume a mathematical 'order' in terms of the RFO form, as given above.

DEFINITION OF 'BEST' AND 'OPTIMAL'

The trade-off requirement TR is defined so as to aid in the identification of the optimal system design solutions when identification on the basis of the combined jR orders alone is not possible. For example, the 'highest performing' system will (at any particular epoch of technological development) usually be more 'costly' than rival 'low performance' design alternatives - there needs to be a 'trade-off'.

System Trade-off. Firstly we need to consider the concept of design 'trade-off'. If a set of candidate system designs S is not empty and jR, $j \in IJS[1,k]$, $k \in IJS^{+}$ are orders over S, then TR is a trade-off between the orders jR if and only if, for x in S and y in S, any clearly established product-order preference is maintained (Wymore 93: sect A1.406 with respect to 'product order'). A system trade-off order is therefore required to satisfy the essential trade-off criteria:

$$x \in S \textit{ and } y \in S,\ x \neq y,\ \textit{if}$$
$$x < y \textit{ modulo } \prod_{j=1}^{j=k} jR,\ \textit{then also}$$
$$x < y \textit{ modulo } TR$$

- or, alternatively expressed as: TR(x,y) = 1 and TR(y,x) = 0. Note that the product-order is defined as follows:

$$(\prod_{j=1}^{j=k} jR)(x,y) = \prod_{j=1}^{j=k} jR\ (x,y).$$

The trade-off order TR is thus required to preserve clear system preferences where it exists in the evaluation product-order, but can be used to 'adjudicate' between candidate systems when either there is an 'equivalence' such that:

$$(\prod_{j=1}^{j=k} jR)(x,y) = (\prod_{j=1}^{j=k} jR)(y,x) = 1$$

- or where systems are determined to be 'not comparable' such that:

$$(\prod_{j=1}^{j=k} jR)(x,y) = (\prod_{j=1}^{j=k} jR)(y,x) = 0$$

'Optimal' System Design. How is an *optimum* system design solution to be recognised?

An 'optimal' system solution y* in S is a system optimum in the sense that, for *any* x in S different from y*, then:

$$(x < y^*) \vee (x \equiv y^*) \textit{ modulo } TR$$

- or alternatively expressed as: TR(x,y*) = 1 and TR(y*,x) = 0, or that TR(x,y*) = 1 and TR(y*,x) = 1. The TR order is required to satisfy the essential trade-off criteria (as above).

Any candidate system that is sub-optimal with respect to *all* the evaluation orders (the product-order) must also be sub-optimal with respect to the trade-off requirement (TR) - see *Theorem 1* in the 'Theorems and Proofs' section. Note also that, in terms of the evaluation product-order, a system that is *not* sub-optimal is *not* necessarily optimal.

There is a corollary to the above theorem in that a system that is a 'best' design solution with respect to all the orders (the product-order) must also be a 'best' solution with respect to the trade-off order - see *Theorem 2* in the 'Theorem and Proofs' section.

It is also the case that, in terms of the trade-off requirement TR, that a system that is not sub-optimal must be optimal - see *Theorem 3* in the 'Theorems and Proofs' section.

'Best' System Design. How then is a *best* system design solution to be recognised? The 'best' system solution y* in S is a system optimum in the sense that, for *any* x in S different from y*, then:

$$x < y^* \textit{ modulo } TR$$

- or alternatively expressed as: TR(x,y*) = 1 and TR(y*,x) = 0. The TR order is required to satisfy the essential trade-off criteria.

Trade-off Example. An example of trade-off optimisation for three candidate system solutions $(S = \{a, b, c\})$ is caricatured in figure 2.

In this line diagram the notation:

$$\prod_{j=1}^{j=k} jR$$

is used to indicate that we are evaluating a product-order over a set of candidate systems S, with respect to a *fixed* number of evaluation criteria (i.e. the k evaluation orders).

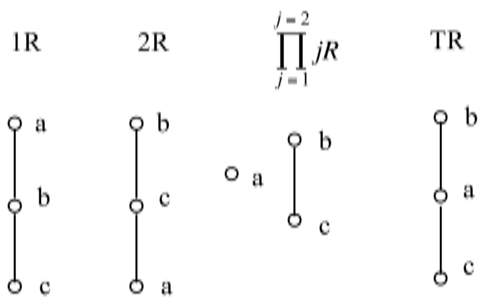

Figure 2 Product-Order, Trade-off Order

Note that, in this example, systems a and b, and a and c are 'not comparable'. The trade-off requirement (TR) is therefore used to adjudicate between these relationships. In this example the trade-off requirement therefore establishes that system b is preferred to system a, and a is preferred to system c. Note also that the trade-off requirement (TR) properly maintains the existing system order preferences such that: c < b (the 'order' is transitive).

DETERMINING THE TRADE-OFF FUNCTION

The trade-off requirement (TR) therefore has an especially important role to play in the comparative assessment of candidate system solutions in terms of providing an 'adjudicating' function to enable a choice to be made between systems, where there is no clear-cut preference in terms of the evaluation product orders.

How, then, are we to quantify this trade-off order TR - by what criteria should we adjudicate between equivalent or non-comparable systems?

The Discriminatory Trade-off. There are many different ways in which a trade-off requirement may be developed - however, in this paper, a relatively simple approach to this problem is described. This approach uses a function of the listed figures of merit so as to provide a *discriminatory* trade-off function (Wymore 1998, Shell 2000). It effectively allows the trade-off requirement to be specified in terms of some function of an 'ensemble' of the figures of merit, as follows:

TR = RFO(S,TFoM), *where*
TFoM(x) = F(1FoM(x), ..., kFoM(x)), *and*
$k \in IJS[1,\infty)$
$x \in S$

Here F is defined as a real-valued, strictly increasing function (see the preliminary part of the 'Theorems and Proofs' section). It can be shown to be entirely correct and consistent to use a trade-off figure of merit specification, expressed in terms of a real-valued strictly-increasing function, to correctly generate a trade-off requirement order (using a real-valued function) - see *Theorem 4* in the 'Theorems and Proofs' section.

This is a true trade-off criteria in the sense that all the system FoM values do not need to have (and almost invariably will not, in practice, have) maximal values for the optimal system solutions. In other words, it can adjudicate between 'non-comparable' candidate systems.

COMPLEX SYSTEM OPTIMISATION

This now brings us to an examination of the Subsystem Trade-off Functional Equation, and its practical application in the determination of optimal candidate complex system solutions.

The objective is always to ensure that each candidate system (in S) is the best architecture (or, at least, optimal) solution for that particular implementation. In terms of finding an overall optimal candidate solution, we want to be able to compare 'the best against the best'.

Consider a particular candidate system solution x, where $x \in S$, with a (fixed) number of components n. Then if, for any m^{th} component, there are r_m options (where $m \in IJS[1,n]$) then the maximum number of potential combinatorial solutions is given by the Cartesian product:

$$\times (r_1, r_2, \dots r_{n-1}, r_n)$$

If the number of options for the various components were assumed to be the same ($r_m = k$) then the number of possible design solution would have the dimensionality of the order k^n. This exponential order of the solution space may require the application of special analysis techniques in order to make the problem of identifying an optimal solution tractable. This is addressed in more detail in later sections of this paper.

Decomposition Of Requirements. This design process can be thought of as 'reductionist' in the sense that the requirement specifications of complex systems are decomposed to the lower-level component parts. Unfortunately, the term 'reductionist' now often seems to attract very negative connotations. However it is important that this particular 'decomposition' process should be seen as a duality of both systems analysis *and* design synthesis. For example, the following has been observed:

> A reductionist approach to phenomena does not simply refer all questions to properties of its constituents as individuals. We must also understand the rules for putting those components together. (Cohen and Stewart 1995, page 34).

It is design theory (in this case with respect to the systems-theoretic design of optimal system solutions) that enables us to establish these 'rules'.

The Subsystem Trade-off Equation. For the purpose of clarity of argument, the original development and analysis of the Subsystem Trade-off Functional Equation (Wymore 1998) was limited to a system with 2 components, and 2 evaluation criteria. It is, in fact, equally applicable to systems with any number of components (including, for the trivial case, a system model containing just one component) and for any number of evaluation criteria.

An extension of the Subsystem Trade-off Functional Equation to a complex implementable system with n components and k evaluation criteria, denoted by nk-STFE, yields the following expression:

$$TFoM(x) = F(1FoM(x), \dots kFoM(x)),$$
where
$$jFoM(x) = Ej(jFoM_1(x_1), \dots jFoM_n(x_n))$$

and, also
$$TFoM(x) = K(F_1(1FoM_1(x_1), \dots kFoM_1(x_1)), \dots F_n(1FoM_n(x_n), \dots kFoM_n(x_n)))$$

for
$j \in IJS[1,k], k \in IJS[1,\infty), n \in IJS[1, \infty)$ *and*
$x \in S, x = (x_1 , \dots x_n)$

There is an exact functional equivalence (Wymore 1993: sect. A1.163) of the above two constructs for TFoM. Given that a choice is made for the function F and each Ej such that the Subsystem Trade-off Functional Equation can be satisfied, then we are able to determine the system function K and the component trade-off functions F_m (this may be done by simple algebraic manipulation). This is effectively a process of design 'decomposition' from system-level to component-level (i.e. from F to the various F_m functions).

The function F is the overall (discriminatory) trade-off function, in terms of the overall system figures of merit jFoM - and will be chosen by the system stakeholders and systems engineers. Function F will be common to *all* candidate systems.

However, it might be that not all the components will contribute to a system evaluation function Ej. It might be that, for the m^{th} component, that $jFoM_m(x_m) = 0$. And, indeed, it could also be that, for the whole system, that we could have $jFoM(x) = 0$ (since our candidate whole-system solution may, in fact, be a 'component' of some wider system design problem).

The evaluation functions Ej are used to specify the way in which the individual attributes of the system components (the figures of merit for each individual component) combine to form the overall system figures of merit. These functions will be determined by analysis/modelling of each implementable system solution (i.e. a 'system sensitivity study') employing such techniques as theoretical analysis, simulation, or prototype experimentation.

The domain of each function Ej contains only a single (undifferentiated) contribution from each component since no assumptions are made with respect to internal component architecture. Each component figure of merit ($jFoM_m(x_m)$, $m \in [1,n]$) will therefore essentially reflect component attribution, and its interconnected relationship to other system components. To have more than one contribution from a component would imply that these would be separately selectable - which is not the case (however component figures of merit may be composite - for example the power-to-weight ratio as a 'performance' figure of merit for an engine). It is possible that different Ej evaluation functions will use the same component figures of merit. For example the 'speed + memory' attributes of a computer system may appear in both the 'performance' and the 'cost' evaluation functions. However, this becomes less likely where composite attributes are used in the determination of figures of merit.

For the nk-STFE formulation, the equivalence of the above two equations for $TFoM$ allows solutions (in terms of real value, strictly increasing functions) for F_m (where $m \in [1,n]$) and, by inference, function K to be determined. In other words, we are able to derive functional expressions for the trade-off figures of merit for *each* individual system component that gives the overall system trade-off figure of merit functional requirement ($TFoM$).

The component trade-off functions F_m are not an arbitrary choice - they are a consequence of the requirements of the whole system problem, together with the manifestations of the system's complex architecture of multiple, interconnected component parts.

'FRACTAL' REQUIREMENT DECOMPOSITION

The reference to a 'fractal' process is made in the sense that there is a repetition of self-similar form with respect to the specification of requirements at different levels of design decomposition (whole-system, components, components of components, etc.). For example we have (although in a somewhat different context) the following observation:

> If this [self-similar patterns] is transferred to specifications it can be argued that a fractal information structure provides a noticeable contribution to complexity control in product planning and development since data is captured in repeatedly appearing information structures. (Ludwig, Kokes, and Bürgel 2000).

Of course, in terms of the objectives being discussed here (the design of optimal systems) we are primarily concerned with the decomposition of trade-off function requirements (i.e. F, F_m , F_{mm} , ... etc.). The fact that this is done in such a way that there is an inclusive mathematical relationship between these requirements across the hierarchical levels of design (through the functions Ej), then this provides a strong argument for describing this process as 'fractal'.

The concern here is that each component should be evaluated in respect of a common set of evaluation criteria. It can be shown that, providing that each component's contribution to each of the system's evaluation criteria is included (it is still included, even if it is effectively zero) then a consequence of the application of the extended Subsystem Trade-off Functional Equation (nk-STFE formulation) is that each component will have the same ordered list (vector) of evaluation criteria, which will be the same size (i.e. of k elements) as the overall system evaluation criteria (see *Theorem 5* in the 'Theorems and Proofs' section). The purpose of the theorem is therefore to show that application of the given form for nk-STFE is consistent with this line of reasoning, and that this supports the notion of a 'fractal' design process.

We choose a system with respect to function F, we also choose a system component with respect to function F_m. Therefore, choosing a multiplicity of types of system evaluation criteria (i.e. that we have: $\#(DMN(F)) >> 1$) allows us greater discriminatory powers not only in selecting optimal whole systems solutions, but also in the selection of system components. Note also that the cardinality of the functional domain of the F_m functions will be the same, even for different candidate system designs, since they will be equal to the cardinality of the function F (which, by definition, is the same for all candidate systems). Obviously the F_m functions themselves will be different since we are dealing with components that will probably (but not necessarily) be differently implemented for different candidate system architectures.

The component trade-off functions F_m are not an arbitrary choice - they are a consequence of the requirements for the whole system, together with the manifestations of the system's complex architecture of multiple, interconnected components.

DESIGN EVALUATION

Method. Systems-theoretic optimisation makes use of ideas fundamental to general systems theory, and to technology-specific engineering and science theories, in order that the properties of a whole system solution can be established from the properties of the (interconnected) component parts. It is therefore possible to make a (predictive) identification of the optimal solution - prior to any commitment to actual system implementation. This is an eminently practical approach, for example:

> In the present writer's mind, GST [General Systems Theory] was conceived as a working hypothesis, being a practicing scientist, he sees the main function of theoretical models in the explanation, prediction, and control of hitherto unexplained phenomena. (von Bertalanffy 1987).

Here our ' hitherto unexplained phenomena' are the various properties of the candidate systems designs, from which we wish to use to identify a 'best' system option. We are therefore required to predict (explain), enumerate and compare these properties in order to identify (and optimise) a system solution.

For complex system solutions a systems-theoretic approach allows us to 'decompose' the trade-off requirements to the individual system components. The component designer is provided with a particular trade-off function (i.e. the m^{th} component F_m) in order to quantify the relative rankings of possible component solutions, and the designer of the whole system solution is provided with an overall trade-off function (i.e. function K) in order to evaluate the relative merits of various (possibly constrained) combinations of system components.

Determining The Evaluation Functions. The goal is to be able to characterise the whole system, based on properties/attributes of the system components - that is, to derive the evaluation functions Ej.

Ideally this should be done through 'theorisation' - that is to say, to use the generic equations and 'algorithms' developed for the purpose of determining the particular properties of complex systems. This is, of course, already a well established approach for systems implemented in particular technologies, for example: electronic circuit theory, thermodynamics and engine-cycle characterisation, machine theory (solid mechanics), artificial neural systems theory - etc. It is also well established for multi-technology systems implementations - for example (and in particular) in the application of linear systems theory.

A good example of the use of evaluation theory is the theoretical treatment of the Hopfield model of associative memory where, within the engineering discipline of neural computation (Hertz, Krogh and Palmer 1991), we can have the following expression for maximum memory storage capacity:

$$p_{max} = n / (4 \log n)$$

So here we can calculate a 'storage capacity' figure of merit value (i.e. p_{max}) in terms of the number of units/neurones (n), as a consequence of the deductive analysis of neural computation theory.

In general, the objective is to achieve a 'symbiotic collaboration' of theory-based design methods under the aegis of a formal, mathematically based, general systems theory.

For those cases where a 'theoretic' approach is not available, or is not practical, then we will need to consider the use of parametric system modelling to determine each generic evaluation function Ej. This requires that adequate modelling of the particular properties of the specific candidate system implementations can be produced - and that these properties can be related to the properties of the system's component parts. These models can then be used to determine the evaluation criteria functions Ej. There are various 'directed-search' techniques that can be used to assist in this determination - for example the conventional *Design of Experiment* (DoE) methods, or by use of recent developments in evolutionary computing (see, for example, Parmee, 2001).

Choosing The Evaluation Criteria. The aim should be to choose evaluation criteria (the $jFoM$ functions) that are independent of any particular system implementation, and are relevant for all levels of design decomposition being considered. For example, we may want to consider the following as figures of merit: 'accuracy', 'timeliness', 'availability', 'capacity', 'development risk', 'operating cost', 'disposal cost' - and so on.

Unfortunately, during the requirements capture phase the systems engineer may encounter examples of vague, qualitatively defined evaluation requirements - particularly in terms of stakeholder needs. For example the system operator may ask for a 'safe' or 'user-friendly' system, or the project manager may insist on a 'low-risk' design option, etc.

It is one of the primary tasks of systems engineering to transform these qualitative requirements into quantitative design criteria - in terms of the jFoM evaluation figures of merit. This is almost always a practicable and achievable task (see, for example, Gilb 1999). The notion of 'risk' can, for example, be mathematically quantified in terms of a combination of likelihood (as a probability of occurrence) and impact (i.e. cost of failure or of risk mitigation). This process will necessarily require close co-operation between the various project stakeholders.

Choosing the Trade-off Function. At the project level of systems design it will be incumbent upon the system stakeholders to agree on a specification for the optimisation requirements. This effectively means an agreed specification for the trade-off function F, including a detailed specification for the evaluation criteria (i.e. the 'types' of system figures of merit jFoM) - and an F function which is 'decomposable' in terms of the nk-STFE extended Subsystem Trade-off Equation.

However, it would be naive to believe that the choice of evaluation criteria and trade-off criteria can be entirely objective:

> When speaking of design optimality, one should realise that different people may value different criteria when judging the optimality of an organisation, i.e. the notion of optimality is subjective. (Levchuk, Pattipati, Kleinman 1999: pp. 80)

This subjectivity applies whether we are considering the design optimisation of an 'organisation' or of a 'hard' system problem. The important thing is therefore that the optimisation parameters should properly reflect the viewpoints of all the system stakeholders.

Of course, it may well be that the system under development is 'merely' a component of some wider system problem - in which case it may also be the case that the function F should be viewed as a component function 'F_m' of the wider system problem (see later section on 'Context Dependency').

DESIGN IMPLEMENTATION CONSTRAINTS

There is a widely held belief that component optimality has, almost always, to be 'traded-off' (usually in a highly recursive process) in order to achieve optimality for the whole system, for example:

> He [the systems engineer] will constantly need to emphasise that the optimisation of each sub-system independently will not in general lead to a system optimum. More strongly, he will have to emphasise that improvement to a particular sub-system may actually worsen the overall system. (Jenkins 1987: pp. 159)

Of course, it all depends on what is meant by an "improvement" to a sub-system - in other words, how the evaluation criteria and the trade-off requirements are to be specified for individual sub-systems (system components). It also depends on what constraints (if any) are imposed on the choice of system component combinations.

Ideally we would wish to be able to construct an implementable system solution using a free choice of (optimal) system components. However, it is highly likely that our choice of components, and component combinations (for a particular overall system architecture) will be limited - either for reasons of basic physical viability, or because of non-technical restraints on component choice.

Implementation Constraints. It has been shown that an application of the Subsystem Trade-off Functional Equation (nk-STFE) allows for trade-off requirements to be decomposed to the individual components. If implementation constraints are *not* taken into consideration, then this technique would allow us to optimise the whole system by optimising the individual components (according to the special assignment of the 'decomposed' requirements for each component). However, the introduction of constraints complicates the task. We cannot select individual components, without regard to a choice for the other components - our choice of component combinations is constrained. The result is that, according to the product-order for the trade-offs for the components, there may be 'not comparable' relationships - and we can apply the result of nk-STFE (resp. function K) to ensure that we have the proper means to identify the optimal system solution (to discriminate between 'not comparable' candidate system solutions, according to the component trade-off product-order).

It is in this context that the function K will effectively 'adjudicate' between a set of implementable design alternatives - where it is not possible simply to implement a system architecture comprised entirely of optimal components. It may be, for example, that we may have two (or more) combinations of components that are 'not comparable' on the basis of the product-order of the components of each system design solution. It is the application of the function K that resolves this problem, and which can identify the optimal design solution(s) according to the trade-off requirements (see Theorem 6 in the 'Theorems and Proofs' section).

A good example of combinatorial constraint is that of the need for an electrical power supply unit (component) to be 'rated' with respect to the demands of the other system components (or, of course, vice-versa, for the choice of the other components to be constrained with respect to the capabilities of a given electrical supply unit). The constraint is therefore in terms of the available energy - which, ultimately, will be sourced from outside of the system (including stored energy devices such as batteries or fuel-cells).

The choice for one component implementation, for a particular system architecture, therefore can (and usually does) affect the choice of some of (and perhaps all of) the other component implementations.

It is therefore highly likely that our choice of components, and component combinations (for a particular overall system architecture) will be limited - for reasons of system-level constraints on component choice. Although we may wish to consider a 'continuum' of possible candidate component designs (especially for custom-built 'bespoke' system solutions) it is often the case that a candidate system solution can only be built from a limited space of system component combinations.

A real-valued function order (i.e. such that $RFO \in FNS(S2, \{0,1\})$ can therefore also be expressed in terms of the set of candidate system designs (S) and the figure of trade-off function with respect to each system component (F_m) as follows:

$$RFO(S, F_m) = \{ ((a, b), y) : (a, b) \in S{\cdot}2; \; y \in \{0, 1\}; \; y = 1 \Leftrightarrow F_m(a) \leq F_m(b) \}$$

- and hence, for the m th order, we have:

$$mR = RFO(S, F_m)$$

Again, a line diagram can be used to illustrate the relationships been the component orders, and the system trade-off requirement. The diagram below is for three alternative design options for a system solution (a^X, a^Y, a^Z) built using combinations of components (1 and 2).

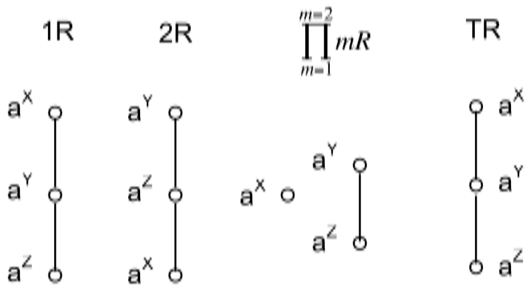

Figure 3 Component-based Product-Order

In this line diagram the notation:

$$\prod_{m=1}^{m=n} mR$$

is used to indicate that we are evaluating a product-order over a set of system design options, with respect to a *fixed* number of component trade-off orders (i.e. the n components for the particular candidate system architecture). The function K is applied to this product-order to produce the trade-off order result, TR.

Combinatorial Constraints. The 'closure' implicit on the space of the implementation space (on S) - as a consequence of system-level constraints - can therefore effectively enforce combinatorial constraints on system implementation. A good example of this is the acceptance of finite resources, as specified in the technology requirement (TYR), which will place resource allocation/sharing constraints on the buildable system solutions.

What, then, is the formal relationship between the requirements for the whole system, and the combinatorial constraints on system architecture? If, for example, we specify some finite limit on a figure of merit such as performance, affordability, resource utilisation, reliability, etc., such that we have:

$$jFoM(x) \geq jMIN, \text{ where } jMIN \in RLS$$

then the 'decomposition' of this requirement leads to an expression for the evaluation function of the form:

$$Ej(jFoM_1(x_1) \; .... \; jFoM_n(x_n)) \geq jMIN$$

The consequence of this will be some combinatorial constraint on component choice.

It may be that *all* the possible component combinations that satisfy one constraint might not include *any* component combination that satisfies another constraint. In other words, there may be *no* viable design options. In practice, constraints are usually applied to all evaluation criteria - for example, to express *minimum* values for affordability, for reliability, and for performance.

The design solution for a system component can therefore be formally described as combinatorially constrained (within the context of a particular system architecture) if *any* implementable solution for that component is constrained to include some particular choices for the other system component(s). The choice of one component implementation in a particular system architecture can (and often does) effect the choice of some of (and perhaps all of) the other component implementations (see, also, the later section on 'Context Dependency').

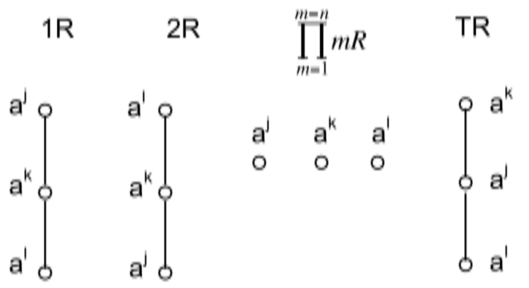

Figure 4 Example of 'Non-comparability'

A consequence of this approach to design optimisation, with an imposed constraint on component-combination choice, is therefore that the chosen 'best' or 'optimal' implementable system design may therefore have to include components which are, in fact, sub-optimal. For example:

> Optimality of system interfaces cannot always be achieved by optimising the individual interfaces. It may be necessary to make lower level interfaces suboptimal to ensure that high level user requirements and associated system requirements definitions are realised by the implementation system. (Sage, Lynch 1998: 216)

A consequence of this approach to design optimisation, with an imposed constraint on component-combination choice, is therefore that the chosen 'best' or 'optimal' implementable system design may have to include components which are sub-optimal. In fact it is entirely possible that, on the basis of the 'decomposed' trade-off requirements for the individual components, that *none* of the components of an optimal system solution are, in themselves, optimal component designs (re. Shell 2002).

This is the very antithesis of the 'one size fits all' philosophy that is so prevalent in today's industrial, financial and service-industry systems (for example, that all 'business units' of a particular enterprise should endeavour to maximise profit).

Consider an example of 3 candidate system designs ($S = \{a^j, a^k, a^l\}$) each built using two types of components (1 and 2), and where we may derive the following order relationship:

$$TFoM(x) = k(F_1(x),F_2(x)), \ \ x \in S$$

$$TFoM(x) = 1.0 \times F_1(x) + 2.0 \times F_2(x)),$$

- we determine the following component trade-off figures of merit values, in the context of the system implementation:

$$F_1(a^j) = 0.5, \ F_1(a^k) = 0.4, \ F_1(a^l) = 0.05$$

$$F_2(a^j) = 0.1, F_2(a^k) = 0.2, F_2(a^l) = 0.3$$

Therefore, when we apply the trade-off function for the whole system, we get the following for each of the design options:

$$TFoM(a^j) = 0.7, TFoM(a^k) = 0.8$$

$$TFoM(a^l) = 0.65$$

This relationship is caricatured in figure 4, using the appropriate line diagram:

Clearly system a^k is sub-optimal with respect to component trade-offs F_1 and F_2 - but is the 'best' system solution with respect to the overall trade-off requirement TR.

Component Availability. There may be constraints on the permitted combinations of system components for non-technical reasons (commercial, business, contractual, legal, ethical). For example, there may be a commercial preference for the use of COTS components, or to incorporate existing 'legacy' components, or to use 'standard' parts. These constraints would also need to be reflected in the TYR technological requirements of the system requirement specification (and hence be reflected in the 'buildability space' of system candidate designs).

Component Tailoring. A consequence of the different representations of the system evaluation function Ej for different candidate systems is that the same *type* of component may be implemented differently to provide for an 'optimal' host system solution.

This could mean, for example, that a component supplier may offer two different versions of a component for integration into two different candidate system solutions. However, in each case, the particular component design may be the desired design solution - with respect to the particular requirement specifications placed on the component supplier. The concept of component 'preference' is entirely dependent on the context - the host system into which it is to be integrated.

COMPONENT FEASIBILITY CANDIDATURE

For each evaluation function there will be a limitation imposed on the system solution in terms of a technology constraint (as part of the TYR specification), such that:

$$Ej(jFoM_1(x_1), ... jFoM_n(x_n)) \geq jMIN$$

where $jMIN \in RLS$ *and* $j \in IJS[1,k]$

and, from the nk-STFE formulation:

$$F(1MIN, ... kMIN) \geq MIN$$

For example, a system may be required to have a minimum level of reliability, a minimum performance capability, and a minimum level of affordability.

Now, if we take (for example) the first evaluation criteria ($j = 1$) and make the assumption that all the components except the first component ($m \neq 1$) are implemented using the *best available technology* - then, providing their figures of merit can be properly quantified, we can evaluate (using function E1) a *minimum feasible constraint* for this first component: $1MIN^{\dagger}_1$.

If we extend the above argument to all criteria, then we can obtain a complete list for component 1 for *all* evaluation criteria:

$$jMIN^{\dagger}_1 , j \in IJS[1, k].$$

And we can repeat this exercise for every component such that *all* minimum feasibility constraints for *all* components are evaluated:

$$jMIN^{\dagger}_m, j \in IJS[1, k] \text{ and } m \in IJS[1, n]$$

The identification of a minimum feasible constraint for every component, and for every evaluation criteria, allows the system designer to exclude those components which cannot be a feasible part of a whole system solution - as illustrated below.

In this example a^X is excluded because the first component of a^X has one or more evaluation criteria values that are less than the minimum feasible constraint. Similarly, a^Y is excluded because the second component of a^Y has one or more evaluation criteria values that are less than the minimum feasible constraint.

In this way the space of candidate designs that needs to be searched can be reduced, and the identification of an optimal solution can be made a more practical proposition.

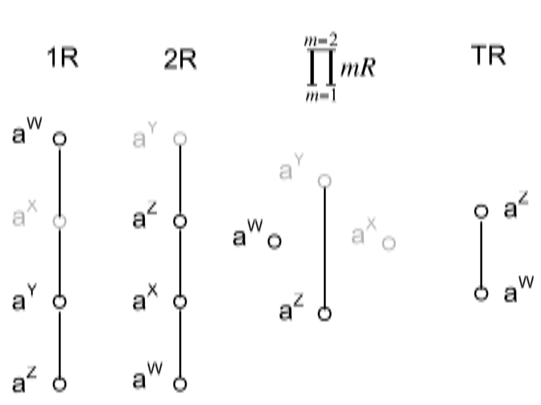

Figure 5 Exclusion of Unfeasible Candidates

Of course, even with a considerably reduced number of candidate systems, the system designer will still (in principle, at least) be effectively faced with a complex (NP-hard) task of identifying the optimal solution - and appropriate search techniques (i.e. approximation algorithms, genetic algorithms, DoE, etc.) may have to be employed.

THE CLASSIFICATION OF COMPLEX SYSTEM DESIGN AS NP-COMPLETE

If we are seeking the optimal choice of possible systems solutions within the implementable space S of design candidates, then that search can be simply conducted in polynomial time. If the number of candidate solutions is represented by n such that $n = \#(S)$ then the problem is a 'trivial' one of making $n-1$ comparisons.

However the problem of identifying the optimal solution for *each* of the candidate designs (so that we can 'compare the best with the best') is potentially much more difficult. This is a direct consequence of the possibility of combinatorial constraints on the complex system architectures. If we could be sure that there are no constraints then we would 'simply' optimise a system design by optimising each of the system's components.

For complex system design a proof has been proposed to demonstrate that system design optimisation is an NP-complete class of problem (Chapman, Rozenblit, Bahill 2001). This is based on the argument that the Knapsack Problem (which is known to be NP-complete) can be sensibly mapped to an instance of the optimal systems design problem. The proof therefore leads to a reasoned assumption that no polynomial-time optimisation algorithms are available.

In fact if we wish to represent complex systems design in the form of the 'knapsack' problem, then we can be very specific - we can represent the complex design optimisation as an exact n-item knapsack problem (E-nKP) where n is the fixed number of system components. Here we wish to maximise the value (profit) of the knapsack's content, but are constrained by a limit on the overall size (capacity) - and possibly to constraints on multiple (multi-dimensional) criteria. This, therefore, results in combinatorial constraints on our choice of items (components).

It is certainly the case that the formal description of complex systems design (as outlined in this paper) can be generally represented as a linear programming (LP) problem. However that does not, in itself, rule out the possibility of finding a generic 'approximation' algorithm that can accurately identify optimal solutions in polynomial time.

Perhaps the question that needs to be asked is whether NP-completeness should necessarily (or even usually) prevent the identification of optimal (highly complex) system solutions from being a reasonably tractable task - even for multiple-variable and multi-objective optimisation problems? It could be argued that for most *practical* cases NP-completeness should not preclude a truly optimal design from being accurately identified, in a reasonable time-scale. We have, for example, the following observation:

> In practice, many NP-hard problems can reliably be solved to near optimality by fast heuristic algorithms whose performance in practice is far better than their worst-case performance (Karp 1998: pp. 156)

It has been shown that, for this NP-hard class of problem, that there exists both a generic polynomial time approximation scheme (PTAS) and a fully polynomial time approximation scheme (FPTAS) that can be applied to solve the problem to optimality (Caprara, Kellerer, Pferschy, Pisinger 1998, sect. 4).

A more recent area of research has been the investigation into the use of Gentic Algorithms (GA) for fast and accurate identification of optimal solutions for systems that are complex, combinatorially constrained, and evaluated with respect to multiple objectives.

The feasibility (and practicality) of using GA for solving NP-hard problems has been thoroughly addressed, from a theoretical bases, by previous researches (for example, De Jong and Spears 1989). More recent research effort has been directed towards investigations of various GA implementations and operational strategies with regard to real industrial applications (for example, Hajela 2002).

CONTEXT DEPENDENCY

In the discussion so far, the identification of the space of implementable candidate solutions (i.e. the space S) has been described simply in the context of the 'stakeholder requirements'. In particular the input/output, the technology, and the trade-off requirements (i.e. IOR, TYR and TR) have been used as a way to formulate the needs of the stakeholders.

However system solutions should be seen as constituents of some wider system 'problem'. Just as components are parts of the whole system, then systems will be parts of some wider system context.

This concept of system implementation context, and of an extensive systems hierarchy, raises two important issues: firstly, of being able to properly relate optimisation criteria and trade-off functions between the different levels; and secondly, of allowing for possible combinatorial constraints of component architectures at each of the various levels.

The first issue concerns the need to provide a proper rationale for the choice (and representation) of evaluation criteria, and of the trade-off function, within any particular systems context. As we have seen, we may be able to use a systems-theoretic approach to 'decompose' the system's optimisation requirements into an appropriate form for the system's component parts. However that then begs the question - if a systems-theoretic approach is not feasible, what then should be our terms of reference for establishing the appropriate 'local' optimisation requirement; and what would the consequences be for the higher-level systems implementation?

The second issue concerns the possibility of there being combinatorial constraints within any particular systems context. Then the requirement may be to produce a sub-optimal system solution (or solutions) at the next, lower 'child' level in order to produce an optimal solution at the 'parent' level. We cannot always assume that the immediate task should be to identify and implement an optimal system solution - it is entirely context-dependent.

Implicit in both of the above key issues is the perceived problem of trying to keep the systems engineering task focused within an appropriate context. The danger is seen to be that of an open-ended process of expanding the problem domain to include ever more interconnected 'systems'.

These difficulties of systems optimisation, that result from context-dependency, will occur regardless of whether an optimal solution is to be achieved by systems-theoretic prediction, or by experimentally recursive selection. However the fundamental questions will remain, namely: what is the proper rationale for the choice of evaluation criteria and trade-off function; and will optimisation at a particular level of system hierarchy provide for (or prevent) an optimal choice of system solution with respect of some wider system context?

THEOREMS AND PROOFS

It can often seem that a ‘formal’ approach to system design simply states the obvious - and that it does so at considerable length. However it is worth considering that (as with many other branches of engineering science) that those assertions which would seem to be ‘intuitively obvious’ are not necessarily correct, and those that are correct are not necessarily obvious.

Real-valued Functions. These are functions whose range (co-domain) is a set of real numbers. In other words, for a function f, $\mathrm{RNG(f)} \subseteq \mathrm{RLS}$.

Strictly Increasing Function. A function f is a real-valued strictly increasing function in the sense that if $r < s$ then $f(r) < f(s)$ - for example a polynomial with positive coefficients.

Theorems. The following theorems, and theorem proofs, are provided so as to substantiate particular statements given within this paper. They are not intended to be an exhaustive exposition of the theory that underpins this mathematical approach to system design. The interested reader should refer to the referenced texts (i.e. Johnson 1998, Wymore 1993 and 1998).

<u>*Theorem 1*</u>. Any candidate system that is sub-optimal with respect to all the orders (i.e. the product-order) must also be sub-optimal with respect to the trade-off requirement order.

Proof:

For any system x, where $x \in S$, if

$$\exists\, y \;\; y \in S \;\; y \neq x \;\; x < y \textit{ modulo} \prod_{j=1}^{j=k} jR$$

then, from the essential pre-requisite criteria of a trade-off requirement,

$$x < y \textit{ modulo } \mathrm{TR}$$

- and hence system x must also be sub-optimal with respect to TR.

<u>*Theorem 2*</u>. A corollary to the above theorem 1 is that a system that is ‘best’ with respect to the product-order will also be ‘best’ with respect to the trade-off order.

Proof.

For a system y*, where $y^* \in S$, if

$$\forall\, x \;\; x \in S \;\; x \neq y^* \;\; x < y^* \textit{ modulo} \prod_{j=1}^{j=k} jR$$

then, also from the essential pre-requisite criteria of a trade-off requirement,

$$x < y^* \textit{ modulo } \mathrm{TR}$$

- and hence system y* must also be the ‘best’ system solution with respect to TR.

<u>*Theorem 3*</u>. In terms of the trade-off requirement TR, that is expressed as a real-valued order (RFO), then a system that is not sub-optimal must be optimal.

Proof.

We are given a definition for the trade-off requirement TR based on a real-valued function order (i.e. RFO(A,f)). Here f is a real-valued, strictly increasing functions, which does not permit a ‘not comparable’ relationship within TR - hence there has to be an optimal candidate system solution (or solutions). Then, if for two candidate systems $x \in S$ and an optimal solution $y^* \in S$ (which must, by definition, exist) and system x is <u>not</u> sub-optimal with respect to system y* (in terms of the trade-off requirement TR), then it must be the case that:

$$x \equiv y^* \;\vee\; x \textit{ ntc } y^* \;\vee\; x > y^* \textit{ modulo } \mathrm{TR}$$

However a ‘not comparable’ relationship is not permitted for a TR trade-off requirement that is expressed in terms of a real-valued order (RFO) of a real-valued, strictly-increasing function (F) of the system figures of merit criteria. In other words, a comparison on the basis of a simple number value must be ‘comparable’. Also, by definition, system x cannot be ‘greater’ than an optimal solution y*. Therefore, it must be the case that:

$$x \equiv y^* \textit{ modulo } \mathrm{TR}$$

Therefore, system x must also be an optimal solution.

<u>*Theorem 4*</u>. If TFoM is a strictly increasing function defined over RLS^k and {1FoM, ... kFoM} is a subset of real-valued functions defined over S, then RFO(S, TFoM) is a trade-off ordering amongst the orderings generated by the jFoM’s, (where jR = RFO(S, jFoM) for $j \in \mathrm{IJS}[1, k]$).

Proof:

If we have a trade-off figure of merit expressed as follows: if

TFoM(x) = F(1FoM(x), ..., kFoM(x)),
where
$k \in \mathrm{IJS}[1,\infty)$
$x \in S$

- and a trade-off requirement order:

$$TR = RFO(S,TFoM)$$

Then we need to show that the following statement is true: if

$$x < y \textit{ modulo } \prod_{j=1}^{j=k} jR$$

then

$$x < y \textit{ modulo } TR$$

However, for the first line of the above to be true, it must be that the following statement:

$$\forall j \;\; jFoM(x) \leq jFoM(y) \;\; \wedge$$

$$\exists j \;\; jFoM(x) < jFoM(y)$$

is also true.

Note that in this statement we are using the symbols $<$ and $\leq$ in the 'trivial' sense - here comparisons are made in terms of simple number value, since we are dealing with figure of merit terms computed using real-valued functions.

But function TFoM is defined to be a real-valued, strictly increasing function, therefore it must also be true that:

$$TFoM(x) < TFoM(y)$$

and hence that:

$$x < y \textit{ modulo } TR$$

<u>*Theorem 5*</u>. The number of evaluation criteria Ej_m for each and every component of a system will be the same as the number of evaluation criteria Ej for the whole system.

We assume the given definition of the extended Subsystem Trade-off Functional Equation, where the argument of *each* Ej function *must* include all the components.

Proof.

From the extended Subsystem Trade-off Equation we have:

$$DMN(F) = RLS^k$$

- i.e. the k figures of merit for the whole system.

However, this equation is defined in such a way that each component will have only one j^{th} figure of merit assigned to it - by means of the arguments of each Ej function. There are k figures of merit for the whole system, and therefore there are k evaluation functions Ej - therefore, for each component there can only be k figures of merit - the same as for the whole system:

$$DMN(F_m) = RLS^k$$

Note also that any 'zero' assignments of the evaluation criteria needs to be included - i.e. cases where $jFoM = 0$, $jFoM_m = 0$.

<u>*Theorem 6*</u>. Where there are *combinatorial constraints* with respect to the choice of system components (as defined in the section 'Closure of the Implementation Space'), then for any candidate system solution, the function K provides for the identification of an optimal system solution with respect to the overall system trade-off requirements specified by F and Ej.

Proof.

This proof is a scholium to the proofs of theorems 1 and 2, but with the relationships determined over the product orders of the component trade-off requirements.

Therefore, for a sub-optimal system x, where $x \in S$,

$$\exists y \;\; y \in S \;\; y \neq x \;\; x < y \textit{ modulo } \prod_{m=1}^{m=n} mR$$

then,

$$x < y \textit{ modulo } TR$$

and, for a 'best' system y*, where $y^* \in S$, we have

$$\forall x \;\; x \in S \;\; x \neq y^* \;\; x < y^* \textit{ modulo } \prod_{m=1}^{m=n} mR$$

then,

$$x < y^* \textit{ modulo } TR$$

In this case TR is, effectively, a function K of the component trade-off figures of merit F_m - since TR is functionally equivalent (i.e. by means of algebraic manipulation) to the F function of the evaluation criteria Ej (Wymore 1993: sect. A1.163 re. definition of equivalent functions). Therefore the orders determined by the two forms of TR will be identical, including the identification of the 'best' particular system option, and the 'best' overall system solution.

NOTATION

The following table provides a summary of the notation used throughout this paper. A more detailed description of these terms is available within the referenced texts (Johnson 1998, Wymore 1993).

{ , , ... }	Set
$\cap$	Intersection of sets
$\cup$	Union of sets
$\subseteq$	is a sub-set of .... (the set)

$\varnothing$	The 'empty' set, also denoted as {}
$\times$	Cartesian product, viz: {1,2}×{a,b} = {(1,a), (1,b), (2,a), (2,b)}
$\in$	is a member of ... (the set), belongs to ...
#	Number of elements of a set (or vector), set 'size', or 'cardinality'
$<$	less than
$>$	greater than
$\leq$	less than, or equivalent to
$\equiv$	equivalent to
$\Leftrightarrow$	if, and only if
~	negation, 'not'
$\vee$	or
$\wedge$	and
$\forall$	for all (each, every, any)
$\exists$	there exists, there is
Π	product
DMN	domain of a function
Ej	the jth evaluation function
F	trade-off function
F_m	mth component trade-off function
FNS(A,B)	Set of *all* functions, mapping from domain set A to, or into, the range set B
IJS[j,k)	Set of integer numbers: { i : i ≥ j , i < k }
IJS$^+$	Set of all positive (non zero) integer numbers
jFoM	the jth figure of merit evaluation criteria
jR	the jth order function, such that jR ∈ FNS(S^2, {0, 1})
k	number of evaluation criteria
n	number of system components
ntc	not comparable to
RFO	Real-Function Order. An order defined in terms of some real-valued function
RLS[x,y]	Set of real numbers: { w : w ≥ x , w ≤ y }
RLS	Set of all real numbers
RNG	Range (co-domain) of a function
S	the set of candidate implementable system designs
TFoM	Trade-off figure of merit evaluation criteria
TR	Trade-off Requirement, an order over the set of implementable system designs.

SUMMARY

This paper focuses on developments of a formal (mathematical) systems theoretic method for the specification, identification, design and development of optimal solutions for complex system problems.

Throughout this paper special emphasis is placed on the real, practical difficulties of engineering optimal system design solutions, with particular regard to the following: the work required (in terms of specific theoretical analysis/prediction, modelling, experimentation) in order to determine the evaluation criteria and trade-off functions at each level of design decomposition; and, also, the effective closure of the space of candidate implementable system designs as a consequence of necessarily providing physically viable design solutions subject to constraints (e.g. finite resources, environmental regulations, commercial limitations, etc.)

This 'optimisation-orientated' approach to complex system design can help an engineering team define, organise, and execute the systems engineering effort. It is especially helpful in the early stages of the identification and capture of the required design models (for example: life-cycle cost, performance, reliability, risk) for both the overall system and the system components.

This approach also permits an *effective* top-down design strategy to be employed. This eliminates (or at least will minimise) the need for design recursion. There is provision for formal traceability from top-level requirements down to component design. A proper rationale can be given for each design decision, particularly in terms of the identification of optimality for a candidate system solution.

A mathematically based systems-theoretic method of complex system design has a natural affinity with other, established, specialist engineering sciences. It is seen as being

particularly suited to those design problems where a conventional theoretic approach can be applied to the system, and the system components, in order to comprehensively 'predict' the properties of the various design implementations.

REFERENCES

von Bertalanff, L., "General System Theory - A Critical Review", in Systems Behaviour, Ed. by the Open Systems Group, The Open University, Harper & Row Pub. Ltd., London, 1981. pp. 59-79

Chapman W. L., Rozenblit J., Bahill A. T., "System Design is an NP-complete Problem", Systems Engineering, Vol. 4, issue 3, 2001, pp. 222-229

Capara A., Kellerer H., Pferschy U., Pisinger D., "Approximation Algorithms for Knapsack Problems with Cardinality Constraints", 1998. `http://www.diku.dk/research/published/98-4.ps.gz`

Cohen, J., Stewart, I., "The Collapse Of Chaos", Penguin Books Inc., New York, 1995.

De Jong, K. A., Spears, W. M., "Using Genetic Algorithms to Solve NP-Complete Problems", Proc. 3rd Int. Conf. on Genetic Algorithms, George Mason University, Fairfax, VA, 1989, pp. 124-132

Fertig, J., and Zapata, R., "A Mathematical Foundation For Systems Synthesis", Proceedings Of The 1st International Conference On Applied General Systems Research, 1977.

Ganter, B., Wille, R., "Formal Concept Analysis - Mathematical Foundations", Springer-Verlag Pub., Berlin Heidelberg, 1999.

Gilb, T., "Quantifying The Qualitative: How To Avoid Vague Requirements", 9th International Symposium, INCOSE, Brighton 1999, pp. 879-886.

Hajela, P., "Search Efficiency in Genetic Algorithms", in Optimization in Industry, Ed. by Parmee I. and Hajela P., Springer-Verlag, London, 2002.

Hertz, J., Krogh, A., Palmer, R., "Introduction To The Theory Of Neural Computation", Addison-Wesley Pub. Co., 1991.

Jenkins, G., M., "The Systems Approach", in Systems Behaviour, Ed. by the Open Systems Group, The Open University, Paul Chapman Pub. Ltd., 1987.

Johnson, D., L., "Elements of Logic via Numbers and Sets", Springer-Verlag Pub., London, 1998.

Karp R. M., "The Mysteries of Algorithms", in People and Ideas in Theoretical Computer Science, Ed. by Calude C. S., Springer-Verlag, Singapore, 1998.

Klir, G., "A Review Of Model-Based Systems Engineering", International Journal Of General Systems, Vol. 25, No 2, 1996.

Levchuk, Y. N., Pattipati, K. R., Kleinman, D. L., "Analytic Model Driven Organizational Design and Experimentation in Adaptive Command and Control", Jnl. Systems Engineering, Vol. 2, N° 2, 1999, pp. 78-107.

Lloyd, S., "Measures of Complexity - a Non-exhaustive List", Dept. Mech. Eng., MIT, Cambridge, MA, 1996

Ludwig, L., Kokes, M., Bürgel, H., "Complexity Control By Fractal Specification (FraSp) - A Practicable Approach?", 2nd European Systems Engineering Conference, INCOSE, Munich, September 2000, pp. 59-64.

Parmee, I. C., "Evolutionary and Adaptive Computing in Engineering Design", Springer-Verlag, London, 2001.

Sage, A. P., Lynch C. L., "Systems Integration and Architecting: An Overview of Principles, Practices, and Perspectives", Jnl. Systems Engineering, Vol 1, N° 3, 1998, pp. 176-227.

Shell, A. D., "Function Based Design Of Complex, Complicated Systems", 9th International Symposium, INCOSE, Brighton, England, June 1999.

Shell, A. D., "A System Theoretic Process For Efficient Design Of Optimal Systems", 2nd European Systems Engineering Conference, INCOSE, Munich, Sept. 2000, pp. 165-176

Shell, A. D., "System Function Implementation and Behavioral Modeling: A Systems Theoretic Approach", Jnl. Systems Engineering, Vol 4, N° 1, 2001, pp. 58-75.

Shell, A. D. "The Direct Synthesis Of Optimal Design Solutions For Complex System", in Optimization in Industry, Ed. by Parmee I. and Hajela P., Springer-Verlag, London, 2002.

Wymore, A. W., "Model-Based Systems Engineering", CRC Press Inc., 1993.

Wymore, A. W., "Subsystem Optimization Implies System Suboptimization: Not!", 1998. `http://www.sie.arizona.edu/sysengr/wymore`

BIOGRAPHY

Tony Shell has a BSc degree from Loughborough University Of Technology. He has acquired almost 30 years experience of systems engineering work within the UK aerospace industry with companies that include Sperry Gyroscope (UK), BAe Space & Comms., Ferranti Aircraft Equipment Division, BAe System & Equipment and Rolls Royce MAE. He is currently working for BAE SYSTEMS on the specification and design of advanced avionics systems. He has been a member of the International Council On Systems Engineering since 1995 and has recently founded the organisation DarkLake Synectics as a wholly independent enterprise, with the aim of promoting the use of systems-science methods within the systems engineering community.

[Note: This research paper was published as a chapter within the following book 'Optimization in Industry', Dr Ian C Parmee and Dr Prabhat Hajela (Eds.), Springer-Verlag, London 2002, pages 175 to 188, ISBN 1852335343]