System Function Implementation and Behavioral Modeling - A Systems Theoretic Approach

Tony Shell

ABSTRACT

Systems theory is now a mature part of the discipline of general systems engineering science, with a substantial amount of research effort having been undertaken in the last forty years - however there is still very little evidence of the widespread practical use of systems-theoretic methods within the engineering industry. This is despite there being strong evidence that many of the current problems in the delivery of acceptable (or even usable) large, complex, systems solutions result from a failure to apply a rigorous systems-science approach.

This paper therefore introduces some *practical* ideas for the effective use of an established *mathematical* systems theory to the specification and design of engineered system solutions. In particular the following areas are explored: the capture of system requirement (and in particular ways of ensuring a *proper and comprehensive* specification of input/output requirements); the modelling of system (complicated) behaviours, including anomalous behaviours arising as a consequence of *real* system implementation; and the formal relationship between a comprehensive input/output requirement specification and the 'complicated' behaviours of the candidate system design solutions.

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

[Note: This research paper was first published in 'Systems Engineering', The Journal of The International Council on Systems Engineering, Volume 4, Number 1, 2001, pages 58 to 75, ISSN 1098-1241]

1. Introduction

Descriptions of the initial phases of a project life-cycle are variously described by a broad range of systems engineering standards and process models (e.g. EIA 632 1997, sect. 5.3 'System Design Process'). These phases can reasonably be summarised as follows: an initial capture of requirements; requirements analysis and validation; the identification of the candidate designs; and finally trade-off studies to select a 'best' (optimal) implementable system solution.

It is well documented that even seemingly minor errors and omissions during the critical early stages of the project life-cycle can result in substantial delays and cost over-runs, frequently to the extent of undermining the chances of a successful conclusion to a project. Unfortunately, for many projects, this underlying 'truth' is only recognised or acknowledged in hindsight. A notable case in point was the failure of the inaugural launch of the Ariane 5 rocket, where 'specification and design errors in the software of the inertial reference system' necessitated the further expenditure of some $320million in management and engineering changes - in addition to the particular losses incurred for the launcher and payload (CNES/ESA report 1996, and Flight International 1996).

Failures in system specification and design (as demonstrated by Ariane 501) are not unusual for large, complex systems. These types of failures have particular relevance to the discussions within this paper - in particular with regard to the problems of the proper capture of input/output requirements, and of the design modelling of complicated system solutions. This paper therefore specifically explores the way in which complicated behaviour (i.e. multiple, interrelated system modes of an implementable system design) can be expressed such that the satisfaction of complex functional requirements (in terms of the input/output requirements) can be determined in a rigorous, formal manner.

The adoption of an established, and mathematically based, systems theory can provide the formulae, theorems and proofs required to underpin the systems engineering processes and decisions. An established systems-theoretic approach (Wymore 1993) is therefore employed extensively throughout this article to provide a substantiated and formal (mathematical) basis for the development of ideas.

The content of this paper addresses the following topics with respect to the practical application of a systems-science methodology:

a) Systems-theoretic principles, and characterisation of system design attributes (sects. 2 to 4).

b) Specification of overall system requirements (sect. 5)

c) Specification of the functional (input/output) system requirements (sects. 6).

d) Satisfaction of the functional (input/output) system requirements by implementable system design (sect. 7).

e) Specification and verification of design solution conformance with respect to complicated system behaviour of the implementable system solution (sects. 8 to 10).

The paper concludes with an explanatory listing of the specific (systems theoretic) mathematical notation (sect. 12), and a worked-example in the form of the preliminary design of a gas-turbine (jet) engine (sect. 13).

2. Practicable Systems Theory

The practical application of a 'systems science', particularly in terms of the use of a rigorously defined and comprehensive systems theory, is still not widespread within industry. For example, it has been observed that:

> Currently systems science appears to be directed towards problems solving in organisations but without reference to general problem-solving methods. It operates mostly in terms of descriptive, rather vaguely defined theoretical constructs and models which are difficult to relate to observations. As such, it operates at a metaphysical level, fragmented and remote from well-established branches of knowledge. [Korn 1997]

This is an unfortunate state of affairs since the adoption of a mathematically-based systems theory can provide the formulae, theorems and proofs required to underpin the systems engineering processes and design decisions. In fact the application of a mathematically based systems theory to systems design is a well researched and proven approach - the viability of which has already been extensively reviewed (for example: Klir 1996, Bahill and Dean 1996, Fertig and Zapata 1977).

It is most important that the theory underlying any general systems engineering (formal) method should be relevant to the problems being addressed. It must be able to adequately represent real physical phenomena where necessary, and must provide a *practical* means of deriving implementable system design solutions.

Of course, it has to be accepted that even the most rigorous, scientifically based engineering methods will have their limitations. It is extremely difficult, if not impossible, to comprehensively codify any scientific method - however that does not prevent us from developing and applying certain practical rules, built on logical argument, and tested by observation, experiment and experience.

Unfortunately many of the theoretical methods being actively promoted as 'systems engineering' formal methods are, in fact, closely associated with particular technologies (in terms of modelling paradigms and notation), or have limited scope (and, especially, with little or no reference to design optimisation), or are particularly concerned with specific implementable system design techniques (for example, for computer-based systems solutions). It would appear that these concerns are shared by Prof. Joseph Goguen who says in respect to general systems theory (GST) that:

> But it was (and still is) disappointing to me that so few people felt any need for concepts and theories of such generality; they seem happy to have (more or less) precise ideas about specific systems or small class of systems, with little concern for what concepts like system, behaviour and interconnection might actually mean [Calude 1998, page 98]

It is therefore for these reasons, in particular, that this paper advocates the development and use of a *general* systems methodology - built upon basic axiomatic rules of mathematical systems theory. This should, as Professor Goguen puts it, address such modelling concepts as:

> .... system, behaviour, and interconnection, formalized in such a way as to include hierarchical whole/part relationships. [Calude 1998, page 98]

Therefore an established systems theoretic approach (i.e. Wymore 1993) is employed extensively throughout this paper so as to provide a substantiated and formal (mathematical) basis for (and in particular) the development of ideas concerning complicated system behaviours.

However, this paper is *not* intended to be a detailed exposition of systems theory - its primary purpose is to describe how an effective systems theoretic approach can be practicably applied to provide practical solutions to 'real-world' problems. The systems-theoretic mathematical formulations, their derivations, and the relevant theorems and proofs are available within the referenced systems engineering text (Wymore 1993).

3. Formal Method

3.1 System Definitions and Specifications. The notion of system definition and system specification are concepts that are fundamental to the proper understanding and correct application of a systems-theoretic system design method. It is therefore worthwhile to revue their meanings, and proper useage.

The term 'definition' refers to providing a description of *meaning* (formal or informal) to some statement. The term 'specification' refers to the detailed description of a *particular thing* or *instance*. These literal meanings are therefore required to be reflected in the use of any mathematical expressions used within a 'formal' context. For example a system is *defined* (in the context of the systems-theoretic method used throughout this paper) as the quintuple:

$$Z = (SZ, IZ, OZ, NZ, RZ) \qquad [3.1.1]$$

- where this *discrete* form of a system model is defined in terms of sets SZ, IZ, and OZ for the system states, inputs, and outputs respectively - and the functions NZ and RZ for the next-state and read-out functions. This mathematical construct is clearly a definition - a generic 'system' description. Importantly, it also provides a definitive 'template' for *specifications* of particular system design solutions.

These types of formal (mathematical) definitions can, in the context of systems engineering theory, be employed to describe the 'meaning' of a wide range of theoretical constructs, including requirement specifications, systems (as above), system resultants (of complex designs), system modes of behaviour, design implementation, optimal system solutions - and so on.

A systems engineering *specification*, therefore, will be a particular example (instance) of a systems-theoretic construct, with its elements given specific sets of attributes, and specific functional relationships, expressed according to the (mathematical) rules of the appropriate, and particular, *definition* for that construct.

3.2 Semantics and Syntax. System design requires an ability for the imaginative creation of mental models of systems engineering constructs. These 'imagined' solutions are required to provide sufficiently accurate representations of the real-world phenomena (for example, the system behaviours) and be formally recorded (i.e. using mathematical, grammar, graphical rule, etc.) in a widely understood, communicable language, that provides appropriate contextual meaning (semantics), within an established set of rules (syntax).

It is therefore widely accepted that 'formal method' refers to any specification and design method that incorporates a logically consistent set of rules - for example we have from the following definition provided by J. P. Calvez:

> Formal Models: A system may be specified by a set of statements expressed in a formal language (grammar rule, algebra rule, etc.). [Calvez 1993, pp. 160]

Indeed it could be conceivably argued that (for example) engineering drawings are a type of formal method of design specification - with the graphical constructs prescribed by certain rules, and given precise contextual *meaning*.

However, the interest here is obviously in terms of a *mathematical* formal method with, in this case, the mathematical statements prescribed by fundamental (axiomatic) rules, and given precise meaning - in terms of the systems-theoretic definitions.

3.3 Proof of Correctness. Systems engineering is an engineering science (a 'formal method' for engineering design) and as such there is a need for the particular design artefacts to be proved to have been properly derived and specified. There is, of course, nothing new here - it is a long-accepted part of the practice of recording 'design rationale' (of keeping, for example, engineering log books for the duration of a system's development and operational life). It is therefore by such means that the references, assumptions, mathematical derivations, and particular proofs (of correctness) can be formally recorded.

It is proposed that the proper venues for confirming that 'proof of correctness' has been established are the system design reviews and design audits routinely carried out at various stages of the design process. It is unfortunate that too often these reviews and audits can turn out to be little more that a check that the design task follows some predetermined plan (or 'procedure'). Indeed there is a real danger that design reviews can often involve little more than the 'ticking off' of completed tasks, with little proper regard for providing a system design 'proof of correctness'.

4. System Complexity and Complication

There are two particular properties of 'real world' systems that often cause system designers significant problems. These can be broadly categorised as system *complexity* and system *complication*. An overview of these concepts, in systems-theoretic terms, is given below.

4.1 Systems-theoretic System Complexity. It is proposed that a 'measure' of system complexity can be formally expressed in terms of a component count (Shell, 1999). Although this might at first seem to be a rather 'un-scientific' proposition it is, in fact, an entirely apposite and mathematically rigorous (systems-theoretic) definition. However care must be taken not to misinterpret this description - this is *not* simply a tally of the number of physical parts of a system. The 'components' are, in themselves, formally identifiable as individual (non-trivial) systems, with their own behavioural characteristics. In terms of the given systems-theoretic modelling paradigm - 'system components are component systems'.

$$\text{complexity} \Rightarrow \text{many components} \Rightarrow \text{mode (behaviour) hologenicity}$$

It is in this context that the concept of complexity in terms of the interactions of the behavioural modes of the components, such that an overall system mode of behaviour is exhibited, is formally introduced (i.e. system mode hologenicity).

It is *not* intended to include the issues of design complexity within this paper. Although it forms an essential part of the systems-theoretic systems design process, it has already been comprehensively addressed elsewhere (for example, see Wymore 1993, chapters. 3 and 5). Systems complexity is not specifically relevant to the subject matter of this paper.

4.2 Systems-theoretic System Complication. A *complicated* system is defined as an implementable system solution that, at a particular level of abstraction, is perceived to exhibit many different modes of behaviour. The concept of system *complication* can be formally captured in terms of the number of modes of the system that have to be expressed in order to comprehensively describe the system's functional behaviour.

$$\text{complication} \Rightarrow \text{many (behaviour) modes} \Rightarrow \text{many mode interactions}$$

For the purpose of this study, a *complicated* system is therefore defined as a system whose overall functionality is best described in terms of multiple modes of operation, each of which exhibits a different, and distinct, pattern of behaviour.

For *complicated* systems the principal concern is of ensuring *reachability and sustainability* of the required (multiple) system modes - such that the functional behaviour of the implemented system can provide complete (or at least more extensive) satisfaction of the functional system design, and hence of the input/output requirements.

It is the problems that arise in the design and implementation of *complicated* systems that is specifically addressed within later sections of this paper.

5. System Requirements

5.1 Requirements Specification. In terms of the systems-theoretic method used throughout this paper (i.e. Wymore 1993), the overall system requirement is *defined* in the form of a sextuple:

$$\mathrm{SDR} = (\mathrm{IOR}, \mathrm{TYR}, \mathrm{PR}, \mathrm{CR}, \mathrm{TR}, \mathrm{STR}) \qquad [5.1.1]$$

- where IOR is an input/output requirement, TYR a technology requirement, PR a performance requirement, CR a cost requirement, TR a trade-off requirement, and STR a system test requirement.

Therefore, specifying a particular input/output requirement involves using the IOR formulation to describe, for every possible (identifiable) system input trajectory, a set of eligible output trajectories.

A technology requirement will be specified in terms of a particular set of (complex) system models that describe the technological solutions that are mandated (or, as is more usual, prohibited) in terms of acceptable component choices and design architectures (the TYR). The system designer is therefore constrained to use specific component designs in order to build a resultant system solution.

The performance, cost, and trade-off requirements (PR, CR and TR respectively) are all defined in terms of comparative orders over a set of candidate system solutions, such that preference between any two systems can be expressed. These requirements are therefore generally specified using real-valued functions of the performance and cost figures of merit, determined for the particular candidate set of implementable system solutions.

The system test requirement (STR) is used to specify the tests to be conducted so as to demonstrate design conformance, and real implemented system compliance, to the requirement specification. The extent (coverage) of testing and the test procedures will, of course, be agreed between the various system stakeholders (notably the system operators). It is also noted that in the referenced methodology (Wymore 93) that the need for testing is incorporated into the processes of the development of the candidate designs, and to the identification of the optimal system solution.

The process of identifying acceptable candidate system design solutions will therefore be formally defined in terms of the identification of those 'buildable' system designs that can satisfy the technology requirement - and that can exhibit a mode of behaviour consistent with an elaboration of some functional system design, such that the input/output requirement is satisfied (see section 7).

5.2 Requirements Validation. The specification of requirements and the specification of the compliant system solutions are obviously inextricably linked. All system design possibilities will be directly determined by a detailed specification of requirements - and the requirements will themselves be validated by the identification of at least one implementable system solution. Every effort should therefore be made to complete a detailed, and validated, description of the system-level requirements.

The validation of requirements is more than ensuring the (provably) correct derivation and specification of requirements in terms of some formal language. The danger of relying solely on the (verifiably) correct use of formal method is that the requirements can merely become a 'wish-list' - albeit expressed in a rigorously precise and consistent (formal) way. We need to be sure that the requirements are specifying something that is functionally achievable, and which is technologically buildable (that does not contravene accepted laws of physics, for example).

For example, in a recent paper that explores the process of systems engineering we have the following description of 'requirement validation':

> Validating requirements means that the set of requirements is consistent, that a real-world solution can be built that satisfies the requirements, and that it can be proven that such a system satisfies its requirements. If Systems Engineering discovers that the customer has requested a perpetual-motion machine, the project should be stopped. [Bahill, Dean 1996]

Of course it should be obvious that to validate the requirements involves identifying a *feasible* system solution. That is fundamentally what this activity is all about - to show (in formal terms) that the implementability space (as specified by the input/output *and* technology requirements) is not empty. It is also the case that (subsequently) the candidate system design solutions will also have to be shown to be *feasible* system solutions with respect to these same (validated) requirements criteria.

We must take care not to confuse the *attribution* of 'feasibility' with the *activities* of 'requirements validation' and of 'system designs identification' (for example: EIA 632-1, 1997 sect. 5.3.1.2. 'Define and Validate System Technical Requirements', and subsequently EIA 632-1, sect. 5.3.2. 'Design Definition Activity ... determine potential solution alternatives'). Both of these activities are concerned with identifying *feasible* system solutions (which may, or may not, already exist).

It should be noted that we need only to identify one feasible solution to validate the requirements, and also that that solution may already (physically) exist. However it might not be a *viable* candidate solution - for other than engineering reasons (i.e. legal, ethical, business, etc.).

6. The Satisfaction Of Input/Output Requirement

6.1 Input/output Scope. A major difficulty faced by a system designer is that of ensuring that the input/output requirements adequately describe the 'real' problem - in terms of capturing the input trajectories that the system will be responsive to, and specifying the resultant output trajectories that the system is allowed to produce. It is this problem (and in particular the inclusion of the system behaviours that result from 'non-operational' or 'anomalous' input trajectories) that is specifically addressed by this paper.

To attempt to limit (or ignore) the true scope of the input/output specification courts disaster. For example, we have the following statement within the Ariane 501 Inquiry Board report which (it is suggested) is very pertinent to the arguments contained above:

> The *specification* of the inertial reference system and the tests performed at equipment level did not specifically include the Ariane 5 [flight] trajectory data. Consequently the realignment function was not tested under simulated flight conditions, and the design error was not discovered. [CNES/ESA 1996, sect 3.1.r]

Systems theory (e.g. Wymore 1993, chapter 6) provides for a mathematically rigorous approach to the specification of an input/output requirement (IOR). In essence this involves the use of an 'eligibility' function (ER) such that for every identified system input trajectory (i.e. $f \in ITR$ where $ITR = FNS(TSR,IR)$) a set of eligible output trajectories can be expressed (i.e. $ER(f)$, where $ER(f) \subseteq OTR$ is a sub-set of the output trajectories, and $OTR \subseteq FNS(TSR,OR)$). A rigorous, comprehensive and detailed functional specification is therefore possible using the mathematical elements of the input/output requirement (IOR).

As will be seen later (section 7) it is the system designer's task to identify functional system designs (FSD) which, if presented with an input trajectory f (where $f \in ITR$) will respond with an 'eligible' output trajectory g (where $g \in ER(f)$).

An important advantage of using a formal (mathematical) approach to the definition of the input/output requirements (IOR) is that the method does not impose any limitations with respect to the size of the problem (as is often the case with textual or graphic-based formal methods). The proper use of mathematical systems theory will also ensure that the expression of the input/output requirements can be rigorously, comprehensively and concisely specified.

For example, given that an input trajectory ITR is *defined* as a subset not empty of FNS(TSR, IR) - where TSR = IJS[0, OLR), the life-cycle operation phase is $OLR \in IJS^{+} \cup \{\infty\}$ and where IR is the set of inputs of the IOR, then the following (simple) example provides a *specification* of a complete set of input trajectories - in this case, as constant inputs with values over the range of real numbers of 0.0 to 10.0.

IR = RLS[0.0,10.0] and OLR = ∞

ITR = {f : f ∈ FNS(TSR,IR), there exists p ∈ RLS[0.0,10.0] such that f = CNS(TSR,p)}

Incidentally, it should be noted that in this example an *infinite* number of input trajectories are specified, i.e. such that #(ITR) = ∞

It is important that the IOR should include the required (or permitted) responses of the system to *all* identified inputs. This includes ‘anomalous’ input trajectories that are not desired or expected ‘user’ input demands, but which are physically feasible (such as those arising from fault or failure conditions of other, externally-connected, systems).

System designers, who attempt to determine a set of possible candidate systems designs on the basis of an incomplete or insufficiently detailed definition of the IOR, should therefore accept the risk that they may encounter significant (perhaps insurmountable) difficulties - for reasons that are summarised below.

6.1 Maintenance Of System Candidature. What happens if it is decided to reduce the range of permitted system outputs, after a set of (previously acceptable) candidate designs have been determined?

A deletion from the sets of eligible output trajectories (i.e. to produce a *proper subset* of ER(f)) may mean that a previously acceptable set of candidate functional designs (FSDs) may no longer apply - one (or more) system solutions may not now be allowed. To ensure that the full (original) set of possible FSDs is retained *may* necessitate the deletion of the corresponding input trajectory, or trajectories, from the previously accepted requirement specification (i.e. f ∈ ITR). The alternative might have to be to accept a reduced set of candidate designs.

This conclusion is consequential upon the formal definitions for IOR sub-requirements and super-requirements (Wymore 1993, sects. 6.50, 6.51).

6.2 Extensions Of The IOR. What are the possible consequences if it is decided to make (later) additions to the input/output requirement? How does this effect the current candidate designs?

Because of the inclusive nature of the IOR, no new FSDs should become eligible since they would already be candidate FSDs for the original IOR. This is very important since the inference is that as the IOR definition is made more ‘complete’, then the number of potential candidate FSDs will decrease (or may possibly remain the same) - but should definitely *not* increase in number. Therefore, any addition to the IOR set of input trajectories, together with the corresponding (eligible) output trajectories, *may* disqualify some of the original FSDs.

For example, it could be argued that decisions made on code re-use for Ariane 5 (or, more particularly, on the retention of redundant code and the lack of comprehensive design rationale) was a major contributory factor in the system failure. In particular the Inquiry Board report states that:

> The same requirement [for continued IRS gyro-compass alignment calculations after lift-off] does not apply to Ariane 5, which has a different preparation sequence and it was maintained for commonality reasons, presumably based on the view that, unless proven necessary, it was not wise to make changes in software which worked well on Ariane 4. [CNES/ESA 1996, sect 2.2, para. 10]

Therefore, in the case of Ariane 5 we have a functional system design that was presumably compliant on the basis of an original IOR specification (for Ariane 4) but which was not an acceptable system solution for an IOR extended to encompass the Ariane 5 flight envelope.

6.3 IOR Elaboration. It is not uncommon for initial design studies to be undertaken on the basis of a simplified (un-elaborated) set of input/output requirements. The IOR is correctly ‘scoped’ in terms of the number of input and output ports, their general content, and the relationships between input trajectories and corresponding (eligible) output trajectories. For example, we may begin with an original input set of the form:

$\mathrm{IR}_{\mathrm{original}}$ = {low power, op power, excess power} where $\#(\mathrm{IR}_{\mathrm{original}}) = 3$

However, the IOR is then subsequently elaborated during the course of further systems analysis work. For example, the above input may be elaborated into the form:

$\mathrm{IR}_{\mathrm{elaborated}} = \mathrm{RLS}[0.0,16.0) \cup \mathrm{RLS}[16.0, 32.0) \cup \mathrm{RLS}[32.0, 50.0)$ such that $\#(\mathrm{IR}_{\mathrm{elaborated}}) = \infty$

- where this input homomorphism (i.e. $\mathrm{IR}_{\mathrm{original}} = \mathrm{HI}(\mathrm{IR}_{\mathrm{elaborated}})$) would be reflected in a corresponding change to the eligibility function (ER), so as to maintain the input/output relationships. The output set OR could also, of course, be subject to elaboration. What, then, are the possible consequences of this elaboration of the input/output requirements?

Consider first the case where no functional system designs (FSDs) can be identified so as to satisfy the original IOR. The consequences of a later elaboration of the IOR (consistently undertaken such that $\mathrm{IOR}_{\mathrm{original}} = \mathrm{HIMIO}(\mathrm{IOR}_{\mathrm{elaborated}}, \mathrm{HI}, \mathrm{HO})$) *may* be that viable functional system designs are (now) found to be possible. The danger is, of course, that without the necessary IOR elaboration, work may be prematurely abandoned on the false premise that no system solution is possible.

Consider, also, the alternative scenario where functional system designs (FSDs) *are* identified that satisfy the IOR. In this case the consequences of a later elaboration of the original IOR may be that *no* viable functional system designs are (now) possible. The danger is therefore that much nugatory work might be undertaken before it is realised that no viable system solutions are, in fact, possible (at least with respect to the requirements as defined).

These observations on the possible consequences of IOR 'elaboration' are based on, and consistent with, ideas developed within the referenced systems theory (e.g. Wymore 1993, sects. 6.71 to 6.77).

6.4 IOR 'Completeness'. Is it possible to be sure that the input/output requirement is 'complete'?

Although it is entirely possible to prove that any given input/output requirement is completely specified, it is not possible to prove that the 'complete' input/output requirement has been captured (i.e. to prove that 'the thing is being done right', but not that 'the right thing is being done'). There is no practical way of testing for requirements 'completeness':

> We know of no objective criteria for determining the completeness of requirements.
> [Garcia, Laplue, Rhodes 1995, p 402, "Determining Requirements Completeness"]

We cannot know that the requirement is incomplete, without knowing what the complete requirement is. Unfortunately a part of the requirement can be omitted (from the SDR) without affecting the consistency, correctness or *validity* of the remaining requirement specification.

However this is not an argument for a laissez-faire approach to requirement capture - every effort should be made to complete a detailed description of the 'system-level' input/output requirements *before* any design process is begun (for the reasons given above).

Unfortunately it is common practice to engender an expectation that the system requirements are very likely to be incomplete at the start of the design process. For example: an 'eighty per cent rule' is increasingly being used as the requirements-capture criteria for commencement of each next design iteration. Indeed it is often planned that the customer's requirements will be 'firmed up' as the preliminary designs and prototypes are developed and demonstrated.

It is the author's considered opinion that this use of prototyping (and simulation/emulation) to elicit requirements is an unsound and unsafe approach to requirements capture. The tendency is that the 'problem' then becomes one of making the prototype acceptable to the customer instead of concentrating on the identification of the true (technology-independent) customer requirement - including the essential input/output requirement. To quote from Alan Cooper:

> You can get a better design with pencil and paper and a good methodology than you
> can with any amount of prototyping. [Cooper 1999, pp. 56]

However prototyping can certainly have an important part to play in the system design process - particularly in terms of requirements *validation*, design model verification, and in the comparative assessment (trade-off studies) of the candidate system solutions.

7. The Identification Of Implementable System Designs

The principal objective here is to develop an implementable system design (ISD) in the form of a mathematical description of a technological solution that satisfies a functional requirement (as prescribed by an input/output requirement, IOR).

The technology requirement (TYR) is used to specify the technologies that are mandated (or, as is more usual, prohibited) in terms of acceptable component choices and design architectures. The system designer is therefore constrained to use a sub-set of available component designs to build the resultant system solution.

Obviously there may be more than one (possibly many) implementable system design solutions that satisfy both the input/output requirement and the technology requirement. These candidate system solutions will be subject to trade-off studies using the performance (PR), cost (CR) and trade-off requirements (TR) so as to determine the 'best' (or an optimal) system solution.

An acceptable and implementable system design (ISD), built using the permitted (and possibly mandated) technology (TYR), therefore has to be able to exhibit a mode of behaviour that provides for a functional system design (FSD) solution that will satisfy the IOR, as illustrated in figure 1.

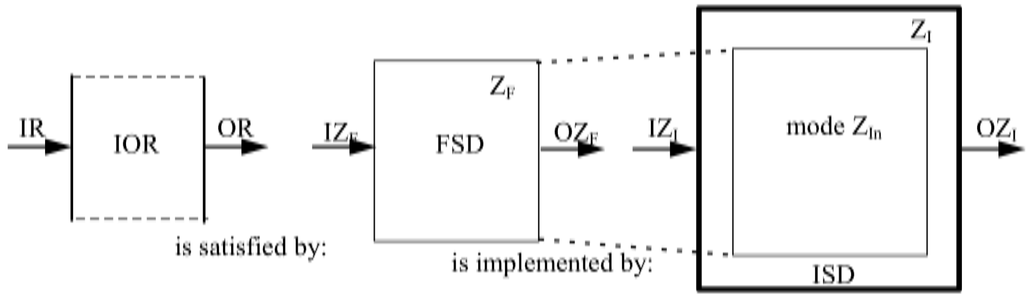

Figure 1 System Design

In system theoretic terms a system (e.g. Z_F) is *implemented* by any buildable system design (e.g. a system Z_I that satisfies the technology requirement TYR) which is capable of exhibiting a mode of behaviour (Z_{in}) such that the implemented system (Z_F) is a homomorphic image of the exhibited system mode. It should be noted that a homomorphic image of a system is a system that exhibits the same functional capability as that system - in this case defining the functional equivalence between Z_{in} and Z_F. A formal description of system implementation is available within the referenced text (i.e. Wymore 1993, sections 5.69 to 5.75)

For the exhibited function (Z_F) to be able to satisfy the input/output requirement it will be necessary to also specify a start condition (system state DSZ) and a time-scale for the function duration (TSZ). Therefore, if Z_I can effectively express a functional system design (FSD) that satisfies the input/output requirement, where $FSD = (Z_F, DSZ_F, TSZ)$, Z_F is the implemented system function, DSZ is the initial state and TSZ is the system time-scale of satisfaction - then Z_I provides for an implementable design solution that satisfies both the input/output (IOR) and the technology requirement (TYR).

[Note that in the referenced systems theoretic method the Z_F, Z_I and Z_{In} system models are all expressed in the form of *discrete* system models - see section 3.1]

The initial (i.e. 'start-up') conditions for the functional design solution are defined as the state DSZ_F, where $DSZ_F \in SZ_F$. In terms of the initial conditions for the implementable system solution (ISD) provided by Z_I , the initial conditions will be a set of states given by the homomorphic relationship:

$$SI \subseteq SZ_{In}, \text{ where } HS(SI) = DSZ_F \qquad [7.0.1]$$

It is required that the sets of inputs and outputs defined for the IOR and the FSD are *identical* to ensure proper conformance to the functional requirements. This will therefore necessitate the homomorphic mapping between Z_I and Z_F being of a form defined by:

$$HI(IZ_I) = IZ_F \text{ , and } IZ_F = ID(IR),$$

$$HO(OZ_I) = OZ_F \text{ , and } OZ_F = ID(OR). \qquad [7.0.2]$$

The implications of the above assertions is that the designer must endeavour to ensure that the proposed real (implementable) system solution is capable of exhibiting behaviour that *completely* satisfies the functional requirement of the IOR (i.e. over the whole of the input/output operational time-scale). Ways of addressing this problem are explored in the remainder of this paper.

8. Invoking System Function

The following mathematical developments are directly concerned with the expression of 'complicated' behaviours of an implementable system design - and of describing how these behaviours can be shown to satisfy the input/output requirement.

The required system function (i.e. the FSD) is provided by means of an exhibited mode of behaviour of the implementable system design (ISD). However this mode of behaviour does not necessarily encompass all possible behaviours of the ISD - there may be other 'non-operational' behaviours that are different, and distinct, from those required to satisfy the particular IOR. These are introduced as a necessary consequence of the technology of the implementable system design (for example 'diagnostic test' or 'calibrate') - or they may be necessary such that the implementable system has the capability of providing other, alternative, functionality (i.e. to serve some 'other purpose'). Additional 'anomalous' modes of behaviour, that are manifest as a result of the physical limitations of the system design (including fault or failure modes), may also be exhibited by the implemented system.

It is therefore important that the possibility of transference between the required FSD mode and these other behavioural modes (i.e. the 'reachability' of one mode from another) is adequately modelled.

The systems theoretic approach can be used to express the system behaviour mode Z_{In} - and this can include the conditions required to both sustain the mode, and for the Z_{In} mode to be reachable. This is developed using the parameterized form of system mode such that: $Z_{In} = SYSMO(Z_I,SMBF_{In},OZ_{In})$, given the system mode behaviour function $SMBF_{In} \in FNS(SZ_{In} \times IZ_{In}, ITZ_I \times TZ^+_I)$, and where $SZ_{In} \subseteq SZ_I$, $IZ_{In} \subseteq IZ_I$, $OZ_{In} \subseteq OZ_I$, $NZ_{In} \subseteq NZ_I$, and $RZ_{In} \subseteq RZ_I$.

The design task is, in effect, to specify the Z_{In} mode states, inputs, outputs, and (most importantly) the system mode behaviour function. The SMBF maps elements of the Cartesian product of mode state and input values (i.e. $(x,p) \in SZ_{In} \times IZ_{In}$) to corresponding elements of system input trajectory and time-index values (i.e. $(f,t) \in ITZ_I \times TZ^+_I$) that are required to sustain that mode. The SMBF function will be specified either by an explicit set of functional pairings or (as would be more usual) by a generic mathematical expression of the required SMBF functional mapping.

8.1 Direct Mode Transfer. The manner in which access to the required behavioural mode (e.g. Z_{In}) is achieved, and the conditions necessary to sustain the mode, are expressed through the identification of particular mode sub-domains. This process is described in formal terms by the development of the following expressions.

If an implementable system design (e.g. Z_I) exhibits a required mode (i.e. Z_{In}), together with some other modes (e.g. Z_{Im}) such that:

$$Z_{In} = SYSMO(Z_I,SMBF_{In},OZ_{In}), \text{ and } Z_{Im} = SYSMO(Z_I,SMBF_{Im},OZ_{Im}), \text{ where}$$

$$SZ_{n\cap m} = SZ_{In} \cap SZ_{Im}, \text{ and } SZ_{n\cap m} \neq \emptyset \qquad [8.1.1]$$

- then this defines all inter-mode transfer states between the two modes. [Note that if there exists a 'mode of a mode', e.g. $Z_{Im} = SYSMO(Z1_{In},SMBF_{Im},OZ_{Im})$, then by definition $SZ_{n\cap m} = SZ_{In} \cap SZ_{Im}$ and $SZ_{n\cap m} \neq \emptyset$ must follow since $SZ_{Im} \subseteq SZ_{In}$.]

If the sets of system states for two modes are disjoint ($SZ_{n\cap m} = \emptyset$) then *direct* transfer between the modes is not possible - however indirect transfer may occur via some other (intermediary) system modes.

8.2 Mode Transfer Reachability. An important design task is to specify the conditions under which inter-mode states are reachable from other parts of the mode domain (if at all). These conditions will be realised as a result of inputs to the system (including, possibly, 'null' inputs) in combination with the current mode states. Given that two system modes are as described above, the mode sub-domain conditions $D_{n \to n|m}$ for entry to the inter-mode states can be defined as follows:

Let $S_{n \to n} \subseteq (SZ_{In} - SZ_{n|m})$, and $D_{n \to n} = S_{n \to n} \times IZ_{In}$, such that

for every $(x,p) \in D_{n \to n}$, if $y = NZ_{In}(x,p)$, and $y \in S_{n \to n}$,

then $D_{n \to n|m} = ((SZ_{In} - SZ_{n|m}) \times IZ_{In}) - D_{n \to n}$ then

$S_{n \to n|m} = \{y$: for every $y = NZ_{In}(x,p)$, where $(x,p) \in D_{n \to n|m}$, and $y \in Sz_{n|m}\}$ *[8.2.1]*

- where $S_{n \to n|m}$ are the accessed inter-mode states on transfer out of Z_{In} to Z_{Im}.

Similarly, the mode sub-domain conditions for re-entry from the inter-mode states are defined as follows:

Let $S_{n|m \to n|m} \subseteq SZ_{n|m}$, and $D_{n|m \to n|m} = S_{n|m \to n|m} \times IZ_{In}$, such that

for every $(x,p) \in D_{n|m \to n|m}$, if $y = NZ_{In}(x,p)$, and $y \in S_{n|m \to n|m}$

then $D_{n|m \to n} = (SZ_{n|m} \times IZ_{In}) - D_{n|m \to n|m}$

$S_{n|m \to n} = \{y$: for every $y = NZ_{In}(x,p)$, where $(x,p) \in D_{n|m \to n}$, and $y \in (SZ_{In} - Sz_{n|m})\}$
[8.2.2]

- where $S_{n|m \to n}$ are the accessed states on transfer back into mode Z_{In}.

The Venn diagram in figure 2 is a simple example of a dual-mode system that illustrates the system design constructs described above. It is used to show the state-space relationships between two system modes. For the purposes of clarity, only mode Z_{In} behaviour conditions are shown - the transfer conditions for mode Z_{Im} could be developed and shown in exactly the same way.

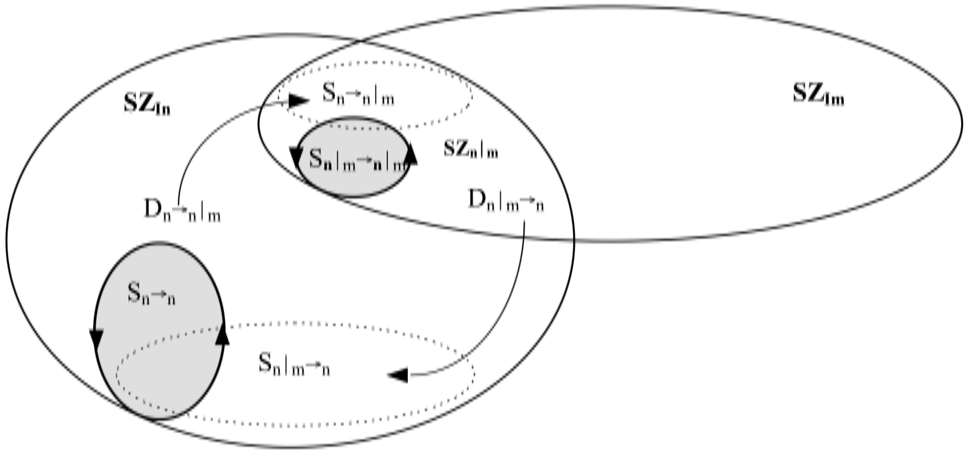

Figure 2 Mode Transference

From the above expressions it is seen that since, by definition:

$S_{n \to n} \subseteq (SZ_{In} - Sz_{n|m})$ and $S_{n|m \to n|m} \subseteq SZ_{n|m}$

then it must be the case that

$S_{n \to n} \cap SZ_{n|m} = \emptyset$ and

$S_{n|m \to n|m} \cap (SZ_{In} - SZ_{n|m}) = \emptyset$ *[8.2.3]*

8.3 Isolating Mode Sub-domains. An isolating mode sub-domain is identified when the condition $D_{n \to n} \neq \emptyset$ is satisfied (and hence, by definition, $S_{n \to n} \neq \emptyset$).

A good example of an isolating mode sub-domain might be found in a system's 'power-up self-test' mode. A detected critical failure would result in the system being prevented from entering an 'operational' mode, regardless of the (operational command) inputs to the system. The system would have to be subject to (perhaps) 'diagnostic test', and/or decommissioned for repair.

8.4 Inevitable Mode Transfers. It may be decided, however, to go further and specify that it would be undesirable to have *any* possibility of a 'sustained' system mode - in other words, to ensure that $S_{n\to n} = \varnothing$ for *any sub-set* of IZ_{In} . How could this be specified? One way would be to broaden the definition for $D_{n\to n}$ through an extension of equation *[8.2.1]* such that:

$IZ_{In} \subseteq IZ_I$

$S_{n\to n} \subseteq (SZ_{In} - SZ_{n|m})$ then, for any

$D_{n\to n} = S_{n\to n} \times I'$, where $I' \subseteq IZ_{In}$ then

for every $(x,p) \in D_{n\to n}$, then $y = NZ_{In}(x,p)$, and $y \in S_{n\to n}$ *[8.4.1]*

Then, in the particular instance where $D_{n\to n} = \varnothing$ (and hence $S_{n\to n} = \varnothing$) there can be no sub-domain of mode Z_{In} which can be sustained - *regardless of the mode input trajectory*. The mode behaviour will be such that it will *inevitably* transfer to the inter-mode states $SZ_{n|m}$.

An inevitable mode transfer may be stipulated where mode transfer is deemed to be mandatory. For example, an 'over-speed' mode of behaviour for an aircraft jet engine would be considered to be extremely hazardous (most certainly a 'worst-case scenario').

The system designers may therefore stipulate an inevitable mode transfer to a safe 'normal operating' mode - which would therefore occur regardless of any further demanded inputs to the engine (see, for example, equation *[13.3.3]*.).

8.5 Other Transfer Possibilities. However it is interesting to note the *possibility* of the relationship: $(S_{n\to n} \cap S_{n|m\to n}) \neq \varnothing$, i.e. an 'isolated' mode sub-domain could be directly reachable from the mode transfer states (as shown in figure 2) - immediately inhibiting further mode transfers from Z_{In} to Z_{Im}.

Note also that it is possible for $D_{n\to n|m} = \varnothing$ which would specify that transfer *out* from mode Z_{In} to Z_{Im} is only possible from the intersecting states $SZ_{n|m}$. It is also possible for $D_{n|m\to n} = \varnothing$ such that once that transfer states $SZ_{n|m}$ are entered, then the Z_{In} mode will remain within those states regardless of the subsequent mode Z_{In} inputs (unless, of course, mode Z_{Im} - or some other intersecting mode is engaged).

9. Function Initiation.

The question remains of how to specify access to the mode 'initiation' states SI (i.e. those states SI defined by: $HS(SI) = DSZ_F$, and $FSD = (Z_F, DSZ_F, TSZ)$, and where FSD satisfies the IOR as defined previously). In other words, how are the 'initiation' states SI to be reachable from regions of the mode domain of Z_{In} ?

In fact, the same form of mathematical expressions as those developed above to define transfer possibilities between modes can be used:

Let $S' \subseteq (SZ_{In} - SI)$, and $D_{S'\to S'} = S' \times IZ_{In}$, such that

for every $(x,p) \in D'$, then $y = NZ_{In}(x,p)$, and $y \in S'$,

then $D_{S'\to SI} = ((SZ_{In} - SI) \times IZ_{In}) - D_{S'\to S'}$ then

$S_{S'\to SI} = \{y$: for every $y = NZ_{In}(x,p)$, where $(x,p) \in D_{S'\to SI}$, and $y \in SI\}$ *[9.0.1]*

- where $S_{S'\to SI}$ defines the immediately accessed 'initiation' states of SI.

Note that it is not necessarily required that SI can be reached only from within Z_{In} - it may be sufficient that the states SI are accessible by direct transfer from some other mode of Z_{I}. Both scenarios are illustrated in figure 3, below.

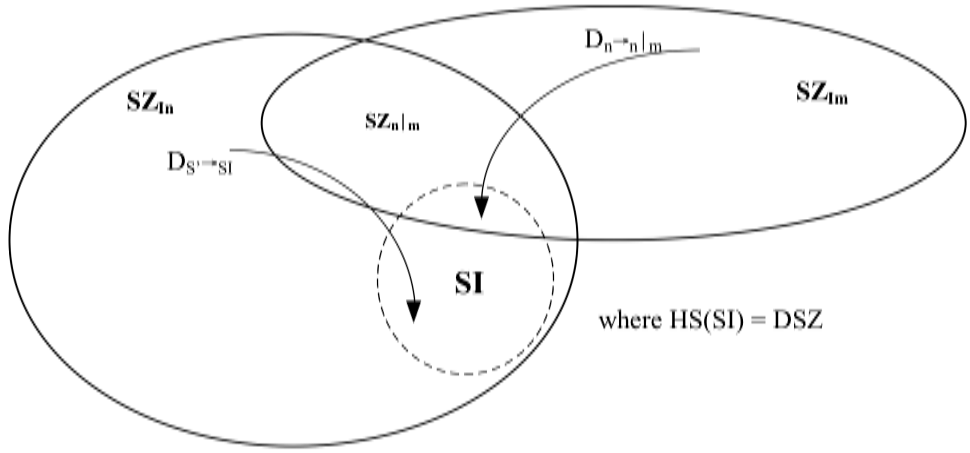

Figure 3 Initiation State Access

10. Functional Partitioning

It is often an expedient design strategy to partition the functional design into ‘function sub-domains’. The partitioning will usually reflect the operational requirements of the system user, as well as (perhaps) the physical characteristics of the implementable system solution.

However, it should be noted that this concept of ‘functional partitioning’ is entirely distinct from ‘functional decomposition’ - which is a consequence of design implementation (i.e. design *complexity*). The issues concerning system complexity, and of ‘functional decomposition’, are outside of the scope of *this* paper.

As with the initial identification of the operational mode Z_{In} to provide satisfaction of the IOR, the partitioning of this implementation mode Z_{In} can be used to express both operational requirements, together with the various physical behaviours of the real, implementable system solution.

10.1 Mode Sub-domains. It is proposed that the implementation of functional partitioning is represented in terms of mode sub-domains of the ISD mode such that, if Z_{In} is developed to represent the required overall mode of behaviour of the implementable system (as defined previously), then a mode $\mathsf{Z}_{\mathsf{In'}}$ can (by definition) be defined as follows:

$\mathsf{Z}_{\mathsf{In'}} = \mathsf{SYSMO}(\mathsf{Z}_{\mathsf{In}},\mathsf{SMBF}_{\mathsf{In'}},\mathsf{OZ}_{\mathsf{In'}})$, i.e. the parameterized system sub-mode, where

$\mathsf{SMBF}_{\mathsf{In'}} \in \mathsf{FNS}(\mathsf{D'}, \mathsf{ITZ}_{\mathsf{I}} \times \mathsf{TZ}_{\mathsf{I}})$, such that

$\mathsf{SMBF}_{\mathsf{In'}} \subset \mathsf{SMBF}_{\mathsf{In}}$, where $\mathsf{D'} \subset (\mathsf{SZ}_{\mathsf{In}} \times \mathsf{IZ}_{\mathsf{In}})$ *[10.1.1]*

This sub-mode will, together with the specified (and unmodified) homomorphic relationship, provide for an implementation of the sub-function $\mathsf{Z}_{\mathsf{F'}}$ - such that $\mathsf{Z}_{\mathsf{F'}}$ is implemented by Z_{I} with respect to the system mode $\mathsf{Z}_{\mathsf{In'}}$ and the homomorphic mappings HS, HI and HO. [Note, therefore, that the overall system mode behaviour function ($\mathsf{SMBF}_{\mathsf{In}}$) and the homomorphic mapping ($\mathsf{HS},\mathsf{HI},\mathsf{HO}$) remain unchanged. The only change required to express the *partitioned* behaviour of $\mathsf{Z}_{\mathsf{In'}}$ is in the specification of $\mathsf{D'}$, the sub-domain of the mode Z_{In}.]

10.2 Sub-domain Reachability. Obviously the concept of 'reachability' can also be applied to the mode 'sub-domains'. The example illustrated in figure 4 shows the interaction of three sub-domains of a mode Z_{In} - i.e. for modes $Z_{In'}$, $Z_{In''}$, and $Z_{In'''}$. This example gives a graphic representation of the different conditions where transfer between the sub-modes may be possible, given the operating conditions.

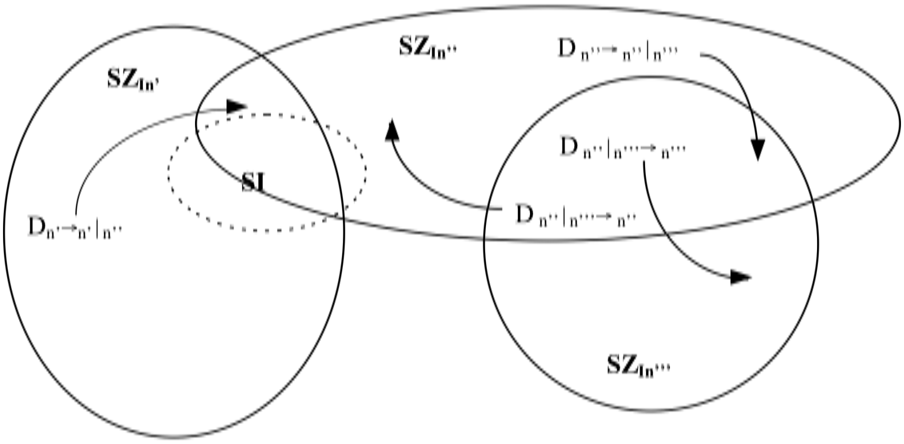
$SZ_{In'}$
$SZ_{In''}$
$D_{n''\rightarrow n''|n'''}$
$D_{n''|n'''\rightarrow n'''}$
SI
$D_{n''|n'''\rightarrow n''}$
$D_{n'\rightarrow n'|n''}$
$SZ_{In'''}$

Figure 4 Sub-mode Interactions

In the example illustrated in figure 4 the initial state space SI is shown as being directly accessible from either sub-mode $Z_{In'}$ or $Z_{In''}$ since (in this case) $SI \subset (SZ_{In'} \cup SZ_{In''})$.

[It should be noted that the use of these mode transfer diagrams are for *illustrative* purposes only - they are *not* an acceptable substitute for the full mathematical definitions.]

11. Functional Design Verification

Previous sections of this paper have investigated methods of providing a comprehensive specification of the input/output requirements, and for specifying functional behaviour for candidate system designs - such that the input/output requirement can be satisfied.

In particular these required system behaviours have been characterised in terms of modes, and mode interactions, of the implementable system solutions. In general terms these broad-based attributes of system behaviour are modelled as follows:

a) System function - as an exhibited mode of the implementable system solution.

b) Mode reachability.

c) Isolated mode behaviours.

d) Inevitable mode transfers.

e) Sub-modes (functional partitioning).

The modelling of these system behaviours therefore provides for the specification of general 'design rules' with respect to the (complicated) functionality of the implementable system solution.

It will be incumbent on the designers of the real, implementable system solution (and the system component designers) to provide proof that their implemented systems will actually exhibit the anticipated (and required) behavioural characteristics - prior to any commitment to manufacture.

This conformation of 'design correctness' will be by means of theoretical analysis and/or by design model experimentation (system simulation/emulation) or by prototype demonstrations.

For example, it may be possible to establish a substantial part of the fundamental behavioural characteristics of a candidate system design by theoretical analysis. Consider the case where the candidate system is implemented as a *continuous* system, and the design mode can be adequately modelled in terms of a linear, time-invariant system. The conventional control techniques of state-space analysis can therefore be usefully used to examine the system mode behavioural characteristics of the system. The system mode is therefore modelled in the standard MIMO form as follows: $\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu}$ and $\mathbf{y} = \mathbf{Cx}$ where $\mathbf{A}$ is the system matrix, $\mathbf{B}$ is the distribution matrix, $\mathbf{C}$ is the output matrix, $\mathbf{x}$ is the state vector, $\mathbf{y}$ is the output vector and $\mathbf{u}$ is the input vector.

For the attribute of ‘controllability’ to be satisfied, where ‘controllability’ is defined in terms of the ability to use system mode input trajectories to cause the system to ‘migrate’ from any arbitrary mode state to any other mode state, via some controlled state trajectory (see, for example Richards 1979, sect. 7.2) then, for complete controllability, the following design conditions need to be satisfied:

$$[\ \mathbf{B}\ :\ \mathbf{AB}\ :\ \ldots\ldots\ :\ \mathbf{A}^{n-1}\,\mathbf{B}\ ]\ \text{must be of rank } n\text{, where } n = \#(\mathbf{x}).$$

A similar analytical procedure may be used to demonstrate complete system controllability for a ‘digitally controlled’ continuous system implementation (see, for example: Franklin and Powell 1980, sect 6.7), or for a purely ‘digital/discrete’ system implementation (see, for example: Szidarovszky and Bahill 1992, sect. 5.2).

[Note that the above analysis is in terms of mathematical modelling of the real, implementable design solutions. This is independent from the decision of using *discrete* modelling for the chosen systems-theoretic method (see section 3.1) which is a consequence of the need to provide for unique mathematical solutions to certain systems-theoretic constructs, and is applicable regardless of the eventual form of system implementation - i.e. continuous, digital, or hybrid.]

If a system satisfies this criteria for ‘complete controllability’ then (by definition) it must be the case that all system mode states are reachable (i.e. $D_{n\text{-}n} = \varnothing$) - there cannot be any ‘isolating’ mode domains (as defined in sections 8.2, 8.3).

Similarly, since the condition of complete controllability is satisfied, then it must be possible to force a return to a previously exhibited system mode state - therefore a ‘controllable’ system mode *cannot* exhibit ‘inevitable’ mode transition (as defined in section 8.4).

It should also be noted that, according to the ‘controllability’ criteria, that a *system* can be completely controllable, but a *system mode* might not be. Similarly, a *system mode* can be completely controllable, even when the *system* is not. For example, we may have the situation of a completely controllable system that exhibits a system mode with an ‘inevitable transition’ sub-domain.

These issues of design validation and verification, in terms of confirmation of mode exhibition, reachability, and sustainability, as outlined in this section, are to be addressed in more detail in a forthcoming paper. In particular the practical use of a broad range of techniques are to be discussed - including theoretical analysis using established engineering science methods, the use of design experimentation/simulation, the *discrete* modelling of ‘continuous’ systems, the analysis of ‘hybrid’ systems, and problems regarding real number modelling.

12. Notation

The following table provides a summary of the systems-theoretic notation used throughout this paper. A more detailed description of these terms is available within the referenced text (i.e. 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 {}
$\Rightarrow$	'implies' ...
×	Cartesian product, for example: $\{1, 2\}\times\{a, b\} = \{ (1, a), (1, b), (2,a), (2, b) \}$
$\in$	is a member of ... (the set), belongs to ...
#	Number of elements of a set (or vector), the set 'size'
BSD	Buildable System Design.
CNS	Constant function, a constant value - over some specified timescale
CR	Cost Requirement, an order over the space of implementable designs
D_n	Mode sub-domain 'n', where $D_n = SZ_{kn} \times IZ_{kn}$ for mode 'n' of system 'k'
$D_{n\to m}$	Mode sub-domain 'n', from which mode sub-domain 'm' is reachable
DSZ	Functional System Design, start state
ER	Eligibility function (input/output requirement)
FNS(A,B)	Set of *all* functions, mapping from domain A over the range B
FSD	Functional System Design
HI	Input Homomorphism
HIMIO	Input/output homomorphism, such that: $HO(ER_{elaborated}(f)) = ER_{original}(HI(f))$, $f \in ITR_{elaborated}$
HO	Output Homomorphism
HIMSY	System homomorphism, such that: $HS(NZ_{elaborated}(x,p)) = NZ_{original}(HS(x), HI(p))$, for every $x \in SZ_{elaborated}$, $p \in IZ_{elaborated}$ and: $HO(RZ_{elaborated}(x)) = RZ_{original}(HS(x))$, $x \in SZ_{elaborated}$
HS	State Homomorphism
ID	Identity Homomorphism
IJS[j,k)	Set of integer numbers: $\{ i : i \geq j , i < k \}$
IOR	Input/output Requirement
IR	Input Requirement, the set of all inputs
ISD	Implementable System Design
ITR	Input Trajectory Requirement
ITZ	System input trajectory, where $ITZ = FNS(TZ, IZ)$
IZ	System Input set
NZ	System next-state function, $NZ \in FNS((SZ \times IZ), SZ)$
OLR	Operational life (time) for the IOR. A non-zero integer value

OR	Output Requirement
OTR	Output Trajectory Requirement
OTZ	System output trajectory, $OTZ(f,x) = RZ(STZ(f,x))$ where $f \in ITZ$, $x \in SZ$
OZ	System Output set
PR	Performance Requirement, an order over the set of implementable designs
RLS[x,y)	Set of real numbers: $\{ w : w \geq x , w < y \}$
RZ	System Read-out function: $RZ \in FNS(SZ, OZ)$
$S_{n \mid m}$	Intersection of states of system modes ‘n’ and ‘m’
$S_{n \to m}$	Mode ‘m’ states, directly accessed from mode ‘n’ states
SDR	System Design Requirement
SI	Implementable system mode state(s), to initiate input/output function exhibition
SMBF	System Mode Behaviour Function $\in FNS((SZ_{mode} \times IZ_{mode}), (ITZ_{host} \times TZ^{+}_{host}))$
STR	System Test Requirement
STZ	System state trajectory, $STZ(f,x) \in FNS(TZ, SZ)$ where $f \in ITZ$, $x \in SZ$
SYSMO	System Mode Parameterisation, such that $Z_{kn} = SYSMO(Z_k, SMBF, OZ_{kn})$
SZ	System States
TR	Trade-off Requirement
TSR	Input/output requirement time-scale set
TSZ	Functional System Design time-scale
TYR	Technology Requirement
TZ	System time-scale
TZ^{+}_{kn}	System mode time index
Z_k	System ‘k’
Z_{kn}	Mode ‘n’ of system ‘k’

13. Case Study - Aircraft Jet Engine Design

The example of the preliminary analysis and design of an aircraft jet engine can be used to demonstrate some of the principles of system specification and design that have been discussed within this paper (note that this example is provided for demonstration purposes only - there is, of course, no intention to produce a 'real' system design solution).

13.1 The Functional Design. We first suppose that the customer requirement is for propulsive thrust to be output in response to pilot throttle demands - and that this is to be expressed within the IOR specification. The inputs to the system may therefore be described in terms of the (pilot) throttle demand IR_{demand}, and the engine outputs as OR_{thrust}.

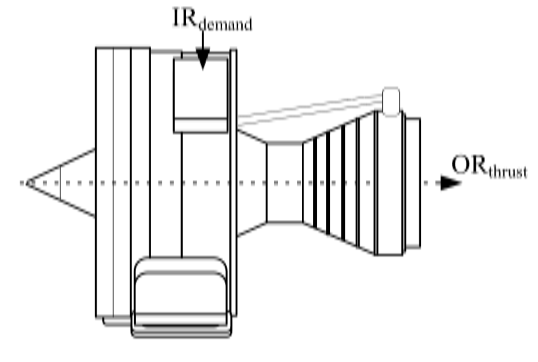

Figure 5 'Jet Engine' Case Study

The overall objective is to identify an implementable system design that will provide a functional design solution (an FSD) that will satisfy the input/output requirement for the engine (see section 6.1), where:

$$IOR = (OLR, IR_{demand}, ITR_{demand}, OR_{thrust}, OTR_{thrust}, ER) \qquad [13.1.1]$$

[Note that an IOR construct, in terms of the operational life (OLR), the input and input trajectory sets (IR, ITR), the output and output trajectory sets (OR, OTR), and the eligibility function (ER) is fully described in the referenced text (Wymore 1993, sect. 6.5).]

13.2 An Implementable Design Solution. An implementable engine design solution is expressed as a mathematical model of a physical solution (i.e. Z_{JET}) that exhibits a mode of behaviour (Z_{NORM}) to provide for an FSD solution that satisfies the IOR (see sect. 7.2), where:

$$Z_{NORM} = SYSMO(Z_{JET}, SMBF_{NORM}, OZ_{NORM}) \text{ where } OZ_{NORM} \subseteq OZ_{JET} \text{ and where}$$

$$SMBF_{NORM} \in FNS(SZ_{NORM} \times IZ_{NORM}, ITZ_{JET} \times TZ^{+}{}_{JET})$$

$$Z_F = HIMSY(Z_{NORM}, HS, HI, HO)$$

$$FSD = (ZF, DSZ, TSZ) \text{ and}$$

$$DSZ = HS(SI), SI \subseteq SZ_{NORM} \text{ and } TSZ = TSR \text{ and}$$

$$IR_{demand} = HI(I1Z_{JET}) \text{ and } OR_{thrust} = HO(O1Z_{JET}) \qquad [13.2.1]$$

The inputs to the implementable engine design solution are extended to include the air mass-flow $I2Z_{JET}$, the provision of external power $I3Z_{JET}$, and the provision of fuel $I4Z_{JET}$. Similarly, the outputs are extended to include an auxiliary power output $O2Z_{JET}$.

The input and output sets for the implementable system design Z_{JET} are therefore given as the following Cartesian products of system inputs and outputs:

$$IZ_{JET} = I1Z_{JET} \times I2Z_{JET} \times I3Z_{JET} \times I4Z_{JET} \text{ and}$$

$$OZ_{JET} = O1Z_{JET} \times O2Z_{JET} \qquad [13.2.2]$$

- and the system input and output trajectories as:

$$ITZ_{JET} = FNS(TZ_{JET}, IZ_{JET}) \text{ and}$$

$$OTZ_{JET}(f,x) = RZ_{JET}(STZ_{JET}(f,x)) \text{ where } f \in ITZ_{JET} \text{ and } x \in SZ_{JET} \qquad [13.2.3]$$

13.3 Design Complication (Engine Behaviours). The engine modes will reflect both the required behaviour to satisfy the IOR input/output requirement, together with other behaviours that result from the implementable system design. For the purpose of this study, therefore, the engine modes are described as: (1) the engine normal operation Z_{NORM} ; (2) and engine stall Z_{STALL} and (3) an engine over-speed $Z_{O/SPEED}$. The actual form for the engine 'normal' mode function ($SMBF_{NORM}$) would, of course, be required to be identified by the system designer. The other principal engine modes Z_{STALL} and $Z_{\text{O/SPEED}}$ are similarly specified.

The designer then specifies the mode sub-domains such that:

$IZ_{\text{O/SPEED}} \subseteq IZ_{\text{JET}}$ then

$D_{\text{O/SPEED}\rightarrow\text{NORM}|\text{O/SPEED}} = (SZ_{\text{O/SPEED}} - SZ_{\text{NORM}|\text{O/SPEED}}) \times IZ_{\text{O/SPEED}} - D_{\text{O/SPEED}\rightarrow\text{O/SPEED}}$

$D_{\text{O/SPEED}\rightarrow\text{NORM}|\text{O/SPEED}} \neq \varnothing$ (and assuming $D_{\text{O/SPEED}\rightarrow\text{O/SPEED}} = \varnothing$), therefore

$SZ_{\text{NORM}|\text{O/SPEED}} \neq \varnothing$ and $SZ_{\text{O/SPEED}} \not\subset SZ_{\text{NORM}}$ and $SZ_{\text{O/SPEED}} \neq SZ_{\text{NORM}}$ *[13.3.1]*

This result is not surprising. It is effectively stipulated (in the third line) that *if* the engine over-speeds, then a capability of reverting to normal operation is to be possible. This analysis confirms that, if there is no 'isolated' over-speed mode sub-domain, then for this particular control to be realisable, the engine over-speed states must *not* be defined as a sub-set of normal operation - which, of course, is intuitively correct.

The designer further stipulates that:

$IZ_{\text{O/SPEED}} \subseteq IZ_{\text{JET}}$

$D_{\text{NORM}|\text{O/SPEED}\rightarrow\text{O/SPEED}} = (SZ_{\text{NORM}|\text{O/SPEED}} \times IZ_{\text{O/SPEED}}) - D_{\text{NORM}|\text{O/SPEED}\rightarrow\text{NORM}|\text{O/SPEED}}$

$D_{\text{NORM}|\text{O/SPEED}\rightarrow\text{O/SPEED}} = \varnothing$ therefore

$(SZ_{\text{NORM}|\text{O/SPEED}} \times IZ_{\text{O/SPEED}}) = D_{\text{NORM}|\text{O/SPEED}\rightarrow\text{NORM}|\text{O/SPEED}}$ since

$SZ_{\text{NORM}|\text{O/SPEED}} \neq \varnothing$ *[13.3.2]*

Again, this result is not surprising. It is stipulated (in the third line) that the implementation is to be such that an over-speed condition is *not* to be reachable from normal operation. This analysis confirms that for this to be possible then any 'over-speed' behaviour is constrained to remain within the intersecting (boundary) states (i.e. $SZ_{\text{NORM}|\text{O/SPEED}}$) - assuming, that is, that the set-spaces for the normal and over-speed modes are not disjoint. Once again these conclusions are intuitively correct.

It may be decided, however, to go further and specify that it would be undesirable to have *any* possibility of a 'sustained' engine over-speed - in other words, to ensure that $S_{\text{O/SPEED}\rightarrow\text{O/SPEED}} = \varnothing$ for *any* sub-set of IZ_{JET} . This can be achieved by use of the 'inevitable transfer' scenario, as discussed in section 8.5. We then have the following model constructs:

$IZ_{\text{O/SPEED}} \subseteq IZ_{\text{JET}}$

$S_{\text{O/SPEED}\rightarrow\text{O/SPEED}} \subseteq (SZ_{\text{O/SPEED}} - SZ_{\text{NORM}|\text{O/SPEED}})$ then, for any

$D_{\text{O/SPEED}\rightarrow\text{O/SPEED}} = S_{\text{O/SPEED}\rightarrow\text{O/SPEED}} \times I'$, where $I' \subseteq IZ_{\text{O/SPEED}}$ then

for every $(x,p) \in D_{\text{O/SPEED}\rightarrow\text{O/SPEED}}$, then

$y = NZ_{\text{O/SPEED}}(x,p)$, and $y \in S_{\text{O/SPEED}\rightarrow\text{O/SPEED}}$ *[13.3.3]*

Then, in this particular instance, the expression $D_{\text{O/SPEED}\rightarrow\text{O/SPEED}} = \varnothing$ is taken to mean that there can be no sub-domain of mode $Z_{\text{O/SPEED}}$ in which an over-speed condition can be sustained (regardless of the engine input trajectory). Engine operation is to be such that it will inevitably revert to a 'safe operating condition' within state-space $SZ_{\text{NORM}|\text{O/SPEED}}$.

The other interactions between the engine modes Z_{NORM} and Z_{STALL} and between Z_{STALL} and $Z_{\text{O/SPEED}}$ may be developed in a similar manner as for the interactions between Z_{NORM} and $Z_{\text{O/SPEED}}$ developed above.

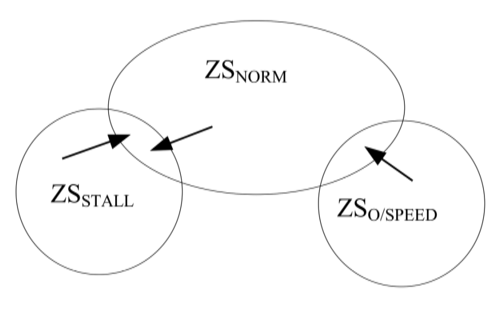

Figure 6 Engine Implementation Modes

13.4 Normal Operation Sub-modes. A Z_{NORM} engine mode may be implemented in the design of a real system solution that provides the function of 'controlled engine thrust' - with a pilot input of throttle position to initiate engine start, modulate the thrust, and to shut-down the engine.

It might therefore be judged to be expedient to represent the provision of this function in terms of sub-modes, such as: (1) engine unassisted run-up or run-down Z_{FREE}; (2) assisted engine run-up Z_{ASST}; (3) engine ignition Z_{IGN}; (4) thrust operation Z_{THRUST}. These sub-modes would therefore be expressed in terms of the mode Z_{NORM}, for example:

$$Z_{THRUST} = SYSMO(Z_{NORM}, SMBF_{THRUST}, OZ_{THRUST}) \text{ and}$$

$$OZ_{THRUST} \subseteq OZ_{NORM} \qquad \textit{[13.4.1]}$$

- with similar expressions for Z_{FREE}, Z_{ASST}, and Z_{IGN}.

These sub-modes of Z_{NORM} are defined in terms of the sub-mode domains and inputs required to sustain them (as per the form defined by equation *[10.1.1]*) as follows:

$$SMBF_{THRUST} \subset SMBF_{NORM} \text{ and } DMN(SMBF_{THRUST}) \subset (SZ_{NORM} \times IZ_{NORM}) \qquad \textit{[13.4.2]}$$

- and again, with similar expressions for $SMBF_{FREE}$, $SMBF_{ASST}$, and $SMBF_{IGN}$.

The interactions between the engine operational sub-modes is defined in terms of the intersection of sub-mode states, together with definitions for sub-domains as follows:

$$SZ_{FREE|ASST} \neq \varnothing,\ SZ_{ASST|IGN} \neq \varnothing,\ SZ_{IGN|THRUST} \neq \varnothing,\ SZ_{THRUST|FREE} \neq \varnothing,$$

$$SZ_{FREE|IGN} \neq \varnothing,\ SZ_{ASST|THRUST} = \varnothing, \text{ and}$$

$$D_{FREE\rightarrow FREE|ASST} \neq \varnothing,\ D_{ASST\rightarrow FREE|ASST} \neq \varnothing,$$

$$D_{ASST\rightarrow ASST|IGN} \neq \varnothing,\ D_{IGN\rightarrow ASST|IGN} \neq \varnothing,$$

$$D_{IGN\rightarrow IGN|THRUST} \neq \varnothing,\ D_{THRUST\rightarrow IGN|THRUST} = \varnothing,$$

$$D_{THRUST\rightarrow THRUST|FREE} \neq \varnothing,\ D_{FREE\rightarrow THRUST|FREE} = \varnothing,$$

$$D_{FREE\rightarrow FREE|IGN} \neq \varnothing,\ D_{IGN\rightarrow FREE|IGN} \neq \varnothing,$$

$$D_{ASST\rightarrow THRUST|ASST} = \varnothing, \text{ and } D_{THRUST\rightarrow THRUST|ASST} = \varnothing \qquad \textit{[13.4.3]}$$

Note that some of the operational mode sub-domains are defined as empty ($\varnothing$). This signifies that there will be no provision for direct transfer from that sub-mode to the other intersecting sub-mode. For example $D_{THRUST\rightarrow IGN|THRUST} = \varnothing$ is used to signify that if the engine is thrusting (i.e. the engine is 'lit') then there is no purpose in (re)engaging an engine ignition mode.

Obviously the above expressions form only the preliminary framework for an implementable system design solution. However they can be usefully used to set the 'ground rules' such that the design can proceed logically - and in accordance with the overall specification of system requirements. Much further work would be required in order that this top-level design description could be fully defined - such that the specification of requirements for the system components could be subsequently produced.

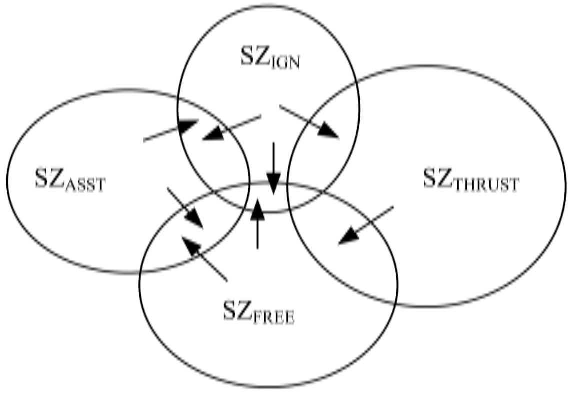

Figure 7 Engine Operational Sub-modes

This 'case-study' demonstrates, in particular, the importance of being able to rigorously define (in terms of mathematical modelling) the real mode behaviour, together with the mode sub-domain conditions under which mode transfers can occur.

14. Summary

Throughout this paper a particular approach to the identification of functional requirements, and to the subsequent design of (compliant) system solutions, has been advanced and recommended - particularly with respect to the formal mathematical modelling of 'complicated' system behaviours such that the satisfaction of the system input/output requirement can be demonstrated. These recommendations can be summarised as follows:

a) Use of a formal method, preferably one based on an established mathematical systems theory, to provide the necessary rigor and discipline to the process of systems specification and design.

b) Emphasis on the completion of a detailed mathematical specification of requirements (and in particular the formulation of the IOR) *before* the search for candidate design solutions is begun.

c) Ensuring that a comprehensive 'high-fidelity' mathematical model of the real-world system solution is produced. The development of this model has to be an integral part of the top-level design process. The model will need to be sufficiently detailed to be able to reproduce all the significant, dynamic behaviours of a real system.

This paper has tended to concentrate on the *practical application* of mathematical systems theory, rather than a detailed examination/explanation of the theory itself. Detailed descriptions of the axiomatic principles behind systems theory are available from the references sources (in particular Fertig and Zapata 1977, and Wymore 1993).

A systems-theoretic approach has therefore been used to develop a set of formal (mathematical) 'rules' which can aid the system designer in the production of compliant, correct, and unambiguous system design solutions. These 'rules' concern the proper determination of the functional requirements of a system, and the design of implementable system solutions (possibly with complicated behaviour characteristics) needed to satisfy those requirements.

The decision to employ systems-theoretic methods within any systems engineering endeavour is not some vacuous exercise to give a 'badge of respectability' to the work (in terms of comparisons with other, specialist, engineering sciences). The reason is much more basic and pragmatic - the use of systems theory can help us to consistently, and efficiently, produce good systems design solutions.

Of course no theory can be totally inclusive or accurate with respect to describing 'real world' phenomena. It is an aid to the understanding of these phenomena, and (in the context of systems theory) it can give us valuable insights into the nature of complex, and complicated, system behaviours. Most importantly, systems theory can help us where our 'common sense' intuition can lead us to incorrect assumptions, and bad design decisions.

It is, perhaps, unfortunate that much of the current debate on the difficulties of providing for successful systems engineering design concentrates exclusively on issues of 'process management'. Relatively little consideration is given (particularly outside of academia) with respect to the development of systems-science methods - or to the practical application of these methods to 'real world' problems. I would argue that it is the use of an established (and practical) systems-science methodology - within the context of a sensible and efficient 'process management' régime - that is the key to the reliable design, development (and effective deployment) of good solutions to difficult 'systems' problems.

References

Bahill A. T., Dean F. F. "Discovering System Requirements", http://www.sie.arizona.edu/sysengr/requirements

Bahill A. T., Dean F. F. "What Is Systems Engineering? A Consensus Of Senior Systems Engineers", INCOSE, Proc. 6th Annual Int. Symp., 1996

Calude C. S. (Ed.) "People & Ideas in Theoretical Computer Science', Springer Verlag Pub., 1998

Calvez J. P., "Embedded Real-Time Systems, A specification and Design Methodology", J. Wiley & Sons, 1993.

CNES/ESA "Board Of Inquiry Report on ARIANE-501", July 1996.

Cooper A., "The Inmates Are Running The Asylum", Macmillan Computer Publishing, 1999

Garcia R. A., LaPlue L., Rhodes R. "A Rigorous Method For Formal Requirements Definition", 5th International Symposium, INCOSE, St. Louis 1995.

EIA Standard 632-1, "Process for Engineering a System, Part 1: Process Characteristics", July 1997.

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

Flight International "Untested Software Is Blamed For Failure Of Ariane 5 Launch", 31st July 1996.

Franklin G. F., Powell J. D. "Digital Control Of Dynamic Systems", Addison-Wesley Publishing, 1980.

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

Korn J., "Problem Of Identity Of Systems Engineering", INCOSE (UK) Symposium, 1997.

Richards R. J., "An Introduction To Dynamics And Control", Longman Group Ltd., 1979.

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

Szidarovszky F., Bahill A. T. "Linear Systems Theory", CRC Press Inc., 1992

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

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 Systems & 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.