Article

Modelling Functional Design in MBSE

James Craig Wrigley [*]

Posted Date: 23 October 2025

doi: 10.20944/preprints202510.1758.v1

Keywords: MBSE; functional decomposition; system modelling; system design; systems theory; DEVS; linear systems theory

Article

Modelling Functional Design in MBSE

James Craig Wrigley

Loughborough University; c.wrigley@lboro.ac.uk

Abstract

This article addresses the problem of placing functional decomposition in Model Based Systems Engineering (MBSE) on a formal mathematical basis in order to support the development of effective mathematical models of digital twins. This has necessitated the re-examination of a number of concepts used within functional decomposition, in-cluding the representation of open systems, the specification of user requirements, the definition of inputs and outputs and the meaning of systems engineering "functions". This resulted in the proposal to consider interactions of the system with its environ-ment rather than input/output requirements and to correctly identify the systems engineering concept of "function" with the mathematical concept of "functional". By doing this we have developed a unified approach to modelling in MBSE that integrates a number of existing approaches, including Wymore's, DEVS and Linear Systems Theory. This results in a mathematical formalism that results in a systems model that traceably satisfies the user requirements.

Keywords: MBSE; functional decomposition; system modelling; system design; systems theory; DEVS; linear systems theory

1. Introduction

Model-Based Systems Engineering (MBSE) is a term coined by Wymore for the use of models to represent problems in Systems Engineering [1]. It promises to improve the application of Systems Engineering in several ways, for example [2]:

- Improved communications
- Increased ability to manage system complexity
- Improved product quality
- Enhanced knowledge capture
- Improved ability to teach and learn Systems Engineering fundamentals

Wymore's work has not been widely adopted by the Systems Engineering community with the result that many Systems Engineering methods are still document-based and qualitative in nature, with a limited mathematical foundation. Salado has embarked on the development of a more rigorous basis for MBSE, starting from the work of Wymore and the DEVS model []. This paper re-examines some of the definitions used in the traditional approaches to system modelling leading to a unified mathematical approach that extends the work of Wymore to be applicable to the modern practice of Systems Engineering, in particular to the development of complete, traceable mathematical models (or "digital twins") of systems under development.

2. Overview of the Functional method in System Design

System Design

We use *system design* to mean the activity of identifying a collection of components that can operate together to satisfy the user requirements. The components may be off-the-shelf or still to be developed. In the latter case, the activity should result in a sufficiently detailed specification to lead to system implementation.

The Functional Method

The Functional method – also called Function Based Systems Engineering [3] is one of the earliest methods for systems engineering and is widely used. In this method,the behaviour of the system, represented by the system function, is decomposed into a set of sub-behaviours, represented by subfunctions, suitably linked. These required sub-behaviours can then be used to select suitable components. The main steps are:

1. Specification of user functional requirements
2. Transformation of user functional requirements into system functional requirements
3. The identification of the system function(s) from system functional requirements
4. Decomposition of system functions
5. Allocation of system subfunctions to components (off-the-shelf or bespoke).

3. The Specification of User Functional Requirements

The meaning of user, stakeholder and system requirements has been widely discussed in the literature, see for example [4]. In document-based systems engineering (DBSE), requirements are expressed in natural language, possibly subject to formal rules. In some texts there are different types of requirements known as Mission , Stakeholder or Operational requirements [3]. In general, these cover statements by the stakeholders in natural language about the purpose, performance and constraints of the system to be developed. We will summarise these as “User Requirements”, recognising that “User” may include a wide range of stakeholders - people and organisations affected by the system, and that the requirements may be of different levels of specificity.

User Functional Requirements express what the Users which to achieve by operating the system. Salado [4] describes the achievements as *outcomes*. They result from the changes that the system makes to its environment as a result of its interactions with the environment. For example an elevator changes the height of its contents. A water fountain changes the drinker of the water from being thirsty to being satisfied.

For this paper we use the definition of a *system* as a collection of *parts* that are related to each other, as in Bertallanfy’s definition of “*a set of elements standing in interrelation*”[5]. INCOSE has developed this to become “*an arrangement of parts or elements that together exhibit behavior or meaning that the individual constituents do not*”[2].

Let us explicitly represent those parts in the operational environment that have the system as a target as $\{\rho_i\}$, and those that are targets of the system as $\{\tau_i\}$. These parts may be systems, humans or may be passive objects. Call this the *Context Set*, $C(\sigma) = \{\rho_i\} \cup \{\tau_i\}$.

The User Functional Requirement can then be framed as the requirements to cause specific changes to the $\{\tau_j\}$ resulting from the effects on the system of the $\{\rho_i\}$. In this paper, the changes are modelled as changes of *state* of the $\{\tau_j\}$ where *state* is used in a general sense. In a physical system, *state* could include measurable quantities such as

1. Temperature
2. Velocity or
3. Height.

In a computer system state could refer to

1. The value of a variable or
2. An abstract state.

In the human context state can be interpreted as the state of a User’s knowledge, or thirsty/satisfied in the above example.

The User Functional Requirement is then expressed as the set of changes of state necessary to achieve the user’s needs, responding to a set of interactions impinging on the system from the environment.

$$UFR = \{(S(\tau_j), S'(\tau_j))\}|\{(\rho_i)\} \tag{2}$$

Here the | symbol is used to mean “given” or “as a result of”.

In practice, the UFR this would be extracted from natural language statements of the user. For example, the primary functional need of an elevator would be "to move items between floors of a building". In this case the τ_j are items to be moved (goods or people) and the changed state represents the people or goods on a different floor of the building. The ρ_i represent any interactions that act on the system, for example the pressing of control buttons.

$UFR = \{(Items\ on\ Floor\ m, Items\ on\ Floor\ n)\}|\{(User\ button\ presses)\}$(3)

The performance requirements can also be interpreted in this way – the performance is measured by quantifying the change, in terms of extent, speed etc. In the elevator example we may add the requirements that the movement between floors is effected with a limit to the accelerations applied, for example.

The expression for UFR given above does not fully describe the User Functional Requirement, since the time-dependence of the changes is not shown. This information is also needed and would need to be elicited from the User through the use of (for example) interaction diagrams. This is discussed later.

The Conversion of User Functional Requirements to System Functional Requirements

Having identified the User Functional Requirements as in the previous section, we now link these to the requirements on the *system* to cause the changes (and respond to the stimuli) that comprise the User Requirement.

Interactions of the System

We start from the IEEE statement [7] that the purpose of a system is "*achieved by the interaction of the system with its operational environment*". The key element here is the word "*interaction*". The *nature* of the *interaction* is essential to the further analysis of the system problem.

Following Lin [6] a *system* can be represented mathematically as a *structure,*

$$S = (M, R) \qquad (1)$$

where M is a set of parts and R represents a set of relations between the *parts*. *Parts* may be *systems. Parts* may be any object in the domain in which the system is defined, for example in a physical system the *parts* are physical items; in a conceptual system the *parts* may be logical functions. In cyber-physical systems *parts* in the physical domain are physical items including the physical item hosting the computer processing. Within that processing part, the behaviour can be abstractly represented by a different type of *part*, for example, Unified Modelling Language (UML) objects.

A system designed with a purpose must *interact* with the environment to achieve that purpose, therefore it must be an *open* system. The previous representation (1) is of a *closed* system. A suitable representation of the required *open* system interacting with the environment is given by the structure:

$$S^+ = (M, R; C, E), \qquad (4)$$

where E is the set of *interactions* between the elements of M and the parts C. $S^+$is not necessarily a system of the form (M, R), because the interactions between the elements of C are not specified.

Let us represent an *interaction* of a system σ as

$$\left(\sigma, \tau, \widetilde{M}(\sigma, \tau)\right) \in E, \qquad (5)$$

where σ is the source of a directed *interaction* and τ is the target - part of the environment. $\widetilde{M}(\sigma, \tau)$ is the "nature" of the *interaction*, which represents the type and strength of the *interaction*. *Interactions* include the following types:

1. The application of force
2. A flow of energy or materials
3. A flow of information or a signal in a cyber system
4. Through a field (e.g. electric, magnetic, gravitational)

These can be quantified and assigned a numerical value called the strength of the *interaction*, for example a flow of electricity measured in Amperes.

The structure, shown in Figure 1, including those entities that σ affects and those it responds to is:

$$S^+ = \left(\sigma, \{\}\,;\; C(\sigma), \left\{\left(\rho_i, \sigma, \tilde{X}(\rho_i, \sigma)\right)\right\} \cup \left\{\left(\sigma, \tau_j, \tilde{Y}(\sigma, \tau_j)\right)\right\}\right) \quad (6)$$

The $\tilde{Y}(\sigma, \tau_j)$[1] and $\tilde{X}(\rho_i, \sigma)$ represent the *interactions* of σ with the *Context Set*. The purpose of the system is achieved through these *interactions*. If the *interactions* with the *Context Set* are to have a measurable effect, they must cause changes to the state of elements of the *Context Set*:

$$\tilde{Y}(\sigma, \tau_j): S_{\tau_j} \xrightarrow{\tilde{Y}} S'_{\tau_j} \quad (7)$$

In this expression, $\tilde{Y}(\sigma, \tau_j)$ is viewed as an operation that changes the state S_{τ_j} of τ_j. The combination of these effects $\tilde{Y}(\sigma, \tau_j)$ cause the changes specified in the User Functional Requirement.

The nature of the *interactions* depend on the domain of the system under consideration. For a computer system, could represent something that updates the state of the operator's knowledge, e.g. a value posted to a display. In a physical system, the *interactions* represent something that causes the required changes in the environment, by changing the physical states of elements of the *Context Set*. For example $\tilde{Y}$ could represent an effect that changes the velocity of an object, and would be implemented through a force. In object-oriented computing, the $\tilde{Y}$ are messages originated by σ that result in a change of state of τ_j. In cyber-physical systems the interface between the physical and processing elements involves *parts* that are *transducers* – converting signals into physical actions and vice versa.

The System requirements for external *interactions* is then captured through

$$\{\tilde{Y}(\sigma, \tau_j)\} \cup \{\tilde{X}(\rho_i, \sigma)\}, \quad (8)$$

analogous to an input/output requirement. Note that this has been derived directly and unambiguously from the User Requirements, by identifying the interactions that can cause the required changes.

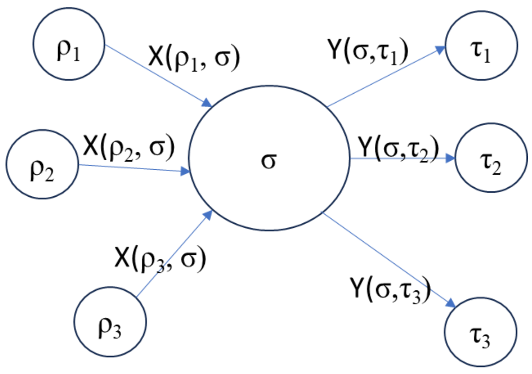

Figure 1. Context diagram.

Comparison with Other Approaches

Buede and Miller [3] view the derivation of system requirements from user requirements as simply a restatement of the requirements from the system perspective. This suggests a form of semantic conversion that requires a degree of linguistic analysis and may be difficult to formalise.

Salado [4] draws the distinction that User Requirements (Stakeholder Needs) have solutions in a space of *outcomes*, while System Requirements have solutions in the space of *functions*. Salado views *outcomes* as changes of state within the set of parts $S_C = \sigma \cup C$. This is similar to the approach adopted here, except that we do not require S_C to be a closed system.

In the traditional functional approach, the nature of the *interaction* is normally that the system produces outputs that affect the environment in such a way as to achieve the desired purpose and also responds to inputs in the desired way. The system problem is then simplified into the problem

[1] The possible time-dependence of $\tilde{Y}$ is suppressed for clarity.

of supporting the required inputs and outputs [1,3]. This is an abstraction that considers the system only up to its boundary. Therefore, the use of the input/output view provides a convenient basis for "factorising away" the *Context Set.* This is not consistent with Salado's definition of *outcomes* [4], since *outcomes* expressly involve the affected elements of the environment.

Here we suggest that the *effects* – changes of state - on and from the environment should be a primary consideration, with the inputs and outputs viewed solely as possible mechanisms for achieving those effects. The designer then has the latitude to select the optimum interaction that can achieve the desired change of state. This change of emphasis allows a more direct linkage of User and System requirements and also allows for a more general system solution, as will be demonstrated.

The removal of the *Context Set* limits the scope of the designer to develop alternative methods of causing the required changes, resulting in a lack of generality in the solution. For example, if the required *outcome* of an alarm system is to deter burglars, this could be achieved by an audible alarm or a flashing light. If the context of "burglars" is removed I may restrict myself unnecessarily to a system that only produces an audio alarm as an output. Another example is a system to heat water. The heating effect could be obtained by the conduction of heat into the water, or by the absorption of RF energy as in a microwave. Without knowing the purpose of the output we cannot optimise the delivery mechanism. If we select the conduction mechanism, this further informs us that the heat must be produced by a heating element that is designed for immersion in water.

This emphasises the importance of retaining references to the *Context Set* where possible, and using an effect-based rather than an output-based design strategy. The System Designer must then consider all possible outputs that create the required changes, and must optimise the delivery mechanism. At the design stage, a knowledge of the application of the system can provide important contextual information relating to the selection of suitable parts, and that the information on the elements of the *Context Set* can be generalised as necessary to avoid dependence of the solution on one particular example of the *Context Set*.

Functions

Traditional functional approach

In traditional Systems Engineering, the strictly mathematical definition of a function as a mapping from elements of a domain to elements of a codomain is broadened to indicate an object that can transform inputs, represented by text descriptors, to outputs labelled in the same way [8]. The intended process is indicated by the name of the function – often a verb/noun pair, as in IDEF0 [9]. In this approach the inputs/outputs and purpose of the function are all implied by text labels, with the human designer required to provide the necessary interpretation in physical terms.

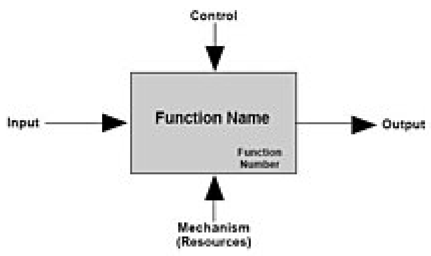

Figure 2. IDEF0 Representation of a function.

Wymore provided a more precise mathematical definition of a logical function using a state machine approach, where the functional definition of a system is given by:

$$Z = (SZ, IZ, OZ, NZ, RZ) \quad (9)$$

where IZ and OZ are sets of input and output states, SZ is the internal state of Z and NZ and RZ are parts of the state machine representing the behaviour of Z. The system relates the "Input trajectory" – the sequence of IZ elements presented to Z, to the output trajectory – the sequence of OZ.

Although a more mathematical formalism, it is sometimes difficult to relate to physical processes, since it relies on the tokenisation of physical items and the use of discrete events to represent the dynamics of the behaviour, which is often more naturally represented by continuous functions.

The mathematical theory presented by Wymore was chosen primarily for pedagogical reasons. According to Wymore's autobiographical account, referenced by Estafan [10] "*restrict[ing] all of MBSE to discrete time and that the basic set of system models would be the set of sequential machines with Moore readout functions. These decisions freed me from topological considerations occasioned by aspects of continuous time to allow me to concentrate on the mathematical structures of the objects of interest to systems engineers*".

In the interactional view presented in this paper, we use *functions* to represent the behaviours of systems or parts in terms of the conversion of interactions acting on the system or part into interactions originating from the system or part. This is different from the standard input/output approach as discussed in the following.

Inputs and Outputs

The previous examples use the standard concepts of *input* and *output.* These concepts are readily understood in the context of computing systems, where typically the *inputs* are the arguments of a function and the *outputs* are the returned values. However, when we try to apply this metaphor to interacting physical systems there are a several subtleties.

Consider the example of a kettle. The conventional statement of the kettle function is that it transforms cold water (the *input*) into boiling water (the *output*). The transformation happens as a result of the kettle acting on the cold water. In this paper, we focus on the interaction of the kettle system (an element of S) with the water (an element of C). Throughout the process, the water remains an element of the environment, C: *input* refers to the initial state of the water and *output* to the final state. The only sense in which the water "goes into" the system is that it is temporarily contained by the system, S: it is never part of the system. This also applies in the elevator example, where the passengers could be considered inputs and outputs of the operation of the system in the given environment, but in the model used in this paper they remain elements outside the system that are acted on by the system (see Figure 3).

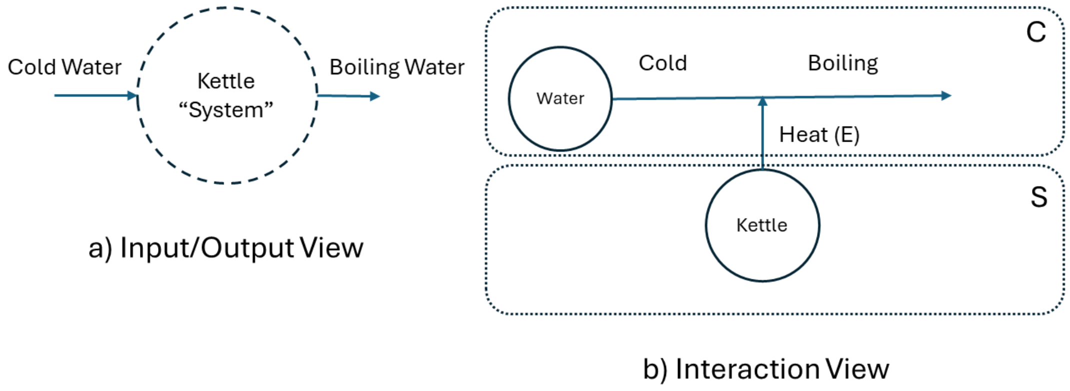

Figure 3. Input/Output vs Interaction.

An *input* continues to be part of the environment that temporarily *interacts* with the system. A way of describing this is that the *input* (target) is *associated with* the system until the *interaction* changes the state of the *input* to an *output* – a *Transformative Function* according to Kossiakoff et al. [11] We model the temporary association by a containment interaction that changes the state of the target from "free" to "contained".

In some cases *inputs* and *outputs* are not evident. The Earth orbits the Sun in the Solar System by virtue of a gravitational *interaction* between them but there is no *input* and *output* in the conventional sense. Another semantic problem is the direction of flow suggested by the terms *input* and *output*. For example, the suction produced by a vacuum cleaner is a flow of air into the system, but since it performs the function of the system, it should perhaps be classed as an output. Similarly for the cooling effect of a refrigerator which actually represents a flow of heat from the content *into* the refrigerator system, so again should be classed as an input. In these cases we again propose that *interaction* is a more appropriate term.

The Identification of the System Function

In the *interaction* view of this paper, we use a function as an expression of the behaviour of a system that describes how the *interactions* originated by the system (*actions*) are generated, as a result of *interactions* originated by external *parts* acting on the system (*reactions*). We need to quantify the *interactions* to place them in a mathematical framework. We have established that the analogue of the system input/output requirement can be represented by

$$\{\tilde{Y}(\sigma,\tau_j)\} \cup \{\tilde{X}(\rho_i,\sigma)\} \tag{10}$$

$\tilde{Y}(\sigma,\tau_j)$ is interpreted as the strength of an *interaction* (e.g. the force applied, the field strength, the heat flow, a number). Let the set $\{\tilde{Y}(\sigma,\tau_j)\}$ be organised as a vector with a time dependence $\boldsymbol{Y}(t)$.

In general, the *actions* at any time may depend on the sequence and history of the *reactions*. In this case we need to transform from entire history of the time-dependent incoming function $\boldsymbol{X}(t)$ into an outgoing function . Mathematically this is a transform or *functional* – a type of function that takes the values of a function over time as its argument and produces a specific value at a given time.

$$\boldsymbol{Y}(t_0) = \boldsymbol{\mathcal{F}}([\boldsymbol{X}(t)]_{t=-\infty}^{t=t_0}) \tag{11}$$

This may include an integral transform of $\boldsymbol{X}(t)$**.**

In the same way, the current state of the part described by the function can only depend on the history of the incoming interactions and the original state:

$$\boldsymbol{S}(t_0) = \boldsymbol{S}([\boldsymbol{X}(t)]_{t=-\infty}^{t=t_0}, \boldsymbol{S}_0) \tag{12}$$

$\boldsymbol{\mathcal{F}}$ is a procedure for converting the evolving input stream $\boldsymbol{X}(t)$ into an output stream $\boldsymbol{Y}(t)$. This can be implemented in a number of ways:

When the system *actions* at time *t* depend only on the *reactions* at time t (i.e. with no history), we can write $\boldsymbol{Y}(t) = \boldsymbol{\mathcal{F}}(\boldsymbol{X}(t))$, where $\boldsymbol{\mathcal{F}}$ is a simple (mathematical) function. As an example, suppose σ is an electrical heating element with input current *X(t)* and output heat *Y(t)*, we can write $Y(t) = X^2(t)R$, where R is the resistance of the element.

For linear systems, with a state vector $\boldsymbol{S}$ we can write [12]

$$\dot{\boldsymbol{S}}(t) = \boldsymbol{A}\boldsymbol{S}(t) + \boldsymbol{B}\boldsymbol{X}(t) \tag{13}$$

$$\boldsymbol{Y}(t) = \boldsymbol{C}\boldsymbol{S}(t) + \boldsymbol{D}\boldsymbol{X}(t) \tag{14}$$

Where the description of the system is event-based, for example in a cyber system, a Finite State Machine may be used to calculate $\boldsymbol{S}(t)$and $\boldsymbol{Y}(t)$. This is the approach selected by Wymore for pedagogical purposes [1] and by Ziegler for the DEVS model [13], where the *actions* and *reactions* are treated as input and output events, for example a function is modelled as:

$$M = <\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{S}, \boldsymbol{S}_0, ta, \delta_{ext}, \delta_{int}, \lambda>, \tag{15}$$

where $\boldsymbol{X}, \boldsymbol{Y}$ are sets of input and output events and $\boldsymbol{S}, \boldsymbol{S}_0, ta, \delta_{ext}, \delta_{int}, \lambda$ are the elements of the state machine. Given a sequence of input events $\boldsymbol{X}(t)$, the state machine generates a sequence of output events $\boldsymbol{Y}(t)$.

The complete expression of the behaviour of σ, shown as $\hat{\sigma}$ is then given by:

$$\hat{\sigma} = (\boldsymbol{X}(t), \boldsymbol{\mathcal{F}}, \boldsymbol{Y}(t)) \tag{16}$$

This expression represents the functional requirements on the System σ. In addition to the *actions*, $\boldsymbol{Y}(t)$, and *reactions*, $\boldsymbol{X}(t)$, the functional $\boldsymbol{\mathcal{F}}$ describes how the *actions* depend on the stream of *reactions* through time. While the *actions* and *reactions* required can be deduced form the User Requirements for external changes of state, the requirement for $\boldsymbol{\mathcal{F}}$ results from the User's Requirements for the sequencing and magnitude of the *actions* and *reactions*.

The Decomposition of the System Function

The aim of functional decomposition is to break the system into subfunctions that represent the behaviour of known *parts* that can then work together to achieve the required system behaviour. The required functional behaviour can be decomposed using a conceptual system for which the *parts*, $\hat{M}$, are *functions* that can ultimately be assigned to real components. In cyber systems designed using an object-oriented approach such as UML, it is more common to use a study of the requirements language within the domain to derive *analysis classes* directly, which can then be refined iteratively into *design classes*. In this cases the *design classes* or object instances form the *parts* in the system model.

Let the *system function* $\hat{\sigma}$ be decomposed into a further set of *subfunctions*, $\{\hat{\sigma}_k\}$. These *functions* must together satisfy the system functional requirements. The functional model, including its *interactions*, is now shown by:

$$\begin{aligned} \widehat{S^+} &= (\hat{M}, R, \hat{C}, E) \\ &= (\{\hat{\sigma}_1 \ldots \hat{\sigma}_k\}, \\ &\{(\hat{\sigma}_\gamma, \hat{\sigma}_\delta, \boldsymbol{W}(\sigma_\gamma, \sigma_\delta))\}, \\ &\{\hat{\rho}_1 \ldots \hat{\rho}_m\} \cup \{\hat{\tau}_1 \ldots \hat{\tau}_n\}, \\ &\{(\hat{\rho}_i, \hat{\sigma}_\alpha, \boldsymbol{X}(\rho_i, \sigma))\} \cup \{(\hat{\sigma}_\beta, \hat{\tau}_j, \boldsymbol{Y}(\sigma, \tau_j))\}) \end{aligned} \tag{17}$$

where R is the set of *interactions* between *subfunctions*, and E is the set of *interactions* with the environment. $\boldsymbol{W}(\sigma_\gamma, \sigma_\delta)$ represent the internal links between the$\{\hat{\sigma}_k\}$.

In this expression there must be a link for each $\hat{\rho}_i$, $(\hat{\rho}_i, \hat{\sigma}_\alpha, \boldsymbol{X}(\rho_i, \sigma))$ to one of the new *functions* that replaces the link $(\hat{\rho}_i, \hat{\sigma}, \boldsymbol{X}(\rho_i, \sigma))$ to the original *system function*. Similarly for each τ_j, $(\hat{\sigma}_\beta, \hat{\tau}_j, \boldsymbol{Y}(\sigma, \tau_j))$ replaces $(\hat{\sigma}, \hat{\tau}_j, \boldsymbol{Y}(\sigma, \tau_j))$. This is a level-balancing or mereological law [14]. In addition, the behaviours of the subfunctions, $\boldsymbol{\mathcal{F}}_\beta$ must combine to produce the required behaviour of the system, $\boldsymbol{\mathcal{F}}$. Proof of this typically requires detailed modelling, for example using the DEVS model [13], a consideration of the system homomorphisms [1] or analysis of the combination of transforms that arise, using (e.g.) Laplace transforms [15].

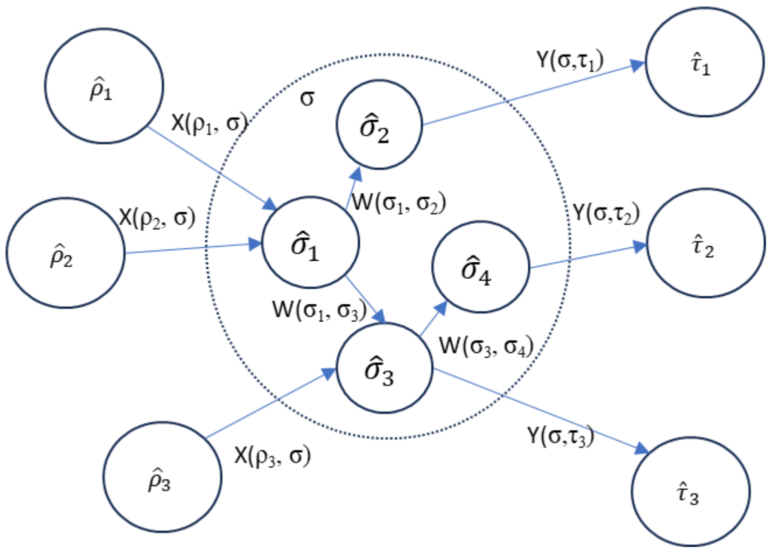

Figure 4. Functional Decomposition.

The original function $\hat{\sigma}$ is replaced entirely by the sub functions $\{\hat{\sigma}_k\}$. and can now be disregarded. There must be no remaining links to or from the original function. The (inner) boundary functions $\hat{\sigma}_\alpha$ and $\hat{\sigma}_\beta$ characterise the *interactions* with external parts. There should be no links from subfunctions to new elements outside σ: these would represent side-effects of the system . Equally, all incoming links must connect to a subfunction. This means that *the interactions required of σ are produced by the boundary subfunctions*. The benefit of the *interaction* approach is that we can use the knowledge of the required effects of the system to give insight into the type of boundary part that can most effectively provide the required *interaction*. If the SOI is an electric kettle, we know that a boundary function must provide heat to the water in the kettle. Knowing that the target is water, we must therefore find an immersible heating element. If we only know that the output is heat, any number of unsuitable parts could be considered.

Further decomposition of any of the resulting subfunctions is possible, and follows the same pattern as the decomposition of the system function, as discussed in [16] . This provides a scale-invariant approach since the *Context Set* for each subfunction is now the union of the remaining subfunctions and the functions from the original *Context Set*:

$$\hat{C}(\hat{\sigma}_k) = \hat{C}(\hat{\sigma}) \cup \{\hat{\sigma}_i \mid i \neq k\} \tag{18}$$

Further decomposition results in subfunctions of the form:

$$\hat{\sigma}_k = (\{Z(\sigma_\mu, \sigma_k)\},\ \mathcal{F}_k,\ \{Z(\sigma_k, \sigma_\nu)\}), \tag{19}$$

where $\hat{\sigma}_\mu, \hat{\sigma}_\nu \subset \hat{C}(\hat{\sigma}_k)$

The end result of the decomposition process is a set of “leaf” subfunctions. The aim is to define “leaf” subfunctions that can be implemented through known physical *parts* or software components.

Salado [4] does not agree with this approach to decomposition: “*The SOI is conceived as an open system that executes a function. Establishing stakeholder needs for the components (and, hence, a problem space of outcomes for the SOI) would require conceiving the SOI as a closed system, which is impossible to do without losing information (removing external inputs and outputs into/from the SOI). Therefore, the SOI’s function is decomposed into a set of functions (those to be executed by its components) that cannot be converted into outcomes.*”

Our position is that the *system* plus its *Context Set* ($S_C = \sigma \cup C$) - equivalent to the Context System for the SOI in Salado’s terminology - does *not* necessarily need to be conceived of as a *closed* system (or even a system), because elements of the *Context Set* may further interact with other elements that are outside C. In effect, the *outcome* required by the SOI Stakeholder may be an intermediate *outcome*. The change in state within C may result in a further *interaction* outside S_C. If I boil water in a “kettle SOI”, the resulting boiling water can further interact with (say) coffee in a “coffee-making SOI” to form brewed coffee. There are two (nested) *outcomes* – the water is boiled (1) and the coffee is made (2).

Further, we do not need to establish Stakeholder Needs (User Requirements) for each component, since we already know how the components will respond to and generate *interactions* with other components (from their specifications – see the next section). We do know that the boundary functions are subject to the User Functional Requirements since they are the functions that respond to and generate the *interactions* that satisfy the User Functional Requirements. The boundary functions *must* produce the *outcomes* required of the system.

Allocation of System Subfunctions to Components

The functional model developed up to this point is a conceptual or logical model of the required system, expressed as *functions* – the required behaviours of *parts*. Ideally we should be able to find or design parts that meet the functional requirements. However, it must be recognised that the functional decomposition cannot be completely arbitrary, it must be guided by the designers knowledge of feasible (buildable) parts. In Wymore’s tricotyledon theory [1], the system must be buildable from the set of available parts and must also meet the requirement of the functional model. We anticipate that there is a set of available *parts* with known *functions.* For automation of the design process, the functions of known parts could be catalogued to support a search function. Where no *part* exists with the required *function,* the *function* can be used as a basis for the specification of a new part, or potentially further decomposition into a set of new *parts*.

Assume we have a set of pre-defined *parts* with *functions*:

$$(\{X(\sigma_M, \sigma_A)\},\ \mathcal{F}_A,\ \{Y(\sigma_A, \sigma_N)\}), \tag{20}$$

For each $\hat{\sigma}_\alpha$ in the functional model we need to find σ_A such that:

$$\{X(\sigma_M, \sigma_A)\} \supseteq \{X(\sigma_\mu, \sigma_\alpha)\} \tag{21}$$

$$\{Y(\sigma_A, \sigma_N)\} \supseteq \{Y(\sigma_\alpha, \sigma_\nu)\} \tag{22}$$

and $\mathcal{F}_A \approx \mathcal{F}_\alpha$ (23)

In other words, the required part must provide at least the *interactions* in the functional model. In addition the behaviour of the part must be equivalent to the behaviour specified by the functional model.

In these expressions, σ_M and σ_N represent classes of parts with which σ_A has a known *interaction*. These are selected as representative of the modelled *parts* σ_μ and σ_υ. For example, assuming the SOI is a hi-fi system, the functional analysis will result in something that amplifies the audio signal and something that converts the signal to sound energy. Amplifiers are specified according to their performance when paired with speakers of a particular impedance.

Where the part has additional (unwanted) *interactions*, it is necessary to find a way of either suppressing or absorbing them. For example, many electrical components generate unwanted heat, which make it necessary to provide cooling. Electrical components may also require electrical power as an input. The behaviour of the part must be such that the relationship between its *interactions* matches that required in the functional model to the extent that it is necessary to result in the same overall system behaviour.

Design Strategy

The observation that the boundary functions provide the behaviour required of the system can be used to support a composition approach to design where suitable parts are identified that produce the boundary *interactions* and the system is then constructed to link those parts together – this is known as functional *composition*.

Coffee Machine Example

The following provides a simple example of the application of this approach to the design of a coffee machine.

Specification of User Functional Requirements

Consider the example of a coffee machine. The User Requirement for changes to the environment can be expressed as:

$$UFR = \left\{(Free_{water}, Coffee\ Solution_{water}\ )|\{Off_{switch,}On_{switch}\}\right\}$$

The user requirement can be refined at the user level by specifying individual steps in the process:

$$UFR = \left\{\begin{matrix}(Free_{water}, Contained_{water}), \\ (Contained_{water}, Heated_{water}), \\ (Heated_{water}, Tranferred_{water}\ ), \\ (Tranferred_{water}, Coffee\ Solution_{water})\end{matrix}\right\}|\{Off_{switch,}On_{switch}\}(24)$$

The changes in the state of the water can be shown (for example in Capella [17]) as in Figure 5:

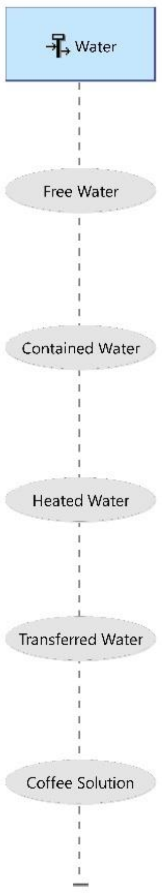

Figure 5. User Requirements for Changes of State.

The Conversion of User Functional Requirements to System Functional Requirements

We therefore require that the system supports the *actions* that cause these changes of state. Specifically we require:

$$Y_1: Free_{water} \rightarrow Contained_{water} \tag{25}$$
$$Y_2 : Contained_{water} \rightarrow Heated_{water} \tag{26}$$
$$Y_3: Heated_{water} \rightarrow Tranferred_{water} \tag{27}$$
$$Y_4: Free_{coffee} \rightarrow Contained_{coffee} \tag{28}$$
$$Y_5: Tranferred_{water} \rightarrow \ Coffee\ Solution_{water} \tag{29}$$

and has a *reaction* to

$$X: Off_{switch,} \rightarrow On_{switch} \tag{30}$$

As illustrated in Figure 6:

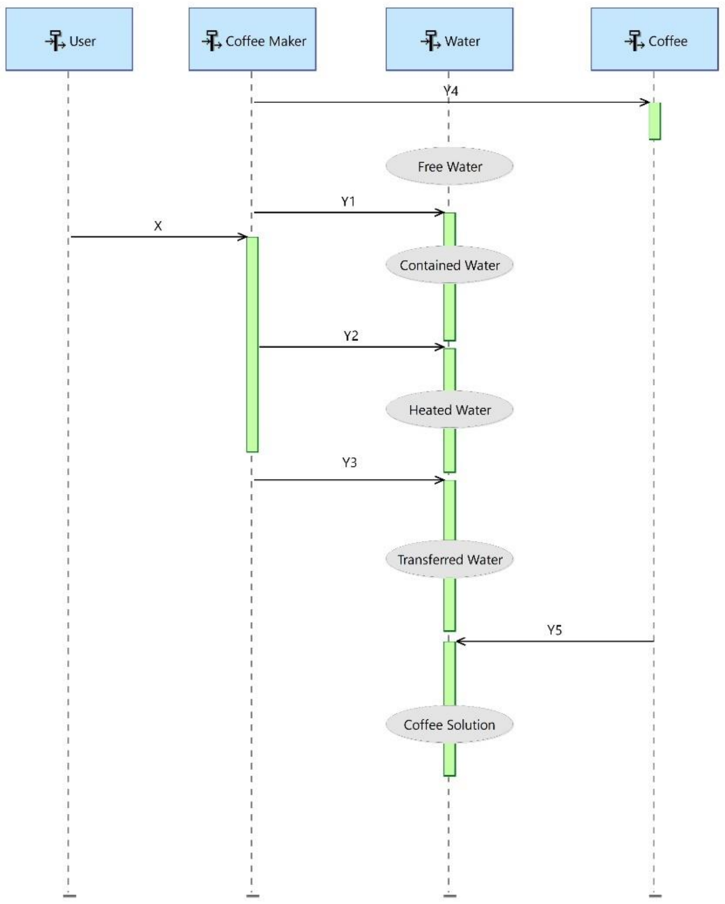

Figure 6. Interactions of Coffee Maker.

The Identification of the System Function from the System Functional Requirements

This results in the system function $(\boldsymbol{X}(t), \boldsymbol{\mathcal{F}}, \boldsymbol{Y}(t))$ illustrated in Figure 6. Here $\boldsymbol{\mathcal{F}}$ is the functional describing how the *actions* $\boldsymbol{Y}(t)$ depend on the *reaction* $X(t)$.

We do not have an explicit mathematical form for $\boldsymbol{\mathcal{F}}$ but we can deduce the sequencing from the User Requirements for the coffee-making process, and that the process should be initiated by the on/off switch.

Now we proceed to quantify the *actions* that the system must produce. From the User Requirement we can determine that the system must produce the *action* Y_1 that changes the state of the Water from *Free* to *Contained*

$$Y_1 : Free_{water} \rightarrow Contained_{water} \tag{31}$$

This can trivially be modelled by a state machine defined by

$$(incoming\ Y, current\ state) \rightarrow next\ state \tag{32}$$

as:

$$(Y_1, Free_{water}) \rightarrow Contained_{water} \tag{33}$$

$$(Y_1, Contained_{water}) \rightarrow Contained_{water} \tag{34}$$

The system must also produce the interaction that causes the following change of state:

$$Y_2 : Contained_{water} \rightarrow Heated_{water} \tag{35}$$

The water is modelled as a linear system $\theta(t) = \boldsymbol{\mathcal{S}}([Y_2(t)]_{t=0}^{t=t}, \theta_0)$, with the state variable θ representing the temperature of the water. The water is *Heated* when θ reaches a user-defined value. We now consider options for Y_2, an interaction that changes the temperature of the water. As previously discussed, this could be through microwave heating or a conduction or convection solution.

We find that if Y_2 is a flow of heat that acts on the water then:

$$\dot{\theta} = HY_2 \quad (36)$$

Where H is the heat capacity of the water. If we integrate this we find:

$$\theta = \int_0^t HY_2\, dt, \quad (37)$$

illustrating that indeed θ is a functional of Y_2. This allows us to quantify the required heat flow, based on the mass of water to be heated and the time required. We continue using a similar approach for Y_3 and Y_4.

We now consider Y_5 where coffee acts on the heated water coffee to dissolve in it. The water is now modelled as another linear system $(Y_5, \mathcal{S}, C)$, where the state variable C represents the concentration of the coffee.

C may be modelled as

$$\dot{C} = k(C_{max} - C) \quad (38)$$

So that

$$C = C_{max}(1 - e^{-kt}) \quad (39)$$

This allows us to quantify further the brewing time for the coffee.

Finally we consider the incoming interaction X, where the user controls the operation of the coffee maker by pushing a button. We propose a simple state machine for this:

$$\left(Off_{switch}, Active_{coffe\ maker}\right) \rightarrow Inactive_{coffe\ maker} \quad (40)$$

$$\left(On_{switch}, Inactive_{coffe\ maker}\right) \rightarrow Active_{coffe\ maker} \quad (41)$$

Decomposition of the System Function

We now require to decompose the system into *parts*, with the quantitative knowledge of the changes that the boundary parts are required to produce.

The functional requirements of the boundary parts are then:

1. Y_1 causes the water to be contained
2. Y_2 – causes the water to be heated
3. Y_3 – causes the water to be transferred (to mix with the coffee)
4. Y_4 – causes the coffee to be contained
5. X – causes the coffee machine to switch between active and inactive

An initial trial decomposition is shown in Figure 7, where each (boundary) interaction has been assigned to a different part.

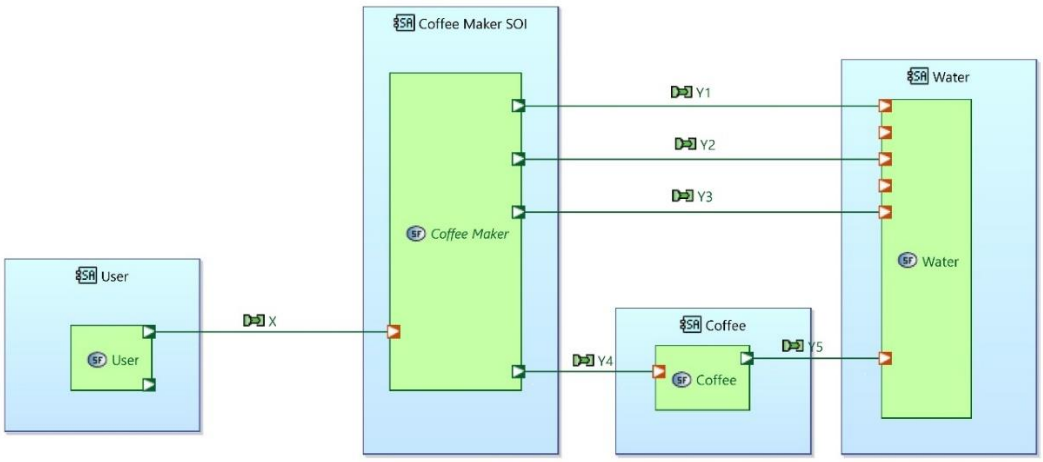

Figure 7. System Function for Coffee Maker.

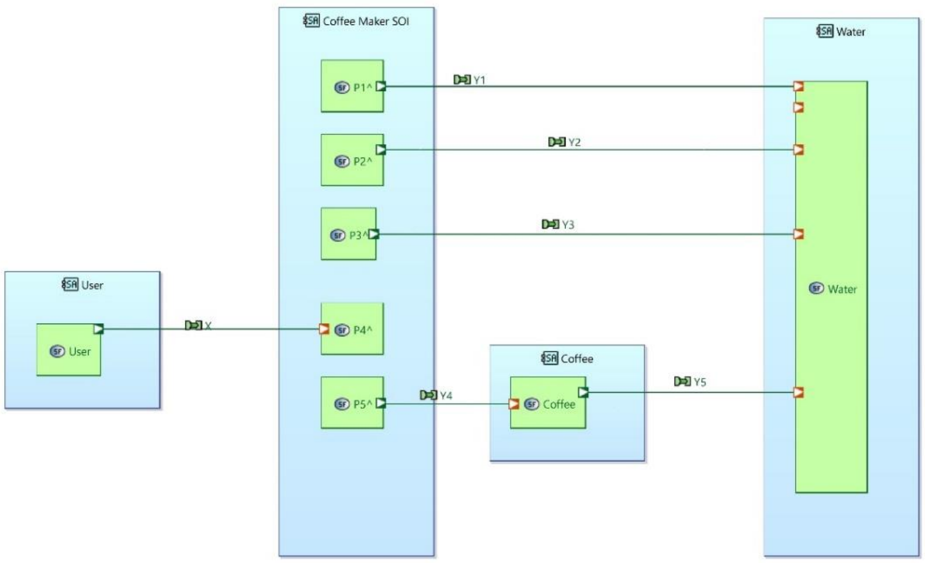

Figure 8. Trial Decomposition.

This results in the following subfunctions:

$$\hat{P}_1 = (R_1, \mathcal{F}_1, Y_1), \tag{42}$$
$$\hat{P}_2 = (R_2, \mathcal{F}_2, Y_2), \tag{43}$$
$$\hat{P}_3 = (R_3, \mathcal{F}_3, Y_3), \tag{44}$$
$$\hat{P}_4 = (X, \mathcal{F}_4, A_4), \tag{45}$$
$$\hat{P}_5 = (R_5, \mathcal{F}_5, Y_4), \tag{46}$$

Here the Y_m are the previously quantified *actions*, A_4 is an unspecified action, X is the specified *reaction*, the R_n are unknown *reactions*.

Allocation of System Subfunctions to Components

We now attempt to find parts (components) with these functional behaviours:

$\hat{P}_1$ supports an *action* to contain water. Any suitable container has this *action*. The size of the container can be determined by refining the user requirement to specify the quantity of coffee to be produced. R_1 is the *reaction* that supports this containment *action* – this is the force supporting the weight of the container provided by the surface it is resting on.

$\hat{P}_2$ acts on the contained water through Y_2 to heat it up. We consider possible off the shelf parts that provide the required interaction – i.e. that cause the water to be heated. Possible solutions include an electric heating coil, a gas heater or a microwave. For simplicity we select an electrical heating element. In this case we know that R_2 is a potential difference applied to the element and therefore that:

$$Y_2 = \frac{{R_2}^2}{R_e} \tag{47}$$

where R_e is the resistance of the element.

$\hat{P}_3$ produces an action on the water to transfer it to be in contact with the coffee. This can be achieved in several ways, for example by using steam pressure to move the water, or by just pouring the water, or perhaps the use of a pump.

$\hat{P}_4$ reacts to a lever being pressed by making the coffee machine active. To embody this we need a component that takes as an input an electrical potential, V and acts to provide a potential of 0 when the lever is up and , V when it is down.. When connected to $\hat{P}_2$ this will cause the water to heat up. The obvious candidate is a simple switch.

$\hat{P}_5$ is simply the behaviour of a container for the coffee.

This analysis outlines the proposed approach. Many details remain to be further analysed, such as the mechanism for controlling the temperature of the water used to brew the coffee. These fundamentals are illustrated:

1. The statement of User Requirements in terms of changes to the environment that the system is required to accomplish,
2. The identification of the *interactions* that are required to cause those changes, based on a quantitative analysis of the behaviour of the targets' *reaction* to the system's *action*.
3. The identification of components able to provide the required *actions* and *reactions*

Discussion

In this paper, we re-evaluated a number of standard assumptions regarding the process of functional decomposition and have proposed a number of departures from the existing approach.

In particular, we have proposed to base the analysis on the changes to the environment that the system is required to make and the system *interactions* that cause those changes. This gives a more precise perspective than the simpler input/output view and allows us to accommodate interactions such as those involving fields that cannot be expressed in input/output terms.

Since the effects of the *interactions* apply to elements outside the system, we are proposing that the restriction of the analysis to the system boundary be reconsidered. We have given a number of examples where the reduction to a simple input/output view can force premature design decisions. The retention of information about the system's immediate environment allows a better optimisation of the interaction between the system and the environment.

We also addressed the concept of *function* in Systems Engineering which at present is defined rather loosely. We have proposed a precise mathematical definition as a *functional* that relates the history of the *reactions* of the system to its *actions*. This unifies various existing approaches such as Wymore's, DEVS and Linear Systems Theory. We have shown how Linear Systems Theory integrates naturally with our approach and have given examples of its application. We have found that the system's external *interactions* take effect through boundary functions, and that the selection of parts to fulfil those functions is aided by a knowledge of the required effects of the *actions* on the environment resulting from the *reactions* to the environment.

Conclusion

We have developed a formal mathematical model of the process of functional decomposition for physical systems and have shown how it can provide a link between User Functional Requirements and the selection of parts which can form a system that satisfies the User Functional Requirements.

The model also shows how to incorporate DEVS modelling and Linear Systems Theory as a natural consequence of the analysis process. The design process follows a series of logical analytical steps and therefore should be straightforward to automate, resulting in a digital twin of the system to be constructed.

We conclude that the *interaction* view proposed by this paper serves to unify and formalise and extend a number of existing approaches to functional decomposition, leading eventually to the ability to fully automate the process of developing a digital twin of the required system and ultimately the construction of the system itself.

References

1. A. W. Wymore, *Model-Based Systems Engineering*. 2018. doi: 10.1201/9780203746936.
2. INCOSE, *Systems Engineering Handbook*, 4th ed. San Diego: Wiley, 2015.
3. 'Overview of the Systems Engineering Design Process', in *The Engineering Design of Systems*, John Wiley & Sons, Ltd, 2009, ch. 2, pp. 49–72. doi: https://doi.org/10.1002/9780470413791.ch2.
4. A. Salado, 'A systems-theoretic articulation of stakeholder needs and system requirements', *Systems Engineering*, vol. 24, no. 2, 2021, doi: 10.1002/sys.21568.

5. L. von Bertallanfy, *General system theory: Essays on its foundation and development*. 1968.
6. Y. Lin, *General Systems Theory: A mathematical approach*, vol. 12. Springer Science and Business Media, 2006.
7. ISO, 'ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering', 2018.
8. 'Functional Architecture Development', in *The Engineering Design of Systems*, John Wiley & Sons, Ltd, 2009, ch. 7, pp. 211–251. doi: https://doi.org/10.1002/9780470413791.ch7.
9. 'IDEFØ – Function Modeling Method – IDEF'. Accessed: Aug. 14, 2025. [Online]. Available: https://www.idef.com/idefo-function_modeling_method/
10. J. A. Estefan, 'Survey of Model-Based Systems Engineering (MBSE) Methodologies', 2008.
11. A. Kossiakoff, W. N. Sweet, S. J. Seymour, and S. M. Biemer, *Systems Engineering Principles and Practice: Second Edition*. 2011. doi: 10.1002/9781118001028.
12. R. L. Williams and D. A. Lawrence, 'State-Space Fundamentals', in *Linear State-Space Control Systems*, 2007, pp. 48–107. doi: https://doi.org/10.1002/9780470117873.ch2.
13. B. P. Zeigler, 'Hierarchical, Modular Discrete-Event Modelling in an Object-Oriented Environment', *Simulation*, vol. 49, no. 5, 1987, doi: 10.1177/003754978704900506.
14. A. Yang and W. Marquardt, 'An ontological conceptualization of multiscale models', *Comput Chem Eng*, vol. 33, no. 4, 2009, doi: 10.1016/j.compchemeng.2008.11.015.
15. B. M. Chen, Z. Lin, and Y. Shamash, *Linear Systems Theory: A Structural Decomposition Approach*. 2004.
16. J. Craig Wrigley, 'Requirements Decomposition using a Graphical method', in *SOSE 2020 - IEEE 15th International Conference of System of Systems Engineering, Proceedings*, 2020. doi: 10.1109/SoSE50414.2020.9130462.
17. 'Capella | Open Source MBSE Tool'. Accessed: Aug. 04, 2025. [Online]. Available: https://mbse-capella.org/