Interactive Functional System Modelling II:

Toward an Executable Modelling of Emergence

David Slater and Rees Hill

Abstract

Complex socio-technical systems often produce outcomes that cannot be explained by decomposing them into independent components or linear event sequences. FRAM addressed this problem by modelling systems as aspect-mediated configurations of coupled functions, but further execution semantics are required if such interactions are to be explored as evolving system state rather than as static dependency structures. This paper proposes Interactive Functional System Modelling (IFSM) as an executable modelling layer for interaction-centred systems analysis. IFSM represents systems as coupled constellations of bounded, stateful functional abstractions whose behaviour emerges through repeated signal-propagation cycles. The paper introduces the Turon as the executable modelling primitive of IFSM: a bounded, aspect-aware functional process that receives typed metadata-bearing signals, evaluates guard conditions, updates internal state and emits outputs that influence other Turon. The contribution of IFSM is to make assumptions about functional interaction explicit, executable, traceable and revisable. The paper defines the IFSM paradigm, outlines its execution semantics, illustrates its use through a simple execution trace, and discusses implications for safety analysis, resilience engineering, MBSE/SysML integration and interaction-centred digital twins.

Keywords: *FRAM; Interaction System Modelling; functional resonance; executable modelling; emergence; resilience engineering; digital twins; socio-technical systems; MBSE; SysML.*

1. Introduction

Complex socio-technical systems rarely behave as collections of independent components. Their behaviour emerges from interactions among technical processes, human adjustments, organisational controls, information flows, resource constraints and timing conditions. This is evident in aviation, healthcare, process safety, autonomous systems and critical infrastructure, where important outcomes often arise not from the failure of one element, but from the way multiple functioning elements influence one another under changing conditions.

This creates a persistent problem for system modelling. Many established methods in reliability engineering and accident analysis are powerful when the task is to represent defined failures, event sequences, barrier functions or component failure probabilities. They are less well suited to situations in which the system-level outcome emerges from interactions among processes that are each locally plausible. This problem is central to the development of FRAM, which was introduced to address disproportionate, non-linear outcomes that conventional accident models based on simple cause-effect reasoning struggled to explain (Hollnagel, 2012; Patriarca et al., 2018).

FRAM addressed this problem by shifting the modelling unit from components to functions. A FRAM model represents a system as an **aspect-mediated configuration of activities** related through Inputs, Outputs, Preconditions, Resources, Controls and Time. These relationships should not be mistaken for formal graph edges. They arise because the output of one function

may become a condition, resource, control, precondition or timing influence for another. The analytical question changes from “what failed?” to “how was performance produced, and how did variability propagate?” FRAM is therefore not simply an accident model, but a method for representing functional dependency, adjustment and resonance.

Important developments have already been made in giving FRAM models more formal, interpretable and executable properties. The FRAM Model Interpreter can be used to interpret a FRAM model, evaluate the consequences of specified couplings and explore how potential couplings are realised as actual couplings under particular conditions. Metadata-based extensions have also shown how FRAM functions can be enriched with numerical and semantic attributes so that quantitative behaviours can be modelled and explored (Hill and Slater, 2024). These developments are important because they show that FRAM should not be treated merely as a qualitative diagramming method.

Nevertheless, a further methodological step is needed. Digital infrastructures, autonomous systems, cyber-physical operations and data-rich organisations increasingly require models that can explore how interactions unfold over time: how signals propagate, how feedback stabilises or destabilises performance, how local changes alter global behaviour, and how systems move between operating regimes. The challenge is therefore not simply to make FRAM quantitative by adding numerical values to a diagram. That risks preserving a static model while giving it a false appearance of precision. The deeper challenge is to make interaction itself executable.

Slater has proposed **Interactive System Modelling (IFSM)** (Slater , 2026), as one response to that challenge. IFSM is a modelling paradigm in which systems are represented as **coupled ensembles of bounded, stateful functional abstractions** whose interactions generate emergent system behaviour over repeated propagation cycles. It builds on FRAM’s interaction-based insight by adding an explicit executable layer for stateful signal propagation.

The paper makes four contributions. First, it defines IFSM as an executable interaction layer for systems analysis. Second, it introduces the Turon as a stateful, aspect-aware representation of a functional process. Third, it defines typed metadata-bearing signals as the coupling mechanism between Turons. Fourth, it specifies a cycle-based execution semantics for propagating state through an **aspect-mediated arrangement of coupled functions** and discusses its relevance for safety analysis, resilience engineering, MBSE/SysML integration and epistemic digital twins.

The originality of the approach lies in the combination of these elements within a single interaction-centred modelling paradigm. IFSM does not merely add quantitative metadata to FRAM, nor does it replace FRAM with a conventional simulation technique. It proposes an executable semantics in which FRAM-style functions are compiled into Turons, aspect categories are represented as typed signal roles, and model behaviour is generated through repeated propagation of metadata-bearing signals across a coupled functional configuration. This allows functional resonance, delay, saturation, inhibition, amplification and regime shift to be explored as traceable execution phenomena rather than only as post hoc analytical interpretations.

This paper therefore addresses a specific methodological gap: FRAM can describe functional coupling and variability, but it does not by itself define a full execution semantics for stateful signal propagation across repeated cycles. IFSM is proposed as a response to that gap. Its

novelty lies in treating functions as executable, bounded, stateful abstractions; treating couplings as typed metadata-bearing signal packets; and treating system behaviour as an evolving trace of interaction rather than as a fixed diagram or linear sequence. The paper first explains the transition from FRAM to executable interaction modelling, then defines IFSM and the Turon, specifies the execution semantics, illustrates the approach through a simple example, and finally discusses its position in relation to FRAM, SysML/MBSE, digital twins and safety analysis.

2. From Functional Resonance to Executable Interaction Modelling

The transition from FRAM to IFSM depends on one central move: the FRAM function must be interpreted not only as an analytical unit, but also as an executable one. A FRAM model can show how functions depend on one another, how outputs become conditions for other functions, and how variability may propagate through couplings. But if the model is to be run as an evolving interaction system, each function must also be able to receive signals, evaluate constraints, change internal state and produce outputs that affect other functions over time.

The analogy with automata provides one useful bridge. A formal automaton can be described in terms of inputs, outputs, internal states, transition rules and output rules. FRAM functions already contain part of this structure: they have Inputs and Outputs, and their performance is shaped by Preconditions, Resources, Controls and Time. The automaton analogy therefore adds a formal language for describing how a function might transform incoming conditions into outgoing effects.

This analogy should not be overextended. It does not imply that human, organisational or socio-technical performance is reducible to machine logic. Human work remains adaptive, contextual and partly indeterminate (Hollnagel and Woods, 2005; Woods et al., 2010). The value of the analogy is methodological rather than ontological: it provides a disciplined way to make a model function executable while preserving the FRAM concern with context, variability and adjustment.

A second analogy comes from signal-processing architectures inspired by neural systems. Biological neurons receive signals, integrate them in relation to local conditions and produce outputs that influence other neurons. IFSM does not use this analogy in the artificial neural-network sense. It does not require hidden layers, opaque learned weights or statistical training over large datasets. The useful idea is simpler and more transparent: complex behaviour may emerge from **coupled fields of bounded processing units** that receive, transform and emit signals over repeated cycles.

A third conceptual source is the idea of bounded conditional state and boundary-mediated interaction. In many systems, the relevant modelling unit is neither a component nor an actor, but a functional region that has a partially bounded internal state and interacts across an interface. This is where the term “Markovian” becomes useful. It points to the modelling discipline of treating each functional abstraction as having local state and boundary-mediated interaction, while recognising that the total system state emerges from the **configuration of such local states and their interactions**.

IFSM therefore brings together three ideas: Turing-style executability, signal-propagation architectures inspired by neural systems, and FRAM-style functional coupling. A function

becomes an executable functional abstraction. It receives typed signals, evaluates them in relation to its internal state, applies transition and output rules, and emits labelled outputs to other functions. System behaviour then emerges through repeated propagation through the **aspect-mediated configuration**.

3. Defining Interactive Functional System Modelling

Interactive Functional Systems Modelling has been defined (Slater 2026):

"as a recursive modelling perspective in which systems are represented as dynamically bounded organisations of interacting functions. It begins by establishing purposes, perspectives and boundaries; formalises transformations, regulatory roles, couplings and control relationships; links those functions where appropriate to empirical state data; and may progressively develop qualitative models into executable representations of nested and interacting system behaviour".

This definition contains four important commitments.

First, IFSM is functional rather than component-centred. Its primary modelling unit is not the physical part, organisational role or software object, but the function: an activity or process required for system performance. A person, machine, software module or organisation may perform several functions; conversely, one function may be distributed across several actors or artefacts. This preserves FRAM's emphasis on activity, dependency and performance rather than physical decomposition alone.

Second, IFSM is stateful. A function is not treated as a **passive transfer point** that simply passes inputs to outputs. It has a local state that may include activation level, confidence, workload, resource sufficiency, delay, degradation, saturation, uncertainty or other domain-specific variables. The same incoming signal may produce different outputs depending on this state.

Third, IFSM is interaction-centred. Couplings are not merely static dependencies. They are executable pathways through which typed signals propagate. These signals may carry values, metadata, timestamps, confidence levels and source information. System behaviour arises through the repeated interaction of these signals with local functional states.

Fourth, IFSM is emergent. The behaviour of the system is not specified as a single top-down sequence. It emerges from propagation through the **coupled functional configuration**. Some trajectories may stabilise. Others may oscillate, saturate, dampen, amplify or shift into new regimes.

IFSM is therefore not "FRAM with numbers added". Adding numerical values to a static model does not make it dynamic. IFSM changes the status of the interaction structure itself. Functions become stateful processes, couplings become signal pathways, and the model output becomes a trace of evolving behaviour rather than only a diagram of dependency.

At the same time, IFSM is not a replacement for FRAM. FRAM remains valuable as a method for eliciting, structuring and interpreting functional dependencies. IFSM is a proposed execution semantics for exploring what follows from a specified interaction structure. In this sense, FRAM provides the functional grammar; IFSM provides an executable interaction layer.

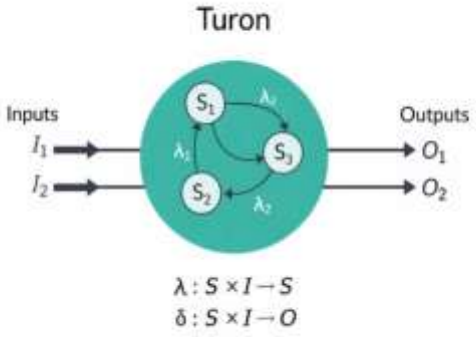
4. The Turon as Executable Functional Abstraction

The executable unit of IFSM is the **Turon**. The Turon is not a discovered natural unit of systems. It is a modelling construct: a bounded, aspect-aware representation of a functional process. A Turon receives labelled signals, evaluates them in relation to its internal state, applies transition and output rules, and emits labelled outputs to other Turons. It is not a person, team, machine or organisation. It is a modelling unit that represents a functional process under declared assumptions.

The Turon has an intellectual lineage in biological neurons, automata theory and state-based computational models. At the biological level, the neuron provides the intuition of a processing unit that receives signals, integrates them within a current state and produces an output. At the formal level, Turing-machine and automata theory provide the language of input, output, state, memory and transition (Turing, 1936; Hopcroft, Motwani and Ullman, 2006). Structurally, the Turon is closest to a Mealy-type machine, in which output depends on both current input and internal state. This is important because socio-technical performance cannot be understood as either a passive response to stimulus or the simple expression of an internal state. It is produced through the interaction of current conditions, accumulated state and contextual constraints.

The Turon should therefore be understood not merely as a **passive transfer point** that receives an input and produces an output, but as an aspect-aware constraint processor. In FRAM, aspects such as Input, Precondition, Resource, Control and Time are represented as separate interaction sites around a function. In IFSM, these aspect categories become typed semantic roles attached to incoming signals. A downstream Turon can therefore read metadata from connected upstream functions and determine whether the necessary activation, control, resource, timing and permission conditions are satisfied.

The Turon does not require separate visual interaction sites corresponding to the FRAM hexagon. Instead, it evaluates typed signal packets and their metadata through an activation or guard function. This preserves FRAM's aspect logic while allowing the executable model to remain compact.



Formally, a Turon may be represented (Figure 1) as:

$$T = (\Sigma_{in}, \Sigma_{out}, S, \gamma, \delta, \lambda, \Theta)$$

where Σ_{in} is the set of incoming typed signal packets, Σ_{out} is the set of emitted signal packets, S is the internal state space, γ is the guard or activation function, δ is the state-transition function, λ is the output function, and Θ is the parameter set, including thresholds, gains, delays, priorities, decay rates and merge rules.

Figure 1 – The Turon – an executable function as a node

A signal packet may be represented as:

$$\sigma = (label, role, value, metadata, timestamp, confidence, source)$$

where the label identifies the coupling, the role identifies the FRAM/IFSM aspect type, the value contains the propagated condition or quantity, metadata contains additional attributes, timestamp records cycle or time, confidence expresses uncertainty or validity, and source identifies the producing Turon or boundary input.

The guard function γevaluates whether the required input, precondition, resource, control, timing and permission conditions are present and sufficient. The transition function δdetermines how the Turon's internal state changes. The output function λdetermines what metadata-bearing outputs are emitted. In this formulation, the Turon is executable, but not ontologically reductive. Its rules are modelling hypotheses, not claims that real socio-technical work is reducible to deterministic state transition.

5. Execution Semantics

A IFSM model executes through repeated propagation cycles. At each cycle, Turons evaluate incoming signal packets, update internal state, emit outgoing signals and write trace records. A simplified execution cycle is as follows:

1. The model reads the signal buffer at cycle (t).
2. Each Turon evaluates incoming signal packets using its guard function (\gamma).
3. If the guard condition is satisfied, the Turon updates its internal state using (\delta).
4. If the guard condition is partially satisfied, the Turon may update to a degraded, delayed or uncertain state.
5. If the guard condition is not satisfied, the Turon remains inactive or enters a blocked state.
6. The output function (\lambda) emits labelled metadata-bearing signals.
7. Propagation rules apply delay, decay, amplification, inhibition, priority and merge logic.
8. Signals are written to the buffer for cycle (t+1).
9. A trace record is written, including active Turons, state changes, emitted signals and unresolved constraints.
10. Execution stops when an exit condition, cycle limit, stable state, oscillatory pattern or externally defined termination condition is reached.

The global state of the IFSM model at cycle $t$can be represented as:

$$X(t+1) = F(X(t), U(t), \Theta)$$

where $X(t)$is the current state of the Turon network, $U(t)$represents external inputs, and Θrepresents parameters such as thresholds, gains, delays, decay rates, priorities and merge rules. The function $F$is not a single hidden equation. It is the collective result of the guard, transition, output and propagation rules distributed across the Turon network.

This formulation matters because it allows emergence to be treated as the evolution of a network state rather than as a chain of events. Repeated signal propagation produces

trajectories through state space. Some trajectories may settle into stable patterns. Others may oscillate, saturate or shift into new regimes. Such behaviour is consistent with established accounts of nonlinear and networked systems, where local interactions may produce equilibria, limit cycles, attractor-like patterns or regime transitions (Mitchell, 2009; Strogatz, 2015).

Execution semantics also require explicit handling of scheduling and conflict. Turons may be updated synchronously, where all nodes evaluate cycle (t) before any outputs are made available at (t+1), or asynchronously, where selected nodes update according to event triggers, priorities or timing constraints. Synchronous updating is simpler and more reproducible; asynchronous updating may better represent some operational settings but requires additional assumptions.

Similarly, when multiple upstream Turons emit signals with the same label, merge rules must specify whether values are averaged, prioritised, accumulated, inhibited, selected by confidence or treated as conflicting. These rules are not technical details. They are part of the model's theory of interaction and must therefore be inspectable.

This reconnects IFSM with the original FRAM concern with functional resonance. FRAM shows how ordinary variability may combine through coupling. IFSM provides a way to explore resonance-like behaviour dynamically. A delayed corrective signal may arrive too late and reinforce instability. Several weak signals may accumulate until a threshold is crossed. A stabilising function may become saturated. A feedback loop may dampen disturbance in one configuration but amplify it in another.

The important claim is not that IFSM predicts complex systems with certainty. It does not. Its more defensible claim is that it can provide a parameterised executable modelling framework, where parameters are explicitly justified, inspected and revised. The model makes assumptions about functional interaction explicit enough that their consequences can be traced, compared and challenged.

6. Methodological Discipline and Validation

For IFSM to be useful, construction must be disciplined. An executable interaction model cannot simply be drawn and allowed to run. The modeller must define the system boundary, identify the functional processes to be represented as Turons, specify the signals that couple them, define relevant state variables, and make transition and output rules explicit. Without these disciplines, execution risks becoming arbitrary. With them, IFSM becomes a reproducible way of exploring interaction dynamics.

The first discipline is boundary definition. The modeller must state what is inside the system of interest, what is treated as an external input, and what outputs leave the model. This is not merely a scoping exercise. The boundary determines what the model can explain and what it must treat as context.

The second discipline is functional identification. Turons should represent functional processes rather than components or actors alone. A pilot, nurse, controller, software module or organisation may perform several functions; conversely, one function may be distributed across several actors or artefacts. This preserves the FRAM orientation toward activity and dependency rather than physical decomposition.

The third discipline is signal control. Since couplings in IFSM are formed through labelled signals, the signal vocabulary must be explicit and stable. A signal name is not casual text; it is the semantic connector of the executable model. If the same condition is described using different labels, intended couplings may be lost. If different conditions are given the same label, false couplings may be created. Signal dictionaries therefore become part of the model's audit trail.

The fourth discipline is rule specification. Each Turon requires state variables and rules describing how incoming signals update state and generate outputs. These rules may initially be simple, ordinal or judgement-based; they may later be refined using data, expert review or calibration. What matters is that they are explicit. A transition rule should be open to challenge: why this threshold, this delay, this gain, this merge rule, this output? The interpretability of IFSM depends on the answer being inspectable.

Validation should therefore be understood as a continuing process rather than a final proof. A IFSM model may be assessed structurally, by asking whether the Turons and couplings are credible; semantically, by asking whether signal labels and state variables make operational sense; computationally, by checking whether the model executes consistently; and empirically, by comparing generated traces with observed work-as-done, incident reconstructions, simulations, telemetry or expert judgement.

Calibration should follow the same principle. It should not become curve-fitting. If a delay is inserted, the modeller should ask what real process the delay represents. If a threshold is changed, the modeller should ask whether this reflects altered sensitivity, resource degradation, changed policy, workload, fatigue or a different operating regime. The goal is not to force the model to match data, but to refine an executable theory of interaction.

A reproducible IFSM model should therefore include at least the following information: the system boundary, the Turon list, the signal dictionary, state variables, guard rules, transition rules, output rules, parameter values, merge rules, delay assumptions, execution schedule and trace outputs. Without these, the model may appear executable but remain scientifically opaque.

7. Illustrative Example: Making Tea as a IFSM Execution Trace

A simple tea-making model illustrates the principle. In a FRAM model, the function "To boil water" may receive a kettle resource, a water-in-kettle precondition, a heating-time aspect and a target-temperature control. In IFSM, this function is represented as a Turon that calculates the achieved water temperature and emits a hot_water signal containing metadata.

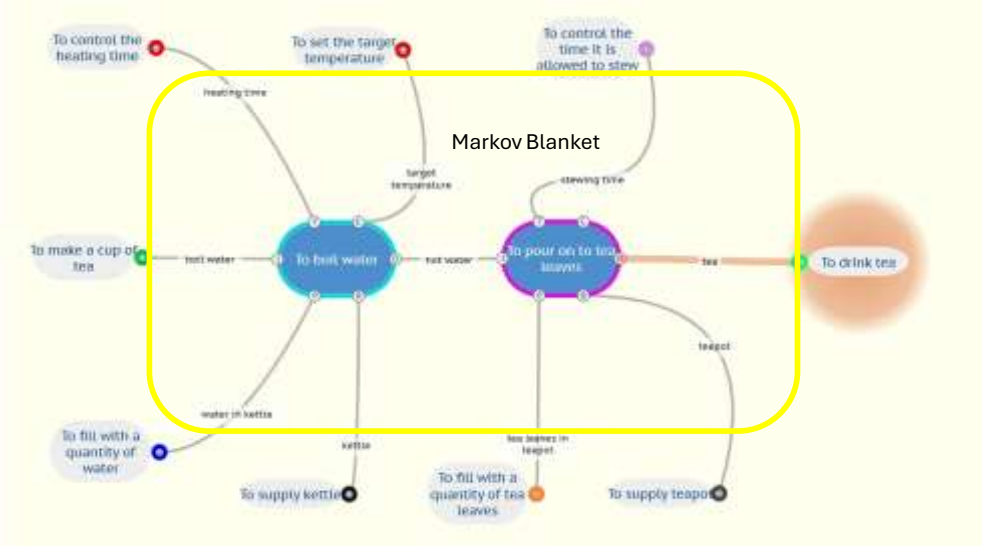
With an initial temperature of 15°C, kettle power of 1000 W, water mass of 1000 g and heating time of 300 s, the achieved temperature is approximately 86.7°C, below a nominal 100°C target. The downstream function "To pour onto tea leaves" therefore receives not only the label hot_water, but the achieved state of that water. If tea quality depends on actual temperature, the downstream Turon can distinguish between intended target temperature and achieved temperature. The example shows how IFSM turns FRAM couplings into executable signal pathways and how traces can reveal discrepancies between intended and actual functional state.

A simplified IFSM version of the tea-making FRAM model can be represented using three main Turons: one to heat the water, one to infuse the tea, and one to consume or evaluate the final output.

Table 1 shows the executable “functions” and Table 2 the metadata produced when the system cycles through the process

Table 1. – The executables

Turon	FRAM source function	Main input signals	Main output signal	State variables
(T_{heat})	To boil water	kettle, water_in_kettle, heating_time, target_temperature	hot_water	actual temperature, target temperature, heating time
(T_{infuse})	To pour onto tea leaves	hot_water, tea_leaves, teapot, stewing_time	tea	infusion quality, temperature adequacy, total time
(T_{drink})	To drink tea	tea	tea consumed or tea unsatisfactory	acceptability, final quality



TA simplified execution trace may be represented as follows.(Table 2)

The same FRAM structure may therefore generate different IFSM trajectories depending on metadata values. If heating time is increased, the hot_water.temperature_actual value rises and the downstream tea-quality state may improve. If the water quantity is increased without increasing heating time, the propagated temperature may fall and downstream quality may degrade. If the downstream function incorrectly reads temperature_target rather than temperature_actual, the model may falsely classify the tea as acceptable. The trace therefore

exposes a central modelling issue: executable couplings must propagate achieved functional state, not only intended control values.

Cycle	Active Turon	Metadata read	Metadata produced	Interpretation
(t=0)	(T_{heat})	power = 1000 W; water = 1000 g; start temperature = 15°C; heating time = 300 s; target = 100°C	hot_water.temperature_actual ≈ 86.7°C; hot_water.temperature_target = 100°C	Hot water is produced, but achieved temperature is below target
(t=1)	(T_{infuse})	hot_water.temperature_actual ≈ 86.7°C; stewing-time metadata; tea-leaf and teapot availability	tea.quality_temperature = weak/degraded if actual temperature threshold is not met	Tea quality depends on propagated actual temperature, not merely intended target
(t=2)	(T_{drink})	tea-quality metadata	tea_acceptable = true/false/conditional	Final output reflects accumulated upstream state
(t=3)	Trace review	observed final state compared with predicted functional state	discrepancy flags if expected and actual quality diverge	Model assumptions can be inspected and revised

This compact example shows the practical meaning of IFSM. A compliant FRAM model provides the functions, aspects, couplings and metadata. IFSM compiles these into Turons, labelled signals, state variables and transition/output rules. The resulting execution trace shows how functional state propagates through the model and how discrepancies between intended and achieved state can be detected.

The tea-making example is intentionally simple. It is not offered as evidence that IFSM is sufficient for system analysis. Its purpose is pedagogical: to show how an interpretable FRAM coupling can be transformed into an executable signal pathway. A safety-relevant demonstration would need to show the same principle in a domain such as unstable approach management, clinical deterioration, process alarm handling or autonomous system handover.

8. Position within the Modelling Landscape

IFSM has affinities with several modelling traditions, but differs in its choice of modelling unit and its emphasis on interpretable functional interaction. Reliability models provide disciplined accounts of component failure, barrier performance and event sequences, but they often abstract away the interaction dynamics through which complex systems operate. System dynamics represents feedback and accumulation, but usually at a more aggregated level than function-to-function coupling. Agent-based modelling represents interacting entities, whereas IFSM represents interacting functional processes. Petri nets provide formal token-based execution over graph-like structures, but IFSM emphasises contextualised, stateful functional performance rather than token flow alone. Recurrent neural models share the broad idea of

stateful signal propagation, but IFSM remains explicitly constructed and interpretable rather than statistically trained and opaque (Peterson, 1981; Goodfellow, Bengio and Courville, 2016).

IFSM differs from actor-oriented and component-based simulation approaches because its primary modelling unit is not the physical, organisational or software component, but the function: an activity required for system performance, represented as a stateful, aspect-aware Turon whose execution depends on typed signals, internal state and explicit guard, transition and output rules.

Its closest conceptual ancestor is FRAM. FRAM asks how functions are coupled and how variability may resonate. IFSM asks how an aspect-mediated configuration of stateful coupled functions behaves when executed over repeated cycles. The relationship is therefore one of extension rather than replacement. FRAM remains valuable as a method for eliciting, structuring and interpreting functional dependencies. IFSM gives such dependencies execution semantics.

The execution semantics of IFSM share structural similarities with several established classes of computational models. Petri nets represent systems as graph-like arrangements of places, transitions and tokens. System behaviour unfolds through the firing of transitions when specific conditions are satisfied, causing tokens to propagate through the structure (Peterson, 1981). IFSM shares the idea of rule-governed activation and propagation. However, whereas Petri nets typically represent discrete state changes associated with resource flow or process synchronisation, IFSM focuses on stateful functional elements whose behaviour depends on both incoming signals and accumulated internal state. This allows the model to represent contextualised performance rather than purely discrete event transitions.

A second comparison is with recurrent neural models. In such systems, processing units receive signals from other processing units and update their internal states according to activation functions, producing outputs that propagate through the arrangement. Complex behaviour emerges through repeated cycles of signal propagation and state updating (Goodfellow et al., 2016). IFSM shares the architectural principle of stateful units interacting through signals. However, recurrent neural models typically learn their behaviour through parameter optimisation over large datasets, whereas IFSM models are constructed through explicitly defined functional interactions and rule-based activation logic. In this sense, IFSM retains interpretability while adopting a broadly similar propagation architecture.

A third comparison arises with coupled dynamical systems, particularly coupled sets of oscillators or interacting state variables studied in nonlinear dynamics (Strogatz, 2015). In such systems, the behaviour of each element depends on the states of neighbouring or interacting elements, and collective patterns such as synchronisation, oscillation or regime shifts may emerge from the coupled set. The propagation cycles of IFSM models create a comparable structure in which system behaviour evolves through repeated interaction among stateful functional abstractions. The resulting activation traces may therefore exhibit patterns analogous to those studied in dynamical systems theory.

Taken together, these comparisons suggest that IFSM sits at the intersection of rule-based execution, stateful signal propagation and interaction-driven system dynamics. Its distinctive contribution lies in combining these properties within a modelling framework designed specifically for interpretable interaction modelling of complex socio-technical systems.

9. SysML as an Authoring and Governance Layer for IFSM

IFSM models need not be authored only in a bespoke notation. A practical implementation could use SysML, particularly SysML v2, as an authoring and governance layer. SysML can represent system structure, behaviour, requirements, constraints, analysis cases and verification cases within a traceable model-based systems engineering environment. A IFSM profile or transformation could then identify selected behaviours as Turons, type their incoming and outgoing signals according to FRAM/IFSM aspect roles, attach state variables and transition rules, and compile the result into an executable IFSM configuration.

In this architecture, SysML provides the systems-engineering model and traceability framework, while IFSM provides the runtime semantics for stateful signal propagation. This would allow IFSM to integrate with established model-based systems engineering practice while preserving the FRAM-derived focus on functional interaction.

The distinction is important. SysML can represent behaviour, structure and constraints, but IFSM defines how coupled functional processes execute over repeated cycles, how metadata-bearing signals propagate, and how traces are generated for comparison with observed system behaviour. SysML is therefore not a replacement for IFSM; it is a suitable model-authoring and interoperability layer from which IFSM models can be generated.

This creates a plausible bridge to systems engineering practice, particularly where traceability, requirements linkage, verification cases and configuration control are required. The risk of losing FRAM aspect logic can be reduced by typing SysML signals with FRAM/IFSM roles. The Turon should, however, be described as an aspect-aware constraint processor rather than as an autonomous intelligent agent. This keeps the concept interpretable and avoids implying a degree of agency or intelligence that the formalism does not require.

10. Implications for Interaction-Centred Digital Twins

One implication of IFSM is the possibility of interaction-centred digital twins. Conventional digital twins often represent physical configuration, asset state, measurable parameters or process behaviour (Boschert and Rosen, 2016; Tao et al., 2019). These remain valuable, but they may not capture the coordination dynamics through which socio-technical systems actually operate. A IFSM-based twin would represent how functions interpret signals, update state, respond to changing conditions and influence one another over time.

This would not replace physical or process twins. It would add an interaction layer. In aviation, flight data could be mapped onto functions such as maintaining energy state, monitoring approach stability, maintaining mode awareness and deciding whether to continue or go around. In healthcare, physiological observations could be mapped onto detecting deterioration, interpreting significance, escalating care and initiating intervention. In process safety, plant variables could be mapped onto measuring process state, interpreting alarms, diagnosing deviation, allocating operator attention and restoring control.

The key distinction is between observed state and predicted functional state. Observed data populate the model with what appears to be happening. The IFSM model then generates an expected evolution of functional state based on its current assumptions. Discrepancies between observed and expected functional state are analytically valuable. They may indicate

missing couplings, wrong thresholds, delayed signals, saturated functions, weak assumptions or a shift into a different operating regime.

In this sense, an interaction-centred twin is not merely a digital mirror. It is an epistemic instrument. It represents what the modeller currently believes about how the system functions, then tests that belief against observed or simulated behaviour. This implication should be treated as a future direction rather than the main claim of the paper. The primary contribution is the executable modelling paradigm. Interaction-centred digital twins are one possible operational expression of that paradigm.

11. Discussion

The central contribution of IFSM is not that it makes complex systems fully predictable. It does not. Nor does it replace FRAM or established modelling approaches. Its contribution is more specific: it makes assumptions about functional interaction explicit, executable, traceable and revisable.

This matters because quantification can mislead as well as clarify. A numerical trace may look more objective than a qualitative account, but it is only as meaningful as the assumptions embedded in the model. IFSM therefore depends on modelling discipline, validation and calibration. The value of execution lies not in the production of numbers, but in exposing the consequences of stated assumptions.

The strongest philosophical position for the approach is therefore quantification without reductionism. IFSM borrows from automata the discipline of input, output, state and transition, but it does not claim that human or organisational performance is reducible to machine logic. This is consistent with the broader tradition of cognitive systems engineering, which treats human-machine systems as adaptive joint systems rather than as separable mechanical components (Hollnagel and Woods, 1983; Roth, Bennett and Woods, 1987; Hollnagel and Woods, 2005).

A Turon should therefore be understood as a modelling representation, not an ontological claim about the real world. It represents a functional process within a defined model boundary. Its rules are hypotheses. Its outputs are consequences of those hypotheses. Its trace is not the system itself, but a structured account of how the modelled interactions unfold under specified assumptions.

The limitations follow directly. IFSM is only as good as the functional analysis on which it is built. Important functions may be omitted. Couplings may be mis-specified. Signal labels may drift. State variables may be poorly chosen. Human and organisational processes may resist stable formalisation. Large models may become difficult to validate. Poorly justified parameters may create spurious precision. These limitations should be made explicit because they strengthen, rather than weaken, the argument. They show that IFSM is a support for disciplined reasoning, not an authority.

The promise lies in the middle ground. IFSM avoids reducing complex systems to deterministic machinery, but also avoids leaving interaction as an unexecutable diagram. It offers a way to study how systems stabilise, adapt, oscillate, saturate and sometimes shift into new regimes. It does not remove uncertainty. It gives analysts a better way to work with uncertainty by making interaction assumptions explicit enough to be tested.

12. Conclusion

This paper has proposed Interaction Functional System Modelling as an executable modelling layer for interaction-centred systems analysis. Its first finding is that FRAM's functional grammar can be extended without abandoning its central insight: system behaviour is produced through the interaction of functions rather than through linear chains of component failure. Its second finding is that the Turon provides a useful executable abstraction for this purpose, because it represents a function as a bounded, stateful process that evaluates typed incoming signals, updates internal state and emits metadata-bearing outputs.

A third finding is that IFSM changes the status of couplings. Couplings are not treated merely as static dependencies in a diagram, but as signal pathways through which achieved functional state is propagated over repeated cycles. This makes it possible to inspect how delays, weak signals, saturation, inhibition, amplification and conflicting inputs alter the evolution of system behaviour. A fourth finding is that execution traces are the main analytical product of IFSM. They allow modellers to compare expected and observed functional state, identify discrepancies, revise assumptions and calibrate rules without claiming deterministic prediction.

The contribution is therefore modest but significant. IFSM does not make complex socio-technical systems fully predictable, nor does it replace FRAM, MBSE or existing simulation approaches. It provides a parameterised, interpretable and revisable way to explore how emergent, non-linear and non-predetermined behaviour may arise from coupled functional processes. FRAM provides the grammar of coupled functions; the Turon provides the executable modelling primitive; and IFSM provides the propagation architecture needed to trace, compare and revise interaction behaviour over time.

References

Barabási, A.-L. (2016) *Network Science*. Cambridge: Cambridge University Press.

Boschert, S. and Rosen, R. (2016) 'Digital Twin—The simulation aspect', in Hehenberger, P. and Bradley, D. (eds.) *Mechatronic Futures: Challenges and Solutions for Mechatronic Systems and their Designers*. Cham: Springer, pp. 59–74.

de Ruijter, A. and Guldenmund, F. (2016) 'The bowtie method: A review', *Safety Science*, 88, pp. 211–218.

Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep Learning*. Cambridge, MA: MIT Press.

Hill, R. and Slater, D. (2024) 'Using a metadata approach to extend the Functional Resonance Analysis Method to model quantitatively, emergent behaviours in complex systems', *Systems*, 12(3), 90.

Hollnagel, E. (2004) *Barriers and Accident Prevention*. Aldershot: Ashgate.

Hollnagel, E. (2012) *FRAM: The Functional Resonance Analysis Method: Modelling Complex Socio-technical Systems*. Farnham: Ashgate.

Hollnagel, E. and Woods, D.D. (1983) ‘Cognitive systems engineering: New wine in new bottles’, *International Journal of Man-Machine Studies*, 18(6), pp. 583–600.

Hollnagel, E. and Woods, D.D. (2005) *Joint Cognitive Systems: Foundations of Cognitive Systems Engineering*. Boca Raton, FL: CRC Press.

Hopcroft, J.E., Motwani, R. and Ullman, J.D. (2006) *Introduction to Automata Theory, Languages, and Computation*. 3rd edn. Boston, MA: Pearson.

Leveson, N.G. (2011) *Engineering a Safer World: Systems Thinking Applied to Safety*. Cambridge, MA: MIT Press.

Mitchell, M. (2009) *Complexity: A Guided Tour*. Oxford: Oxford University Press.
Patriarca, R., Bergström, J., Di Gravio, G. and Costantino, F. (2018) ‘Framing the FRAM: A literature review on the Functional Resonance Analysis Method’, *Safety Science*, 102, pp. 2–19.

Perrow, C. (1984) *Normal Accidents: Living with High-Risk Technologies*. New York: Basic Books.

Peterson, J.L. (1981) *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall.
Rasmussen, J. (1997) ‘Risk management in a dynamic society: A modelling problem’, *Safety Science*, 27(2–3), pp. 183–213.

Roth, E.M., Bennett, K.B. and Woods, D.D. (1987) ‘Human interaction with an “intelligent” machine’, *International Journal of Man-Machine Studies*, 27(5–6), pp. 479–525.

Salehi, V., Smith, D., Veitch, B. and Hanson, N. (2021) ‘A dynamic version of the FRAM for capturing variability in complex operations’, *Safety Science*, 143, 105419.

Slater, D., (2026), The Evolution of Functional Systems Modelling, ResearchGate, https://www.researchgate.net/publication/406952025_The_Evolution_of_Functional_Systems_Modelling

Strogatz, S.H. (2015) *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. 2nd edn. Boulder, CO: Westview Press.

Tao, F., Zhang, H., Liu, A. and Nee, A.Y.C. (2019) ‘Digital twin in industry: State-of-the-art’, *IEEE Transactions on Industrial Informatics*, 15(4), pp. 2405–2415.

Turing, A.M. (1936) ‘On computable numbers, with an application to the Entscheidungsproblem’, *Proceedings of the London Mathematical Society*, 2(42), pp. 230–265.

Woods, D.D., Dekker, S., Cook, R., Johannesen, L. and Sarter, N. (2010) *Behind Human Error*. 2nd edn. Farnham: Ashgate.

Appendix

A Procedure for Implementing IFSM Models

Interactive Functional System Modelling can be implemented by converting a functional model into an executable configuration of **Turons**, **typed signal packets**, **state variables**, **rules**, **parameters** and **execution traces**. The aim is not to produce deterministic predictions of a complex system, but to make assumptions about functional interaction explicit, executable, inspectable and revisable.

1. Define the model boundary

Define the system or process of interest and specify what lies inside and outside the executable model. The boundary should be treated as a **model interface**, not simply as a visual perimeter. External influences are represented as **source signals** entering the model. Final outputs, completed states or consequences leaving the model are represented as **sink signals**.
This boundary may be understood as analogous to a **Markov blanket**, in the limited sense that interaction between internal model states and the external environment is mediated through declared interface states.

2. Identify the functional processes

Identify the activities, tasks or processes required for the system to perform its intended work. These should be expressed as **functions**, not merely as actors, components, departments or technologies. A person or component may perform several functions, and a single function may be distributed across several actors or artefacts.
Each selected function should make a clear contribution to system performance and should be capable of receiving conditions, evaluating them and producing an output state.

3. Convert functions into Turons

Each executable function is represented as a **Turon**. A Turon is a bounded, stateful modelling unit that represents a functional process. It is not a person, machine or organisation; it is an executable abstraction of a function under declared assumptions.

Each Turon may be treated as a local **Markov "bubble"**: a bounded state-processing unit whose next state is computed from its current internal state, declared incoming signals and parameters, and whose influence on other Turons occurs only through typed outgoing signal packets.

Each Turon may thus be regarded as its own bounded state-processing unit. For each Turon, specify:

- function name;
- internal state variables;
- incoming signal types;
- output signals;
- guard or activation rule;
- state-transition rule;
- output rule;
- parameters such as thresholds, delays, gains, priorities and decay rates.

Complex functions may be decomposed into sub-Turons or grouped Turons where finer resolution is required.

4. Define typed signal packets

Interactions between Turons are represented as **typed metadata-bearing signals**. A signal packet should contain:

signal = (
label,
role,
value,
metadata,
timestamp,
confidence,
source)

The **label** identifies the coupling. The **role** identifies the aspect type, such as Input, Control, Precondition, Resource or Time. The **value** carries the propagated condition or quantity. The **metadata** carries additional information such as units, thresholds, uncertainty, achieved state, target state or validity. The **timestamp**, **confidence** and **source** provide traceability.

The key discipline is that IFSM propagates **achieved functional state**, not merely intended control values.

5. Build the signal and metadata dictionary

Create a controlled dictionary of signal labels, metadata fields, permitted values, units and merge rules. The dictionary prevents accidental semantic drift.

For each signal, define:

- signal label;
- producing Turon or boundary source;
- consuming Turon or sink;
- signal role;
- value type;
- metadata fields;
- units and valid ranges;
- default values;
- merge rule where multiple signals share a label;
- evidence basis.

All Turons share the same dictionary as a semantic reference. However, each Turon should only access its declared incoming signals, its own internal state, declared boundary inputs and permitted parameters. This avoids hidden couplings.

6. Specify rules and parameters

Each Turon requires three rule types.

The **guard rule** determines whether the Turon activates, delays, degrades, blocks or remains inactive.

The **transition rule** determines how the Turon's internal state changes after evaluating incoming signals.

The **output rule** determines what signals and metadata the Turon emits.

Rules may initially be simple, ordinal or expert-defined. They may later be refined using telemetry, simulation, statistical models, MATLAB functions, lookup tables or other specialist models. The essential requirement is that the rules remain explicit and open to review.

7. Select the execution environment

A spreadsheet can be used as an initial implementation environment because it provides transparency, formula execution, tabular state storage and easy inspection. In such a prototype, Excel may hold:

- Turon register;
- signal dictionary;
- state variables;
- guard rules;
- transition rules;
- output rules;
- parameter table;
- current-cycle signal buffer;
- next-cycle signal buffer;
- execution trace;
- calibration log.

For larger models, the same structure may be implemented in a database-backed execution engine, SysML-linked model, Python interpreter or dedicated IFSM/FMI-style execution tool.

8. Execute propagation cycles

The model runs through repeated execution cycles. However, in IFSM the term “Markovian” is specifies that each cycle is computed from the current declared model state, current boundary inputs and declared parameters. It does not imply that the real socio-technical system is memoryless.

At each cycle:

1. Read the current signal buffer.
2. For each Turon, retrieve declared incoming signals and current internal state.
3. Evaluate the guard rule.
4. Update internal state using the transition rule.
5. Emit output signals using the output rule.
6. Apply delay, decay, amplification, inhibition, priority or merge rules.
7. Write emitted signals to the next-cycle buffer.
8. Record active Turons, state changes, emitted signals and unresolved constraints.
9. Advance from cycle t to cycle t+1.
10. Stop when an exit signal, stable state, cycle limit, oscillation, divergence or defined termination condition is reached.

In compact form: $X(t+1) = F(X(t), U(t), \Theta)$

where $X(t)$ is the current Turon-state configuration, $U(t)$ represents boundary inputs, and Θ contains declared parameters. F is not a hidden global equation; it is the combined result of the Turons’ guard, transition, output and propagation rules.

9. Store the execution trace

Each model run should produce an execution trace. The trace records how the model behaved under the declared assumptions.

The trace should include:

- cycle number;
- active Turons;
- blocked, delayed or degraded Turons;
- signals read;
- state before and after execution;
- signals emitted;
- merge or conflict events;
- boundary outputs;
- discrepancy flags;
- reviewer comments.

The trace is the main analytical product of IFSM. It allows the modeller to inspect how functional state propagated, where constraints accumulated, where signals arrived late or weakly, and where the model diverged from observed behaviour.

10. Validate, calibrate and revise

Validation is continuous rather than final. A IFSM model should be reviewed structurally, semantically, computationally and empirically.

- **Structural validation** asks whether the Turons and couplings credibly represent the system.
- **Semantic validation** asks whether labels, metadata and state variables are coherent.
- **Computational validation** checks whether the model executes reproducibly.
- **Empirical validation** compares traces with observations, incident reconstructions, simulations, telemetry or expert judgement.

Calibration should not become curve-fitting. Any change to a threshold, delay, gain, priority or merge rule should be justified in terms of the real functional process it represents.