7/8/2026

FRAMsynt

TURONS: Executable Functions, Couplings and Functional Resonance in Systems

Turons: Executable Functions, Couplings and Functional Resonance in Realising Systems

David Slater

Abstract

The Functional Resonance Analysis Method (FRAM) provides a powerful way of understanding complex sociotechnical systems as patterns of interdependent functions rather than as collections of components. It shows that system behaviour emerges from the couplings between functions and from the variability of everyday function performance (Hollnagel, 2012). However, conventional FRAM remains primarily descriptive. It identifies possible functions, aspects and couplings, but it does not by itself specify how functions become active, how couplings are achieved at runtime, how variability is carried through the system, or how realisation traces can be stored, compared and learned from.

*This paper introduces the concept of the **Turon** as a development of the FRAM function. A Turon is an open, stateful, executable function that waits for required inputs of specified types, uses the metadata carried by those inputs, changes its own state or the wider system state, and produces outputs that become available to other functions within a defined system workspace. In this formulation, a coupling is not a fixed line between two functions. It is an achieved runtime binding that occurs when an available output satisfies the requirement of a waiting Turon.*

The Turon concept provides a formal basis for representing function performance variability and functional resonance. Each candidate coupling can be assessed in terms of availability, information effectiveness, tolerance, weighting and contextual modification. CREAM Common Performance Conditions can be incorporated as task, agent and environmental modifiers of coupling and firing probabilities (Hollnagel, 1998). Functional resonance can then be understood as the propagation, combination, damping or amplification of variable performance signatures through achieved Turon couplings.

The practical implication is that Functional Systems Modelling can move from static representation towards executable, inspectable and learnable system realisation. FMV can visualise possible functions and relationships; FRAIXL can hold Turon definitions, coupling rules, metadata and live state; FMI can operate as the realisation engine; and DuckDB can store runtime traces for comparison with Work-As-Done. The result is a route from qualitative FRAM modelling to auditable functional digital twins capable of representing how complex systems actually realise behaviour.

Key words - Turons; FRAM; Functional Systems Modelling; functional resonance; runtime coupling; performance variability; CREAM; Bayesian updating; digital twins; Markov blankets; Work-As-Done.

Introduction

Complex systems do not behave simply because components are connected. They behave because functions are performed, conditions are met, resources become available, information is interpreted, constraints are applied, and outputs from one part of the system become usable by another. This is the central insight that FRAM has brought to safety science, resilience engineering and sociotechnical systems modelling (Hollnagel, 2012, 2014). Systems are not best understood as static collections of parts, but as dynamic patterns of functional interdependence.

FRAM captures this by representing functions, aspects and couplings. It allows the modeller to ask what a function needs, what it produces, and how variability in one function may affect others. This has been a major advance over component-centred modelling because it shifts attention from structural decomposition to functional interaction. However, if Functional Systems Modelling is to become executable, traceable and learnable, a further step is required. We need to distinguish between the relationships that are shown in a model and the relationships that are actually realised in a specific instance of system behaviour.

The concept of the Turon is proposed as that next step. It retains the FRAM insight that systems are patterns of interdependent functions, but makes the function open, stateful and executable. A Turon is not simply a box in a diagram or a node in a graph. It is a function waiting for the conditions that allow it to be performed. It waits for required inputs of specified types. These inputs may include demand signals, resources, control instructions, timing conditions, preconditions, state signals or background metadata. They resemble FRAM aspects, but they are better understood as input requirements rather than fixed ports.

When the required inputs are sufficiently available, the Turon may fire. In firing, it uses the matched inputs, the metadata carried by those inputs and its own internal state. It may update its own state or the wider system state, and it produces outputs that become available within the system workspace. Those outputs may then satisfy the requirements of other Turons. The system is therefore realised through repeated cycles of matching, coupling, firing and output production.

The crucial shift is that a coupling is not treated as a pre-existing edge. A coupling is an achieved runtime binding. It occurs only when an available output satisfies the requirement of a waiting Turon. A drawn model may indicate that one function can provide something needed by another, but the actual system trace shows whether that relationship was realised, whether the output arrived in time, whether it was sufficiently complete and trustworthy, and whether the receiving function actually used it.

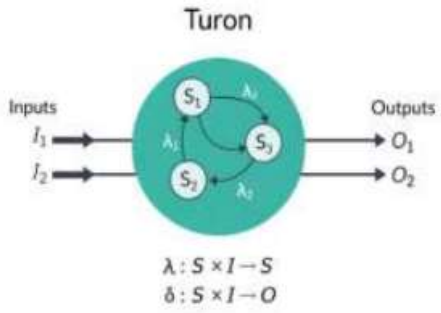
In simple terms, the realisation cycle is:

available output → matched requirement → coupling achieved → Turon fires → new output becomes available

This may appear sequential, but it need not be linear. Several Turons may become enabled at once. Some may compete for the same resource. Some outputs may persist, while others may expire, decay or be consumed. Some Turons may fire normally, while others may fire late, partially, incorrectly or in degraded mode. A Turon therefore has more explanatory power than a simple static function. It can remember what has happened before. It can be ready, overloaded, degraded, confident, uncertain, failed or learning. Its behaviour can change over time.

A concise definition is therefore:

A Turon is an open, stateful, executable function that waits for required inputs of specified types, uses the metadata carried by those inputs to execute its function, changes its own state or the system state, and produces outputs that become available to other functions within the relevant system workspace.



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.

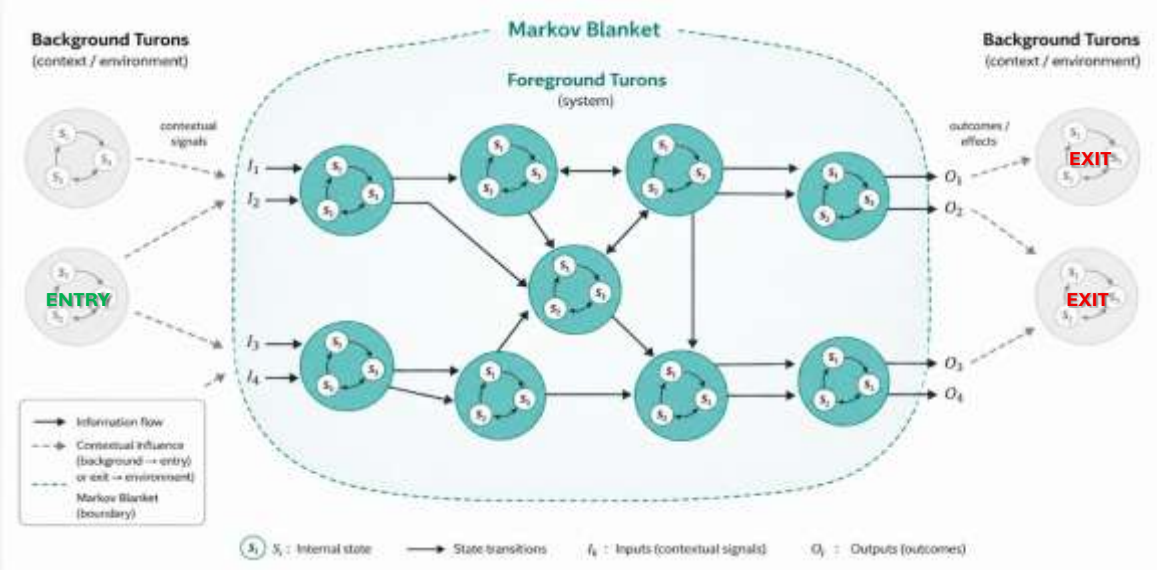
A Turon-Based System

A Turon-based system is defined by a modelling boundary that separates the foreground system from its background context. This boundary can usefully be understood as a Markov blanket. In its original and subsequent uses, the Markov blanket provides a way of distinguishing the internal states of a system from the external states with which they interact through mediating boundary states (Friston, 2010; Kirchhoff et al., 2018). In the present context, the Markov blanket is not treated as a hard physical wall, but as a modelling distinction between what is treated as part of the foreground system, what is treated as background, and how the two are allowed to influence one another.

Within the blanket are the foreground Turons. These are the Turons directly participating in the system realisation being modelled. They can match with and use any available outputs in the foreground workspace, subject to type, metadata, timing, quality, confidence and sufficiency rules.

Outside the blanket are background Turons. These may donate information, resources, constraints, demands, timing conditions, context or metadata across the boundary. They may also consume, receive or respond to outputs, actions and effects exported by the foreground system. In general, background Turons are not part of the foreground firing trace. They condition it, inform it, trigger it or receive from it.

Entry and Exit Turons are special active background Turons. They sit at the boundary and organise how the foreground system is coupled to its context. An Entry Turon can initialise, initiate, prompt, recycle, reset or feed back into the foreground system. It may collect background outputs and metadata, translate them into usable foreground inputs, and make them available inside the foreground workspace.



An Exit Turon performs the complementary role. It receives outputs, actions or consequences exported from foreground Turons and translates them into meaningful external outcomes, effects, records, feedback or changes in the background environment. Exit Turons may also support recycling and feedback by passing results back towards Entry Turons or other background Turons.

A realised system is therefore the trace of foreground Turons firing within a Markov blanket, shaped by background donations and active Entry mediation, and exported, completed or recycled through Exit Turons.

Basic Execution Logic

The simplest mathematical implementation of a Turon system needs only a few elements.

First, there is a set of available outputs in the foreground system. This can be thought of as the current workspace of the system. Each available output has a type, a value, a source, a time, a quality, a confidence level and associated metadata. At the start of a process sequence, these available outputs may be provided by an Entry Turon outside the foreground system.

Second, each Turon has a set of requirements. These describe what kinds of inputs it needs before it can fire. The requirements may be strict or flexible. A Turon may need all of its required inputs, any one of several alternatives, or enough evidence to pass a threshold.

Third, there is a matching rule. This asks whether the current available outputs are sufficient to satisfy the Turon's requirements. If they are, the Turon becomes enabled.

Fourth, there is a firing rule. When the Turon fires, it uses the matched inputs, their metadata and its own internal state to produce new outputs. It may also update its own state or the state of the wider system.

Fifth, there is an update rule for the workspace. New outputs are added. Some old outputs may remain available, some may be consumed, some may expire, and some may lose quality or confidence over time.

The basic cycle is therefore:

1. The foreground workspace contains available outputs.

2. A Turon checks whether available outputs match its requirements.
3. If the requirements are satisfied, a coupling is achieved.
4. The Turon fires.
5. The Turon produces new outputs.
6. The workspace is updated.
7. Other Turons may now become enabled.

This gives a simple but powerful execution logic. The system does not run because pre-drawn lines say that one box is connected to another. It runs because outputs become available, requirements are satisfied, couplings are achieved, and functions fire.

Coupling Variability

A systematic treatment of variability is now possible using this approach. It first requires a distinction between an **indicated coupling** and an **achieved coupling**.

- **An indicated coupling is a possible relationship shown in the model.**
- **An achieved coupling is a runtime binding that occurs only when an available output satisfies the requirement of a waiting Turon.**

This develops the FRAM emphasis on functional coupling and performance variability, but moves it towards an executable interpretation (Hollnagel, 2012). The distinction is important because the presence of a line in a model does not mean that the corresponding information, resource, control, timing condition or precondition will actually be present, usable, timely or sufficient in the realised system.

For each candidate coupling, **the first question is whether the coupling is available**. The model may indicate that a particular output can satisfy a requirement, but at runtime the output may be missing, delayed, expired, consumed, inaccessible, outside the relevant boundary, or present in a form that cannot be used. Availability can therefore be represented as a probability attached to the candidate coupling. This probability expresses the likelihood that the required output will actually be present in the workspace when the receiving Turon needs it.

The second question **is whether the available coupling is effective**. An output may be present without being functionally useful. It may carry insufficient confidence, poor quality, ambiguous meaning, incomplete metadata, excessive delay or a level of uncertainty that prevents the receiving Turon from using it normally. The coupling therefore has not only a probability of availability, but also a probability of effectiveness. This captures the difference between an input being present and an input being good enough to support the function that depends on it.

The third question is **whether the receiving Turon will in fact fire**. A Turon may have sufficient inputs and still fail to fire because of its internal state, workload, inhibition, lack of resource, degraded mode, uncertainty, control constraints or timing conditions. Function firing should therefore be treated as a separate probability, conditional on the availability and effectiveness of the required couplings. This allows the model to distinguish between a missing coupling, an ineffective coupling, and a function that was enabled but nevertheless did not perform.

The fourth question concerns **the tolerance limits of the receiving Turon**. Each Turon should have a defined tolerance profile for the variability it can absorb in its inputs. These tolerances may apply to timing, precision, completeness, quality, confidence, resource sufficiency, sequence, load, or any other domain-relevant property. If an input lies within tolerance, the Turon may fire normally. If it lies close to the tolerance boundary, the Turon may fire in

degraded or cautious mode. If it lies outside tolerance, the Turon may fail to fire or produce an altered output. This makes explicit the FRAM insight that system performance depends not only on whether functions occur, but on how much variability downstream functions can absorb.

The fifth question concerns **weighting.** A Turon may receive several similar or alternative couplings capable of satisfying the same requirement. These should not automatically be treated as equal. The receiving Turon may assign different weights to different sources according to trust, relevance, precision, authority, recency, redundancy or learned past performance. A high-confidence output from a trusted source may carry more weight than a low-confidence output from a secondary source. This allows the model to represent substitution, redundancy, conflict, dependency and source priority.

The sixth question concerns **the conditions** under which the coupling and function performance occur. CREAM Common Performance Conditions can be incorporated as contextual modifiers rather than as separate functions (Hollnagel, 1998). They can be organised in three layers: the task layer, the agent layer and the environmental or organisational layer. The task layer describes the difficulty and structure of the work, including complexity, procedures, available time and simultaneous goals. The agent layer describes the condition and capability of the performer, including training, experience, fatigue, workload and collaboration. The environmental and organisational layer describes the wider setting, including working conditions, interface quality, organisational support, resource conditions and external pressure.

These CPC layers can modify the probability that a coupling is available, the probability that the available information is effective, the probability that the Turon fires, the tolerance limits of the Turon, and the variability of the output produced. Poor task conditions may reduce tolerance. Poor interface conditions may reduce information effectiveness. Fatigue or lack of training may reduce firing reliability. Organisational pressure may change weighting, priorities or timing. The CPCs therefore become shaping factors for runtime functional performance rather than general background descriptions.

In formal terms, each candidate coupling can be represented by **an availability probability, an effectiveness probability, a tolerance acceptance function and a coupling weight**.

The receiving Turon aggregates these coupling scores according to its aspect logic, such as ALL, ANY, threshold or weighted evidence. If the aggregate score is sufficient, the Turon becomes enabled. Whether it actually fires is then determined by its firing probability, internal state and CPC-modified performance conditions. The output produced by the Turon carries its own performance signature, including timing, quality, confidence, precision, completeness and degradation status.

Estimating and Implementing Coupling Probabilities

There are several possible ways to estimate and implement the probabilities attached to Turon couplings. These should be understood as a progression from simple auditable logic to more adaptive and data-driven calibration.

The simplest implementation is a fault-tree or Boolean probability model. Each indicated coupling can be treated as a success or failure event: the required output is either available or unavailable, the information is either effective or ineffective, the input is either within or outside tolerance, and the receiving Turon either fires or fails to fire. This allows the model to use familiar AND, OR and threshold logic. For example, a Turon may require all critical inputs to be available, any one of several alternative inputs, or enough successful couplings to pass a

defined sufficiency threshold. This is transparent and suitable for a first demonstrator. Its limitation is that it can imply a sharper distinction than exists in practice and may assume independence between events that are actually related.

A more powerful implementation is Bayesian. In this case, the probability that a coupling is achieved is conditional on the state of the source Turon, the quality of the output token, the state of the receiving Turon, the timing window, the relevant CPCs and the wider system context. The probability that the receiving Turon fires can then be represented as conditional on the availability and effectiveness of its required couplings, its internal state, its tolerance limits and the prevailing task, agent and environmental conditions. This is a better fit with Turons because the model is explicitly stateful and because each runtime trace provides potential evidence for updating the probability estimates. Bayesian implementation also makes it possible to begin with expert judgement and then revise the probabilities as Work-As-Done evidence accumulates (Pearl, 1988; Gelman et al., 2013; Vehtari, Gelman and Gabry, 2017).

Matrix approaches, including DSM and adjacency-matrix representations, however, cannot be considered as the formal implementation route. They may be useful as static summaries of possible dependencies, and DSM has long been used to represent structural dependencies in complex engineering systems (Steward, 1981; Eppinger and Browning, 2012). However, such methods normally treat the model as a fixed network of relations. The Turon argument is different. A coupling is not a permanent edge between two functions; it is an achieved runtime binding that occurs only when an available output, carrying suitable timing, quality, confidence and metadata, satisfies the requirement of a waiting Turon. Matrix methods can show that one function may depend on another, but they cannot adequately show whether that dependency was realised, whether it was effective, whether it was within tolerance, or how its variability propagated. They may therefore be used as descriptive aids, but they should not be used as the formal basis for Turon variability or functional resonance.

The most ambitious implementation is a digital twin or intelligent calibration approach. Once FRAIXL, FMI and DuckDB can generate and store many realisation traces, the model can be repeatedly compared with observed Work-As-Done. The probabilities of coupling availability, information effectiveness, function firing, coupling weights, tolerance bands and CPC modifiers can then be adjusted in the light of observed performance. This may be done through structured Bayesian updating, optimisation, simulation experiments or machine-learning methods such as artificial neural networks. However, the learning layer should not replace the Turon model. Its role should be to calibrate, test and challenge the parameters of the model while the Turon structure remains the interpretable representation of functional behaviour. This is consistent with the wider digital twin idea of using a digital representation to observe, simulate and improve the behaviour of a physical or operational system over time (Grieves and Vickers, 2017; Tao, Zhang and Nee, 2019).

The practical proposal is therefore layered.

Begin with Boolean success/failure logic for coupling achievement. Extend this into conditional Bayesian probabilities as the role of metadata, tolerance and CPCs becomes clearer. Use matrix representations only as supporting structural summaries, not as the main model of realisation. Finally, use digital twin traces and machine-learning calibration to refine the probabilities, weights and tolerances against observed Work-As-Done. This gives a route from transparent first implementation to progressively more intelligent and empirically grounded Functional Systems Modelling.

Functional Resonance

Functional resonance can now be given a more formal interpretation. In FRAM, functional resonance refers to the way normal variability in multiple functions may combine and produce disproportionate system effects (Hollnagel, 2012). In Turon terms, it can be understood as the propagation, combination, damping or amplification of variable performance signatures through achieved runtime couplings. A single small variation may be absorbed by downstream tolerance. Several small variations may combine to push a Turon close to a threshold. A delayed but effective input may produce a late output that narrows the timing window for another function. A low-confidence output may be accepted but given reduced weight. A function operating under adverse CPCs may convert acceptable input variability into degraded output variability.

This means that resonance is not treated as a vague emergent effect. It becomes an inspectable pattern in the runtime trace: which coupling was available, how effective it was, how it was weighted, whether it was within tolerance, whether the function fired, and what variable output was produced.

This formal treatment should, however, be introduced progressively. There is a risk that the Turon model could become too parameter-heavy if every coupling, tolerance, weight, probability and contextual modifier is specified in detail from the outset. A practical implementation should therefore begin with a small number of high-value parameters, such as coupling availability, information effectiveness and function firing status, and then add tolerance limits, coupling weights and CPC modifiers as the model matures. The CPCs should not be treated as arbitrary adjustment factors; they need to be calibrated through domain expertise, observed Work-As-Done, operational traces, incident data or structured judgement. Similarly, coupling weights and tolerance limits should be reviewed with domain experts and, where possible, refined empirically as FRAIXL accumulates run data. The aim is not to create a mathematically over-specified model, but to provide a disciplined, inspectable and learnable structure for representing how variability arises, is absorbed, is propagated, or becomes resonant across the system.

FRAIXL Implementation

In FRAIXL this can be implemented directly. The coupling table can hold the indicated source Turon, output type, receiving Turon, receiving aspect, availability probability, effectiveness probability, tolerance limits, coupling weight and relevant CPC modifiers. The Turon table can hold the function state, firing probability, aspect logic, degradation modes, output variability model and current CPC context. The runtime trace can then record, for each cycle, which couplings were achieved, which were missing, which were ineffective, which were outside tolerance, which Turons fired, and how their outputs varied. This provides a practical route from qualitative FRAM variability to executable, auditable and eventually learnable Turon-based Functional Systems Modelling (Hollnagel, 2012; Grieves and Vickers, 2017).

A further advantage of the Turon formulation is that the state of the system can be represented instantaneously as the set of current states and metadata values of all Turons within the foreground workspace, together with the boundary Turons through which the system is coupled to its background. In FRAIXL, this can be made visible on a single spreadsheet. Each Turon can be represented as a row, with columns showing its current state, required inputs, matched inputs, missing inputs, output tokens, confidence, timing, quality, degradation mode

and firing status. The spreadsheet therefore becomes more than a model description. It becomes a live functional state surface.

This gives Functional Systems Modelling an important practical advantage. A diagram can show possible functional relationships, but the Turon state table can show the current realisation condition of the system. It turns the model from a static representation into an inspectable, updateable and auditable operational state-space. The system does not have to disappear into code. It can remain visible, editable and interpretable. This is particularly important in safety, healthcare, aviation, resilience and other sociotechnical domains, where the model must remain explainable to domain experts.

Demonstrator

The implementation does not need to begin with a complex mathematical engine. A first demonstrator could be simple. Each existing FRAM function could be translated into a basic Turon. Each Turon would have a name, a small set of required input types, a simple internal state such as waiting, enabled, fired or failed, and a list of outputs it can produce. The execution engine would maintain a list of available outputs, check which Turons are enabled, fire them, update the workspace, and record the resulting trace.

The first demonstrator could use an existing FMV/FMI model and run it twice. First, it could be run as a conventional FRAM/FMI model. Second, it could be run as a Turon realisation model. The test would be whether the Turon version reproduces the visible FMI sequence while also providing richer information about which couplings were actually achieved, which inputs were missing, which functions became enabled, and which failed to fire.

Once that simple version works, it can be extended. Inputs can carry confidence and quality. Outputs can persist, expire, degrade or be consumed. Turons can have richer internal states. Firing can become probabilistic. Degraded modes can be added. Run data can be stored in DuckDB and used to calibrate the model against observed Work-As-Done.

Insights

The Turon formulation gives us a different way to think about system behaviour. The system is not a fixed diagram of connected boxes. It is a dynamic field of functional possibilities. At any moment, some outputs are available and some functions are waiting. The realised behaviour of the system is the path created as functions become enabled, fire, change state and produce further outputs.

A drawn model may show possible relationships. The actual system realisation shows which relationships were achieved. That distinction is central. The diagram shows what might happen. The Turon trace shows what did happen.

This is particularly useful for comparing Work-As-Imagined with Work-As-Done. A Work-As-Imagined model may say that one function should provide the conditions for another. A Turon-based trace can show whether the required output was actually available, whether it arrived in time, whether it had sufficient quality, whether the receiving function recognised it, whether the expected function fired, and whether the resulting output changed the future path of the system.

This turns functional modelling into something more analytical. We can ask which Turon was waiting, which required input was missing, which output was available but not used, which coupling was expected but did not occur, which unexpected coupling did occur, which function fired late or in degraded mode, which outputs changed the future path of the system, which background conditions shaped the foreground realisation, and which exported outputs became meaningful outcomes.

This is where FRAIXL, FMV, FMI, Excel and DuckDB become important. FMV can still visualise functions and possible relationships. Excel can hold the Turon definitions, requirements, output types, rules and metadata. FMI can become the realisation engine that performs matching, coupling, firing and workspace updating. DuckDB can store the realisation traces: which Turons fired, which outputs were used, which couplings were achieved, what state changed, and what outputs were produced.

Summary

The Turon proposal can be summarised in a few central claims.

A FRAM function can be understood as a special case of a Turon. A Turon is an open executable function waiting for sufficient typed inputs. These inputs are not fixed ports connected by permanent edges; they are requirements that may be satisfied by available outputs in the system workspace. A coupling is therefore not a pre-existing line in a diagram, but an achieved runtime binding.

A Turon-based system is the realised trace of Turons firing within a defined modelling boundary. This boundary may be understood as a Markov blanket separating the foreground system from its background context. Entry and Exit Turons mediate the relationship between the foreground system and its background by importing demands, constraints, resources and metadata, and exporting outcomes, effects, records and feedback.

The important extension beyond conventional FRAM is that Turons are stateful and executable. They can be waiting, enabled, firing, delayed, degraded, inhibited, failed, confident, uncertain or learning. Their outputs carry not only values, but also metadata concerning timing, quality, confidence, precision, completeness and degradation status. This makes it possible to treat variability as a formal property of runtime system realisation.

Functional resonance is then no longer simply a qualitative description of emergent variability. It becomes the traceable propagation and interaction of variable performance signatures through achieved couplings. A single variation may be absorbed. Several variations may combine. A degraded output may be accepted but weighted differently. A late output may narrow the timing window for downstream functions. Contextual conditions may alter availability, effectiveness, tolerance or firing probability. The realised behaviour of the system is the path formed by these functional achievements.

Conclusion

The Turon is a natural development from FRAM. FRAM gives us the language of functions, aspects, couplings and variability (Hollnagel, 2012). Turons make those functions open, executable and stateful. FMI then becomes not just an interpreter of a FRAM diagram, but a realisation engine for functional systems.

The central claim is that a FRAM function is a special case of a Turon. A Turon is an open executable function waiting for sufficient typed inputs. A coupling is not a static edge, but an achieved runtime binding. A Markov blanket defines the foreground system workspace. Background Turons donate, consume, prompt or receive across the boundary. Entry and Exit Turons are active boundary Turons. A system realisation is the trace produced by repeated matching, coupling, firing, output production and state updating.

This gives Functional Systems Modelling a stronger foundation. It keeps the qualitative power of FRAM but adds a way to execute, trace, compare and eventually learn from functional behaviour. It also avoids reducing the system to components or static connections. The system is instead understood as a dynamic field of functions, conditions, outputs, background influences, achieved couplings and variable performance signatures.

In its simplest form, the Turon proposal says this: systems work when functions find the conditions they need, fire, and create the conditions for other functions to fire. What we call the system is the realised pattern of those functional achievements.

References

Eppinger, S.D. and Browning, T.R. (2012) *Design Structure Matrix Methods and Applications*. Cambridge, MA: MIT Press.

Friston, K. (2010) 'The free-energy principle: a unified brain theory?', *Nature Reviews Neuroscience*, 11, pp. 127–138.

Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A. and Rubin, D.B. (2013) *Bayesian Data Analysis*. 3rd edn. Boca Raton, FL: Chapman and Hall/CRC.

Grieves, M. and Vickers, J. (2017) 'Digital Twin: Mitigating unpredictable, undesirable emergent behavior in complex systems', in Kahlen, F.J., Flumerfelt, S. and Alves, A. (eds) *Transdisciplinary Perspectives on Complex Systems*. Cham: Springer, pp. 85–113.

Hollnagel, E. (1998) *Cognitive Reliability and Error Analysis Method: CREAM*. Oxford: Elsevier Science.

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

Hollnagel, E. (2014) *Safety-I and Safety-II: The Past and Future of Safety Management*. Farnham: Ashgate.

Kirchhoff, M.D., Parr, T., Palacios, E., Friston, K. and Kiverstein, J. (2018) 'The Markov blankets of life: autonomy, active inference and the free energy principle', *Journal of The Royal Society Interface*, 15(138), 20170792.

Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann.

Steward, D.V. (1981) 'The design structure system: A method for managing the design of complex systems', *IEEE Transactions on Engineering Management*, EM-28(3), pp. 71–74.

Tao, F., Zhang, M. and Nee, A.Y.C. (2019) *Digital Twin Driven Smart Manufacturing*. London: Academic Press.

Vehtari, A., Gelman, A. and Gabry, J. (2017) 'Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC', *Statistics and Computing*, 27, pp. 1413–1432.

Appendix A - Minimal Formal Notation

The Turon concept can be expressed using a small amount of formal notation. The purpose of the notation is not to replace the descriptive model, but to make clear what is being executed, traced and calibrated.
Let:
T = the set of Turons in the foreground system.
Each Turon is denoted:
$\mathbf{T_j}$
where *j* identifies a particular Turon.

At any runtime step *t*, the system workspace contains a set of available output tokens:
$\mathbf{O_t = \{o_1, o_2, \ldots, o_n\}}$
Each output token may be represented as:
o = (type, value, source, time, quality, confidence, metadata, status)
where *status* may include available, consumed, expired, degraded or exported.

Each Turon has a set of requirements:
$\mathbf{R_j = \{r_j^1, r_j^2, \ldots, r_j^k\}}$

These requirements correspond to the typed inputs needed by the Turon before it can become enabled. They may include information, resource, control, precondition, timing, state or background metadata requirements.

A candidate coupling can then be represented as:
$\mathbf{c_{ij}^a}$
meaning a possible coupling from an output produced by Turon $\mathbf{T_i}$ to requirement or aspect **a** of Turon $\mathbf{T_j}$.

The model may indicate that this coupling is possible:
$\mathbf{I_{ij}^a = 1}$
but this does not mean that the coupling has been achieved. An achieved coupling at runtime requires that an available output token satisfies the receiving Turon's requirement:
$\mathbf{A_{ij}^a(t) = 1}$ **if** $\mathbf{o_i(t)}$ **satisfies** $\mathbf{r_j^a}$

Otherwise:
$\mathbf{A_{ij}^a(t) = 0}$
This makes explicit the distinction between an indicated coupling and an achieved coupling. The former belongs to the model; the latter belongs to the runtime trace.

The probability that an indicated coupling is available may be written as:
$\mathbf{P(A_{ij}^a(t) = 1) = p\_avail(c_{ij}^a)}$

The effectiveness of the information or metadata carried by the coupling can be represented as:
$\mathbf{E_{ij}^a(t) \in [0, 1]}$
where 0 means that the available output is ineffective for the receiving requirement, and 1 means that it is fully effective.

The receiving Turon's tolerance for variability in that input can be represented by a tolerance function:

$\mathbf{Tol_j^a(o_i(t)) \in [0, 1]}$

This expresses whether the input lies within the receiving Turon's acceptable range for timing, precision, quality, confidence, completeness, sequence, resource sufficiency or other domain-relevant properties.

Where several similar or alternative couplings can satisfy the same requirement, each can be assigned a weight:

$\mathbf{w_{ij}^a \in [0, 1]}$

with weights normally summing to 1 across comparable alternatives:

$\mathbf{\Sigma w_{ij}^a = 1}$

The contribution of a candidate coupling to the receiving Turon's requirement can therefore be expressed as:

$\mathbf{S_{ij}^a(t) = A_{ij}^a(t) \times E_{ij}^a(t) \times Tol_j^a(o_i(t)) \times w_{ij}^a \times M\_CPC(t)}$

where **M_CPC(t)** represents the relevant Common Performance Condition modifier at that runtime step.

The receiving Turon aggregates the scores of relevant couplings according to its aspect logic:

$\mathbf{S_j^a(t) = Aggregate(S_{1j}^a, S_{2j}^a, \ldots, S_{nj}^a)}$

The aggregation rule may be ALL, ANY, threshold, weighted evidence, or another domain-specific rule.

The Turon becomes enabled when its required aspect scores satisfy its enabling rule:

$\mathbf{Enabled(T_j, t) = 1 \text{ if } \Phi_j(S_j^1, S_j^2, \ldots, S_j^k) \geq \theta_j}$

where $\mathbf{\Phi_j}$ is the Turon's sufficiency rule and $\mathbf{\theta_j}$ is its enabling threshold.

Whether the enabled Turon actually fires is represented separately:

$\mathbf{P(F_j(t) = 1) = f(Enabled(T_j, t), State_j(t), Resources_j(t), Controls_j(t), M\_CPC(t))}$

This preserves the important distinction between a Turon being enabled and a Turon actually firing.

When the Turon fires, it updates its own state and produces new output tokens:

$\mathbf{State_j(t+1) = g_j(State_j(t), Inputs_j(t), M\_CPC(t))}$

$\mathbf{O_{t+1} = O_t \cup h_j(F_j(t), Inputs_j(t), State_j(t))}$

where $\mathbf{h_j}$ is the output-generation rule for the Turon.

Each produced output carries a performance signature:

$\mathbf{V(o_j) = (timing, quality, confidence, precision, completeness, degradation, uncertainty)}$

Functional resonance can then be represented as the propagation, combination, damping or amplification of these performance signatures through achieved couplings:

$\mathbf{V(T_j, t+1) = \rho_j(V(inputs), State_j(t), Tol_j, M\_CPC(t))}$

where $\mathbf{\rho_j}$ describes how the receiving Turon absorbs, transmits, transforms or amplifies incoming variability.

This notation gives a compact formal interpretation of the Turon proposal. The system model defines possible Turons, requirements and candidate couplings. The runtime trace records which couplings were achieved, how effective they were, whether they were tolerated, how they were weighted, whether the receiving Turon fired, and what variable output was produced. Functional resonance is then represented as the traceable propagation of variable performance signatures through achieved runtime couplings.