Ontology-Grounded Capability Interaction Graphs: From Knowledge Graphs to Fault Trees

Manzi Aimé Ntagengerwa, Georgiana Caltais, Mariëlle Stoelinga, University of Twente, The Netherlands

***Abstract*—The development of Cyber-Physical Systems (CPSs) is inherently multidisciplinary, involving expertise from domains such as software engineering, electrical engineering, and mechatronics, throughout the lifecycle of the system, from design to deployment. Ensuring system reliability in Cyber-Physical Systems (CPSs) requires the identification and analysis of potential failures and their cascading effects. However, reliability modeling remains a challenging and error-prone activity, as it often depends on tacit expert knowledge, incomplete documentation of failure modes, and limited consideration of interactions between subsystems.**

To address these challenges, this paper introduce the Capability Interaction Graph (CIG), an ontology-driven representation of CPS architectures grounded in the Unified Foundational Ontology (UFO). Due to its graph-based structure, a CIG is naturally represented as a knowledge graph (KG), enabling the explicit capture of functional dependencies and system semantics.

Building upon this representation, we propose an automated synthesis algorithm for generating Fault Trees (FTs) directly from CIGs encoded as knowledge graphs. Fault Tree Analysis provides an effective mechanism for evaluating critical failure properties, including failure propagation paths and minimal cut set sets. Our approach reduces this complexity by leveraging CIGs and knowledge graphs. We provide a common semantic representation across engineering domains and support the automated generation of reliability models.

I. Introduction

Cyber-Physical Systems (CPSs) are engineered through the interaction of multiple disciplines, including software engineering, electrical engineering, mechanics, and control. During early design stages, engineers must reason about system reliability despite incomplete knowledge, heterogeneous models, and evolving architectures. Yet many reliability analyses are still performed only after substantial design decisions have already been fixed, when changes become costly and difficult to implement [16].

Fault tree analysis and knowledge graphs: Fault Tree Analysis (FTA) is one of the most widely used techniques for analyzing system reliability and safety. Fault trees (FTs) provide a structured representation of how lower-level faults combine into system-level failures, enabling analyses such as minimal cut sets, failure propagation paths, and criticality assessment. Despite the maturity of FTA algorithms, constructing fault trees remains largely manual. In practice, FT development requires extensive coordination between domain experts and depends heavily on tacit engineering knowledge. As a consequence, fault trees are expensive to maintain, difficult to integrate across subsystem boundaries, and often unavailable during the early stages of CPS design.

Model-Based Systems Engineering (MBSE) and knowledge graphs (KGs) offer an opportunity to address this limitation. Modern CPS development already produces large amounts of structured architectural information describing components, interfaces, and interactions. Knowledge graphs are particularly attractive because they provide a unified semantic representation capable of integrating heterogeneous engineering artifacts under a common ontology. However, existing approaches still lack a principled intermediate representation that connects architectural knowledge to fault propagation semantics in a formally grounded way.

Capability Interaction Graphs: This paper introduces the Capability Interaction Graph (CIG), a novel formal model for representing functional dependencies in CPS architectures. The CIG is the central contribution of this work. It provides an ontologically grounded representation of components, capabilities, channels, interfaces, functions, faults, and errors, together with the dependency structure that governs fault propagation. Unlike traditional FT models, CIGs explicitly capture the distinction between:

- Faults and errors,
- Events and situations, and
- Fault creation and fault activation.

These distinctions are essential for accurately modeling how failures emerge and propagate in cyber-physical architectures, yet they are largely absent from conventional fault tree formalisms.

The CIG serves as an intermediate semantic layer between knowledge graphs and synthesized fault trees. Starting from a minimally structured CPS architecture represented in a KG, we automatically construct a CIG that captures the system's functional dependency structure. Fault trees are then synthesized directly from the CIG semantics. This approach enables the automated generation of meaningful FT models under minimal modeling assumptions while preserving traceability to the underlying system architecture.

A key aspect of our approach is its ontological foundation. The CIG model and its supporting ontology are grounded in the Unified Foundational Ontology (UFO), providing precise semantics for notions such as capability manifestation, participation, causality, and error propagation. At the same time, the ontology remains intentionally

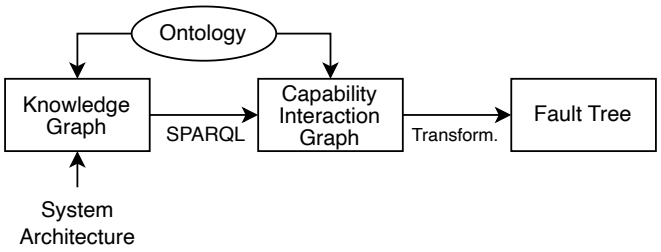


Fig. 1: Proposed pipeline for the automatic synthesis of fault trees from a knowledge graph.

lightweight so that it can absorb information from heterogeneous engineering models and remain applicable in early design stages where detailed behavioral information may not yet exist.

Thus, this paper connects foundational ontology, knowledge-graph reasoning, and fault-tree analysis into a unified framework for modeling and analyzing failures in Cyber-Physical Systems.

Figure 1 summarizes the information flow. Systems architecture descriptions, conforming to the proposed ontology, are represented as a knowledge graph. These KGs are then queried using SPARQL to obtain a CIG, and automatically transformed into fault trees suitable for standard FTA techniques.

A. Overview of the Approach

To derive fault trees from architectural knowledge represented as a knowledge graph, we proceed through the following transformations. Each introduces a new level of abstraction, adding the information required for reliability analysis.

1) **Representation of the system architecture.** We begin with a description of the system architecture consisting of components, channels, and their interactions. The ontology provides explicit semantics for concepts such as components, capabilities, functions, interfaces, faults, and errors.
   At this stage, the model describes *what the system is* and *how its parts are connected*, but it does not yet describe failure propagation.
   This information is later represented as a knowledge graph conforming to an ontology grounded in UFO.
2) **Introduce failure semantics.**
   We then define how faults, fault activations, and errors are represented within the CIG. Faults are associated with capabilities, while errors arise when expected capabilities fail to manifest. Because capabilities depend on resources produced elsewhere in the graph, an error in one location may propagate to dependent capabilities.
   This step transforms the CIG from a purely architectural model into a model that explains *how failures can spread through the system.*
3) **Extract the functional dependency structure.**
   The CIG identifies the capabilities present in the system and the dependencies between them. In particular, it captures how capabilities consume and produce resources, and how resources are exchanged through channels and interfaces.
   The CIG therefore makes explicit *which functions depend on which other functions.*
4) **Characterize the causes of errors.** Using the failure semantics, we derive an error structure function for each capability. This function recursively describes the conditions under which the capability becomes erroneous. A capability may fail because it is directly affected by a fault, or because one of its required resources are unavailable due to failures elsewhere in the CIG.
   The resulting structure provides a complete causal explanation of the capability's failure.
5) **CIGs from KGs.** Now that the structure and semantics of CIGs have been defined, we show how well-formed KGs (those that conform to the ontology) can be queried using SPARQL to construct CIGs.
6) **Synthesize a fault tree.** Finally, we translate the CIG into a fault tree using its error semantics. The capability under analysis becomes the top-level event, direct and indirect causes are recursively expanded, and logical relationships are represented by fault-tree gates.
   Through induction we show that the resulting fault tree is semantically grounded in the original architecture and can be readily analyzed using standard Fault Tree Analysis techniques such as minimal cut-set computation and criticality analysis.

In summary, the development of the paper follows the chain CIG $\rightarrow$ Failure Semantics $\rightarrow$ Error Structure Function $\rightarrow$ KG $\rightarrow$ FT.

Each step enriches the previous representation with additional semantics, ultimately transforming architectural knowledge into a form suitable for reliability analysis.

a) Running example: Figure 2 shows a subsystem of a production printer. There are three components: the power supply unit (PSU), the ink reservoir, and the print head. The PSU has the capability to provide electricity, the ink reservoir can provide ink, and the print head can deposit ink and self-clean. All of these capabilities, except for the self-cleaning of the print head, are manifested as *functions* in the presented situation. The output of the PSU and the ink reservoir reach the print head through a wire and a tube respectively.

b) Organization of the sections: In section II, we introduce explain the concepts of knowledge graphs, formal ontology and fault trees. In section III, we propose a novel ontology that describes the concepts of components, capabilities and component interactions in greater detail. In section IV we combine these ontological concepts to form the formal Capability Interaction Graph (CIG). Section V describes the dynamic evolution of CIGs over time through events, and in section VI we specifically consider fault creation and activation, and error behavior. Section VII

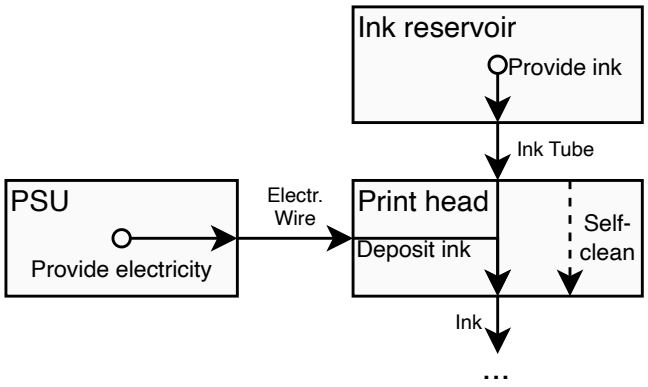


Fig. 2: A simplified diagram of a production print head, illustrating its components, capabilities and interfaces.

formalizes the notion of functional dependency. We introduce the mechanism of error propagation in section VIII, and in section IX we give the formal interpretation of error semantics in CIGs. In section X we show how CIGs are constructed from well-formed KGs. In section XI we give the semantic mapping between CIGs and FTs. In section XII, we mathematically prove completeness of the CIG-FT transformation w.r.t. preservation of error semantics. Finally, section XIII discusses related work.

II. Background

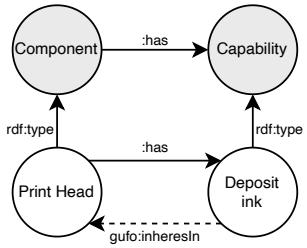


Fig. 3: A small KG that instantiates the ontology based on the running example.

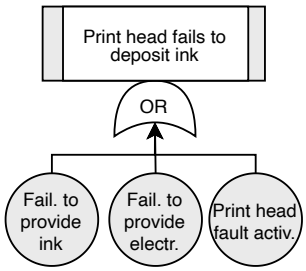


Fig. 4: A simple FT that models the failure modes of the running example.

A. Knowledge Graphs

Knowledge graphs (KGs) are a flexible knowledge-base framework that allow for the integration of knowledge from various sources, and the reasoning over known facts to infer new knowledge. KGs typically contain knowledge over a particular domain. The concepts in this domain, and how they relate to each other, can be encoded explicitly with an ontology. Rule-based fact inference then reasons over the instances of the ontology. A prominent example of a KG is Wikidata [37], the knowledge source of the online encyclopedia Wikipedia.

Figure 3 shows an example of a KG, formalizing a fragment of the diagram shown in Figure 2. The gray elements show the relevant parts of the ontology. White elements are instances, and the black lines between the white elements are factual relationships between instances.

We use gUFO [1], a lightweight OWL implementation of the Unified Foundational Ontology [12], to represent UFO concepts within RDF knowledge graphs. gUFO provides machine-readable semantics for notions such as objects, events, dispositions, and relators, enabling ontology-based reasoning over CPS architectures.

The dashed `gufo:inheresIn` association between the PSU and the capability to *produce electricity* is one we can infer: Imagine an inference rule that states

$$\forall \alpha, f.\ \text{Component}(\alpha), \text{Capability}(f) \implies \text{has}(\alpha, f) \iff \text{inheresIn}(f, \alpha) \quad (1)$$

I.e. if a component has a capability, then we say that the capability inheres in that component and vice versa. Applying this rule on the KG in Figure 3, we insert a new fact $(f, \texttt{gufo:inheresIn}, \alpha)$ that states *"The capability to produce electricity inheres in the PSU component"*.

In this work, we use a GraphDB instance as our KG for its gUFO support. GraphDB is an open standard and it has a straight-forward connection to predicate logic. GraphDB implements a Resource Description Framework (RDF) triple-store [29], which is an open W3C standard. All RDF triples are of the form (*subject*, *predicate*, *object*), where each entity is identified by its uniform resource identifier (URI) [4]. All facts in the KG have the form xPy and can be described in predicate logic as a binary predicate $P(x, y)$. RDF triple-stores are queried using SPARQL [15].

SPARQL (SPARQL Protocol and RDF Query Language) is the standard query language for retrieving and manipulating data stored in RDF graphs. SPARQL enables users to express graph-pattern queries to match triples. The language supports core operations such as selection, filtering, aggregation, optional matching, unions, and federated queries across distributed data sources. Through these capabilities, SPARQL allows both flexible exploration of heterogeneous linked data and precise extraction of structured relationships.

B. Fault Trees and Fault Tree Analysis

Fault trees (FTs) are risk models that enable fault tree analysis (FTA) [36]. There exist many variations of FTs [30]. In this work, we focus on the well-known Static FT (formally defined in subsection XI-A) that allows, among other things, for the modeling of common cause failures [17]. This makes FTs directed, acyclic graphs (DAGs) and not strictly trees, contrary to what their name suggests.

FTs model all foreseen failure modes that result in the particular top-level event (TLE) of interest. The leaves of the trees, so-called basic events (BEs) are stochastically independent events that are not decomposed further. The conditions under which these BEs propagate to result in the TLE are modeled by use of `AND` / `OR` logic gates.

Figure 4 shows an example of a simple fault tree based on the diagram in Figure 2. The TLE is a situation in which the print head fails to deposit ink. The failure of the PSU to provide electricity, the failure of the ink reservoir to provide ink, and the fault activation of the print head component are the basic events. In natural language, the FT expresses that "The print head fails to deposit ink when it is faulty, or when either the PSU or the ink reservoir fail to deliver."

Fault Tree Analysis (FTA) is a widely used deductive safety and reliability engineering technique that calculates metrics over FTs. FTA enables qualitative reasoning about system vulnerabilities by identifying minimal cut sets: the smallest combinations of basic events sufficient to cause the top event. Qualitative analysis also supports ranking critical contributors, revealing single points of failure, exposing common-cause dependencies, and improving understanding of fault propagation paths. Because of FTA's traceable logic, it remains valuable during system design, hazard assessment, and certification activities.

In addition to these structural insights, FTA also supports quantitative evaluation when failure data are available. Probabilities or failure rates can be assigned to basic events and propagated through the tree to estimate the likelihood or frequency of the top event, as well as importance measures for individual contributors. However, in many early design and model-based engineering contexts, qualitative metrics are especially useful because they can be derived without precise statistical data and still guide architectural decisions, redundancy strategies, and risk reduction efforts.

C. The Unified Foundational Ontology (UFO)

The Unified Foundational Ontology (UFO) [12] is a foundational ontology developed to provide precise semantic distinctions for conceptual modeling. UFO has been widely adopted in conceptual engineering and enterprise modeling because it offers a rigorous treatment of objects, events, dispositions and relations. The ontology is accompanied by the modeling language OntoUML [12], which provides graphical constructs grounded in UFO semantics.

UFO distinguishes between endurants and perdurants. *Endurants* are entities that are wholly present whenever they exist, such as physical components, channels, or interfaces. *Perdurants* are entities that unfold in time and possess temporal parts, such as functions or errors. In our setting, CPS components, channels and capabilities are modeled as endurants, whereas functions, fault activations, and errors are modeled as events (i.e., perdurants).

A central notion in UFO is that of moments. *Moments* are dependent entities that cannot exist without another entity, called their bearer. In particular, intrinsic moments inhere in exactly one bearer. An intuitive example is a headache – a person's headache cannot exist without the particular person (and only *that* person) in whom it inheres [14]. Formally, UFO provides the relation

$$inheresIn(m, o)$$

to denote that moment m inheres in object o. In this work, capabilities and faults are modeled as intrinsic moments that inhere in components or channels.

UFO further characterizes dispositions, which are moments that may manifest under certain triggering conditions. A capability such as "providing electricity" is therefore not itself an event, but a disposition that can be manifested through an event. UFO relates manifestations and dispositions through the relation

$$manifests(e, d)$$

meaning that event e is the manifestation of disposition d. In our model, functions are manifestations of capabilities.

Another important distinction concerns situations. A situation represents a snapshot of the world at a particular time point. Events transform one situation into another. In this work, we introduce a syntax that facilitates the description of the preconditions and postconditions of events:

$$\Gamma \xrightarrow{e} \Gamma'$$

where event e begins in situation Γ and brings about situation Γ'. This treatment allows us to formalize fault creation, fault activation, and error propagation as temporally ordered events.

UFO also provides a formal treatment of causality. An event may bring about a situation that subsequently triggers another event, thereby establishing a causal chain. This distinction is particularly important in reliability modeling because it separates:

- the creation of a fault,
- the activation of a fault under operational conditions,
- the resulting error state.

Traditional fault tree approaches often conflate these notions, whereas UFO enables them to be represented explicitly.

Finally, UFO distinguishes between objects, roles, relators, and events. We exploit these distinctions in the proposed ontology:

- components and channels are modeled as objects,
- capabilities and faults are modeled as dispositions,
- producer and consumer are modeled as roles,
- interfaces are modeled as relators mediating interactions, and
- functions, errors, and fault activations are modeled as events.

Grounding the proposed CIG model in UFO provides precise semantics for participation, manifestation, dependency, and causality, while remaining lightweight enough for early-stage CPS architecture modeling.

III. The Ontological Foundations of Capability Interaction Graphs

We use OntoUML to formally visualize the elements of CIGs in Figure 5.We argue that this conceptualization is small enough to capture knowledge in the earliest stages of CPS design, yet generic enough to capture concepts

across modeling languages and engineering domains, and sufficiently rich for the synthesis of meaningful fault trees.

Note that not all boxes in this diagram need to be provided by a user instantiating a CIG. The only input we strictly require (for the purpose of any *meaningful* fault tree synthesis) is a set of components, channels, and a set of labeled edges that connect them. All the other concepts can implicitly be instantiated under a fixed set of assumptions.

A. Formal notation

Central to the CIG model is UFO's notion of situations. In essence, a situation describes a system under study at a single point in time. By convention, we here denote a situation Γ.

Furthermore, we use some standard mathematical notations for our formal definitions. We introduce them here for convenience. Let proj_i be the function that maps a tuple to its i^{th} element. I.e. let T be a tuple, such that

$$T = (x_1, \ldots, x_n)$$

Then

$$\text{proj}_i\ T = x_i$$

Furthermore, we use the common FOL syntactic shorthand for quantifying over members of a set:

$$\forall x \in S.\ \phi(x) \equiv \forall x.\big(S(x) \implies \phi(x)\big)$$
$$\exists x \in S.\ \phi(x) \equiv \exists x.\big(S(x) \wedge \phi(x)\big)$$

Lastly, let $\mathscr{P}$ denote the finite powerset (as we only work with finite sets).

B. Concepts and formal definitions

Endurants (such as objects) are *present in* a situation. E.g. the apple that fell on Sir Isaac Newton's head was present at the moment of impact. That same apple is not present now. We will introduce the elements of a CIG in the context of a particular situation. It suffices to know that this means *the elements of the system that are present at that particular time point*. In this section, we introduce sets of individuals that "are present". For instance, we collect components in the set $\mathcal{C}$ and we indicate the situation in which these objects are present by means of a subscript. I.e. the set $\mathcal{C}_\Gamma$ collects exactly those object which are components and that are present in the situation Γ.

In contrast, perdurants (events) *exist throughout* a time interval. E.g. the event of that particular apple falling began in the situation where it had just detached from the stem, and ended in the situation that it reached Sir Newton's cranium (all the while accumulating temporal parts). Section V formalizes this notion in the context of CIGs.

The bold-print terms introduced below refer to the endurant concepts shown in Figure 5. We use the running example in Figure 2 to help the reader in building an intuition of these concepts.

Resource Types are *higher-order types* that denote the parameter and output types of a function. When instantiating the model, instances of *Resource Type* (e.g. *paper* or *electricity*) are types themselves. To be precise, the instances of resource types are *substantial types* (types whose instances are founded on matter). The assertion that a type is a resource type is given by the unary predicate $\mathit{ResourceType} : \mathit{Individual} \to \{\top, \bot\}$. This predicate comes from the ontology. Looking at Figure 2, we find that $\mathit{ResourceType}(\texttt{Ink}) = \top$, and $\mathit{ResourceType}(\texttt{PSU}) = \bot$.

Resource types generalize any form of energy in the sense of [10], such as materials, fluids or signals. This broad definition covers many of the fluids, gases, control signals, etc. that are involved in the design of a CPS architecture.The only meaningful semantics of resource types are conveyed by their relationships to ontologically richer concepts like capabilities and channels. Figure 2 shows two resource types; *electricity* and *ink*.

Definition 1 (Resource Types)**.** The set of resource types, denoted by the set $\mathcal{R} = \{r \mid \text{ResourceType}(r)\}$ represents a higher-order type.

Elements of $\mathcal{R}$ are sortal *types* themselves. We do not consider individuals that instantiate resource types in this work. For instance, $\mathit{Page} \in \mathcal{R}$ is a valid resource type, but any particular sheet of paper (e.g. *page 42 of my copy of "A Room of One's Own"*) is not. As a result, resource types are purely symbolic elements in $\mathcal{R}$, whose symbol *is* the (nominal) identity of the type. I.e., the elements $\text{Page} \in \mathcal{R}$ and $\text{Ink} \in \mathcal{R}$ are distinct because their symbols are distinct. Likewise, the elements $\text{Ink} \in \mathcal{R}_\Gamma$ and $\text{Ink} \in \mathcal{R}_{\Gamma'}$ are the same type, because their symbols are identical. For this reason, we generally do not subscript $\mathcal{R}$ with a situation.

Capabilities are *dispositions* (or more generally; *modes*) that describe the possible behavior of a component. They inhere in exactly <u>one</u> component. Dispositions are intrinsic moments that are existentially dependent on their bearer. They are particular, i.e. one capability cannot inhere in two distinct bearers.

A capability can be manifested as a function. We distinguish the *Functional* and the *Erroneous* phases of capabilities. A capability is in the erroneous phase only during the existence of an error over that capability (and otherwise it is in the functional phase). The generalization set of these phases is complete and disjoint (components are either functional or erroneous, but not both). The assertion that an individual is a capability is given by the unary predicate $\mathit{Capability}_\Gamma : \mathit{Individual} \to \{\top, \bot\}$. In Figure 2, we have $\mathit{Capability}(\texttt{Provide electricity}) = \top$ and $\mathit{Capability}(\texttt{Ink}) = \bot$.

The assertion that a capability is in the functional or erroneous phase is given by $\mathit{FunctionalCapability}_\Gamma : \mathit{Individual} \to \{\top, \bot\}$ and $\mathit{ErroneousCapability}_\Gamma : \mathit{Individual} \to \{\top, \bot\}$ respectively.

In Figure 2, the capabilities are to *provide electricity*, to *provide ink*, and to *deposit ink* and *self-clean*. They inhere

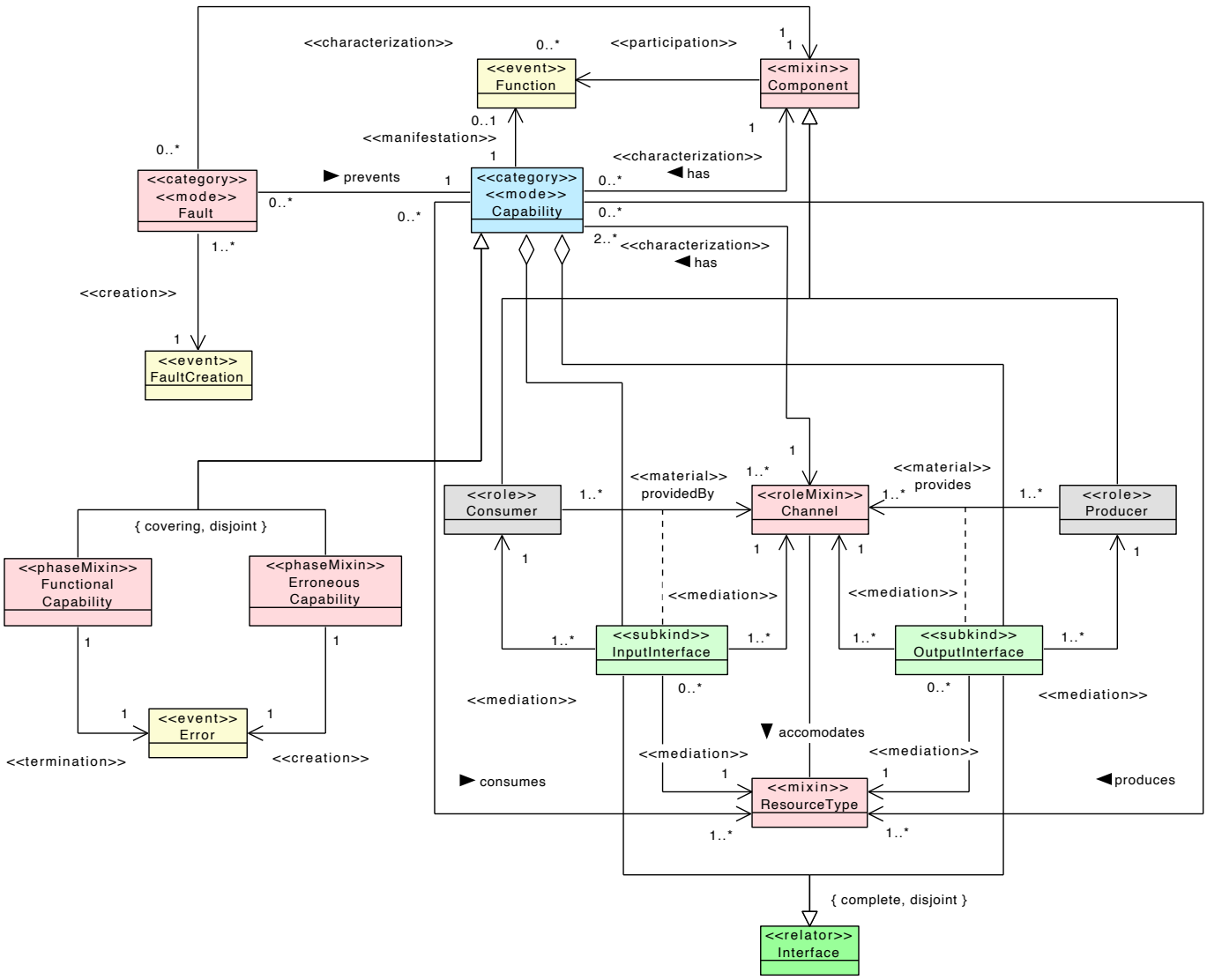


Fig. 5: Our proposed ontology of CPS architecture, introducing the notions of components, capabilities, functions, channels, etc.

in the PSU, ink reservoir and print head respectively. The capability to *self-clean* is not manifested in the presented situation.

Definition 2 (Capabilities)**.** Capabilities $\mathscr{C}_\Gamma \subseteq \text{CapabilityId}_\Gamma \times \mathscr{P}(\mathcal{R}) \times \mathscr{P}(\mathcal{R})$ is a set of tuples, where the first element ranges over $\text{CapabilityId}_\Gamma$.

We define the bijective function ι_Γ that maps the elements in $\mathscr{C}_\Gamma$ to an individual in the ontology.

$$\iota_\Gamma : \mathscr{C}_\Gamma \to \text{CapabilityId}_\Gamma$$

Its inverse is denoted ι_Γ^{-1}.

The second and third element of the tuple respectively denote the resource types its bearing component consumes and produces when the capability is manifested.

Capabilities inhere in a unique object (**D1**, Table III):

$$\forall f \in \mathscr{C}_\Gamma.\ \text{inheresIn}_\Gamma(\iota_\Gamma(f), \beta(\iota_\Gamma(f)))$$

Functions are *atomic events* that are the manifestation of a capability. The bearing component of that capability participates in the function event. UFO-B [14] defines that the begin point of this manifestation is the end point of the activation of the capability it manifests. The conditions for this activations must therefore be met at this begin point, as well as for the entire duration of the function. Such conditions are given by the capability's *consumes* relation.

A function effects a transformation of resources, describing the behavior of a component. In this work, the mechanics of such transformations are not explicitly modeled. The assertion that an individual is a function is given by the unary predicate $\mathit{Function}_\Gamma : \mathit{Individual} \to \{\top, \bot\}$.

Definition 3 (Functions)**.** The set $\mathbb{F}_\Gamma = \text{FunctionId}_\Gamma$ is a set of function identifiers. Elements of this set denote which capabilities are currently manifested as a function event.

The set $\mathbb{F}_\Gamma$ is exactly the set of symbols that represent a function event in Γ:

$$\mathbb{F}_\Gamma = \left\{ \bar{f} \mid \text{Function}_\Gamma(\bar{f}) \right\}$$

The set FunctionId_Γ is derived from the set of capability identifiers $\mathit{CapabilityId}_\Gamma$ by a partial bijection

$$\varrho_\Gamma : \text{CapabilityId}_\Gamma \to \mathbb{F}_\Gamma$$

and

$$\text{dom}\varrho_\Gamma = \{c \mid \text{Capability}_\Gamma(c) \wedge \exists e : \text{Event}.\ \text{manifests}(e, c)\} \quad (2)$$

and

$$\forall f \in \text{dom}\varrho_\Gamma . \forall \bar{f} \in \mathbb{F}_\Gamma . \quad \varrho_\Gamma(f) = \bar{f} \iff \text{manifests}(\bar{f}, f) \quad (3)$$

Where $\text{dom}\varrho_\Gamma$ denotes the domain of ϱ_Γ. Furthermore, since ϱ_Γ is a partial bijection, *if* a function manifests a capability, it is the *unique* manifestation of that capability in Γ.

Let $\varrho_\Gamma^{-1} : \mathit{FunctionId}_\Gamma \to \mathit{CapabilityId}_\Gamma$ be the inverse function of ϱ that returns the unique capability that a given function manifests (**D2**, Table III):

$$\forall \bar{f} \in \mathbb{F}_\Gamma .\ \text{manifests}(\bar{f}, \varrho_\Gamma^{-1}(\bar{f}))$$

Note that ϱ_Γ^{-1} is injective where $\text{dom}\varrho_\Gamma^{-1} = \mathbb{F}_\Gamma$.

In our notation, we use symbol names to indicate the manifestation relationship between a capability and a function. Bar-notation indicates that $\bar{f}$ is a function that manifests the capability f.

Faults are *modes* that describe the possible prevention of the manifestation of a component's capabilities. They inhere in exactly <u>one</u> component. The assertion that an individual is a fault is given by the unary predicate $\mathit{Fault} : \mathit{Individual} \to \{\top, \bot\}$.

Definition 4 (Faults)**.** Faults $\mathscr{F}_\Gamma \subseteq \text{FaultId}_\Gamma \times \mathscr{P}(\text{CapabilityId}_\Gamma)$ are modes that inhere in a component and prevent the manifestation of (some of) that component's capabilities.

We denote by $\iota_\Gamma : \mathscr{F}_\Gamma \to \mathit{FaultId}_\Gamma$ the bijective function that maps the elements in $\mathscr{F}_\Gamma$ to an individual in the ontology. Its inverse is denoted ι_Γ^{-1}.

A fault prevents the manifestation of certain capabilities:

$$\forall \Gamma : \mathcal{G} . \forall (\text{id}_{\not f}, C) \in \mathscr{F}_\Gamma . \forall \text{id}_f \in C .\ \text{id}_f \notin \text{dom}\varrho_\Gamma$$

A fault can only prevent the manifestation of capabilities that inhere in the bearer of that fault:

$$\forall \Gamma : \mathcal{G} . \forall (\text{id}_{\not f}, C) \in \mathscr{F}_\Gamma . \forall f \in C .\ \beta(f) = \beta(\text{id}_{\not f}) \quad (4)$$

We may denote a fault with the striked-out symbol of any of the capabilities whose manifestation is prevented by that fault. I.e. $\not f \in \mathscr{F}_\Gamma$ denotes that the manifestation of capability $f \in \mathscr{C}_\Gamma$ is prevented.

The **Component** type is a *mixin* that describe physical entities with one or more capabilities. We also distinguish the *Consumer* and *Producer* roles. The generalization set of these roles if not disjoint and not complete. These roles are respectively brought about by a material *providedBy* and *provides* relation to a channel. The assertion that an individual is a component is given by the unary predicate $\mathit{Component} : \mathit{Individual} \to \{\top, \bot\}$. For instance, in Figure 2, we find that $\mathit{Component}(\texttt{PSU}) = \top$ and $\mathit{Component}(\texttt{Provide electricity}) = \bot$.

In this work, we do not make an ontological commitment to the parthood of components or the definition of systems. Fault tree synthesis and analysis does not necessarily depend on the mereology of systems, and to keep the ontology minimal we do not involve it.

We adopt an atomistic worldview [9] and define all components to be mereologically atomic, i.e. components do not have proper parts. A common refrain is *"but what is a component to one, is a system to another"*. This is not necessarily incompatible with our perspective. In fact, it is one of the key principles that allows for the modeling of an entire "print head" as a single component, constrained only by its interfaces to other other components.

In Figure 2, the components are the *PSU*, the *ink reservoir*, and the *print head*.

Definition 5 (Components)**.** Components are tuples of the form

$$\mathcal{C}_\Gamma \subseteq \text{ComponentId}_\Gamma \times \mathscr{P}(\mathscr{C}_\Gamma) \times \mathscr{P}(\mathbb{F}_\Gamma) \times \mathscr{P}(\mathscr{F}_\Gamma)$$

where $\text{ComponentId}_\Gamma$ is a set of component identifiers.

Let $\alpha = (\text{id}_\alpha, \mathscr{C}_\Gamma^\alpha, \mathbb{F}_\Gamma^\alpha, \mathscr{F}_\Gamma^\alpha)$ be a component. Then:

$\mathscr{C}_\Gamma^\alpha \in \mathscr{P}(\mathscr{C}_\Gamma)$ is the set of capabilities that inhere in α, and;

$\mathbb{F}_\Gamma^\alpha \in \mathscr{P}(\mathbb{F}_\Gamma)$ is the set of functions that manifest a capability of α, and;

$\mathscr{F}_\Gamma^\alpha \in \mathscr{P}(\mathscr{F}_\Gamma)$ is the set of faults that inhere in α.

We define the bijective function

$$\iota_\Gamma : \mathcal{C}_\Gamma \to \text{ComponentId}_\Gamma$$

that maps components to an instance of *Component* in the situation Γ, and that preserves the mixed identity principles provided by the specializations of the *Component* mixin across different situations. Its inverse is denoted ι_Γ^{-1}.

We denote by $\mathscr{C}_\Gamma^\alpha$ the set of capabilities that inhere in the component $\alpha \in \mathcal{C}_\Gamma$ in situation Γ:

$$\mathscr{C}_\Gamma^\alpha = \{ f \mid f \in \mathscr{C}_\Gamma .\ \iota_\Gamma(\alpha) = \beta(\iota_\Gamma(f)) \}$$

We denote by $\mathbb{F}_\Gamma^\alpha$ the set of functions that manifest a capability of the component $\alpha \in \mathcal{C}_\Gamma$ in the situation Γ:

$$\mathbb{F}_\Gamma^\alpha = \left\{ \bar{f} \mid \bar{f} \in \mathbb{F}_\Gamma .\ \iota_\Gamma(\alpha) = \beta_\Gamma(\varrho_\Gamma^{-1}(\bar{f})) \right\}$$

We denote by $\mathscr{F}_\Gamma^\alpha \subseteq \mathscr{F}_\Gamma$ the set of faults that inhere in the component $\alpha \in \mathcal{C}_\Gamma$ in the situation Γ:

$$\mathscr{F}_\Gamma^\alpha = \{ \not f \mid \iota_\Gamma(\alpha) = \beta_\Gamma(\not f) \}$$

Channels are *role mixins* that interface with a component. The channel role is grounded in the association to a resource type that the channel is said to accommodate. Channels can transport only the resource types they accommodate. The electrical wire that connects the `PSU` and the `Print head` in Figure 2 is an example of a channel. The transport of each resource type is made up of two *distinct* capabilities: a pure consumption capability to *consume* that resource, and a pure production capability to *produce* that resource. Since we only discuss resource types in this work, we do not associate a quantity with this capacity. The predicate *Channel* : *Individual* $\rightarrow \{\top, \bot\}$ asserts whether an individual is a channel.

Definition 6 (Channels)**.** Channels $\vec{\mathcal{C}_\Gamma} \subseteq \text{ChannelId}_\Gamma \times \mathscr{P}(\mathcal{R}) \times \mathscr{P}(\mathscr{C}_\Gamma) \times \mathscr{P}(\mathbb{F}_\Gamma) \times \mathscr{P}(\mathscr{F}_\Gamma)$ are tuples that relate an individual channel to the resource types it accommodates, where ChannelId_Γ is a set of channel identifiers.

Let the function ι_Γ denote the bijective function that maps the elements in $\vec{\mathcal{C}}$ to an individual in the ontology.

$$\iota_\Gamma : \vec{\mathcal{C}_\Gamma} \rightarrow \text{ChannelId}_\Gamma$$

Its inverse is denoted ι_Γ^{-1}.

We denote by $\mathscr{C}_\Gamma^c$ the set of capabilities that inhere in the channel $c \in \vec{\mathcal{C}_\Gamma}$ in situation Γ:

$$\mathscr{C}_\Gamma^c = \{f \mid f \in \mathscr{C}_\Gamma.\ \iota_\Gamma(c) = \beta(\iota_\Gamma(f))\}$$

We denote by $\mathbb{F}_\Gamma^c$ the set of functions that manifest a capability of the channel $c \in \vec{\mathcal{C}_\Gamma}$ in situation Γ:

$$\mathbb{F}_\Gamma^c = \left\{\bar{f} \mid \bar{f} \in \mathbb{F}_\Gamma.\ \iota_\Gamma(c) = \beta_\Gamma(\varrho_\Gamma^{-1}(\bar{f}))\right\}$$

We denote by $\mathscr{F}_\Gamma^c \subseteq \mathscr{F}_\Gamma$ the set of faults that inhere in the channel $c \in \vec{\mathcal{C}_\Gamma}$ in the situation Γ:

$$\mathscr{F}_\Gamma^c = \{\mathscr{f} \mid \iota_\Gamma(c) = \beta_\Gamma(\mathscr{f})\}$$

Let the function $\text{cons}_\Gamma : \vec{\mathcal{C}_\Gamma} \times \mathcal{R} \rightarrow \mathscr{C}_\Gamma$ denote the unique input capability that allows a channel's consumption of a resource type. Let the function $\text{prod}_\Gamma : \vec{\mathcal{C}_\Gamma} \times \mathcal{R} \rightarrow \mathscr{C}_\Gamma$ denote the unique output capability that allows a channel's production of a resource type.

Let $c \in \vec{\mathcal{C}_\Gamma}$ be a channel, and let $r \in \mathcal{R}$ be a resource type that c accommodates. Let $h = \text{cons}_\Gamma(c, r)$. Then by definition, h must be unique:

$$h \in \mathscr{C}_\Gamma^c \wedge (\neg\exists h' \in \mathscr{C}_\Gamma^c.\ h' = \text{cons}_\Gamma(c, r) \wedge h \neq h')$$

Furthermore, h must – if it exists – refer to the same capability across situations:

$$\forall \Gamma, \Gamma' : \mathcal{G}.\ (h = \text{cons}_\Gamma(c, r)) \implies ((\exists h' \in \mathscr{C}_{\Gamma'}.\ \iota_\Gamma(h) = \iota_{\Gamma'}(h')) \implies h = \text{cons}_{\Gamma'}(c, r))$$

The same definitions hold for $k = \text{prod}_\Gamma(c, r)$.

The production and consumption capabilities must actually consume and produce the correct respective resources. Let $c \in \vec{\mathcal{C}_\Gamma}$ be a channel, and let $r \in \mathcal{R}$ be a resource type that c accommodates in the situation Γ. Let $h = \text{cons}_\Gamma(c, r)$ and let $k = \text{prod}_\Gamma(c, r)$. Then:

$$\bigwedge_{x \in \{h,k\}} \text{consumes}_\Gamma(x, r) \wedge \text{produces}_\Gamma(x, r)$$

Furthermore, this resource type r is unique. I.e. there exists no $r' \in \mathcal{R}$ s.t.:

$$r' \neq r \wedge \bigvee_{x \in \{h,k\}} \left(\text{consumes}_\Gamma(x, r') \vee \text{produces}_\Gamma(x, r')\right)$$

The consumption and production capabilities in a channel $c \in \vec{\mathcal{C}_\Gamma}$ are distinct. Let $c \in \vec{\mathcal{C}_\Gamma}$ be a channel, and let $r \in \mathcal{R}$ be a resource type that c accommodates in the situation Γ. Then:

$$\iota_\Gamma(\text{cons}_\Gamma(c, r)) \neq \iota_\Gamma(\text{prod}_\Gamma(c, r))$$

Interfaces are *relators* that describe an interaction between a component and a channel through some resource. They aggregate a component's and a channel's externally dependent modes that are founded by the foundation *an*[1] interface, resulting in the material relations *providedBy* (InputInterfaces) and *provides* (OutputInterfaces). I.e., the component α has the role *consumer of resource* r iff there exists a channel c such that an input interface *in* materializes a *providedBy* relationship that mediates that resource r through c. Similarly, the component α has the role *producer of resource* r iff there exists a channel c such that an output interface *out* materializes a *provides* relationship that mediates that resource r through c.

We can compose functions such that the output of one component's function serves as the input of another component's function. Such patterns describe components exchanging a resource through a channel. We sometimes call this pattern a *resource exchange*.

In Figure 2, there is an exchange of *electricity* between the PSU's *provide electricity* function and the print head's *deposit ink* function. There is also a resource exchange between the ink reservoir's *provide ink* and the print head's *deposit ink*.

Definition 7 (Output interfaces)**.** An output interface $\mathcal{O}_\Gamma \subseteq \text{ComponentId}_\Gamma \times \text{ChannelId}_\Gamma \times \mathcal{R}$ is the relator that the *provides* association between a producer component and a channel is derived from.

An output interface relates a component, a channel and the resource type that the component provides to the channel.

Let $\varpi_\Gamma^o : \mathcal{O}_\Gamma \rightarrow \mathcal{P}(\text{CapabilityId}_\Gamma)$ be the function that returns the set of capabilities that make up the component's qua-individual *component-α-qua-provider-of-o*, such that

$$\forall \Gamma\ :\ \mathcal{G}.\forall o\ \in\ \mathcal{O}_\Gamma.\forall \text{id}_f\ \in\ \varpi_\Gamma^o(o).\ \beta(\text{id}_f)\ =\ \text{proj}_1(o)$$

[1] The founding interface is not necessarily *this* interface.

Let $\varsigma^o_\Gamma : \mathcal{O}_\Gamma \to \text{CapabilityId}_\Gamma$ be the function that returns the unique capability that makes up the channels's qua-individual *channel-c-qua-consumer-of-o*, such that

$$\forall \Gamma : \mathcal{G}. \forall o \in \mathcal{O}_\Gamma. \\ \varsigma^o_\Gamma(o) = \iota_\Gamma(\text{cons}(\iota^{-1}(\text{proj}_2(o)), \text{proj}_3(o)))$$

The identity of an output interface is derived from the sum of the qua-entities it aggregates, given by

$$\iota^o_\Gamma : \mathcal{O}_\Gamma \to \text{InterfaceId}_\Gamma$$

Its inverse is given by $\iota^{o,-1}_\Gamma$.

Lastly, the output interface relator is existentially dependent on the manifestations of the capabilities that make up its qua-entities:

$$\forall \Gamma : \mathcal{G}. \forall o \in \mathcal{O}_\Gamma. \forall \text{id}_f \in (\varpi^o_\Gamma(o) \cup \{\varsigma^o_\Gamma(o)\}) . \\ \exists \bar{f} \in \mathbb{F}_\Gamma.\ \varrho_\Gamma(\text{id}_f) = \bar{f}$$

Definition 8 (Input interfaces)**.** An input interface $\mathcal{I}_\Gamma \subseteq \text{ChannelId}_\Gamma \times \text{ComponentId}_\Gamma \times \mathcal{R}$ is the relator that the *providedBy* association between a channel and a consumer component is derived from.

An input interface relates a channel, a component and the resource type that the channel provides to the component.

Let $\varpi^i_\Gamma : \mathcal{I}_\Gamma \to \mathscr{P}(\text{CapabilityId}_\Gamma)$ be the function that returns the set of capabilities that make up the component's qua-individual *component-α-qua-provided-by-i*, such that

$$\forall \Gamma : \mathcal{G}. \forall i \in \mathcal{I}_\Gamma. \forall \text{id}_f \in \varpi^i_\Gamma(i).\ \beta(\text{id}_f) = \text{proj}_2(i)$$

Let $\varsigma^i_\Gamma : \mathcal{I}_\Gamma \to \text{CapabilityId}_\Gamma$ be the function that returns the unique capability that makes up the channel's qua-individual *channel-c-qua-provider-of-i*, such that

$$\forall \Gamma : \mathcal{G}. \forall i \in \mathcal{I}_\Gamma. \\ \varsigma^i_\Gamma(i) = \iota_\Gamma(\text{prod}(\iota^{-1}(\text{proj}_1(i)), \text{proj}_3(i)))$$

The identity of an input interface is derived from the sum of the qua-entities it aggregates, given by

$$\iota^i_\Gamma : \mathcal{I}_\Gamma \to \text{InterfaceId}_\Gamma$$

Its inverse is given by $\iota^{i,-1}_\Gamma$.

Lastly, like with output interfaces, the input interface relator is existentially dependent on the manifestations of the capabilities that make up its qua-entities:

$$\forall \Gamma : \mathcal{G}. \forall i \in \mathcal{I}_\Gamma. \forall \text{id}_f \in \left(\varpi^i_\Gamma(i) \cup \left\{\varsigma^i_\Gamma(i)\right\}\right) . \\ \exists \bar{f} \in \mathbb{F}_\Gamma.\ \varrho_\Gamma(\text{id}_f) = \bar{f}$$

Expectations. Expectations describe what capabilities *should* be manifested as functions. In this work, we do not explore the justification of *why* a particular capability's manifestation should be expected, nor do we explicitly describe *who* has such expectations. E.g. we may expect the capability *to provide electricity* to be manifested. If it is not – i.e. if this expectation is not met – we call this an *error*.

Definition 9 (Expectations)**.** A set of expectations $\mathscr{E}_\Gamma \subseteq \text{CapabilityId}_\Gamma$ is a subset of capability identifiers.

We say a situation *meets* an expectation if and only if the expected capability is manifested by a function in that situation:

$$\forall \Gamma : \mathcal{G}. \forall e \in \mathscr{E}_\Gamma.\ \text{meets}(\Gamma, e) \iff e \in \text{dom}\varrho_\Gamma$$

Note that, given Definition 9, a CIG can still meet all expectations when the system "does more than expected".

On the sets $\mathscr{C}_\Gamma$, $\mathbb{F}_\Gamma$ and $\mathscr{F}_\Gamma$.

Capabilities inhere in objects. In this work, the only objects we distinguish are components and channels. Hence, all capabilities must inhere in either a component or a channel. The union of component capabilities and channel capabilities therefore equals the set of capabilities $\mathscr{C}_\Gamma$:

$$\forall \Gamma : \mathcal{G}.\ \mathscr{C}_\Gamma = \left(\bigcup_{\alpha \in \mathcal{C}_\Gamma} \mathscr{C}^\alpha_\Gamma\right) \cup \left(\bigcup_{c \in \vec{\mathcal{C}}_\Gamma} \mathscr{C}^c_\Gamma\right) \tag{5}$$

Functions are manifestations of exactly one capability, which in turn inhere in exactly one object. Like with capabilities, the union of component functions and channel functions therefore equals the set of functions $\mathbb{F}_\Gamma$:

$$\forall \Gamma : \mathcal{G}.\ \mathbb{F}_\Gamma = \left(\bigcup_{\alpha \in \mathcal{C}_\Gamma} \mathbb{F}^\alpha_\Gamma\right) \cup \left(\bigcup_{c \in \vec{\mathcal{C}}_\Gamma} \mathbb{F}^c_\Gamma\right) \tag{6}$$

Faults inhere in exactly one object. Like with capabilities, the union of component faults and channel faults therefore equals the set of faults $\mathscr{F}_\Gamma$:

$$\forall \Gamma : \mathcal{G}.\ \mathscr{F}_\Gamma = \left(\bigcup_{\alpha \in \mathcal{C}_\Gamma} \mathscr{F}^\alpha_\Gamma\right) \cup \left(\bigcup_{c \in \vec{\mathcal{C}}_\Gamma} \mathscr{F}^c_\Gamma\right) \tag{7}$$

Recall that CIGs permit components that play the role of a channel. Therefore, the families of sets $\bigcup_{\alpha \in \mathcal{C}_\Gamma} \mathscr{C}^\alpha_\Gamma$, $\bigcup_{c \in \vec{\mathcal{C}}_\Gamma} \mathscr{C}^c_\Gamma$, and $\left\{\bigcup_{\alpha \in \mathcal{C}_\Gamma} \mathbb{F}^\alpha_\Gamma, \bigcup_{c \in \vec{\mathcal{C}}_\Gamma} \mathbb{F}^c_\Gamma\right\}$, and $\left\{\bigcup_{\alpha \in \mathcal{C}_\Gamma} \mathscr{F}^\alpha_\Gamma, \bigcup_{c \in \vec{\mathcal{C}}_\Gamma} \mathscr{F}^c_\Gamma\right\}$ are not *necessarily* partitions of the sets $\mathscr{C}_\Gamma$, $\mathbb{F}_\Gamma$ and $\mathscr{F}_\Gamma$ respectively.

IV. Capability Interaction Graphs

In this section we combine the individual concepts of CPS architecture from the previous section into one large model: the CIG. We use this formalization in the next section, to show how these CIGs can dynamically change over time.

In this section we also present a graphical notation, turning CIGs into readable diagrams which will be useful for examples throughout the paper. Table I provides a glossary of endurant CIG elements and functions.

Notation	Description
$\Gamma : \mathcal{G}$	The factual situation Γ
$\mathcal{C}_\Gamma, \vec{\mathcal{C}_\Gamma}, \ldots$	The set of components, channels, etc. in the situation Γ
ι_Γ	The function that returns the ontological identity of a given CIG element
$\varrho_\Gamma(f)$	The function that returns the Function that manifests a given capability manifests in the situation Γ
$f, \bar{f}, \not{f}$	A capability, a Function manifesting that capability, and a fault preventing the manifestation of that capability resp.
$\text{cons}_\Gamma(c, r)$	The function that returns the unique input capability that allows the channel c's consumption of the resource type r
$\text{prod}_\Gamma(c, r)$	The function that returns the unique output capability that allows the channel c's production of the resource type r
$\varpi^o_\Gamma(o), \varpi^i_\Gamma(i)$	The functions that return, for a given interface, the set of capabilities that make up the component's qua-individual
$\varsigma^o_\Gamma(o), \varsigma^i_\Gamma(i)$	The functions that return, for a given interface, the unique capability that makes up the channels's qua-individual

TABLE I: A glossary of endurant CIG elements and functions.

ID	Definition	Description
10 [12]	$\beta : \text{IntrinsicMoment} \to \text{Individual}$	The function that returns the bearer of a particular intrinsic moment.
a102 [13]	$\text{manifests}(x, y) \implies \text{Perdurant}(x) \wedge \text{Endurant}(y)$	Relates an event to the particular moment it is a manifestation of.

TABLE II: The relevant portion of UFO, taken from [12] and [13].

A. Formal definition

A CIG is a tuple composed of **capabilities** ($\mathscr{C}$), **functions** ($\mathbb{F}$), **faults** ($\mathscr{F}$), **components** ($\mathcal{C}$), **channels** ($\vec{\mathcal{C}}$), **output interfaces** ($\mathcal{O}$), **input interfaces** ($\mathcal{I}$) and **expectations** ($\mathscr{E}$). Such a tuple represents the state of a system, and is typically denoted γ. The type of a CIG is denoted $\mathcal{G}$, and we write $\gamma \in \mathcal{G}$ for instances of $\mathcal{G}$.

We base our notion of time points on that introduced in UFO-B [14]. Simply put, the set *TimePoint* and the *precedes* relation make up a strict total order. We may assign a time point to the state γ. We then write $\Gamma : \mathcal{G}$.

Furthermore, we may denote by a subscript the CIG to which a set belongs. E.g. we mean by $\mathcal{C}_\Gamma$ the set of components that belong to some CIG $\Gamma : \mathcal{G}$.

Additionally, in the places where it is obvious, we do not write the particular situation in which predicates are said to hold. For instance, when we write

$$\forall \alpha \in \mathcal{C}_\Gamma.\ \mathit{Component}(\iota(\alpha))$$

we mean that α is *present at* the time point at which the situation Γ obtains, and that the predicate $\mathit{Component}(\iota_\Gamma(\alpha)$ holds in that situation. We describe time points and situations in more detail in section V.

The formal definition of CIGs is grounded in UFO-A and UFO-B predicates and functions. For reference, the relevant portions of UFO-A are described in Table II. Table III lists the relevant portion of UFO-B.

Definition 10. Consider the set $\mathcal{G} = \mathscr{P}(\mathscr{C}) \times \mathscr{P}(\mathbb{F}) \times \mathscr{P}(\mathscr{F}) \times \mathscr{P}(\mathcal{C}) \times \mathscr{P}(\vec{\mathcal{C}}) \times \mathscr{P}(\mathcal{O}) \times \mathscr{P}(\mathcal{I}) \times \mathscr{P}(\mathscr{E})$. A CIG $\gamma = (\mathscr{C}_\Gamma, \mathbb{F}_\Gamma, \mathscr{F}_\Gamma, \mathcal{C}_\Gamma, \vec{\mathcal{C}_\Gamma}, \mathcal{O}_\Gamma, \mathcal{I}_\Gamma, \mathscr{E}_\Gamma)$ is a tuple that is an element of the set $\mathcal{G}$, denoted $\gamma \in \mathcal{G}$.

An factual CIG Γ (i.e. one that obtains at some time point) is a *situation* in the sense of UFO-B [14] and denoted $\Gamma : \mathcal{G}$. In UFO-B, situations are akin to *states of affairs* that describe a fragment of the world at a particular point in time. However, two situations are distinct if the time point at which they obtain are different – even if the state of affairs they describe are identical.

The multiplicity function $\mu_{\mathcal{G}'} : \mathcal{G} \to \mathbb{N}$ defines the multiset $\mathcal{G}' = (\mathcal{G}, \mu_{\mathcal{G}'})$. Contrary to common sets, a multiset allows for more than one element of the same value. This is an important property, since in UFO two situations are distinct if they describe a different time points [14].

Then let $\mathit{Occ}(\mathcal{G}') \subset \mathcal{G} \times \mathbb{N}$ be the set that relates CIGs to the multiplicity of their occurrences:

$$\text{Occ}(\mathcal{G}') = \{(\gamma, i) \mid \gamma \in \mathcal{G}.\ 0 \leq i \leq \mu_{\mathcal{G}'}(\gamma)\}$$

We introduce the notation Γ to denote an occurrence $(\gamma, i) \in \text{Occ}(\mathcal{G}')$. The notation $\Gamma : \mathcal{G}$ is such that the multiplicity of the occurrence is at least one:

$$\forall\, (\gamma, i) \in \mathcal{G}.\ \Gamma : \mathcal{G} \iff \mu_{\mathcal{G}'}(\Gamma) \geq 1$$

Let the function $\tau : \mathit{Occ}(\mathcal{G}') \to \mathit{TimePoint}$ be the total function that relates occurrences of a CIG to a time point. By the definitions of $\mathit{Occ}(\mathcal{G}')$ and τ, all CIGs such that $\Gamma : \mathcal{G}$ have obtained in a time point:

$$\forall \Gamma \in \mathcal{G}.\ \Gamma : \mathcal{G} \iff \exists t \in \text{TimePoint}.\ \tau(\Gamma) = t$$

In UFO, such situations are termed *facts*.

Two CIGs $\Gamma = (\gamma, i) \in \mathit{Occ}(\mathcal{G}')$ and $\Gamma' = (\gamma', j) \in \mathit{Occ}(\mathcal{G}')$ are structurally equivalent when they describe the same state of affairs (regardless of their time point):

$$\Gamma \equiv \Gamma' \iff \gamma = \gamma'$$

Finally, we define that two CIGs $\Gamma = (\gamma, i) \in \mathit{Occ}(\mathcal{G}')$ and $\Gamma' = (\gamma', j) \in \mathit{Occ}(\mathcal{G}')$ are distinct if they obtain at a different time point:

$$\tau(\Gamma) \neq \tau(\Gamma') \implies i \neq j$$

In this way, the notation $\Gamma : \mathcal{G}$ mirrors the UFO notion of factual situations, whereas $\Gamma \in \mathcal{G}$ denotes the set of all world *possibilia* (i.e. all the states that a system *could* be in).

B. On CIGs as objects

CIGs describe a collection of interacting elements. To argue that these interacting elements can be the focus of a situation, we must argue that this collection itself has object-like properties. We call the elements that a CIG comprises *endogenous* elements. And we call the sum of such endogenous elements *systems*. We do not make any ontological commitments to the nature of systems – such as their mereology – beyond the stipulate that a system is the object comprising all components, channels and interactions in a CIG γ, and that this object is precisely the focus of the situation Γ. We must now be precise: a CIG γ is an **object**. When we discuss the dynamics and evolution of such objects, we describe their occurrences (e.g. Γ) as **situations**.

C. Formal graphical notation of CIGs

A CIG can be interpreted as a graph by having its set of capabilities as vertices, and its interfaces as solid edges. The label of each edge is the resource type that is exchanged. We draw a rectangle around capabilities that have the same bearer (i.e. components). A dashed edge denotes the flow of a resource type between channel input and output capabilities. We overset a capability label with a bar (e.g. $\bar{f}$) to indicate that it is currently manifested as a function. The running example shown in Figure 2 is more formally denoted in Figure 6.

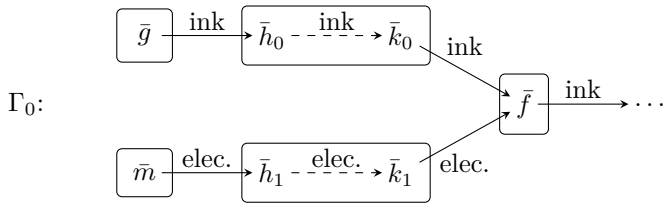


Fig. 6: A formal CIG diagram showing the running example (derived from Figure 2).

D. Further model constraints

To further eliminate unintended model instantiations, we provide additional integrity constraints.

Individuation. The elements of CIGs must have a surjective relation to the instances that the ontology in section III permits for concepts of the same name;

$$\begin{aligned}
\forall r \in \mathcal{R}.\ &\text{ResourceType}(r)\\
\forall f \in \mathscr{C}_\Gamma.\ &\text{Capability}(\iota_\Gamma(f))\\
\forall \bar{f} \in \mathbb{F}_\Gamma.\ &\text{Function}(\bar{f})\\
\forall \not{f} \in \mathscr{F}_\Gamma.\ &\text{Fault}(\iota_\Gamma(\not{f}))\\
\forall \alpha \in \mathcal{C}_\Gamma.\ &\text{Component}(\iota_\Gamma(\alpha))\\
\forall c \in \vec{\mathcal{C}}_\Gamma.\ &\text{Channel}(\iota_\Gamma(c))\\
\forall o \in \mathcal{O}_\Gamma.\ &\text{OutputInterface}(\iota^o_\Gamma(o))\\
\forall i \in \mathcal{I}_\Gamma.\ &\text{InputInterface}(\iota^i_\Gamma(i))\\
\forall e \in \mathscr{E}_\Gamma.\ &\text{Capability}(\iota_\Gamma(e))
\end{aligned}$$

Integrity of resource types. The second and third element of a capability tuple denote the resource types that the capability, when manifested, consumes and produces respectively. Let $\Gamma : \mathcal{G}$ and $(\text{id}_f, R_{in}, R_{out}) \in \mathscr{C}_\Gamma$:

$$\begin{aligned}
\big(\forall r^{\downarrow} \in R_{in}.\ \text{consumes}(\beta(\text{id}_f), r^{\downarrow})\big) \wedge\\
\big(\forall r^{\uparrow} \in R_{out}.\ \text{produces}(\beta(\text{id}_f), r^{\uparrow})\big)
\end{aligned} \tag{8}$$

The second element of a channel tuple denotes the resource types that channel accommodates. Let $\Gamma : \mathcal{G}$:

$$\forall(\text{id}_c, R, \dots) \in \vec{\mathcal{C}}_\Gamma.\ \forall r \in R.\ \text{accommodates}(\text{id}_c, r)$$

The component capabilities mediated by interfaces are compatible with that interface w.r.t. the resource type they produce or consume (for output and input interfaces respectively). Let $\Gamma : \mathcal{G}$:

$$\forall o \in \mathcal{O}.\forall \text{id}_f \in \varpi^o_\Gamma(o).\ \text{proj}_3(o) \in \text{proj}_3(\iota_\Gamma^{-1}(\text{id}_f))$$

and

$$\forall i \in \mathcal{I}.\forall \text{id}_f \in \varpi^i_\Gamma(i).\ \text{proj}_3(i) \in \text{proj}_2(\iota_\Gamma^{-1}(\text{id}_f))$$

On the special case of transport capabilities. The consumption capability of a channel cannot provide a channel, and the production capability of a channel cannot be provided by a channel. Let $\Gamma : \mathcal{G}$ and $c \in \vec{\mathcal{C}}_\Gamma$:

$$\begin{aligned}
\big(\exists r \in \mathcal{R}.\ \text{accommodates}(c, r)\big) \wedge \forall f \in \mathscr{C}^c_\Gamma.\\
\big(f = \text{cons}_\Gamma(c, r) \implies \neg\exists o \in \mathcal{O}.\ \iota_\Gamma(f) = \varsigma^o_\Gamma(o)\big) \wedge\\
\big(f = \text{prod}_\Gamma(c, r) \implies \neg\exists i \in \mathcal{I}.\ \iota_\Gamma(f) = \varsigma^i_\Gamma(i)\big)
\end{aligned}$$

The bearer of transport capabilities. The bearer of the consumption and production capabilities of a channel c, is c. Let $\Gamma : \mathcal{G}$ be a situation, an $c \in \vec{\mathcal{C}}_\Gamma$ be a channel. Let $r \in \mathcal{R}$ be a resource that c accommodates. Then:

$$\beta(\text{cons}_\Gamma(c, r)) = \beta(\text{prod}_\Gamma(c, r)) = c$$

Manifestation of capabilities. Let $\Gamma : \mathcal{G}$. The following equation explicitly connects all members of $\mathbb{F}_\Gamma$ to a unique member of $\mathscr{C}_\Gamma$:

$$\forall \bar{f} \in \mathbb{F}_\Gamma.\exists! f \in \mathscr{C}_\Gamma.\bar{f} = \varrho_\Gamma(\iota_\Gamma(f)) \wedge \text{manifests}(\bar{f}, \iota_\Gamma(f))$$

V. Dynamics and system evolution

In this section, we extend the representation of the system *at some point in time* (formalized in the previous section) with a representation of a system's dynamic evolution *throughout* time.

To encode the dynamic aspects of a system, we can evolve CIGs through the UFO-B notion of events [14]. We typically denote by Γ' the situation that is derived from the situation Γ by the occurrence (or *application*) of some event. For reference, we also list the relevant portions of UFO-B in Table III.

Events transform one situation into another. We call the situation that an event begins in its *antecedent*, and the situation that is brought about by the event its *consequent*.

An atomic event is an event that is not made up of other events (in contrast to *complex* events) [14]. We distinguish two types of atomic events: triggered events and spontaneous events.

Axiom	FOL formula
S1	$\forall s : \text{Situation}, e : \text{Event}.\ \text{triggers}(s, e) \implies \text{obtainsIn}(s, \text{begin-point}(e))$
S2	$\forall s : \text{Situation}, e : \text{Event}.\ \text{brings-about}(e, s) \implies \text{obtainsIn}(s, \text{end-point}(e))$
S3	$\forall e : \text{Event}.\exists! s : \text{Situation}.\ \text{triggers}(s, e)$
S4	$\forall e : \text{Event}.\exists! s : \text{Situation}.\ \text{brings-about}(e, s)$
S5	$\forall s : \text{Situation}.\ \text{fact}(s) \iff \exists t : \text{TimePoint}.\ \text{obtainsIn}(s, t)$
S6	$\forall e, e' : \text{Event}.\ \text{directly-causes}(e, e') \iff \exists s : \text{Situation}.\ \text{brings-about}(e, s) \wedge \text{triggers}(s, e')$
S7	$\forall e, e'' : \text{Event}.\ \text{causes}(e, e'') \iff \text{directly-causes}(e, e'') \vee (\exists e' : \text{Event}.\ \text{causes}(e, e') \wedge \text{causes}(e', e''))$
T1	$\forall t : \text{TimePoint}.\ \neg\text{precedes}(t, t)$
T2	$\forall t, t' : \text{TimePoint}.\ \text{precedes}(t, t') \implies \neg\text{precedes}(t', t)$
T3	$\forall t, t', t'' : \text{TimePoint}.\ \text{precedes}(t, t') \wedge \text{precedes}(t', t'') \implies \text{precedes}(t, t'')$
t4 [5]	$\forall t, t' : \text{TimePoint}.\ \text{immediateSuccessorOf}(t', t) \iff \text{precedes}(t, t') \wedge \neg\exists t'' : \text{TimePoint}.\ (\text{precedes}(t'', t') \wedge \text{precedes}(t, t''))$
T5	$\forall e : \text{Event}.\exists! t : \text{TimePoint}.\exists! t' : \text{TimePoint}.(t = \text{begin-point}(e)) \wedge (t' = \text{end-point}(e))$
P1	$\forall e : \text{AtomicEvent}.\exists! o : \text{Object}.\ \text{dependsOn}(e, o)$
P2	$\forall e : \text{AtomicEvent}, o : \text{Object}.\text{excDepends}(e, o) \iff \text{dependsOn}(e, o)$
P4	$\forall e : \text{Event}.\text{Participation}(e) \iff \exists! o : \text{Object}.\ \text{excDepends}(e, o)$
P5	$\forall o : \text{Object}, p : \text{Participation}.\text{participationOf}(p, o) \iff \text{excDepends}(p, o)$
a102 [13]	$\text{manifests}(x, y) \implies \text{Perdurant}(x) \wedge \text{Endurant}(y)$
D1	$\forall d : \text{Disposition}.\exists! o : \text{Object}.\ \text{inheresIn}(d, o)$
D2	$\forall e : \text{AtomicEvent}.\exists! d : \text{Disposition}.\ \text{manifests}(e, d)$
D3	$\forall s : \text{Situation}, e : \text{AtomicEvent}.\ \text{triggers}(s, e) \iff \exists d : \text{Disposition}.\ \text{activates}(s, d) \wedge \text{manifests}(e, d)$
D4	$\forall d : \text{Disposition}, e : \text{AtomicEvent}, o : \text{Object}.\text{manifests}(e, d) \wedge \text{inheresIn}(d, o) \implies \text{dependsOn}(e, o)$

TABLE III: A subset of axioms of UFO-B, taken from [14].

Definition 11 (Triggered events)**.** We may denote a triggered event e, the situation that triggers it (S3, Table III), and the situation it brings about (S4, Table III) as $\Gamma \xrightarrow{e} \Gamma'$:

$$\forall \Gamma, \Gamma' : \mathcal{G}.\forall e : \text{Event}.\ \Gamma \xrightarrow{e} \Gamma' \iff \text{triggers}(\Gamma, e) \wedge \text{brings-about}(e, \Gamma') \quad (9)$$

Definition 12 (Spontaneous events)**.** We may denote a spontaneous event $\hat{e}$, the situation that precedes it, and the situation it brings about (S4, Table III) as $\Gamma \overset{\hat{e}}{\hookrightarrow} \Gamma'$:

$$\forall \Gamma, \Gamma' : \mathcal{G}.\forall e : \text{Event}.\ \Gamma \overset{\hat{e}}{\hookrightarrow} \Gamma' \iff \text{obtainsIn}(\Gamma, \text{begin-point}(\hat{e})) \wedge \neg\text{triggers}(\Gamma, \hat{e}) \wedge \text{brings-about}(\hat{e}, \Gamma') \quad (10)$$

By S3, for each event there must exist a unique situation that triggers it. However, for spontaneous events we cannot describe this situation since, by Definition 12, we do not know it (if we did, the event would be a *triggered* event).

Furthermore:

- An antecedent situation that triggers an event obtains at the begin point of that event (S1, Table III), and;
- The consequent situation that an event brings about obtains at the end point of that event (S2, Table III).

In certain cases, we need only consider the consequent situation that an event brings about. If we do not need to distinguish between triggered events and spontaneous events, we may write $\Gamma \overset{\dot{e}}{\curvearrowright} \Gamma'$ in lieu of either $\Gamma \xrightarrow{e} \Gamma'$ or $\Gamma \overset{\hat{e}}{\hookrightarrow} \Gamma'$:

$$\forall \Gamma, \Gamma' : \mathcal{G}.\forall e : \text{Event}.\ \Gamma \overset{e}{\curvearrowright} \Gamma' \iff \Gamma \xrightarrow{e} \Gamma' \oplus \Gamma \overset{e}{\hookrightarrow} \Gamma' \quad (11)$$

where $\oplus$ is the `XOR` connective.

Definition 13 (Temporal order of situations)**.** Let the strict partial order $\prec \subseteq \mathcal{G} \times \mathcal{G}$ denote the temporal order of factual situations that obtain at some time point (S5, T1, T2, T3, Table III):

$$\forall \Gamma, \Gamma' : \mathcal{G}.\ \Gamma \prec \Gamma' \iff \text{precedes}(\tau(\Gamma), \tau(\Gamma')) \quad (12)$$

By axiom T5 (Table III), events cannot have zero duration in UFO-B. Therefore, the time point of an event's antecedent always precedes the time point of its consequent. By (12), we obtain:

$$\forall \Gamma, \Gamma' : \mathcal{G}, \dot{e} : \text{Event}.\ \Gamma \overset{\dot{e}}{\curvearrowright} \Gamma' \implies \Gamma \prec \Gamma'$$

We also make use of a weaker relation $\preceq\ : \mathcal{G} \times \mathcal{G}$ that is entailed by $\prec$:

$$\forall \Gamma, \Gamma' : \mathcal{G}.\ \Gamma \preceq \Gamma' \iff \Gamma \prec \Gamma' \vee \tau(\Gamma) = \tau(\Gamma')$$

Definition 14 (Channel depletion)**.** A channel c can only output a resource of type r if that resource is present in the channel. If the resource is not (continuously) replenished by the manifestation of the channel's input capability, then

after some non-zero time δ, its store of r is depleted and the channel's output capability cannot be manifested. We denote this depletion event $\Delta\langle c\rangle$, which has a constant duration δ for all channels. The begin point of $\Delta\langle c\rangle$ is the end point of the manifestation of c's input of r, and the end point of $\Delta\langle c\rangle$ is the first time point at which r is depleted.

Let $c \in \vec{\mathcal{C}_\Gamma}$ be a channel. Then:

$$\forall \Gamma, \Gamma' : \mathcal{G}.\ \hat{\Gamma} \xrightarrow{\Delta\langle c\rangle} \Gamma' \iff \mathrm{cons}_{\hat{\Gamma}}(c) \notin \mathrm{dom}\varrho_{\hat{\Gamma}} \wedge \mathrm{prod}_{\Gamma'}(c) \notin \mathrm{dom}\varrho_{\Gamma'} \quad (13)$$

Where by $\hat{\Gamma}$ in (13) we mean the earliest situation in which $\mathrm{cons}_\Gamma(c)$ is not manifested (see Definition 15).

Definition 15 (Earliest antecedent such that φ)**.** Let φ be a predicate. Let $\Gamma \xrightarrow{e} \Gamma'$ be some event such that φ holds in Γ (denoted $\Gamma \models \varphi$). Let t be the time point such that $\mathrm{obtainsIn}(\Gamma, t)$. Then Γ is the earliest situation in which φ holds iff there does not exist a situation Γ'' that obtains at a time point t'', such that t'' immediately succeeded by t and φ holds in Γ''. I.e.:

$$\neg\exists t'' : \mathit{TimePoint}.\ \exists \Gamma'' : \mathcal{G}.\ \mathrm{obtainsIn}(\Gamma'', t'') \wedge \mathrm{immediateSuccessorOf}(t, t'') \wedge \Gamma'' \models \varphi$$

Γ is then the earliest situation such that φ holds, denoted $\hat{\Gamma}$.

Definition 16 (Causality)**.** Recall from Table III the definitions of direct (S6) and general (S7) causality. This is captured by (14) and (15) respectively.

$$\forall \Gamma, \Gamma', \Gamma'' : \mathcal{G}. \forall \dot{e}, e' : \mathrm{Event}. \quad \Gamma \overset{\dot{e}}{\curvearrowright} \Gamma' \xrightarrow{e'} \Gamma'' \iff \text{directly-causes}(\dot{e}, e') \quad (14)$$

and

$$\forall n \geq 2.\ \forall \Gamma, \ldots, \Gamma_n : \mathcal{G}. \forall \dot{e}_0, e_1, \ldots, e_n : \mathrm{Event}. \quad \Gamma_0 \overset{\dot{e}_0}{\curvearrowright} \Gamma_1 \xrightarrow{e_1} \ldots \xrightarrow{e_n} \Gamma_n \iff \mathrm{causes}(\dot{e}_0, e_n) \quad (15)$$

for any sequence of causal events of length $n+1$.

Note that by (10) and S6, a spontaneous event cannot have a (direct) cause:

$$\forall \Gamma, \Gamma', \Gamma'' : \mathcal{G}. \forall \dot{e}, \hat{e} : \mathrm{Event}. \quad \Gamma \overset{\dot{e}}{\curvearrowright} \Gamma' \overset{\hat{e}}{\nrightarrow} \Gamma'' \implies \neg\mathrm{causes}(\dot{e}, \hat{e})$$

Lastly, we define a causal relationship between the event that a channel is not provided with a resource, and the event that a channel stops producing that resource. Let $c \in \vec{\mathcal{C}_\Gamma}$ be a channel, and $r \in \mathcal{R}$ a resource that it accommodates. Let Γ be the situation in which $h = \mathrm{cons}_\Gamma(c, r)$ and $k = \mathrm{prod}_\Gamma(c, r)$ are manifested as functions $\bar{h}$ and $\bar{k}$. Then let e be the event that terminates $\bar{h}$ in Γ', such that $\Gamma \overset{e}{\curvearrowright} \Gamma'$. Let $\Delta\langle c\rangle$ be the event introduced in (13), and Γ'' be the situation that obtains some time δ after Γ', such that k is not manifested in Γ''. Then by (9), (10), (11), and (S6 and S7, Table III):

$$\Gamma \overset{e}{\curvearrowright} \Gamma' \xrightarrow{\Delta\langle c\rangle} \Gamma'' \implies \mathrm{causes}(e, \Delta\langle c\rangle)$$

In the definitions of CIG events in the following sections, we define preconditions and postconditions in the antecedent and consequent situations of an event, respectively. An event is defined by these conditions. However, we do not describe the full antecedent and consequent situations – only the relevant portion that defines the particular event. For instance, when we write about the pre and postconditions of a channel depletion event like in (13), we do not make any commitments to other components, channels, resource types, etc. that may be part of Γ or Γ'. This naturally raises the question: what happens to the rest of the CIG?

Postulate 1 (Locality of events)**.** The elements of a CIG that are not terms in the pre- or postconditions of an event are unaltered by the event.

Postulate 1 states that the elements of a situation Γ are "copied over" by an event, only changing the (aspects of) elements described in the pre and postconditions. Do note that events are not *necessarily* changes of the elements in a situation, but all non-relational changes of elements of situations *are* events [5].

Postulate 2 (The passive nature of time)**.** In this work, we assume that the passing of time effects no change in the *moments* of objects.

We admit that Postulate 2 is too strong a postulate for physical systems. However, a description of the mechanisms of passing time is not given in UFO and its exploration warrants intent study. We leave this to future work.

Remark (Events as Types). It is worth noting that we have introduced events as *types*, fully characterized by their antecedent and consequent situations. When we write something of the form

$$\forall \Gamma, \Gamma' : \mathcal{G}. \forall e : \mathrm{Event}.\ \Gamma \xrightarrow{e} \Gamma' \iff \ldots$$

The universal quantification indicates that we are introducing a property that holds for all instances of a particular event nature (i.e. an event type). For instance in (9), the combination of universal quantification and bi-implication indicates a (complete) characterization of the event type "triggered event".

When we talk about particular events, we usually do so by means of existential quantification:

$$\exists \Gamma, \Gamma' : \mathcal{G}.\ \Gamma \xrightarrow{e} \Gamma' \implies \ldots$$

Here, the implication indicates that we are deriving a property that holds for the occurrence of a particular event.

Contrary to what (11) suggests, generalization sets over event types are not *necessarily* disjoint.

This position aims to align with the notion of event types as employed in COVER [32]. We leave a more rigorous integration of COVER, as well as a detailed taxonomy of the events we present in this paper, for future work.

VI. On faults, errors and fault activation

The previous section describes the modeling of dynamic aspects of CIGs in a *general* sense. In this section, we specifically describe how faults come about, and how faults and errors evolve over time. We also disambiguate between the core notions of fault creation, errors and fault activation.

Our notions of faults, errors and fault activation is inspired by the taxonomy proposed in [2]. We redefine these concepts in the context of CIGs using the ontological framework of objects and events provided by UFO [12] [14].

Example. An example of *fault creation* in the running example is the power supply unit (PSU) burning out. The created fault (a molten internal winding) is dormant as long as we do not try to use the PSU. *Fault activation* commences at the instant we put a load on the PSU and expect an output current. The fault then prevents the PSU from supplying this expected current, resulting in an *error*.

Note that we do not consider *failures* in this work. A notion of failure pertains to a notion of *service* [2] or *goals* [22], and we do not consider such notions in this work. We do consider *expectations*, which we describe only as an analytical artifact.

Definition 17 (Fault creation)**.** In this work, we consider the *creation* of a set of faults $F \subseteq \mathscr{F}_{\Gamma'}$ in an object $\text{obj} \in \text{ComponentId}_{\Gamma'} \cup \text{ChannelId}_{\Gamma'}$ a spontaneous atomic event, denoted $\Gamma \overset{\lightning F}{\hookrightarrow} \Gamma'$, such that $\forall \mathscr{f} \in F.\ \beta(\iota_{\Gamma'}(\mathscr{f})) = \text{obj}$.

A spontaneous event is a fault creation event, iff the spontaneous event creates faults in a component or channel that did not exist in that object before the event. Let $F \subseteq \mathscr{F}_{\Gamma'}$ be the set of faults that are created. Let $\Gamma, \Gamma' : \mathcal{G}$. Then:

$$\Gamma \overset{\lightning F}{\hookrightarrow} \Gamma' \iff \forall \mathscr{f} \in F.\ \left(\neg\exists \mathscr{f} \in \mathscr{F}_{\Gamma}\right) \wedge \left(\exists \mathscr{f} \in \mathscr{F}_{\Gamma'}\right) \quad (16)$$

In the consequent situation of a fault creation event, the created faults prevent the manifestation of capabilities of the particular object. By the existential dependency of interfaces (as relators) on the manifestation of such capabilities, any interface that is associated with these capabilities cannot exist. Let $\Gamma, \Gamma' : \mathcal{G}$. Then:

$$\Gamma \overset{\lightning F}{\hookrightarrow} \Gamma' \implies \forall(\text{id}_{\mathscr{f}}, C) \in F.\ \forall f \in C.\ \neg\exists o \in \mathcal{O}_{\Gamma'}.\ f \in \varpi^{o}_{\Gamma'}(o) \wedge \neg\exists i \in \mathcal{I}_{\Gamma'}.\ f \in \varpi^{i}_{\Gamma'}(i)$$

I.e.: all material provides and providedBy relations that are existentially dependent on the capabilities covered by F are terminated *synchronously* in the fault creation's consequent situation.

Definition 18 (Errors)**. Errors** are atomic events that are triggered by unmet expectations placed on capabilities in a particular situation. An error event $\varepsilon\langle f \rangle$ describes the period of time during which a capability f remains erroneous. An error event ends as soon as the capability is no longer erroneous.

An error is triggered in the antecedent situation Γ, if and only if we have an expectation for the manifestation of a function in Γ, and that expectation is not met. The situation that an error brings about is one where either i) there is no expectation for the manifestation of the capabilities that the error comprises, or ii) the capabilities that the error comprises are manifested again. I.e. an error *persists* so long as the expectation exists and the capabilities that it comprises are not manifested. Formally:

$$\forall e \in \mathscr{E}_{\Gamma}.\ \exists \Gamma, \Gamma' : \mathcal{G}.\ \hat{\Gamma} \xrightarrow{\varepsilon\langle e \rangle} \Gamma' \iff \neg\text{meets}(\hat{\Gamma}, e) \wedge (e \notin \mathscr{E}_{\Gamma'} \vee \text{meets}(\Gamma', e)) \quad (17)$$

Where by $\hat{\Gamma}$ in (17) we mean the earliest situation Γ in which the expectation e is not met (see Definition 15). Note that $e \notin \mathscr{E}_{\Gamma'} \implies \neg\text{meets}(\Gamma', e)$ – we only include it in the formula to emphasize this special case.

In the consequent situation of $\varepsilon\langle f \rangle$, the phase of the capability that the error comprises has changed from *Erroneous* back to *Functional*:

$$\forall \Gamma, \Gamma' : \mathcal{G}.\ \hat{\Gamma} \xrightarrow{\varepsilon\langle f \rangle} \Gamma' \iff \text{ErroneousCapability}_{\hat{\Gamma}}(f) \wedge \text{FunctionalCapability}_{\Gamma'}(f)$$

In this work, we do not describe mechanisms that change the phase of capabilities from Erroneous to Functional. As a result, erroneous capabilities remain erroneous once they have entered this phase. That is to say: components cannot be repaired, rerouted or replaced. In section IX we touch on the consequences of this w.r.t. the evaluation of our model. One way to look at this is that the number of erroneous capabilities as a function of situations is monotonic (given the order $\preceq$ over the set of situations).

Proposition 1. Capabilities that are erroneous in a situation Γ are erroneous in all situations Γ'' where $\Gamma \preceq \Gamma'' \prec \hat{\Gamma}'$, s.t. $\hat{\Gamma}'$ is the earliest situation in which the error is resolved.

Definition 19 (Fault activation)**.** Fault activation is the complex event in which the situation brought about by a fault creation event triggers an error.

Let $\Gamma \overset{\lightning F}{\hookrightarrow} \Gamma'$ be a fault creation event. The consequences of the fault creation then depend on the expectations we place on the affected capabilities. If there exists any expectation $e \in \mathscr{E}_{\Gamma'}$ such that the created fault inhibits the manifestation of that expectation (i.e. $e \in F$), then by (16), (4) and Definition 18, Γ' triggers an error, and $\lightning F$ is the direct cause for this error (Definition 16):

$$\Gamma \overset{\lightning_F}{\hookrightarrow} \Gamma' \implies \Big(\exists e \in \mathscr{E}_{\Gamma'}.\ e \in F \wedge \neg\text{meets}(\Gamma', e) \implies \exists \Gamma'' : \mathcal{G}.\ \Gamma' \xrightarrow{\varepsilon\langle e\rangle} \Gamma''\Big) \quad (18)$$

If no such expectation exists in Γ', no error is triggered. The distinct cases of *fault activation* are:

1) With a witness to such an unmet expectation $e \in \mathscr{E}_{\Gamma'}$, we obtain from (18):
$$\Gamma \overset{\lightning_F}{\hookrightarrow} \Gamma' \xrightarrow{\varepsilon\langle e\rangle} \Gamma'' \quad (19)$$
and by (14), $\lightning_F$ is then a direct cause for $\varepsilon\langle e\rangle$.
2) Otherwise, the created fault is called *dormant* [2]. Since faults inhere in objects, it will remain dormant as long as the fault persists, the object exists, or the fault is activated in a future situation where an expectation exists for the affected capability and the activation of the (until then dormant) fault causes an error.

We do not aim to encode the dynamics of expectations in this work. We therefore leave the exploration of dormant fault activation to future work. From here on, when we refer to fault activation, we always refer to its first, causal case.

VII. Functional Dependency

In this section, we define the notion of functional dependency. In the next section, we show how this notion combines with the definition of errors from the previous section to form error propagation.

Functional dependency is a complex notion that describes the necessary conditions for the manifestation of a function and comprises i) the prerequisites for the manifestation of a capability as a function, and ii) the fragment of the system architecture that fulfills these prerequisites. We address each constituent notion individually:

i. Recall that, in UFO, an event is the manifestation of a disposition, created and terminated by the activation and deactivation of a disposition [14]. The manifestation of a capability as a function depends on access to certain types of resources. The instant that access to these resources is realized, the capability's manifestation as a function begins. The instant that access to these resources stops, this manifestation is terminated. We call the need for access to this set of resources the *prerequisites* for the manifestation of a capability as a function.
ii. The minimal configuration fragment of the interaction between components (through channels) that provides access to the prerequisites for the manifestation of a capability in a given situation.

Definition 20 (Capability prerequisites)**.** The manifestation of a capability as a function is contingent on access to that capability's prerequisites. We define the function $\psi_\Gamma : \mathscr{C}_\Gamma \to \mathcal{P}(\mathcal{R})$ that returns a set of resource types.

Let $\Gamma : \mathcal{G}$ be a situation, and $f \in \mathscr{C}_\Gamma$ be a capability. Then:

$$\psi_\Gamma(f) = \{r \mid r \in \mathcal{R}.\ \text{consumes}(\iota_\Gamma(f), r)\}$$

or equivalently, by (8):

$$\psi_\Gamma(f) = \text{proj}_2(f)$$

The notion of "access" of a capability to a particular resource type r is formally defined based on the *role*[2] that capability plays:

- For any component capability f, access is defined as the taking part of that capability in an input interface from which it consumes r.
- For any channel *output* capability k, it is defined as the manifestation of that channel's input capability for r[3].
- For any channel *input* capability h, it is defined as the taking part of that capability in an output interface from which it consumes r.

Note that in all three cases, the capability consumes that resource.

The set of capabilities that provide access to a particular resource type is called a *configuration fragment*.

Definition 21 (Component configuration fragments)**.** We define $\pi_\Gamma : \mathscr{C}_\Gamma \times \mathcal{R} \to \mathscr{P}(\mathscr{C}_\Gamma)$ to be the function that returns the set of channel production capabilities that supply a given component capability with a given resource type. Let $\Gamma : \mathcal{G}$ be a situation. Let $f \in \mathscr{C}_\Gamma$ be a component capability and let $r \in \mathcal{R}$ be a resource type that f consumes, such that $\exists \alpha \in \mathcal{C}_\Gamma.\ f \in \mathscr{C}_\Gamma^\alpha$. Then:

$$\pi_\Gamma(f, r) = \{\varsigma_\Gamma^i(i) \mid i \in \mathcal{I}_\Gamma . f \in \varpi_\Gamma^i(i) \wedge \text{proj}_3(i) = r\}$$

Definition 22 (Channel output configuration fragments)**.** Recall that, by Definition 14, there is a delayed dependency between a channel's output capability and its corresponding input capability for a particular resource. By definition, this delay is precisely δ time units.

We define the function $\pi_\Gamma : \mathscr{C}_\Gamma \times \mathcal{R}_\Gamma \to \mathscr{C}_\Gamma$ to be the function that returns the channel's unique input capability that provides its output capability with access to a particular resource type. Let $\Gamma^-, \Gamma : \mathcal{G}$ be situations, such that Γ^- precedes Γ by δ time units. Let $k \in \mathscr{C}_\Gamma$ be a channel output capability and let $r \in \mathcal{R}$ be a resource type that k consumes, such that $\exists c \in \vec{\mathcal{C}_\Gamma}.\ k = \text{prod}_\Gamma(c, r)$. Then:

$$\pi_\Gamma(k, r) = \text{cons}_{\Gamma^-}(\beta_\Gamma(k), r) \quad (20)$$

Definition 23 (Channel input configuration fragments)**.** We define $\pi_\Gamma : \mathscr{C}_\Gamma \times \mathcal{R} \to \mathscr{P}(\mathscr{C}_\Gamma)$ to be the function that returns the set of component production capabilities that provide a given channel input capability with access

[2] "Role", here, in the colloquial sense.

[3] The matter of *when* that input capability must be manifested is discussed in section VIII.

a given resource type. Let $\Gamma : \mathcal{G}$ be a situation. Let $h \in \mathscr{C}_\Gamma$ be a channel consumption capability and let $r \in \mathcal{R}$ be a resource type that h consumes, such that $\exists c \in \vec{\mathcal{C}_\Gamma}.\ h = \text{cons}_\Gamma(c, r)$. Then:

$$\pi_\Gamma(h, r) = \bigcup \{\varpi^o_\Gamma(o) \mid o \in \mathcal{O}_\Gamma.\ h = \varsigma^o_\Gamma(o)\}$$

A capability f is said to have *access* to a resource type r iff (f, r) is in the domain of π_Γ, and $\pi_\Gamma(f, r) \neq \emptyset$ (for component capabilities and channel input capabilities):

$$\text{access}(f, r) \iff (f, r) \in \text{dom}\pi_\Gamma \wedge \pi_\Gamma(f, r) \neq \emptyset$$

Since we only consider resource *types* in this work, we do not explicitly deal with quantities or qualities of individual resources. We capture this assumption in Postulate 3.

Postulate 3 (Sufficient access)**.** Any access to the appropriate resource types is sufficient to satisfy the prerequisites of a capability.

Access to all of a capability's prerequisites necessarily activates (or maintains, if already activated) a capability. Similarly, losing access to any prerequisite necessarily deactivates the manifestation.

Postulate 4 (Liveliness)**.** A capability is necessarily activated, respectively deactivated, when it gains or loses access to all, respectively any, of its prerequisites.

This liveliness is, in a sense, an over-approximation of the behavior of a system.

Let $\Gamma : \mathcal{G}$ be a situation and let $f \in \mathscr{C}_\Gamma$ be a capability. By Postulate 4, we obtain:

$$f \in \text{dom}\varrho_\Gamma \iff \forall r \in \psi_\Gamma(f).\ \text{access}(f, r) \tag{21}$$

The minimal sets of such interactions that satisfy a capability's prerequisites are called *minimal configuration fragments*.

Definition 24 (Minimal configuration fragments)**.** A configuration fragment is a minimal configuration fragment (MCS) w.r.t. to a capability f and a resource type r is a configuration fragment, iff if one element of that set would cease to be manifested, then f would not have sufficient access to r.

Definition 25 (Functional dependency A)**.** The function $\mathscr{D}^A_\Gamma : \mathscr{C}_\Gamma \times \mathcal{R} \to \mathscr{P}(\mathscr{C}_\Gamma)$ is defined to return the minimal configuration fragments that a given capability f is functionally dependent on for access to the resource type r. I.e. if any element is removed from a set $S \in \mathscr{D}^A_\Gamma(f, r)$, the set of channel capabilities S would not provide f with sufficient access to the resource type r, as in (22).

We define the functions $\text{ProvidedQuantity}_\Gamma : \mathcal{R} \times \mathcal{P}(\vec{\mathcal{C}_\Gamma}) \to \mathbb{N}$ and $\text{RequiredQuantity}_\Gamma : \mathcal{R} \times \mathscr{C}_\Gamma \to \mathbb{N}$ to return the quantity of resources that (a set of) capabilities provide and require respectively.

Let $f \in \mathscr{C}_\Gamma$ be a component capability. Let $r \in \psi_\Gamma(f)$ be a resource type. Then:

$$\begin{aligned}&\mathscr{D}^A_\Gamma(f, r) = \\ &\{S \mid S \subseteq \pi_\Gamma(f, r).\ \text{Prov.Q.}_\Gamma(r, S) \geq \text{Req.Q.}_\Gamma(r, f) \wedge \\ &\big(\neg\exists S' \subset S.\ \text{Prov.Q.}_\Gamma(r, S \setminus S') \geq \text{Req.Q.}_\Gamma(r, f)\big)\}\end{aligned} \tag{22}$$

Note that by Postulate 3 all non-empty sets $S \in \mathscr{D}^A_\Gamma(f, r)$ provide the capability f with access to a sufficient quantity of the resource r in Γ. I.e. the set $\mathscr{D}^A_\Gamma(f, r)$ is the set of minimal configuration fragments. To make this explicit, we provide definitions for the functions $\text{ProvidedQuantity}_\Gamma$ and $\text{RequiredQuantity}_\Gamma$ that do not consider any aspects of instances of r:

$$\begin{aligned}\text{ProvidedQuantity}_\Gamma(r, S) &= |S| \\ \text{RequiredQuantity}_\Gamma(r, f) &= 1\end{aligned}$$

These definitions allow us to simplify (22) by substitution:

$$\mathscr{D}^A_\Gamma(f, r) = \{S \mid S \subseteq \pi_\Gamma(f, r).\ |S| = 1\} \tag{23}$$

Finally, if we cannot construct any such minimal configuration fragment, then the prerequisites for the manifestation of a capability are not satisfied and the capability is not manifested. Let $\Gamma : \mathcal{G}$. Let $\alpha \in \mathcal{C}_\Gamma$ and $f \in \mathscr{C}^\alpha_\Gamma$. Then by (21):

$$\forall r \in \psi_\Gamma(f).\ \mathscr{D}^A_\Gamma(f, r) = \emptyset \implies f \notin \text{dom}\varrho_\Gamma$$

Definition 26 (Functional Dependency B)**.** Recall that, by Definition 14 and Definition 22, there is a delayed dependency between a channel's output capability and its corresponding input capability for a particular resource. We define the function $\mathscr{D}^B_\Gamma : \mathscr{C}_\Gamma \times \mathcal{R} \to \mathscr{C}_\Gamma$ such that

$$\mathscr{D}^B_\Gamma(k, r) = \text{cons}_{\Gamma^-}(\beta_\Gamma(k), r) \tag{24}$$

where Γ^- precedes Γ by δ time units.

Definition 27 (Functional Dependency C)**.** The function $\mathscr{D}^C_\Gamma : \mathscr{C}_\Gamma \times \mathcal{R} \to \mathscr{P}(\mathscr{C}_\Gamma)$ is defined to return the minimal configuration fragments that a given channel consumption capability is functionally dependent on such that, if any element is removed from a set $S \in \mathscr{D}^A_\Gamma(h, r)$, the set of channel capabilities S would not provide h with sufficient access to the resource type r.

Following similar steps as for Definition 25, we obtain:

$$\mathscr{D}^C_\Gamma(h, r) = \{S \mid S \subseteq \pi_\Gamma(h, r).\ |S| = 1\} \tag{25}$$

And here too, if we cannot construct any such minimal configuration fragment, then the prerequisites for the manifestation of a capability are not met and the capability is not manifested. Let $\Gamma : \mathcal{G}$. Let $c \in \vec{\mathcal{C}_\Gamma}$ and $h \in \mathscr{C}^c_\Gamma$. Then:

$$\forall r \in \psi_\Gamma(h).\ \mathscr{D}^C_\Gamma(h, r) = \emptyset \implies h \notin \text{dom}\varrho_\Gamma$$

Definition 28 (Functional dependency)**.** We define the transitive relation $\mathscr{D}_\Gamma \subseteq \text{CapabilityId}^2$, which denotes that

for any ordered pair $(\mathrm{id}_f, \mathrm{id}_g) \in \mathscr{D}_\Gamma$, the capability f is functionally dependent on the capability g in the situation Γ.

The relation $\mathscr{D}_\Gamma$ captures all three cascading notions of functional dependency (A, B and C).

Let $\Gamma : \mathcal{G}$ be a situation. Let $f \in \mathscr{C}_\Gamma$ be a capability, and let $\mathrm{id}_f = \iota_\Gamma(f)$. Then:

$$\forall r \in \psi_\Gamma(f).\quad \begin{cases} \forall S^A \in \mathscr{D}^A_\Gamma(f,r).\ \mu(S^A), & \text{if } \exists \alpha \in \mathcal{C}_\Gamma.f \in \mathscr{C}^\alpha_\Gamma \\ (\mathrm{id}_f, \iota_\Gamma(\mathscr{D}^B_\Gamma(f,r))) \in \mathscr{D}_\Gamma, & \text{if } \exists c \in \vec{\mathcal{C}}_\Gamma.\vec{\theta}(c,r,f) \\ \forall S^C \in \mathscr{D}^C_\Gamma(f,r).\ \mu(S^C), & \text{if } \exists c \in \vec{\mathcal{C}}_\Gamma.\overleftarrow{\theta}(c,r,f) \end{cases}$$

Where

$$\begin{aligned} \mu(S) &\iff \forall g \in S.\ (\mathrm{id}_f, \iota_\Gamma(g)) \in \mathscr{D}_\Gamma \\ \vec{\theta}(c,r,f) &\iff \mathrm{prod}_\Gamma(c,r) = f \\ \overleftarrow{\theta}(c,r,f) &\iff \mathrm{cons}_\Gamma(c,r) = f \end{aligned}$$

This notion of functional dependency is intimately tied to that of errors: if there exist prerequisites for a component's capability, and there does not exist a minimal configuration fragment that satisfies these prerequisites, then that capability is not, and cannot be, manifested as a function. If, in any particular situation, there exists an expectation for the manifestation of such a function, then an error is triggered.

On Dependency and Expectation. Knowing that the manifestation of a capability has functional dependencies, we cannot expect a capability to be manifested if such functional dependencies are not satisfied. Therefore, we introduce a cascading notion of expectation, mirroring the transitive relation of functional dependency.

Stipulate 1 (Cascading expectations)**.** If we expect a capability to be manifested, we must also expect the sufficient conditions for that manifestation to hold.

The sufficient conditions for the manifestation of a capability are determined by access to its prerequisites. That is:

1) If we have an expectation for a component's capability f, then we also have an expectation for the channels' capabilities that provide access to f's prerequisites, and;
2) If we have an expectation for a channel to produce a resource, then we also have an expectation for that channel to have consumed that resource, and;
3) If we have an expectation for a channel to consume a resource, then we also have an expectation for some component capabilities to provide that resource.

The cascading of expectations in this manner reflect the three types of functional dependency (A, B and C) captured by the transitive relation $\mathscr{D}_\Gamma$. Formally:

$$\forall \Gamma : \mathcal{G}.\forall (\mathrm{id}_f, \mathrm{id}_g) \in \mathscr{D}_\Gamma.\ \mathrm{id}_f \in \mathscr{E}_\Gamma \implies \mathrm{id}_g \in \mathscr{E}_\Gamma$$

Example. Recall the running example from Figure 2. The `print head`'s function *to deposit ink* is functionally dependent on the `PSU`'s function *to provide electricity*, and the `ink reservoir`'s function *to provide ink*.

Should we expect the print head to deposit ink, then we should also expect the PSU to provide electricity and the ink reservoir to provide ink.

VIII. Error propagation

In this section, we combine the definitions of errors and functional dependency to construct a notion of error propagation. This forms the theoretical backbone for CIG analysis and FT synthesis.

Error propagation is a sequence of causal events, where each situation that is the antecedent of such an event triggers an error. This is similar in structure to the causal chain shown in (15), Definition 16.

Definition 29 (Error propagation)**.** Error propagation is a complex event, denoted $\Gamma_i \overset{E}{\rightsquigarrow} \Gamma_{i+n}$, that comprises a causal sequence of events

$$\Gamma_i \overset{e_i}{\curvearrowright} \ldots \xrightarrow{e_{i+n-1}} \Gamma_{i+n}$$

such that each consequent situation Γ_{i+j}, $0 < j \leq n$ in the sequence triggers an error, i.e. there exists some capability f_j such that

$$\Gamma_{i+j} \xrightarrow{\varepsilon\langle f_j \rangle} \Gamma_{n_j}$$

Then *all* these events are mereological parts of E, i.e.

$$\text{has-part}(E, e_i); \qquad \text{has-part}(E, \varepsilon\langle f_j \rangle)$$

Errors and functional dependency. By Definition 18, when a producer of a particular resource experiences an error over the capability that produces it, it no longer provides that resource to any other capabilities. And through functional dependency, the manifestation of a capability as a function depends on access to certain types of resources.

These facts together mean that error propagation is observed if in an antecedent situation Γ, the following holds:

1) A capability g provides a channel c with a resource r, and;
2) A capability f, with r as prerequisite, is provided with r by c, and;
3) Let $h = \mathrm{cons}_\Gamma(c,r)$ be the input capability of channel c, such that there exists no minimal configuration fragment for h that does not include g, and;
4) Let $k = \mathrm{prod}_\Gamma(c,r)$ be the output capability of cannel c, such that there exists no minimal configuration fragment for f that does not include k, and;
5) There exists an expectation for the manifestation of f (whereby there exist expectations for k, h and g).

Then...

1) If in the situation Γ fault creation occurs over g, i.e. $\Gamma \overset{\lightning\{g\}}{\hookrightarrow} \Gamma'$, then;

Fault activation occurs, such that an error $\varepsilon\langle g\rangle$ is triggered over g, i.e. $\Gamma' \xrightarrow{\varepsilon\langle g\rangle} \Gamma'_n$, and;
Also in Γ' an error $\varepsilon\langle h\rangle$ is triggered over h, i.e. $\Gamma' \xrightarrow{\varepsilon\langle h\rangle} \Gamma'_{n'}$.

2) Then in the situation Γ', the capability h is no longer manifested, and a depletion event Δ is triggered over the bearing channel, i.e. $\Gamma' \xrightarrow{\Delta\langle\beta(h)\rangle} \Gamma''$, and;
In the situation Γ'', k is no longer manifested and an error is triggered over k, such that $\Gamma'' \xrightarrow{\varepsilon\langle k\rangle} \Gamma''_j$, and;
In that situation Γ'', f is no longer manifested and an error $\varepsilon\langle f\rangle$ is triggered over f, i.e. $\Gamma'' \xrightarrow{\varepsilon\langle f\rangle} \Gamma''_{j'}$.

Let $\Gamma_i \overset{E}{\rightsquigarrow} \Gamma_{i+n}$ denote a causal chain of error propagation, such that

$$\Gamma_i \overset{E}{\rightsquigarrow} \Gamma_{i+n} \iff \Gamma_i \overset{e}{\curvearrowright} \Gamma_{i+1} \xrightarrow{\Delta_{i+1}} \dots \xrightarrow{\Delta_{i+n-1}} \Gamma_{i+n}$$

where the event e and the depletion events Δ_{i+1} and Δ_{i+2} combine to form the complex error propagation event:

$$\text{has-part}(E, e)$$

and

$$\forall j.\ i < j < i+n.\ \text{has-part}(E, \Delta_j)$$

Lastly, we call such a causal chain an *error propagation* iff in the consequent situation of each event an error is triggered:

$$\forall j.\ i \leq j < i+n.\ \exists \Gamma' : \mathcal{G}.\ \Gamma_j \xrightarrow{\varepsilon_j} \Gamma' \wedge \text{has-part}(E, \varepsilon_j)$$

The situation this complex event brings about is determined by the temporal order of the situations brought about by the events it comprises. We find that the error propagation E, with $n \geq 3$ events and $n+1$ situations $\Gamma_i, \dots, \Gamma_{i+n}$, is a sequence over $\prec$, bringing about the latest situation Γ_{i+n} in that sequence.

Example of error propagation. In Figure 7 we show a smaller version of the running example introduced in Figure 6. Let g be the symbol for the `Ink reservoir`'s capability to *provide ink*. Let f denote the `Print head`'s capability to *deposit ink*. Let there be a channel c between them that accommodates `Ink` (denoted r). Let $h = \text{cons}_\Gamma(c, r)$ be the channel's input capability, and let $k = \text{prod}_\Gamma(c, r)$ be the channel's output capability. Lastly, recall that $\bar{g}$ and $\bar{f}$ denote the manifestation of the capabilities g and f as a function respectively.

The left-hand side of Figure 7 labels each situation, and the edges between those labels depict events (e.g. the event $\Gamma \overset{\lightning\{\not{g}\}}{\hookrightarrow} \Gamma'$ is depicted).

We wish to place an expectation on the manifestation of f. By Stipulate 1, we must then also place an expectation on k, h and g. Figure 7 then shows how an error over g propagates, through c, to an error over f.

A solid arrow in the diagram denotes the presence of an interface. As the errors propagate, the interactions between capabilities are terminated and the arrows are removed. The dashed arrow signifies a satisfied functional dependency between the output and input capabilities of a channel (for a particular resource type).

Furthermore, if $\bar{f}$ provides another expected function in Γ'', the propagation continues.

Remark (The end point of an error). Note that in UFO, all events must have an endpoint. Events are also immutable. Foregoing the philosophical debate on determinism v.s. free will, this only makes sense if these points are necessarily in the past. Therefore, these error events must have obtained in a situation before we "query" (or "factually know") anything about such error events.

Remark (The causality of errors). There are no causal relationships between error events themselves. The causal aspect of an error propagation is carried by the causality between fault creation and depletion events across functional dependency.

Proposition 2 (Direct causes for errors)**.** For any function, fault creation (Definition 19) and propagation of depletion events through unmet functional dependency (Definition 29) make up the two possible direct causes for errors in CIGs.

IX. CIG Analysis

In this section we provide an error function that builds upon the notion of error propagation from the previous section. This error function is instrumental in shaping the FT synthesis algorithm.

The error function of CIGs answers a diagnostic question: *"What events could lead to an error over this capability?"* This question is, in a way, the dual of the predictive question answered by the definitions of fault activation and error propagation: *"What are the consequences of an error over this capability?"* Since we have at present already developed that "predictive" framework, we use it now to answer the diagnostic question.

Table IV provides a glossary of the perdurant CIG concepts introduced earlier.

Definition 30 (CIG error function)**.** We use the notation $[\![f]\!]^{\Gamma_n}_{\Gamma_0} \in \{\top, \bot\}$ to indicate that a capability f is erroneous in the situation Γ_n:

$$\forall \Gamma_n : \mathcal{G}. \forall f \in \mathscr{C}_{\Gamma_n}.\ [\![f]\!]^{\Gamma_n}_{\Gamma_0} \iff \exists \Gamma_{n-i}, \Gamma_{n+j} : \mathcal{G}.$$
$$\Gamma_0 \preceq \Gamma_{n-i} \preceq \Gamma_n \prec \Gamma_{n+j} \wedge \Gamma_{n-i} \xrightarrow{\varepsilon\langle f\rangle} \Gamma_{n+j}$$

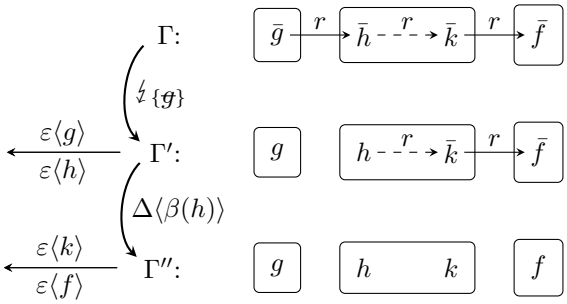


Fig. 7: An example of error propagation.

Notation	Description
$\Gamma \prec \Gamma'$	Situation Γ precedes the situation Γ'
$\Gamma \preceq \Gamma'$	Situation Γ precedes, or coincides with, Γ'
$\Gamma \overset{e}{\curvearrowright} \Gamma'$	Event e with antecedent Γ and consequent Γ'
$\Gamma \xrightarrow{e} \Gamma'$	A triggered event e
$\Gamma \overset{\hat{e}}{\hookrightarrow} \Gamma'$	A spontaneous event $\hat{e}$
$\Gamma \xrightarrow{\Delta\langle c\rangle} \Gamma'$	A depletion event over the channel c
$\Gamma \overset{\lightning F}{\hookrightarrow} \Gamma'$	A fault creation event, creating the faults F
$\Gamma \xrightarrow{\varepsilon\langle e\rangle} \Gamma'$	An error event over the expectation e
$\Gamma \overset{\lightning F}{\hookrightarrow} \Gamma' \xrightarrow{\varepsilon\langle e\rangle} \Gamma''$	Fault activation over $e \in F$
$\Gamma_i \overset{E}{\rightsquigarrow} \Gamma_{i+n}$	Error propagation

TABLE IV: A glossary of perdurant CIG concepts.

A structure function implementation. In its current form, the definition of the error function given in Definition 30 is still not very helpful in answering the diagnostic question. We have described in Proposition 2 that, to find all possible causes for an error $\Gamma_{n-i} \xrightarrow{\varepsilon\langle f\rangle} \Gamma_{n+j}$ that spans the situation Γ_n, we must consider both i) fault activation of a fault created in its bearer and ii) error propagation through functional dependency. I.e. given $[\![f]\!]_{\Gamma^n} = \top$, we know that there must exist a previous situation Γ_{n-i} from which either fault creation or error propagation occurred.

To cite the principal NASA handbook on Fault Tree construction [34]:

> *"In this way the analyst proceeds down the tree continually transferring the point of view from mechanism to mode, and continually approaching finer resolution in defining mechanisms and modes, until ultimately, the limit of resolution of the tree is reached. This limit consists of basic component failures of one sort or another, and the tree is now complete."*

We now split on fault creation and the three cases of functional dependency to derive such a recursive structure function.

First, by definition of errors (Definition 18) there must exist an expectation $f \in \mathscr{E}_{\Gamma_n}$. I.e.:

$$[\![f]\!]_{\Gamma_0}^{\Gamma_n} \implies f \in \mathscr{E}_{\Gamma_n} \tag{26}$$

Then, let E_{Γ_n} be a set of erroneous capabilities in Γ_n:

$$E_{\Gamma_n} = \{f \mid f \in \mathscr{C}_{\Gamma_n}.\ \text{ErroneousCapability}(\iota_{\Gamma_n}(f))\} \tag{27}$$

By definition of the erroneous phase of capabilities, there exists an error for each element of E_{Γ_n} whose begin point is in Γ_n or precedes Γ_n, and whose end point is after Γ_n:

$$\forall f \in E_{\Gamma_n}.\exists \Gamma_{n-i}, \Gamma_{n+j} : \mathcal{G}.\ \Gamma_{n-i} \preceq \Gamma_n \prec \Gamma_{n+j} \wedge \Gamma_{n-i} \xrightarrow{\varepsilon\langle f\rangle} \Gamma_{n+j} \tag{28}$$

Let $\Gamma_0, \Gamma_n : \mathcal{G}$ be CIGs, such that $\Gamma_0 \preceq \Gamma_n$. Let $f \in \mathscr{C}_{\Gamma_n}$ be a capability. By Definition 30, it is clear that

$$[\![f]\!]_{\Gamma_0}^{\Gamma_n} \iff f \in E_{\Gamma_n}$$

Fault Creation. Let $f \in \mathscr{E}_{\Gamma_n}$ be an expectation for the manifestation of f in Γ_n. Then by (18):

$$\left(\exists \Gamma_{n-i}, \Gamma'_{n-i} : \mathcal{G}.\ \Gamma_{n-i} \overset{\lightning\{f\}}{\hookrightarrow} \Gamma'_{n-i}\right) \implies f \in E_{\Gamma_n} \tag{29}$$

where $\Gamma'_{n-i} \preceq \Gamma_n$.
Furthermore, error propagation can occur;

Propagation case A (input propagation).
The capability f is a component capability:

$$\exists \alpha \in \mathcal{C}_{\Gamma_n}.\ f \in \mathscr{C}^{\alpha}_{\Gamma_n}$$

Let $f \in \mathscr{E}_{\Gamma_n}$ be an expectation for the manifestation of f in Γ_n. Then:

$$\exists r \in \psi_{\Gamma_n}(f).\ \left(\forall S^A \in \mathscr{D}^A_{\Gamma_0}(f, r).\ S^A \subseteq E_{\Gamma_n}\right) \implies f \in E_{\Gamma_n}$$

By (28), $S^A \subseteq E_{\Gamma_n}$ implies there is an error event for all capabilities in each set of minimal configuration fragments in $\mathscr{D}^A_{\Gamma_0}(f, r)$. I.e.:

$$\exists r \in \psi_{\Gamma_n}(f).\ \left(\bigwedge_{S^A \in \mathscr{D}^A_{\Gamma_0}(f,r)} S^A \subseteq E_{\Gamma_n}\right) \implies f \in E_{\Gamma_n} \tag{30}$$

A conjunction over the empty set is vacuously true, i.e.:

$$\bigwedge_{S^A \in\ \emptyset} = \top$$

whereby $\mathscr{D}^A_{\Gamma_n}(f, r) = \emptyset$ implies that Equation 30 is (vacuously) true. This matches the semantics that "if we cannot construct any such minimal configuration fragment, then the prerequisites for the manifestation of a capability are not met and the capability cannot be manifested" (Definition 25).

Propagation case B (depletion propagation).
The capability f is a channel production capability:

$$\exists c \in \vec{\mathcal{C}}_{\Gamma_n}.\exists r \in \mathcal{R}.\ f = \text{prod}_{\Gamma_n}(c, r)$$

Given that $f \in \mathscr{E}_{\Gamma_n}$:

$$\left(\exists \Gamma_{n-i}, \Gamma'_{n-i} : \mathcal{G}.\Gamma_{n-i} \xrightarrow{\Delta\langle\beta(f)\rangle} \Gamma'_{n-i}\right) \implies f \in E_{\Gamma_n}$$

where $\Gamma'_{n-i} \preceq \Gamma_n$.

Propagation case C (output propagation).
f is a channel consumption capability:

$$\exists c \in \vec{\mathcal{C}}_{\Gamma_n}.\exists r \in \mathcal{R}.\ f = \text{cons}_{\Gamma_n}(c, r)$$

Given that $f \in \mathscr{E}_{\Gamma_n}$:

$$\exists r \in \psi_{\Gamma_n}(f).\ \left(\forall S^C \in \mathscr{D}^C_{\Gamma_0}(f, r).\ S^C \subseteq E_{\Gamma_n}\right) \implies f \in E_{\Gamma_n}$$

$$\forall \Gamma_n : \mathcal{G}.\ f \in \mathscr{C}_{\Gamma_n}.\ [\![f]\!]^{\Gamma_n}_{\Gamma_0} \iff \mathrm{id}_f \in \mathscr{E}_{\Gamma_n} \wedge \Bigg(\exists \Gamma_{n-i}, \Gamma'_{n-i} : \mathcal{G}.\ \Gamma_{n-i} \overset{\lightning_F}{\hookrightarrow} \Gamma'_{n-i} \ \vee \bigvee_{r \in \psi_{\Gamma_n}(f)} \Bigg(\Big(\exists \alpha \in \mathcal{C}_{\Gamma_n}.\ f \in \mathscr{C}^{\alpha}_{\Gamma_n} \implies \bigwedge_{S^A \in \mathscr{D}^A_{\Gamma_0}(f,r)} \bigvee_{k \in S^A} [\![k]\!]^{\Gamma_n}_{\Gamma_0} \Big) \vee \Big(\exists c \in \vec{\mathcal{C}}_{\Gamma_n}.\ f = \mathrm{prod}_{\Gamma_n}(c, r) \implies \exists \Gamma_{n-j} : \mathcal{G}.\ [\![\mathscr{D}^B_{\Gamma_0}(f, r)]\!]^{\Gamma_{n-j}}_{\Gamma_0} \Big) \vee \Big(\exists c \in \vec{\mathcal{C}}_{\Gamma_n}.\ f = \mathrm{cons}_{\Gamma_n}(c, r) \implies \bigwedge_{S^C \in \mathscr{D}^C_{\Gamma_0}} \bigvee_{g \in S^C} [\![g]\!]^{\Gamma_n}_{\Gamma_0} \Big) \Bigg) \Bigg)$$

Where $\Gamma_0 \prec \Gamma_{n-i}$ and $\Gamma'_{n-i} \preceq \Gamma_n$, and $\mathrm{id}_f = \iota_{\Gamma_n}(f)$, and $\mathrm{id}_{\lightning} \in F$, and $\Gamma_{n-j} \prec \Gamma_n$.
Additionally, there exists time points t_{n-j} and t_n such that $\mathrm{obtainsIn}(\Gamma_{n-j}, t_{n-j})$ and $\mathrm{obtainsIn}(\Gamma_n, t_n)$, and $t_n - t_{n-j} \geq \delta$.

Fig. 8: The definition of the CIG structure function $[\![\cdot]\!]^{\Gamma_n}_{\Gamma_0}$.

By (27), $S^C \subseteq E_{\Gamma_n}$ implies there is an error event for all capabilities in each set of minimal configuration fragments in $\mathscr{D}^C_{\Gamma_0}(f, r)$. I.e.:

$$\exists r \in \psi_{\Gamma_n}(f). \left(\bigwedge_{S^C \in \mathscr{D}^C_{\Gamma_0}(f,r)} S^C \subseteq E_{\Gamma_n} \right) \implies f \in E_{\Gamma_n} \qquad (31)$$

Recall that we defined conjunction such that $\bigwedge_{S^C \in \emptyset} = \top$. Therefore, $\mathscr{D}^C_{\Gamma_n}(f, r) = \emptyset$ implies that Equation 31 is (vacuously) true. This matches the semantics that "if we cannot construct any such minimal configuration fragment, then the prerequisites for the manifestation of a capability are not met and the capability cannot be manifested" (Definition 27).

Final remarks. For completeness, we must also define the case in which Γ_n *is* the initial situation Γ_0 . This is trivial by the definitions or errors (Definition 18) and causation (Definition 16); let $f \in \mathscr{E}_{\Gamma_n}$. Then:

$$[\![f]\!]^{\Gamma_n}_{\Gamma_0} \iff f \notin \mathrm{dom} \varrho_{\Gamma_n}$$

and $\varepsilon\langle f \rangle$ has no cause.

Furthermore, the evaluation of a capability using this structure function is only possible if there exists a future situation Γ_{n+j} such that $\Gamma_n \prec \Gamma_{n+j}$ in or before which all relevant errors have their end point.

Example. In Figure 9 we show a system with redundant cables between the power supply unit (PSU) and the print head from the running example. Let g denote the `PSU`'s capability to *provide electricity*. Let h_0, k_0, and h_1, k_1 respectively denote `cable harness 1` and `cable harness 2`'s capabilities to *transport electricity*. Finally, let f denote the `ink head`'s capability to *deposit ink*.

We now show that, using Figure 8, we can symbolically determine all causes for an error over $\bar{f}$ in situation Γ^n w.r.t. the situation Γ_0 as depicted in Figure 9. Let $\gamma \in \mathcal{C}_{\Gamma_0}$ and $\alpha \in \mathcal{C}_{\Gamma_0}$ be components, such that $\bar{g} \in \mathbb{F}^{\gamma}_{\Gamma_0}$ and $\bar{f} \in \mathbb{F}^{\alpha}_{\Gamma_0}$. Let $c_0 \in \vec{\mathcal{C}}_{\Gamma_0}$ and $c_1 \in \vec{\mathcal{C}}_{\Gamma_0}$ be channels, such that $h_0, k_0 \in \mathbb{F}^{c_0}_{\Gamma_0}$ and $h_1, k_1 \in \mathbb{F}^{c_1}_{\Gamma_0}$.

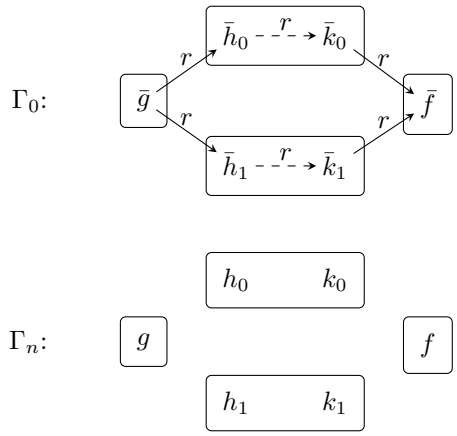


Fig. 9: An example of a system with redundancy.

The arrows in Figure 9 depict interfaces. I.e.: let $o_0 \in \mathcal{O}_{\Gamma_0}$ be an output interface, such that $\varpi^o_{\Gamma_0}(o_0) = \{\iota_{\Gamma_0}(g)\}$ and $\varsigma^o_{\Gamma_0}(o_0) = \iota_{\Gamma_0}(h_0)$ and $\mathrm{proj}_3(o_0) = r$. Let $o_1 \in \mathcal{O}_{\Gamma_0}$ be an output interface, such that $\varpi^o_{\Gamma_0}(o_1) = \{\iota_{\Gamma_0}(g)\}$ and $\varsigma^o_{\Gamma_0}(o_1) = \iota_{\Gamma_0}(h_1)$ and $\mathrm{proj}_3(o_1) = r$.

Let $i_0 \in \mathcal{I}_{\Gamma_0}$ be an input interface, such that $\varpi^i_{\Gamma_0}(i_0) = \{\iota_{\Gamma_0}(f)\}$ and $\varsigma^i_{\Gamma_0}(i_0) = \iota_{\Gamma_0}(k_0)$ and $\mathrm{proj}_3(i_0) = r$. Let $i_1 \in \mathcal{I}_{\Gamma_0}$ be an input interface, such that $\varpi^i_{\Gamma_0}(i_1) = \{\iota_{\Gamma_0}(f)\}$ and $\varsigma^i_{\Gamma_0}(i_1) = \iota_{\Gamma_0}(k_1)$ and $\mathrm{proj}_3(i_1) = r$.

From this, it follows that $\mathrm{cons}_{\Gamma_0}(c_0, r) = h_0$ and $\mathrm{prod}_{\Gamma_0}(c_0, r) = k_0$, and $\mathrm{cons}_{\Gamma_0}(c_1, r) = h_1$ and $\mathrm{prod}_{\Gamma_0}(c_1, r) = k_1$.

Derivation. Table V presents the functional dependency relationships in Γ_0 so that we can refer to them in the derivation. We also use the symbol $\lightning$ as shorthand notation for the existence of a fault creation event;

$$\lightning^{\Gamma_n}_{\Gamma_0}\langle \mathrm{id}_f \rangle \iff \exists \Gamma_{n-i}, \Gamma'_{n-i} : \mathcal{G}.\ \Gamma_{n-i} \overset{\lightning_F}{\hookrightarrow} \Gamma'_{n-i}$$

where $\Gamma_0 \prec \Gamma_{n-i}$ and $\Gamma'_{n-i} \preceq \Gamma_n$, and $\mathrm{id}_{\lightning} \in F$.

Capability	$\mathscr{D}^A_{\Gamma_0}(\cdot, r)$	$\mathscr{D}^B_{\Gamma_0}(\cdot, r)$	$\mathscr{D}^C_{\Gamma_0}(\cdot, r)$
f	$\{\{k_0\}, \{k_1\}\}$	–	–
k_0	–	h_0	–
k_1	–	h_1	–
h_0	–	–	$\{\{g\}\}$
h_1	–	–	$\{\{g\}\}$
g	$\emptyset$	–	–

TABLE V: The functional dependency relationships in Γ_0 of Figure 9.

We evaluate the structure function $[\![f]\!]^{\Gamma_n}_{\Gamma_0}$ for the presence of an error over the capability f;

$$\begin{aligned}
[\![f]\!]^{\Gamma_n}_{\Gamma_0} &\iff \mathrm{id}_f \in \mathscr{E}_{\Gamma_n} \wedge \left(\lightning^{\Gamma_n}_{\Gamma_0} \langle \mathrm{id}_f \rangle \vee \left([\![k_0]\!]^{\Gamma_n}_{\Gamma_0} \wedge [\![k_1]\!]^{\Gamma_n}_{\Gamma_0} \right) \right) \\
[\![k_0]\!]^{\Gamma_n}_{\Gamma_0} &\iff \lightning^{\Gamma_n}_{\Gamma_0} \langle \mathrm{id}_{k_0} \rangle \vee \left(\exists \Gamma_{n-j} : \mathcal{G}.\ [\![h_0]\!]^{\Gamma_{n-j}}_{\Gamma_0} \right) \\
[\![k_1]\!]^{\Gamma_n}_{\Gamma_0} &\iff \lightning^{\Gamma_n}_{\Gamma_0} \langle \mathrm{id}_{k_1} \rangle \vee \left(\exists \Gamma_{n-j'} : \mathcal{G}.\ [\![h_1]\!]^{\Gamma_{n-j'}}_{\Gamma_0} \right) \\
[\![h_0]\!]^{\Gamma_{n-j}}_{\Gamma_0} &\iff \lightning^{\Gamma_{n-j}}_{\Gamma_0} \langle \mathrm{id}_{h_0} \rangle \vee [\![g]\!]^{\Gamma_{n-j}}_{\Gamma_0} \\
[\![h_1]\!]^{\Gamma_{n-j'}}_{\Gamma_0} &\iff \lightning^{\Gamma_{n-j'}}_{\Gamma_0} \langle \mathrm{id}_{h_1} \rangle \vee [\![g]\!]^{\Gamma_{n-j'}}_{\Gamma_0} \\
[\![g]\!]^{\Gamma_{n-j}}_{\Gamma_0} &\iff \lightning^{\Gamma_{n-j}}_{\Gamma_0} \langle \mathrm{id}_g \rangle \\
[\![g]\!]^{\Gamma_{n-j'}}_{\Gamma_0} &\iff \lightning^{\Gamma_{n-j'}}_{\Gamma_0} \langle \mathrm{id}_g \rangle
\end{aligned} \tag{32}$$

Recall that expectation is transitive over functional dependency (e.g. $\mathrm{id}_f \in \mathscr{E}_{\Gamma_n} \implies \mathrm{id}_{k_0} \in \mathscr{E}_{\Gamma_n}$).

Bottom-up substitution of the recursively defined equivalences yields a boolean formula with atomic propositions. E.g.:

$$[\![h_0]\!]^{\Gamma_{n-j}}_{\Gamma_0} \iff \lightning^{\Gamma_{n-j}}_{\Gamma_0} \langle \mathrm{id}_{h_0} \rangle \vee [\![g]\!]^{\Gamma_{n-j}}_{\Gamma_0}$$

becomes:

$$[\![h_0]\!]^{\Gamma_{n-j}}_{\Gamma_0} \iff \lightning^{\Gamma_{n-j}}_{\Gamma_0} \langle \mathrm{id}_{h_0} \rangle \vee \lightning^{\Gamma_{n-j}}_{\Gamma_0} \langle \mathrm{id}_g \rangle$$

There are many sequences[4] of events that satisfy (32). One example is given in Figure 10. The set of configurations that satisfy this formula are exactly the distinct failure modes that satisfy the structure function of the corresponding fault tree.

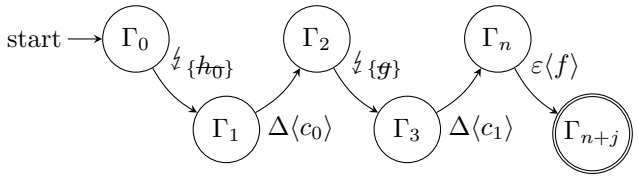


Fig. 10: A sequence of events that satisfies the formula $[\![f]\!]^{\Gamma_n}_{\Gamma_0}$.

[4]"Sequence" here is a loose term that also permits concurrent collections of events.

RDF triple pattern	**CIG construct**
?x rdf:type :`Class`	$x \in \phi(\texttt{Class})_{\Gamma_0}$
?x :has ?f	$f \in \mathscr{C}^x_{\Gamma_0}$
?x :consumes ?r	$r \in \mathrm{proj}_2\ \iota^{-1}(x)$
?x :produces ?r	$f \in \mathrm{proj}_3\ \iota^{-1}(x)$
?c :accommodates ?r	$r \in \mathrm{proj}_2\ \iota^{-1}(c)$
?i :channel ?c	$c = \mathrm{proj}_1\ \iota^{-1}(i)$
?i :component ?a	$a = \mathrm{proj}_2\ \iota^{-1}(i)$
?i :component_capabilities ?f	$f \in \varpi_{\Gamma_0}(\iota^{-1}(i))$
?i :channel_capabilities ?f	$f = \varsigma_{\Gamma_0}(\iota^{-1}(i))$

TABLE VI: The semantic mapping between KGs and CIGs, with $\phi = \{\text{Component} \mapsto \mathcal{C}, \text{Capability} \mapsto \mathscr{C}, \text{ResourceType} \mapsto \mathcal{R}, \text{Channel} \mapsto \vec{\mathcal{C}}, \text{InputInterface} \mapsto \mathcal{I}, \text{OutputInterface} \mapsto \mathcal{O}\}$.

X. KG querying and CIG construction

In this section, we take a side-step and show how we query KGs to instantiate CIGs.

We assume the KG is well-formed with respect to the ontological constraints. Enforcement of these data constraints in the GraphDB database (e.g. using SHACL) is a technical detail that we do not discuss in this work.

A. Semantic mapping

Table VI shows the a straightforward semantic mapping between RDF triple patterns and CIG elements.

It is important to note that the state of the KG represents **only** the initial system configuration. None of the dynamic features of the domain, i.e. the changes in situations that events bring about, are encoded in the KG. The CIG constructed from the KG is the initial situation, denoted Γ_0.

Listing 1 shows the SPARQL query Q that retrieves components, functions and resource exchanges from a KG. We may interpret it, in plain English, as

> *"What are all component capabilities that exchange resources through interfacing with channel capabilities?"*

The formal semantics of SPARQL are given in [27], [28], [31]. The result mapping $\mu : V \to I$, is defined for all variables that we `SELECT` in Listing 1.

We can evaluate Q over an instantiated KG, which yields a result set $\Omega \in \mathcal{P}(\mu)$:

$$[\![Q]\!]_{\mathrm{KG}} = \Omega$$

Where all selected variables are bound, and thus

$$\forall \mu \in \Omega.\ \mathrm{dom}\ \mu = \{?\alpha, ?f, ?c, ?k, ?r, ?h, ?\zeta, ?g, ?i, ?o\}$$

```
PREFIX : <...>
PREFIX rdf: <...>
SELECT ?alpha ?f ?c ?k ?r ?h ?zeta ?g
 ?inputinterface ?outputinterface
WHERE{
 ?r rdf:type :ResourceType.

 ?alpha rdf:type :Component.
  ?alpha :has ?f.
   ?f rdf:type :Capability.
   ?f :consumes ?r.

 ?zeta rdf:type :Component.
  ?zeta :has ?g.
   ?g rdf:type :Capability.
   ?g :produces ?r.

 ?inputinterface rdf:type :InputInterface.
  ?inputinterface :component ?alpha.
  ?inputinterface :channel ?c.
  ?inputinterface :component_capabilities ?f.
  ?inputinterface :channel_capabilities ?k.

 ?c rdf:type :Channel.
  ?c :accommodates ?r.
   ?c :has ?k.
    ?k rdf:type :Capability.
    ?k :produces ?r.
   ?c :has ?h.
    ?h rdf:type :Capability.
    ?h :consumes ?r.

 ?outputinterface rdf:type :OutputInterface.
  ?outputinterface :channel ?c.
  ?outputinterface :component ?zeta.
  ?outputinterface :channel_capabilities ?h.
  ?outputinterface :component_capabilities ?g.
}
```

Listing 1: The SPARQL query Q.

B. CIG construction

We introduce a function $T : \mathcal{P}(\mu) \to \Gamma_0$ that constructs a CIG from a result set over KG. We then build the initial CIG Γ_0 from Ω in parts.

Let the set of partial capabilities be

$$\partial\mathscr{C} = \bigcup_{?x \in \{?f, ?h\}} \{(\mu(?x), \{\mu(?r)\}, \emptyset) \mid \mu \in \Omega\} \cup \bigcup_{?x \in \{?g, ?k\}} \{(\mu(?x), \emptyset, \{\mu(?r)\}) \mid \mu \in \Omega\}$$

Let $\mathrm{CapabilityId}_{\Gamma_0}$ be the set of capability IDs in Γ_0:

$$\mathrm{CapabilityId}_{\Gamma_0} = \{\mathrm{proj}_1\ x \mid x \in \partial\mathscr{C}\}$$

Let $R_{\mathrm{in}}(\mathrm{id})$ be the set of resource consumptions of the capability identified by id:

$$R_{\mathrm{in}}(\mathrm{id}) = \bigcup \{\mathrm{proj}_2\ x \mid x \in \partial\mathscr{C}.\ \mathrm{proj}_1\ x = \mathrm{id}\}$$

Similarly, let $R_{\mathrm{out}}(\mathrm{id})$ be the set of resource productions of the capability identified by id:

$$R_{\mathrm{out}}(\mathrm{id}) = \bigcup \{\mathrm{proj}_3\ x \mid x \in \partial\mathscr{C}.\ \mathrm{proj}_1\ x = \mathrm{id}\}$$

The set of capabilities $\mathscr{C}_{\Gamma_0}$ is then:

$$\mathscr{C}_{\Gamma_0} = \left\{(\mathrm{id}, R_{\mathrm{in}}(\mathrm{id}), R_{\mathrm{out}}(\mathrm{id})) \mid \mathrm{id} \in \mathrm{CapabilityId}_{\Gamma_0}\right\}$$

By Postulate 4, there exists a function for each capability such that this function manifests that unique capability. Let the relation $\vartheta : \mathrm{CapabilityId}_{\Gamma_0} \to \mathrm{FunctionId}_{\Gamma_0}$ return this unique manifestation. Let the set of functions in Γ_0 permit a bijection over ϑ w.r.t. the set of capability IDs $\mathrm{CapabilityId}_{\Gamma_0}$:

$$\mathbb{F}_{\Gamma_0} = \left\{\vartheta(\mathrm{id}) \mid \mathrm{id} \in \mathrm{CapabilityId}_{\Gamma_0}\right\}$$

Let the set of faults $\mathscr{F}_{\Gamma_0} = \emptyset$ be the empty set.

We can now construct the set of partial components $\partial\mathcal{C}$:

$$\partial\mathcal{C} = \left\{(\mu(?\alpha), \{\iota_{\Gamma_0}^{-1}(\mu(?f))\}, \{\vartheta(\mu(?f))\}, \emptyset) \mid \mu \in \Omega\right\}$$

Let $\mathrm{ComponentId}_{\Gamma_0}$ be the set of component IDs in Γ_0:

$$\mathrm{ComponentId}_{\Gamma_0} = \{\mathrm{proj}_1\ x \mid x \in \partial\mathcal{C}\}$$

Let $C(\mathrm{id})$ be the set of capabilities of the component identified by id:

$$C(\mathrm{id}) = \bigcup \{\mathrm{proj}_2\ x \mid x \in \partial\mathcal{C}.\ \mathrm{proj}_1\ x = \mathrm{id}\}$$

Similarly, let $F(\mathrm{id})$ be the set of component functions of the capability identified by id:

$$F(\mathrm{id}) = \bigcup \{\mathrm{proj}_3\ x \mid x \in \partial\mathcal{C}.\ \mathrm{proj}_1\ x = \mathrm{id}\}$$

And the set of components $\mathcal{C}_{\Gamma_0}$ is then:

$$\mathcal{C}_{\Gamma_0} = \left\{(\mathrm{id}, C(\mathrm{id}), F(\mathrm{id}), \emptyset) \mid \mathrm{id} \in \mathrm{ComponentId}_{\Gamma_0}\right\}$$

Next, we construct the set of partial channels $\partial\vec{\mathcal{C}}$. The process is similar as for components:

$$\partial\vec{\mathcal{C}} = \left\{(\mu(?c), \{\iota_{\Gamma_0}^{-1}(\mu(?h)), \iota_{\Gamma_0}^{-1}(\mu(?k))\}, \{\vartheta(\mu(?h)), \vartheta(\mu(?k))\}, \emptyset) \mid \mu \in \Omega\right\}$$

Let $\mathrm{ChannelId}_{\Gamma_0}$ be the set of channel IDs in Γ_0:

$$\mathrm{ChannelId}_{\Gamma_0} = \{\mathrm{proj}_1\ x \mid x \in \partial\vec{\mathcal{C}}\}$$

Let $C'(\mathrm{id})$ be the set of capabilities of the channel identified by id:

$$C'(\mathrm{id}) = \bigcup \left\{\mathrm{proj}_2\ x \mid x \in \partial\vec{\mathcal{C}}.\ \mathrm{proj}_1\ x = \mathrm{id}\right\}$$

Similarly, let $F'(\mathrm{id})$ be the set of channel functions of the capability identified by id:

$$F'(\mathrm{id}) = \bigcup \left\{\mathrm{proj}_3\ x \mid x \in \partial\vec{\mathcal{C}}.\ \mathrm{proj}_1\ x = \mathrm{id}\right\}$$

And the set of channels $\vec{\mathcal{C}}_{\Gamma_0}$ is then:

$$\vec{\mathcal{C}}_{\Gamma_0} = \{(\mathrm{id}, C'(\mathrm{id}), F'(\mathrm{id}), \emptyset) \mid \mathrm{id} \in \mathrm{ChannelId}_{\Gamma_0}\}$$

By Equation 5, Equation 6 and Equation 7, the set of capabilities $\mathscr{C}_{\Gamma_0}$, functions $\mathbb{F}_{\Gamma_0}$ and faults $\mathscr{F}_{\Gamma_0} = \emptyset$ is simply the union of those that inhere in (and manifest modes of) components and capabilities:

$$\mathscr{C}_{\Gamma_0} = \left(\bigcup_{\alpha \in \mathcal{C}_\Gamma} \mathscr{C}_\Gamma^\alpha\right) \cup \left(\bigcup_{c \in \vec{\mathcal{C}}_\Gamma} \mathscr{C}_\Gamma^c\right)$$

$$\mathbb{F}_{\Gamma_0} = \left(\bigcup_{\alpha \in \mathcal{C}_\Gamma} \mathbb{F}_\Gamma^\alpha\right) \cup \left(\bigcup_{c \in \vec{\mathcal{C}}_\Gamma} \mathbb{F}_\Gamma^c\right)$$

The sets output ($\mathcal{O}_{\Gamma_0}$) and input ($\mathcal{I}_{\Gamma_0}$) interfaces are constructed as follows:

$$\mathcal{O}_{\Gamma_0} = \{(\mu(?\zeta), \mu(?c), \mu(?r)) \mid \mu \in \Omega\}$$
$$\mathcal{I}_{\Gamma_0} = \{(\mu(?c), \mu(?\alpha), \mu(?r)) \mid \mu \in \Omega\}$$

Such that

$$\iota^o_{\Gamma_0}((\mu(?\zeta), \mu(?c), \mu(?r))) = \mu(?\text{outputinterface})$$
$$\iota^o_{\Gamma_0}((\mu(?c), \mu(?\alpha), \mu(?r))) = \mu(?\text{inputinterface})$$

Let $\text{OutputInterfaceId}_{\Gamma_0}$ be the set of output interface IDs in Γ_0:

$$\text{OutputInterfaceId}_{\Gamma_0} = \{\mu(?\text{outputinterface}) \mid \mu \in \Omega\}$$

And let $\text{InputInterfaceId}_{\Gamma_0}$ be the set of input interface IDs in Γ_0:

$$\text{InputInterfaceId}_{\Gamma_0} = \{\mu(?\text{inputinterface}) \mid \mu \in \Omega\}$$

The partial component capabilities that the output and input interface identified by id aggregate are:

$$\partial\varpi^o(\text{id}) = \{\mu(?g) \mid \mu \in \Omega.\ \mu(?\text{outputinterface}) = \text{id}\}$$
$$\partial\varpi^i(\text{id}) = \{\mu(?f) \mid \mu \in \Omega.\ \mu(?\text{inputinterface}) = \text{id}\}$$

Whereby the functions that return the component capabilities aggregated by an output or input interface are given by:

$$\varpi^o_\Gamma = \left\{\iota^{o,-1}_{\Gamma_0}(\text{id}) \mapsto \partial\varpi^o(\text{id}) \mid \text{id} \in \text{OutputInterfaceId}_{\Gamma_0}\right\}$$
$$\varpi^i_\Gamma = \left\{\iota^{i,-1}_{\Gamma_0}(\text{id}) \mapsto \partial\varpi^i(\text{id}) \mid \text{id} \in \text{InputInterfaceId}_{\Gamma_0}\right\}$$

And the channel capabilities they aggregate are:

$$\varsigma^o_{\Gamma_0} = \left\{\iota^{o,-1}_{\Gamma_0}(\mu(?\text{outputinterface})) \mapsto \mu(?h) \mid \mu \in \Omega\right\}$$
$$\varsigma^i_{\Gamma_0} = \left\{\iota^{i,-1}_{\Gamma_0}(\mu(?\text{inputinterface})) \mapsto \mu(?k) \mid \mu \in \Omega\right\}$$

Finally, let $\mathcal{R} = \{\mu(?r) \mid \mu \in \Omega\}$ and $\mathscr{E}_{\Gamma_0} = \emptyset$.
The constructed CIG γ_0 is then:

$$(\mathscr{C}_{\Gamma_0}, \mathbb{F}_{\Gamma_0}, \mathscr{F}_{\Gamma_0}, \mathcal{C}_{\Gamma_0}, \vec{\mathcal{C}}_{\Gamma_0}, \mathcal{O}_{\Gamma_0}, \mathcal{I}_{\Gamma_0}, \mathscr{E}_{\Gamma_0}) \in \mathcal{G}$$

which obtains at some arbitrary time point t, such that

$$\Gamma_0 = (\gamma_0, t) : \mathcal{G}$$

And finally,

$$T(\Omega) = \Gamma_0$$

XI. The CIG-FT Transformation

In this section we first introduce the formal FT semantics. We then provide a semantic mapping between the CIG concepts and FT concepts introduced in the previous sections. Lastly, we define the algorithm that performs the synthesis of FTs from CIG instances.

A. Fault Tree Semantics

We present the formal structure and semantics of FTs following the summarization in [30].

CIG concept	FT concept
Capability of Interest	Top-Level Event
Error Propagation	Intermediate Event
Fault Activation	Basic Event

TABLE VII: The semantic mapping from CIG concepts to FT concepts.

Definition 31. A static fault tree (FT) is a 4-tuple $FT = \langle BE, G, T, I\rangle$, consisting of the following components.

BE is the set of basic events.
G is the set of gates, with $BE \cap G = \emptyset$. We write $E = BE \cup G$ for the set of elements.
$T : G \to \text{GateTypes}$ is a function that describes the type of each gate.
$I : G \to \mathcal{P}(E)$ describes the inputs of each gate.

Definition 32. The semantics of a fault tree FT is a function

$$\pi_{\text{FT}} : \mathcal{P}(BE) \times E \to \{0, 1\}$$

where $\pi_{\text{FT}}(S, e)$ indicates whether e fails given the set S of failed BEs. It is defined as follows.

1) **For** $e \in BE$, $\pi_{\text{FT}}(S, e) = e \in S$.
2) **For** $g \in G$ **and** $T(g) =$ **And**, let
$$\pi_{\text{FT}}(S, g) = \bigwedge_{x \in I(g)} \pi_{\text{FT}}(S, x).$$
3) **For** $g \in G$ **and** $T(g) =$ **Or**, let
$$\pi_{\text{FT}}(S, g) = \bigvee_{x \in I(g)} \pi_{\text{FT}}(S, x).$$

B. Semantic mapping

We present the semantic partial mapping to show the correspondence between CIG and FT concepts. Given the greater specificity of concepts in CIGs, we can only propose this partial mapping in one direction. After all, we can construct FTs that do not pertain to systems architectures at all (e.g. an FT about social inequality). That is to say: the partial mapping between CIG concepts and FT concepts is necessarily surjective. Additionally, not all CIG concepts are mapped to an FT concept (e.g. the CIG concept `Fault` does not have a related concept in FTs). Table VII lists the semantic partial mapping.

There are two distinct causes for errors in CIGs. An error can either be the direct result of a fault activation in the bearing component, or the indirect result of error propagation that a capability is dependent on.

Fault trees distinguish three types of events; basic events (BEs), which occur spontaneously, intermediate events (IEs), which are caused by one or more other events [30], and top-level events (TLEs), which are at the root of an FT. IEs and TLEs are associated with a gate type that models the logical behavior of failure propagation.

Top-level events. An FT's TLE is the event of interest in FTA. This is the undesired event we wish to calculate metrics over. Similarly, in CIG evaluation, we choose a capability of interest and place an expectation on it. This comes down to the same thing; the event that this capability is not (or no longer) manifested is the "undesired event". In CIG terminology, an *error*. It is therefore natural to map CIG capabilities of interest to top-level events.

Recall that by placing an expectation on a capability of interest, we transitively also place expectations on the capabilities that it is functionally dependent on. This results in a DAG of potential errors. Each vertex of this DAG (excluding the potential error over the capability of interest) gets mapped to IEs and/or BEs.

Intermediate events. Intermediate events are associated with a gate type. In standard FT's, the intermediate event with an `OR` gate type is observed when any of its input events occur. Intermediate events with an `AND` gate type are observed when *all* of its input events occur. In CIGs we distinguish two scenarios for which, if either of them occur, an error is observed over a capability f:

i. Fault activation over f, or;
ii. Error propagation preventing access to any of f's prerequisite resources.

This maps the error over f to an `OR` type intermediate event with two inputs. Scenario (i) maps to a basic event, and scenario (ii) maps each prerequisite resource type to a distinct `AND` gate.

Scenario (ii) also pertains to the notion of minimal configuration fragments w.r.t. a prerequisite resource type r. By (23), (24) and (25), the size of a minimal configuration fragment is always one. Therefore we map such errors to child events directly, instead of mapping the minimal configuration fragment to an intermediate `OR` gate with a single child event representing the error over the minimal configuration fragment's single element. In the general case, such an `OR` gate should be generated for minimal configuration fragments.

In FTs, `AND` gates describe that all of the input events must happen for the particular intermediate event to be observed. In CIGs, an error over at least one element (recall that in this work, there is only one) of *each* minimal configuration fragment over f for r then means that we observe an error over f. We therefore map the set of minimal configuration fragments over f for r to a single `AND` gate.

Basic events. An FT's basic events are stochastically independent failure events. They are in a causal relationship w.r.t. the TLE of the fault tree. In CIGs, fault creation is a spontaneous event (i.e. it is not caused by any endogenous part of the system). In the focus of this work, a fault creation event is always a direct cause of a fault activation event. This induced fault activation is exclusively dependent on the same object as the fault creation that caused it. All components are susceptible to fault creation, and the fault activation that is, in this work, necessarily triggered by the fault creation over a capability is mapped to the FT concept of the basic event.

It is important to note that, since capabilities cannot be repaired in CIGs, there can occur **at most** one fault activation over each capability.

Remark (Fault Tree Events as Event Types). The fault tree events we synthesize are in fact based on event *types*, rather than particular events. Qualitative FTA, for instance by use of the structure function, utilizes this distinction in a subtle way: membership in the set of failed basic events S represents the *occurrence* of a particular event of a certain type. This is a fundamental insight about the fault trees we synthesize from CIGs, and indeed fault trees analysis in general. Furthermore, this realization opens the door to a future ontological exploration of the quantitative analysis of FTs, e.g. in line with the notion of event type likelihood in COVER [32].

Remark. According to Veseley et al., creating an FT with no intermediate labels between its gate events is *"indicative of sloppy analysis"* [36]. Although the FT in Figure 11b does not show event labels, meaningful labels can be determined as follows:

- A fault activation event over some capability g has the exclusive participation of the bearing object o. The label that we associate with the basic event this fault activation is mapped to may take the form *"Fault activation over g in o"*.
- For an error over a channel c's input capability h that consumes a resource type r, a meaningful label for the synthesized `OR` gate may take the shape *"c fails to consume r"*.
- For an error over that channel's output capability k that produces the resource type r, the label of the corresponding `OR` may be *"c fails to provide r"*.
- For an error over a component capability f's access to a prerequisite resource r, one may choose to label its `AND` gate *"f loses sufficient access to r"*.
- For an error over a component α's output capability (or a capability of interest) ℓ, a label like *"α fails to ℓ"* is appropriate for the generated `OR` gate.

C. Transformation Definition

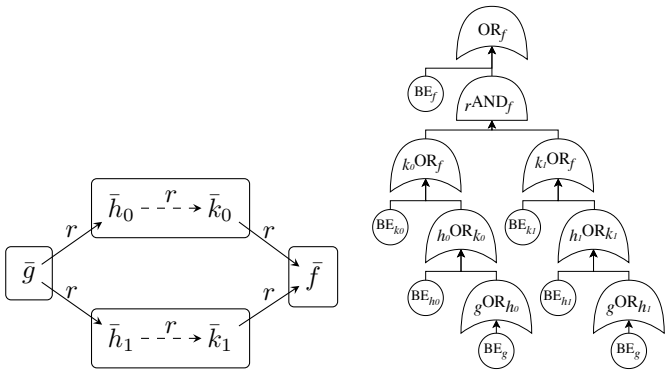
We introduce a transformation function that synthesizes a fault tree from a valid CIG.

Definition 33 (The CIG to FT transformation function)**.** The function $\tau : \mathcal{G} \times \mathscr{C}_\Gamma \to \text{FT}$ transforms a valid CIG to an FT for a given capability of interest. Its precise definition is given in pseudo code in Algorithm 1.

Recall the print head example from Figure 9. Figure 11 illustrates the transformation of that system into a fault tree $\text{FT} = \tau(\Gamma_0, f)$, where the capability of interest f denotes the ink head's capability to deposit ink.

XII. Transformation properties

In this section we prove the soundness and completeness of the synthesis algorithm presented in the previous section, with respect to error semantics.



(a) Example of the print head in its ground state Γ_0.

(b) The FT synthesized from Γ_0.

Fig. 11: Fault tree synthesized from the print head example.

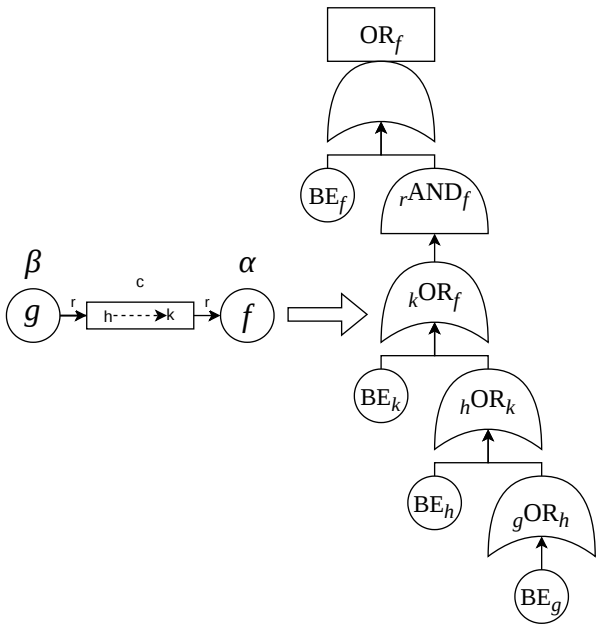


Fig. 12: An illustration of the base case.

Intuitively, the transformation is sound iff the failure semantics of the synthesized fault tree are not spurious. I.e. all positive evaluations of the structure function of the synthesized FT represent a positive evaluation of the CIG's structure function.

Conversely, our transformation is complete iff it preserves all failure semantics of the original system. All failure modes that can occur in the CIG model must be represented by a corresponding failure mode in the generated fault tree.

The transformation is sound *and* complete iff the synthesized fault tree represents the exact error semantics of the CIG it was generated from. To prove this we will make use of an intermediate result, namely that the set of failed basic events can be derived from the set of erroneous CIG components.

Lemma 1 (Derivation of failed basic events from the set of erroneous CIG components)**.** Let S denote the set of failed basic events. Let $E' \subseteq E_{\Gamma_n}$ be a subset of erroneous CIG components. The set S can be derived from $E' \subseteq E_{\Gamma_n}$ s.t.

$$\forall x \in E'.\ \exists! b_x \in S.\ b_x \leftarrow x$$

where a basic event b_x derived from an erroneous component x is denoted $b_x \leftarrow x$. Furthermore, τ does not generate spurious basic events:

$$\forall b_x \in S.\ \exists! x \in E'.\ b_x \leftarrow x$$

I.e. there exists a partial bijection from E_{Γ_n} to S.

Proof sketch for Lemma 1. Suppose $\Gamma_0, \Gamma_n : \mathcal{G}$ are valid CIGs, such that $\Gamma_0 \preceq \Gamma_n$. Let $f \in \mathscr{C}_{\Gamma_n}$ be a capability.

We use the symbol $\lightning$ as shorthand notation for the existence of a fault creation event;

$$\lightning^{\Gamma_n}_{\Gamma_0}\langle \mathrm{id}_f \rangle \iff \exists \Gamma_{n-i}, \Gamma'_{n-i} : \mathcal{G}.\ \Gamma_{n-i} \overset{\lightning\, F}{\hookrightarrow} \Gamma'_{n-i}$$

where $\Gamma_0 \prec \Gamma_{n-i}$ and $\Gamma'_{n-i} \preceq \Gamma_n$, and $\mathrm{id}_{\lightning} \in F$.

By (29), we have that

$$\lightning^{\Gamma_n}_{\Gamma_0}\langle \mathrm{id}_x \rangle \implies x \in E_{\Gamma_n}$$

It is obvious that the set of all capabilities whose fault activation is a term in $[\![f]\!]^{\Gamma_n}_{\Gamma_0}$ is then a subset $E' \subseteq E_{\Gamma_n}$.

Furthermore, from Algorithm 1, we have that τ maps the fault activation over such a capability $x \in E'$ to a unique basic event $b_x \in \mathrm{BE}$[5]. I.e.:

$$\exists! b_x \in S.\ b_x \leftarrow \lightning^{\Gamma_n}_{\Gamma_0}\langle \mathrm{id}_x \rangle$$

By simple composition of these two results, we obtain

$$\exists E' \subseteq E_{\Gamma_n}.\ \forall b_x \in S.\ \exists! x \in E'.\ b_x \leftarrow x$$

□

Theorem XII.1 (The soundness and completeness of τ)**.** Let $f \in \mathscr{C}_{\Gamma_0}$ be a capability. Let $\Gamma_0, \Gamma_n : \mathcal{G}$ be valid CIGs. Then the transformation τ is sound and complete iff

$$\forall f \in \mathscr{C}.\ [\![f]\!]^{\Gamma_n}_{\Gamma_0} \iff \pi_{\tau(\Gamma_0, f)}(S, f)$$

where S is derived from the set of erroneous components E_{Γ_n} by Lemma 1.

We proceed by induction over the size of the CIG, which is the sum of the number of exchanged resource types, channels, and components. By using bi-implication throughout the proof, we show both soundness and completeness at once.

For the base case, observe the CIG of size $I+M+N = 4$ in Figure 12, showing two component capabilities and a single channel between them. We argue that this interaction is the minimal CIG that is meaningful for analysis

[5]Note that the generation of basic events in τ is idempotent.

(the analysis of a single component with no inputs or outputs is trivial). The capability $f \in \mathscr{C}_{\Gamma_n}$ receives resource type r from $g \in \mathscr{C}_{\Gamma_n}$ through channel $c \in \vec{\mathcal{C}}_{\Gamma_n}$.

Let $\mathrm{FT} = \tau(\Gamma_0, f)$ be the fault tree synthesized from the CIG Γ_0. We show that there is a logical equivalence by a one-to-one correspondence between fault creation events and basic events:

Base case. We evaluate the structure function $[\![f]\!]^{\Gamma_n}_{\Gamma_0}$ for the existence of an error over the capability f:

$$\begin{aligned}
[\![f]\!]^{\Gamma_n}_{\Gamma_0} &\iff \lightning^{\Gamma_n}_{\Gamma_0}\langle \mathrm{id}_f\rangle \vee [\![k_0]\!]^{\Gamma_n}_{\Gamma_0}\\
[\![k_0]\!]^{\Gamma_n}_{\Gamma_0} &\iff \lightning^{\Gamma_n}_{\Gamma_0}\langle \mathrm{id}_{k_0}\rangle \vee \left(\exists \Gamma_{n-j} : \mathcal{G}.\ [\![h_0]\!]^{\Gamma_{n-j}}_{\Gamma_0}\right)\\
[\![h_0]\!]^{\Gamma_{n-j}}_{\Gamma_0} &\iff \lightning^{\Gamma_{n-j}}_{\Gamma_0}\langle \mathrm{id}_{h_0}\rangle \vee [\![g]\!]^{\Gamma_{n-j}}_{\Gamma_0}\\
[\![g]\!]^{\Gamma_{n-j}}_{\Gamma_0} &\iff \lightning^{\Gamma_{n-j}}_{\Gamma_0}\langle \mathrm{id}_g\rangle
\end{aligned}$$

Recall that by Equation 26, expectation is vacuous for any capability that we evaluate through the structure function. For readability, we therefore discard the conjunctive term $\mathrm{id}_f \in \mathscr{E}_{\Gamma_n}$. Furthermore, recall expectation is transitive over functional dependency (a mechanism reflected by the structure function). E.g. $\mathrm{id}_f \in \mathscr{E}_{\Gamma_n} \implies \mathrm{id}_{k_0} \in \mathscr{E}_{\Gamma_n}$, also omitted for readability.

Bottom-up substitution of the recursively defined equivalences yields a boolean formula with atomic propositions. E.g.:

$$[\![h_0]\!]^{\Gamma_{n-j}}_{\Gamma_0} \iff \lightning^{\Gamma_{n-j}}_{\Gamma_0}\langle \mathrm{id}_{h_0}\rangle \vee [\![g]\!]^{\Gamma_{n-j}}_{\Gamma_0}$$

becomes:

$$[\![h_0]\!]^{\Gamma_{n-j}}_{\Gamma_0} \iff \lightning^{\Gamma_{n-j}}_{\Gamma_0}\langle \mathrm{id}_{h_0}\rangle \vee \lightning^{\Gamma_{n-j}}_{\Gamma_0}\langle \mathrm{id}_g\rangle$$

Assume a set S of failed basic events, derived from the set of erroneous components as defined in Lemma 1. Recursively applying the fault tree structure function π over the fault tree FT yields:

$$\begin{aligned}
\pi_{\mathrm{FT}}(S, \mathrm{OR}_f) &\iff \mathrm{BE}_f \in S \vee \pi_{\mathrm{FT}}(S, {}_r\mathrm{AND}_f)\\
\pi_{\mathrm{FT}}(S, {}_r\mathrm{AND}_f) &\iff \pi_{\mathrm{FT}}(S, {}_k\mathrm{OR}_f)\\
\pi_{\mathrm{FT}}(S, {}_k\mathrm{OR}_f) &\iff \mathrm{BE}_k \in S \vee \pi_{\mathrm{FT}}(S, {}_h\mathrm{OR}_k)\\
\pi_{\mathrm{FT}}(S, {}_h\mathrm{OR}_k) &\iff \mathrm{BE}_h \in S \vee \pi_{\mathrm{FT}}(S, {}_g\mathrm{OR}_h)\\
\pi_{\mathrm{FT}}(S, {}_g\mathrm{OR}_h) &\iff \mathrm{BE}_g \in S
\end{aligned}$$

By bottom-up substitution, it is clear that for all capabilities $x \in \mathscr{C}_{\Gamma_n}$ such that $\lightning\langle \mathrm{id}_x\rangle$ is a term in $[\![f]\!]^{\Gamma_n}_{\Gamma_0}$, we have a one-to-one correspondence to a derived term $\mathrm{BE}_x \in S$ s.t. $\mathrm{BE}_x \leftarrow x$ in the FT structure function $\pi_{\mathrm{FT}}(S, \mathrm{OR}_f)$. □

Next, let $\Gamma_0 : \mathcal{G}$ be a valid CIG. Let $\mathrm{FT} = \tau(\Gamma_0, f)$. Let $I = |\mathcal{R}_{\Gamma_0}|$, $M = |\vec{\mathcal{C}}_{\Gamma_0}|$, and $N = |\mathscr{C}^{\mathcal{C}}_{\Gamma_0}|$ respectively denote the number of exchanged resource types, channels and component capabilities[6] in the CIG Γ_0. Let $K = I + M + N$ be the finite size of Γ_0. Let $f \in \mathscr{C}_{\Gamma_0}$ be a capability, and let $\mathrm{FT} = \tau(\Gamma_0, f)$ be the synthesized fault tree. The induction hypothesis is that $[\![f]\!]^{\Gamma_n}_{\Gamma_0} \iff \pi_{\mathrm{FT}}(S, f)$ holds for a CIG of some size K. Figure 13 depicts a CIG of size K and the generated FT.

[6] We define $\mathscr{C}^{\mathcal{C}}_{\Gamma_0} = \{f \mid f \in \mathscr{C}_{\Gamma_0}.\ \exists \alpha \in \mathcal{C}_{\Gamma_0}.\ f \in \mathscr{C}^{\alpha}_{\Gamma_0}\}$.

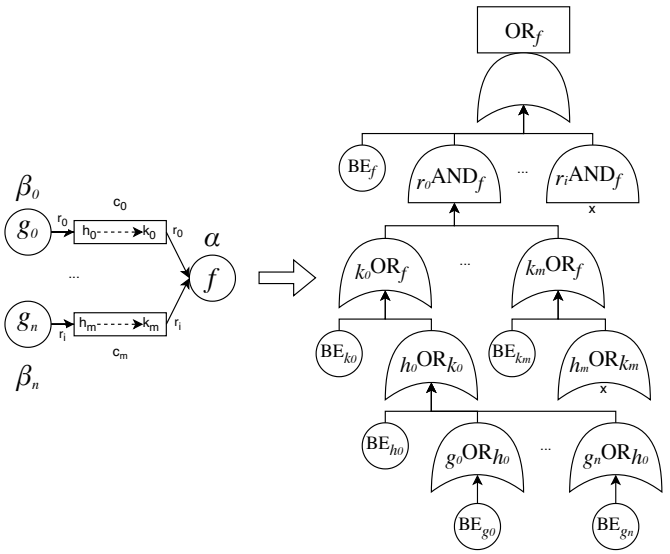


Fig. 13: An illustration of the general case of the transformation.

We derive the structure function for a generic CIG of size K for reusability. We do the same for the structure function of the synthesized FT. **CIG derivation.** We evaluate the structure function $[\![f]\!]^{\Gamma_n}_{\Gamma_0}$ for the existence of an error over the capability f:

$$\begin{aligned}
[\![f]\!]^{\Gamma_n}_{\Gamma_0} &\iff \lightning^{\Gamma_n}_{\Gamma_0}\langle \mathrm{id}_f\rangle \vee\\
&\quad \left(\left(\cdots \wedge [\![k_m^{r_0}]\!]^{\Gamma_n}_{\Gamma_0}\right) \vee \cdots \vee \left(\cdots \wedge [\![k_m^{r_i}]\!]^{\Gamma_n}_{\Gamma_0}\right)\right)\\
[\![k_0^{r_0}]\!]^{\Gamma_n}_{\Gamma_0} &\iff \lightning^{\Gamma_n}_{\Gamma_0}\langle \mathrm{id}_{k_0^{r_0}}\rangle \vee \left(\exists \Gamma_{n-j} : \mathcal{G}.\ [\![h_0^{r_0}]\!]^{\Gamma_{n-j}}_{\Gamma_0}\right)\\
&\cdots\\
[\![k_m^{r_i}]\!]^{\Gamma_n}_{\Gamma_0} &\iff \lightning^{\Gamma_n}_{\Gamma_0}\langle \mathrm{id}_{k_m^{r_i}}\rangle \vee \left(\exists \Gamma_{n-j} : \mathcal{G}.\ [\![h_m^{r_i}]\!]^{\Gamma_{n-j}}_{\Gamma_0}\right)\\
[\![h_0^{r_0}]\!]^{\Gamma_{n-j}}_{\Gamma_0} &\iff \lightning^{\Gamma_{n-j}}_{\Gamma_0}\langle \mathrm{id}_{h_0^{r_0}}\rangle \vee [\![g_0^{h_0^{r_0}}]\!]^{\Gamma_{n-j}}_{\Gamma_0} \vee \cdots \vee [\![g_n^{h_m^{r_i}}]\!]^{\Gamma_{n-j}}_{\Gamma_0}\\
&\cdots\\
[\![h_m^{r_i}]\!]^{\Gamma_{n-j}}_{\Gamma_0} &\iff \lightning^{\Gamma_{n-j}}_{\Gamma_0}\langle \mathrm{id}_{h_m^{r_i}}\rangle \vee [\![g_0^{h_0^{r_0}}]\!]^{\Gamma_{n-j}}_{\Gamma_0} \vee \cdots \vee [\![g_n^{h_m^{r_i}}]\!]^{\Gamma_{n-j}}_{\Gamma_0}\\
[\![g_0^{h_0^{r_0}}]\!]^{\Gamma_{n-j}}_{\Gamma_0} &\iff \lightning^{\Gamma_{n-j}}_{\Gamma_0}\langle \mathrm{id}_{g_0^{h_0^{r_0}}}\rangle\\
&\cdots\\
[\![g_n^{h_m^{r_i}}]\!]^{\Gamma_{n-j}}_{\Gamma_0} &\iff \lightning^{\Gamma_{n-j}}_{\Gamma_0}\langle \mathrm{id}_{g_n^{h_m^{r_i}}}\rangle
\end{aligned} \tag{33}$$

Note that Γ_{n-j} is fresh in each term $[\![h_y^{r_x}]\!]^{\Gamma_{n-j}}_{\Gamma_0}$, though we have simplified its index here for readability.

FT derivation. Assume a set S of failed basic events, derived from the set of erroneous components as defined in Theorem XII.1. Recursively applying the fault tree structure function π over the fault tree $\mathrm{FT} = \tau(\Gamma_0, f)$

yields:

$$
\begin{aligned}
\pi_{\mathrm{FT}}(S, \mathrm{OR}_f) &\iff \mathrm{BE}_f \in S \vee \\
&\quad \big(\pi_{\mathrm{FT}}(S, {}_{r_0}\mathrm{AND}_f) \vee \cdots \vee \pi_{\mathrm{FT}}(S, {}_{r_i}\mathrm{AND}_f)\big) \\
\pi_{\mathrm{FT}}(S, {}_{r0}\mathrm{AND}_f) &\iff \\
&\quad \pi_{\mathrm{FT}}(S, {}_{k_0^{r_0}}\mathrm{OR}_f) \wedge \cdots \wedge \pi_{\mathrm{FT}}(S, {}_{k_m^{r_0}}\mathrm{OR}_f) \\
&\quad \cdots \\
\pi_{\mathrm{FT}}(S, {}_{r_i}\mathrm{AND}_f) &\iff \\
&\quad \pi_{\mathrm{FT}}(S, {}_{k_0^{r_i}}\mathrm{OR}_f) \wedge \cdots \wedge \pi_{\mathrm{FT}}(S, {}_{k_m^{r_i}}\mathrm{OR}_f) \\
\pi_{\mathrm{FT}}(S, {}_{k_0^{r_0}}\mathrm{OR}_f)) &\iff \mathrm{BE}_{k_0^{r_0}} \in S \vee \pi_{\mathrm{FT}}(S, {}_{h_0^{r_0}}\mathrm{OR}_{k_0^{r_0}}) \\
&\quad \cdots \\
\pi_{\mathrm{FT}}(S, {}_{k_m^{r_i}}\mathrm{OR}_f)) &\iff \mathrm{BE}_{k_m^{r_i}} \in S \vee \pi_{\mathrm{FT}}(S, {}_{h_m^{r_i}}\mathrm{OR}_{k_m^{r_i}}) \\
\pi_{\mathrm{FT}}(S, {}_{h_0^{r_0}}\mathrm{OR}_k) &\iff \mathrm{BE}_{h_0^{r_0}} \in S \vee \\
&\quad \pi_{\mathrm{FT}}(S, {}_{g_0^{h_0^{r_0}}}\mathrm{OR}_{h_0^{r_0}}) \vee \cdots \vee \pi_{\mathrm{FT}}(S, {}_{g_n^{h_m^{r_i}}}\mathrm{OR}_{h_m^{r_i}}) \\
&\quad \cdots \\
\pi_{\mathrm{FT}}(S, {}_{h_m^{r_i}}\mathrm{OR}_k) &\iff \mathrm{BE}_{h_m^{r_i}} \in S \vee \\
&\quad \pi_{\mathrm{FT}}(S, {}_{g_0^{h_0^{r_0}}}\mathrm{OR}_{h_0^{r_0}}) \vee \cdots \vee \pi_{\mathrm{FT}}(S, {}_{g_n^{h_m^{r_i}}}\mathrm{OR}_{h_m^{r_i}}) \\
\pi_{\mathrm{FT}}(S, {}_{g_0^{h_0^{r_0}}}\mathrm{OR}_{h_0^{r_0}}) &\iff \mathrm{BE}_{{}_{g_0^{h_0^{r_0}}}} \in S \\
&\quad \cdots \\
\pi_{\mathrm{FT}}(S, {}_{g_n^{h_m^{r_i}}}\mathrm{OR}_{h_m^{r_i}}) &\iff \mathrm{BE}_{{}_{g_n^{h_m^{r_i}}}} \in S
\end{aligned}
\tag{34}
$$

We split the induction step $K+1$ in three cases. Case 1 is the addition of a new prerequisite resource type through existing connected channels and components ($K+1 = (I+1)+M+N$). Case 2 is the addition of a channel with an existing resource type between already connected components ($K+1 = I+(M+1)+N$). Case 3 is the addition of a new component that provides of an existing resource type to an already connected channel ($K+1 = I+M+(N+1)$).

Case 1 ($I+1$). Let Γ_0' be the extension of Γ_0 by a fresh prerequisite resource r_{i+1}. Let $f \in \mathscr{C}_{\Gamma_0'}$ be a capability, and let $\mathrm{FT}' = \tau(\Gamma_0', f)$ be the synthesized fault tree.

Recall the formulae (33) and (34). By adding an additional resource r_{i+1}, we obtain the following extension for the CIG structure function:

$$
\begin{aligned}
[\![f]\!]_{\Gamma_0}^{\Gamma_n} &\iff \lightning_{\Gamma_0}^{\Gamma_n}\langle \mathrm{id}_f \rangle \vee \\
&\quad \Big(\cdots \vee \Big(\cdots \wedge [\![k_m^{r_i}]\!]_{\Gamma_0}^{\Gamma_n}\Big) \vee \Big(\ldots \wedge [\![\boldsymbol{k_m^{r_{i+1}}}]\!]_{\boldsymbol{\Gamma_0}}^{\boldsymbol{\Gamma_n}}\Big)\Big) \\
&\quad \cdots \\
[\![k_m^{r_i}]\!]_{\Gamma_0}^{\Gamma_n} &\iff \lightning_{\Gamma_0}^{\Gamma_n}\langle \mathrm{id}_{k_m^{r_i}} \rangle \vee \Big(\exists \Gamma_{n-j} : \mathcal{G}.\ [\![h_m^{r_i}]\!]_{\Gamma_0}^{\Gamma_{n-j}}\Big) \\
[\![\boldsymbol{k_m^{r_{i+1}}}]\!]_{\boldsymbol{\Gamma_0}}^{\boldsymbol{\Gamma_n}} &\iff \lightning_{\boldsymbol{\Gamma_0}}^{\boldsymbol{\Gamma_n}}\langle \mathbf{id}_{\boldsymbol{k_m^{r_{i+1}}}} \rangle \\
&\quad \vee \Big(\exists \boldsymbol{\Gamma_{n-j}} : \boldsymbol{\mathcal{G}}.\ [\![\boldsymbol{h_m^{r_{i+1}}}]\!]_{\boldsymbol{\Gamma_0}}^{\boldsymbol{\Gamma_{n-j}}}\Big) \\
&\quad \cdots \\
[\![h_m^{r_i}]\!]_{\Gamma_0}^{\Gamma_{n-j}} &\iff \lightning_{\Gamma_0}^{\Gamma_{n-j}}\langle \mathrm{id}_{h_m^{r_i}} \rangle \vee \cdots \vee [\![g_n^{h_m^{r_i}}]\!]_{\Gamma_0}^{\Gamma_{n-j}} \\
[\![\boldsymbol{h_m^{r_{i+1}}}]\!]_{\boldsymbol{\Gamma_0}}^{\boldsymbol{\Gamma_{n-j}}} &\iff \lightning_{\boldsymbol{\Gamma_0}}^{\boldsymbol{\Gamma_{n-j}}}\langle \mathbf{id}_{\boldsymbol{h_m^{r_{i+1}}}} \rangle \vee \ldots \vee [\![\boldsymbol{g_n^{h_m^{r_{i+1}}}}]\!]_{\boldsymbol{\Gamma_0}}^{\boldsymbol{\Gamma_{n-j}}} \\
&\quad \cdots \\
[\![g_n^{h_m^{r_i}}]\!]_{\Gamma_0}^{\Gamma_{n-j}} &\iff \lightning_{\Gamma_0}^{\Gamma_{n-j}}\langle \mathrm{id}_{g_n^{h_m^{r_i}}} \rangle \\
[\![\boldsymbol{g_n^{h_m^{r_{i+1}}}}]\!]_{\boldsymbol{\Gamma_0}}^{\boldsymbol{\Gamma_{n-j}}} &\iff \lightning_{\boldsymbol{\Gamma_0}}^{\boldsymbol{\Gamma_{n-j}}}\langle \mathbf{id}_{\boldsymbol{g_n^{h_m^{r_{i+1}}}}} \rangle
\end{aligned}
$$

For the FT structure function, we obtain a similar extension:

$$
\begin{aligned}
\pi_{\mathrm{FT}}(S, \mathrm{OR}_f) &\iff \mathrm{BE}_f \in S \vee \\
&\quad \big(\cdots \vee \pi_{\mathrm{FT}}(S, {}_{r_i}\mathrm{AND}_f) \\
&\quad\quad \vee \boldsymbol{\pi_{\mathrm{FT}}(S, {}_{r_{i+1}}\mathbf{AND}_f)}\big) \\
&\quad \cdots \\
\pi_{\mathrm{FT}}(S, {}_{r_i}\mathrm{AND}_f) &\iff \pi_{\mathrm{FT}}(S, {}_{k_0^{r_i}}\mathrm{OR}_f) \wedge \ldots \\
&\quad \wedge \pi_{\mathrm{FT}}(S, {}_{k_m^{r_i}}\mathrm{OR}_f) \\
\boldsymbol{\pi_{\mathrm{FT}}(S, {}_{r_{i+1}}\mathbf{AND}_f)} &\iff \boldsymbol{\pi_{\mathrm{FT}}(S, {}_{k_0^{r_{i+1}}}\mathbf{OR}_f)} \wedge \ldots \\
&\quad \wedge \boldsymbol{\pi_{\mathrm{FT}}(S, {}_{k_m^{r_{i+1}}}\mathbf{OR}_f)} \\
&\quad \cdots \\
\pi_{\mathrm{FT}}(S, {}_{k_m^{r_i}}\mathrm{OR}_f)) &\iff \mathrm{BE}_{k_m^{r_i}} \in S \vee \pi_{\mathrm{FT}}(S, {}_{h_m^{r_i}}\mathrm{OR}_{k_m^{r_i}}) \\
\boldsymbol{\pi_{\mathrm{FT}}(S, {}_{k_m^{r_{i+1}}}\mathbf{OR}_f))} &\iff \boldsymbol{\mathbf{BE}_{k_m^{r_{i+1}}} \in S} \\
&\quad \vee \boldsymbol{\pi_{\mathrm{FT}}(S, {}_{h_m^{r_{i+1}}}\mathbf{OR}_{k_m^{r_{i+1}}})} \\
&\quad \cdots \\
\pi_{\mathrm{FT}}(S, {}_{h_m^{r_i}}\mathrm{OR}_k) &\iff \mathrm{BE}_{h_m^{r_i}} \in S \vee \ldots \\
&\quad \vee \pi_{\mathrm{FT}}(S, {}_{g_n^{h_m^{r_i}}}\mathrm{OR}_{h_m^{r_i}}) \\
\boldsymbol{\pi_{\mathrm{FT}}(S, {}_{h_m^{r_{i+1}}}\mathbf{OR}_k)} &\iff \boldsymbol{\mathbf{BE}_{h_m^{r_{i+1}}} \in S} \vee \ldots \\
&\quad \vee \boldsymbol{\pi_{\mathrm{FT}}(S, {}_{g_n^{h_m^{r_{i+1}}}}\mathbf{OR}_{h_m^{r_{i+1}}})} \\
&\quad \cdots \\
\pi_{\mathrm{FT}}(S, {}_{g_n^{h_m^{r_i}}}\mathrm{OR}_{h_m^{r_i}}) &\iff \mathrm{BE}_{{}_{g_n^{h_m^{r_i}}}} \in S \\
\boldsymbol{\pi_{\mathrm{FT}}(S, {}_{g_n^{h_m^{r_{i+1}}}}\mathbf{OR}_{h_m^{r_{i+1}}})} &\iff \boldsymbol{\mathbf{BE}_{{}_{g_n^{h_m^{r_{i+1}}}}} \in S}
\end{aligned}
$$

Apply bottom-up substitution to observe that both resulting boolean formulae are extended with clauses that

are disjunctive w.r.t. (33) and (34) and between whose terms there is a one-to-one correspondence. I.e.:

$$[\![k_0^{r_{i+1}}]\!]_{\Gamma_0}^{\Gamma_n} \wedge \cdots \wedge [\![k_m^{r_{i+1}}]\!]_{\Gamma_0}^{\Gamma_n} \iff \pi_{\mathrm{FT}}(S, {}_{k_0^{r_{i+1}}}\mathrm{OR}_f) \wedge \cdots \wedge \pi_{\mathrm{FT}}(S, {}_{k_m^{r_{i+1}}}\mathrm{OR}_f)$$

And by the IH:

$$[\![f]\!]_{\Gamma_0'}^{\Gamma_n} \iff \pi_{\mathrm{FT}'}(S, f)$$

□

Case 2 ($M+1$). Let Γ_0' be the extension of Γ_0 by a fresh channel c_{m+1}. Let $f \in \mathscr{C}_{\Gamma_0'}$ be a capability, and let $\mathrm{FT}' = \tau(\Gamma_0', f)$ be the synthesized fault tree.

Recall the formulae (33) and (34). By adding an additional channel c_{m+1}, we obtain the following extension for the CIG structure function:

$$\begin{aligned}
[\![f]\!]_{\Gamma_0}^{\Gamma_n} &\iff \lightning_{\Gamma_0}^{\Gamma_n}\langle \mathrm{id}_f\rangle \vee \Big(\cdots \vee \Big(\cdots \wedge [\![k_m^{r_i}]\!]_{\Gamma_0}^{\Gamma_n} \wedge \boldsymbol{[\![k_{m+1}^{r_i}]\!]_{\Gamma_0}^{\Gamma_n}}\Big)\Big)\\
&\cdots\\
[\![k_m^{r_i}]\!]_{\Gamma_0}^{\Gamma_n} &\iff \lightning_{\Gamma_0}^{\Gamma_n}\langle \mathrm{id}_{k_m^{r_i}}\rangle \vee \Big(\exists \Gamma_{n-j} : \mathcal{G}.\ [\![h_m^{r_i}]\!]_{\Gamma_0}^{\Gamma_{n-j}}\Big)\\
\boldsymbol{[\![k_{m+1}^{r_i}]\!]_{\Gamma_0}^{\Gamma_n}} &\boldsymbol{\iff \lightning_{\Gamma_0}^{\Gamma_n}\langle \mathrm{id}_{k_{m+1}^{r_i}}\rangle \vee \Big(\exists \Gamma_{n-j} : \mathcal{G}.\ [\![h_{m+1}^{r_i}]\!]_{\Gamma_0}^{\Gamma_{n-j}}\Big)}\\
&\cdots\\
[\![h_m^{r_i}]\!]_{\Gamma_0}^{\Gamma_{n-j}} &\iff \lightning_{\Gamma_0}^{\Gamma_{n-j}}\langle \mathrm{id}_{h_m^{r_i}}\rangle \vee \cdots \vee [\![g_n^{h_m^{r_i}}]\!]_{\Gamma_0}^{\Gamma_{n-j}}\\
\boldsymbol{[\![h_{m+1}^{r_i}]\!]_{\Gamma_0}^{\Gamma_{n-j}}} &\boldsymbol{\iff \lightning_{\Gamma_0}^{\Gamma_{n-j}}\langle \mathrm{id}_{h_{m+1}^{r_i}}\rangle \vee \ldots \vee [\![g_n^{h_{m+1}^{r_i}}]\!]_{\Gamma_0}^{\Gamma_{n-j}}}\\
&\cdots\\
[\![g_n^{h_m^{r_i}}]\!]_{\Gamma_0}^{\Gamma_{n-j}} &\iff \lightning_{\Gamma_0}^{\Gamma_{n-j}}\langle \mathrm{id}_{g_n^{h_m^{r_i}}}\rangle\\
\boldsymbol{[\![g_n^{h_{m+1}^{r_i}}]\!]_{\Gamma_0}^{\Gamma_{n-j}}} &\boldsymbol{\iff \lightning_{\Gamma_0}^{\Gamma_{n-j}}\langle \mathrm{id}_{g_n^{h_{m+1}^{r_i}}}\rangle}
\end{aligned}$$

For the FT structure function, we obtain a similar result:

$$\begin{aligned}
\pi_{\mathrm{FT}}(S, \mathrm{OR}_f) &\iff \mathrm{BE}_f \in S \vee \big(\cdots \vee \pi_{\mathrm{FT}}(S, {}_{r_i}\mathrm{AND}_f)\big)\\
&\cdots\\
\pi_{\mathrm{FT}}(S, {}_{r_i}\mathrm{AND}_f) &\iff \cdots \wedge \pi_{\mathrm{FT}}(S, {}_{k_m^{r_i}}\mathrm{OR}_f) \wedge \boldsymbol{\pi_{\mathrm{FT}}(S, {}_{k_{m+1}^{r_i}}\mathrm{OR}_f)}\\
&\cdots\\
\pi_{\mathrm{FT}}(S, {}_{k_m^{r_i}}\mathrm{OR}_f)) &\iff \mathrm{BE}_{k_m^{r_i}} \in S \vee \pi_{\mathrm{FT}}(S, {}_{h_m^{r_i}}\mathrm{OR}_{k_m^{r_i}})\\
\boldsymbol{\pi_{\mathrm{FT}}(S, {}_{k_{m+1}^{r_i}}\mathrm{OR}_f))} &\boldsymbol{\iff \mathrm{BE}_{k_{m+1}^{r_i}} \in S \vee \pi_{\mathrm{FT}}(S, {}_{h_{m+1}^{r_i}}\mathrm{OR}_{k_{m+1}^{r_i}})}\\
&\cdots\\
\pi_{\mathrm{FT}}(S, {}_{h_m^{r_i}}\mathrm{OR}_k) &\iff \mathrm{BE}_{h_m^{r_i}} \in S \vee \pi_{\mathrm{FT}}(S, {}_{g_n^{h_m^{r_i}}}\mathrm{OR}_{h_m^{r_i}})\\
\boldsymbol{\pi_{\mathrm{FT}}(S, {}_{h_{m+1}^{r_i}}\mathrm{OR}_k)} &\boldsymbol{\iff \mathrm{BE}_{h_{m+1}^{r_i}} \in S \vee \ldots \vee \pi_{\mathrm{FT}}(S, {}_{g_n^{h_{m+1}^{r_i}}}\mathrm{OR}_{h_{m+1}^{r_i}})}\\
&\cdots\\
\pi_{\mathrm{FT}}(S, {}_{g_n^{h_m^{r_i}}}\mathrm{OR}_{h_m^{r_i}}) &\iff \mathrm{BE}_{g_n^{h_m^{r_i}}} \in S\\
\boldsymbol{\pi_{\mathrm{FT}}(S, {}_{g_n^{h_{m+1}^{r_i}}}\mathrm{OR}_{h_{m+1}^{r_i}})} &\boldsymbol{\iff \mathrm{BE}_{g_n^{h_{m+1}^{r_i}}} \in S}
\end{aligned}$$

Apply bottom-up substitution to observe that both resulting boolean formulae are extended with clauses that are conjunctive w.r.t. (33) and (34) and between whose terms there is a one-to-one correspondence. I.e.:

$$[\![k_{m+1}^{r_i}]\!]_{\Gamma_0}^{\Gamma_n} \iff \pi_{\mathrm{FT}}(S, {}_{k_{m+1}^{r_i}}\mathrm{OR}_f)$$

And by the IH:

$$[\![f]\!]_{\Gamma_0'}^{\Gamma_n} \iff \pi_{\mathrm{FT}'}(S, f)$$

□

Case 3 ($N+1$). Let Γ_0' be the extension of Γ_0 by a fresh component capability g_{n+1}. Let $f \in \mathscr{C}_{\Gamma_0'}$ be a capability, and let $\mathrm{FT}' = \tau(\Gamma_0', f)$ be the synthesized fault tree.

Recall the formulae (33) and (34). By adding an additional component capability g_{n+1}, we obtain the following extension for the CIG structure function:

$$
\begin{aligned}
[\![f]\!]^{\Gamma_n}_{\Gamma_0} &\iff \lightning^{\Gamma_n}_{\Gamma_0}\langle \mathrm{id}_f\rangle \vee \\
&\quad \Big(\big(\cdots \wedge [\![k^{r_0}_m]\!]^{\Gamma_n}_{\Gamma_0}\big) \vee \cdots \vee \big(\cdots \wedge [\![k^{r_i}_m]\!]^{\Gamma_n}_{\Gamma_0}\big)\Big) \\
&\quad \cdots \\
[\![k^{r_i}_m]\!]^{\Gamma_n}_{\Gamma_0} &\iff \lightning^{\Gamma_n}_{\Gamma_0}\langle \mathrm{id}_{k^{r_i}_m}\rangle \vee \Big(\exists \Gamma_{n-j} : \mathcal{G}.\ [\![h^{r_i}_m]\!]^{\Gamma_{n-j}}_{\Gamma_0}\Big) \\
&\quad \cdots \\
[\![h^{r_i}_m]\!]^{\Gamma_{n-j}}_{\Gamma_0} &\iff \lightning^{\Gamma_{n-j}}_{\Gamma_0}\langle \mathrm{id}_{h^{r_i}_m}\rangle \vee \cdots \vee [\![g^{h^{r_i}_m}_n]\!]^{\Gamma_{n-j}}_{\Gamma_0} \\
&\quad \vee [\![\boldsymbol{g^{h^{r_i}_m}_{n+1}}]\!]^{\boldsymbol{\Gamma_{n-j}}}_{\boldsymbol{\Gamma_0}} \\
&\quad \cdots \\
[\![g^{h^{r_i}_m}_n]\!]^{\Gamma_{n-j}}_{\Gamma_0} &\iff \lightning^{\Gamma_{n-j}}_{\Gamma_0}\langle \mathrm{id}_{g^{h^{r_i}_m}_n}\rangle \\
[\![\boldsymbol{g^{h^{r_i}_m}_{n+1}}]\!]^{\boldsymbol{\Gamma_{n-j}}}_{\boldsymbol{\Gamma_0}} &\iff \lightning^{\boldsymbol{\Gamma_{n-j}}}_{\boldsymbol{\Gamma_0}}\langle \mathbf{id}_{\boldsymbol{g^{h^{r_i}_m}_{n+1}}}\rangle
\end{aligned}
$$

For the FT structure function, we obtain a similar extension:

$$
\begin{aligned}
\pi_{\mathrm{FT}}(S, \mathrm{OR}_f) &\iff \mathrm{BE}_f \in S \vee \\
&\quad \big(\cdots \vee \pi_{\mathrm{FT}}(S, {}_{r_i}\mathrm{AND}_f)\big) \\
&\quad \cdots \\
\pi_{\mathrm{FT}}(S, {}_{r_i}\mathrm{AND}_f) &\iff \cdots \wedge \pi_{\mathrm{FT}}(S, {}_{k^{r_i}_m}\mathrm{OR}_f) \\
&\quad \cdots \\
\pi_{\mathrm{FT}}(S, {}_{k^{r_i}_m}\mathrm{OR}_f)) &\iff \mathrm{BE}_{k^{r_i}_m} \in S \vee \pi_{\mathrm{FT}}(S, {}_{h^{r_i}_m}\mathrm{OR}_{k^{r_i}_m}) \\
&\quad \cdots \\
\pi_{\mathrm{FT}}(S, {}_{h^{r_i}_m}\mathrm{OR}_k) &\iff \mathrm{BE}_{h^{r_i}_m} \in S \vee \ldots \\
&\quad \vee \pi_{\mathrm{FT}}(S, {}_{g^{h^{r_i}_m}_n}\mathrm{OR}_{h^{r_i}_m}) \\
&\quad \vee \boldsymbol{\pi_{\mathrm{FT}}(S, {}_{\boldsymbol{g^{h^{r_i}_m}_{n+1}}}\mathrm{OR}_{\boldsymbol{h^{r_i}_m}})} \\
&\quad \cdots \\
\pi_{\mathrm{FT}}(S, {}_{g^{h^{r_i}_m}_n}\mathrm{OR}_{h^{r_i}_m}) &\iff \mathrm{BE}_{g^{h^{r_i}_m}_n} \in S \\
\boldsymbol{\pi_{\mathrm{FT}}(S, {}_{\boldsymbol{g^{h^{r_i}_m}_{n+1}}}\mathbf{OR}_{\boldsymbol{h^{r_i}_m}})} &\boldsymbol{\iff \mathbf{BE}_{g^{h^{r_i}_m}_{n+1}} \in S}
\end{aligned}
$$

Apply bottom-up substitution to observe that both resulting boolean formulae are extended with terms that are disjunctive w.r.t. (33) and (34) and between who there is a one-to-one correspondence. I.e.:

$$
[\![g^{h^{r_i}_m}_{n+1}]\!]^{\Gamma_{n-j}}_{\Gamma_0} \iff \pi_{\mathrm{FT}}(S, {}_{g^{h^{r_i}_m}_{n+1}}\mathrm{OR}_{h^{r_i}_m})
$$

And by the IH:

$$
[\![f]\!]^{\Gamma_n}_{\Gamma'_0} \iff \pi_{\mathrm{FT}'}(S, f)
$$

□

In all three cases, the induction hypothesis holds. We have thus shown that the transformation τ is sound and complete w.r.t. CIG and FT error semantics.

Remark. It should be noted that if K increases with any elements that do not affect the terms of the structure function $[\![f]\!]^{\Gamma_n}_{\Gamma_0}$, then the IH holds trivially. E.g. adding downstream components to the system does not affect f's error function.

XIII. Related work

A. Failure Mode Effects Analysis

Failure Mode Effects Analysis (FMEA) is an inductive reliability analysis method in which experts and analysts consider the potential effects of envisioned failure modes of a system. This knowledge is captured in FMEA tables – a human-readable, textual format, listing a subset of a system's potential failure modes. Due to their natural-language descriptions, FMEA tables cannot be evaluated with classical algorithms, though some progress in mechanized natural-language interpretation has been made [38].

Since manual FMEA is a labor-intensive process, it is usually carried out in the later design phases. By that point, changes to the system design are much more difficult to accommodate and consequently much more costly [16]. As a result, the emphasis often lies on the mitigation of known design flaws, rather than improving the system design.

FTA, in contrast to FMEA is a deductive approach. FT construction starts from the undesired top-level event, and potential causes are explored in a recursive top-down manner. FTs provides an intuitive visualization of how failure modes and their interactions result in system-level failure, making them easier to interpret than FMEA tables – especially for large systems. Furthermore, fault trees are formally defined and they afford many metrics that can be computed algorithmically (i.e. minimal cut sets and minimal path sets [30]). The strength of FTA lies in the combination of visual failure mode interactions and automatic reliability metrics.

If FTA reliability metrics can be obtained with low effort in the early design phases, design changes based on new insights are less costly to implement. Engineers and decision-makers have the ability to make systems more robust by design, and simultaneously to reduce costly "just in case" design decisions because of uncertainty in this early stage. However, even though analysis is fully mechanized, the manual *construction* of fault trees typically suffers from the same labor-intensiveness and consistency challenges associated with manual FMEA construction.

B. Fault Tree and FTA Extensions

Dynamic Fault Trees (DFTs) extend classical FTs by introducing temporal and state-dependent failure semantics [30]. In addition to standard Boolean gates, DFTs incorporate constructs such as priority-AND gates, functional dependencies[7], and spare management, enabling the representation of dynamic system behavior and sequencing constraints. DFTs have been widely applied in the analysis of safety-critical systems. Several approaches have investigated the automatic generation of DFTs from architectural or behavioral models. However, the detailed level of knowledge required to create a DFT is unavailable at early design stages, which makes the DFT an inappropriate formalism for the focus of this work.

[7]The semantics of the term "functional dependency" in DFTs is not the same as how we define it in this work.

A more domain-specific extension of fault trees, called Component Fault Trees [18](CFTs), makes the role of components as the bearers of faults explicit by allowing modelers to encapsulate internal component failure modes. However, the lack of an ontological foundation allows for the construction of CFTs that are unintended by the modeler, e.g. those that have no consistent notion of (component) identity, time or causality. Furthermore, their manual construction confronts stakeholders with the same challenges as in the case of standard FTs.

Nicoletti et al. introduce Probabilistic Fault tree Logic (PFL) – a query language for fault trees, allowing analysts to obtain user-specific and complex metrics of interest [24]. PFL also allows for locally setting evidence in queries without changing the values of the underlying FT. In the context of this work, analysts can use PFL to obtain quantitative FT metrics despite the fact that the FTs synthesized from CIGs do no "contain" numerical failure probabilities.

C. Fault Tree Generation from MBSE

There is extensive work on (semi-)automatically transforming failure-decorated MBSE models to (dynamic) FTs.

Baklouti et al. generate dynamic FTs from decorated SysML models by introducing a novel redundancy profile [3]. However, this requires explicit safety assessment from the modeler (i.e. redundancy does not *follow* from the architecture, it is *assumed* of the architecture) and its tooling is proprietary to SysML (version 1) extended with the authors' profile. Furthermore, despite its practical applicability, the work lacks an ontological foundation.

Castet et al. introduce a fault management ontology, providing engineers at JPL with a common vocabulary of problematic system behavior [7]. A tool that integrates this ontology with SysML tooling is presented in [8]. The fault management ontology describes concepts like constraints, violations, mitigation and explanations. These concepts are complementary to the ontological concepts (e.g. components, capability, faults and errors) that we introduce in this work.

Mhenni et al. [21] describe a method to synthesize FTs from SysML's Internal Block Diagrams (IBD). IBDs represent a system's internal structure and the interactions between its components through ports. The FT synthesis algorithm is done by a combination of standard graph traversal algorithms and generating sub-FTs for known IBD patterns. This is similar in spirit to the transformation we propose in this work. However, the existing approach relies on specific SysML semantics. In contrast, the conceptual model proposed in our work is generally applicable to any KG that contain structural and functional system knowledge, regardless of the modeling language in which such system knowledge is originally expressed. This allows us to integrate different models (in different modeling languages) that describe the same system.

Alternative System Models. The work by Bozzano et al. in [6] allows for the automatic generation of FTs from SLIM models (an extension of AADL). While these FTs are expressive and accurate, the synthesis methodology requires detailed nominal models and explicit error models of a system.

Majdara and Wakabayashi proposee a method to generate FTs from a system model using so-called function tables for components [19]. Such a table describes the output of a component given its current failure status. Additionally, there are state transition tables that describe the state changes that a component can go through. Lastly, they employ a notion of *flow*, which is similar in spirit to our own notion of *resource types*. The inter-connectivity of components that exchange flows is the basis for the FT synthesis algorithm. Though these works address some of the challenges that we also touch on in this paper, the ontological foundation is minimal. Furthermore, the need for function tables and state transition tables, which encode failure states and their consequences, incurs the same modeling burden that many of the MBSE approaches suffer from.

What all these approaches have in common, is that FT synthesis relies on manually decorating system architecture with failure information. In doing so, they allow for the *recording* of expert knowledge about specific failure modes. In contrast, CIGs do not require (and indeed do not facilitate) the manual modeling of failure behavior. Therefore, the ontology we present must encode sufficiently rich semantics to enable the *inference* of failure modes – and this by the very same *mechanisms* that experts may employ.

D. Embedding Fault Knowledge in KGs

When fault semantics are directly encoded in an ontology, like in that proposed by Meng et al. [20], generating FTs from KGs that instantiate such an ontology is relatively straight-forward. The ontology that Shen et al. present in [33] even explicitly encodes the basic FT concepts. However, both these ontologies describe KGs where explicit fault knowledge is already present. The synthesis of FTs from an ontology about the structural and functional aspects of system architectures, *without* specified fault knowledge, is a key value proposition of our approach.

There are many automated methods to extract knowledge graphs from unstructured data [39]. In this work, we assume that a knowledge graph representing the system architecture is available.

E. Ontology for Risk

The Common Ontology of Value and Risk (COVER) [32] proposes an ontological foundation for reasoning about risk, causality, and events in complex socio-technical systems. COVER emphasizes the distinction between risk and value perception, event types, and situations, enabling a semantically rigorous treatment of social risk and value experience. COVER provides explicit conceptual grounding for what objects

participate in risk and value events, and how such events are causally connected. Our work aligns with this perspective by treating fault creation, fault activation, and error propagation as ontologically distinct event types grounded in UFO. COVER also lends validity to the CIG notion of *capabilities*: *"When dispositions are perceived as beneficial (...) they are usually labeled as capabilities."* [11], [32]. However, whereas COVER primarily focuses on high-level conceptual modeling of events and risk, our approach operationalizes such ideas for the automated synthesis of FTs. Furthermore, the domain-specific notions relevant to CPSs are not captured by COVER (though the possibility for alignment of CIGs with COVER has fruitful potential).

Nicoletti et al. explicitly highlight the importance of objects, e.g. components, in FTA and Attack Tree Analysis explicit. The WATCHDOG [23] approach, based on COVER, proposes object-oriented disruption graphs (DOGs). COVER describes how Threat Objects and Objects at Risk can participate in Threat Events and Loss Events. WATCHDOG draws a parallel between the preconditions of such events and the objects that participate in them. However, these conditions are, at the leafs of a DOG, *enablers* of a basic events. The work does not explicitly consider the mechanisms of non-synchronic situations (i.e. the world at different time points), and as such leaves a notion of causality between events underspecified. Because of the gap between "enabling conditions" and causality, this work is incompatible with the type of functional dependency analysis we focus on here. Furthermore, by allowing *any kind of object* in the model, the approach relies on manual, iterative expert modeling for the correct construction of DOGs (e.g. the correct modeling of causal relationships between events is not guided or enforced by the model). In contrast, since CIGs are domain-specific to CPS architecture, they are sufficiently constrained to enable fully automated synthesis of FTs.

Venceslau et al. [35] later extended the work of Majdara and Wakabayashi [19] with an ontology (in a much leaner sense of the word than what we commit to here) and applied it to an industrial plant case study. Though there are works in the direction of an ontology for risk models [26], a comprehensive ontological analysis of CPSs for reliability analysis is lacking.

Another general observation is that most of the state of the art focuses on vertical failure propagation, i.e. propagation of failure along the mereological composition of a system. This in contrast to error propagation along its horizontal integration, which is what we focus on in this work.

XIV. Conclusion

This paper introduced the Capability Interaction Graph (CIG), a semantically and ontologically grounded model for representing functional dependencies and fault propagation in cyber-physical systems (CPSs). By combining principles from Model-Based Systems Engineering, knowledge graphs, and the Unified Foundational Ontology (UFO), the CIG provides a lightweight yet expressive intermediate representation that bridges CPS architectural knowledge and reliability analysis.

Building upon this model, we presented an automated synthesis pipeline that transforms knowledge-graph representations of CPS architectures into fault trees suitable for classical Fault Tree Analysis. In contrast to existing approaches that rely heavily on manually decorated architectural models and expert-specified failure behavior, our approach infers fault propagation structure directly from functional dependencies encoded in the CIG. The model explicitly distinguishes between faults, fault activations, errors, events, and situations, thereby addressing several semantic ambiguities that are commonly conflated in traditional fault tree methodologies.

The proposed ontology and formalization enable the unified representation of heterogeneous engineering knowledge while remaining sufficiently lightweight for early design stages, where detailed behavioral information is often unavailable. Furthermore, grounding the framework in UFO provides rigorous semantics for causality, object participation, event manifestation, and dependency, improving consistency and traceability across synthesized reliability models.

Overall, this work demonstrates that meaningful and semantically rich fault trees can be synthesized automatically from minimally structured CPS architectural knowledge. This opens opportunities for earlier and more scalable reliability analysis, reducing the effort associated with manual fault modeling and supporting more informed architectural decision-making during system development. This article build on our earlier work presented in [25].

Future work includes extending the framework with quantitative reliability information, integrating behavioral and temporal semantics, incorporating uncertainty and probabilistic reasoning, and validating the approach on large-scale industrial CPS case studies. In addition, future research may explore tighter integration with MBSE toolchains and automated extraction of CIGs from heterogeneous engineering artifacts and natural-language system documentation.

A. Acknowledgments

The authors would like to thank dr. Claudenir Morais Fonseca for the insightful discussions and valuable advice on OntoUML and the Unified Foundational Ontology (UFO), which greatly contributed to the development of this work.

This publication is part of the project ZORRO with project number KICH1.ST02.21.003 of the research programme Key Enabling Technologies (KIC) which is (partly) financed by the Dutch Research Council (NWO).

References

[1] João Paulo A. Almeida, Giancarlo Guizzardi, Tiago Prince Sales, and Claudenir M. Fonseca. gufo: A gentle foundational ontology for semantic web knowledge graphs, 2026.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, 2004.

[3] Anis Baklouti, Nga Nguyen, Faïda Mhenni, Jean-Yves Choley, and Abdelfattah Mlika. Dynamic Fault Tree Generation for Safety-Critical Systems Within a Systems Engineering Approach. IEEE Systems Journal, 14(1):1512–1522, March 2020.

[4] Tim Berners-Lee, Roy T. Fielding, and Larry M Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, January 2005.

[5] A Botti Benevides, JPA Almeida, and Giancarlo Guizzardi. Towards a unified theory of endurants and perdurants: UFO-AB. In Proceedings of the Joint Ontology Workshops 2019. Episode V: The Styrian Autumn of Ontology, volume 2518 of CEUR Workshop Proceedings. CEUR-WS, 2019.

[6] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. Safety, dependability and performance analysis of extended AADL models. Comput. J., 54(5):754–775, 2011.

[7] Jean-Francois Castet, Magdy Bareh, Jeffery Nunes, Steven Jenkins, and Gene Lee. Fault Management Ontology and Modeling Patterns. American Institute of Aeronautics, 2016.

[8] Jean-Francois Castet, Magdy Bareh, Jeffery Nunes, Shira Okon, Larry Garner, Emmy Chacko, and Michel Izygon. Failure analysis and products in a model-based environment. In 2018 IEEE Aerospace Conference, pages 1–13. IEEE, 2018.

[9] Aaron J Cotnoir and Achille C Varzi. Mereology. Oxford University Press, 2021.

[10] Richard P. Feynman, Robert B. Leighton, Matthew Sands, and E. M. Hafner. The Feynman Lectures on Physics; Vol. I. American Journal of Physics, 33(9):750–752, 09 1965.

[11] Business Model Generation. A handbook for visionaries. Game Changers, and Challengers, 2010.

[12] Giancarlo Guizzardi. Ontological foundations for structural conceptual models. PhD thesis, University of Twente, 2005.

[13] Giancarlo Guizzardi, Alessander Botti Benevides, Claudenir M. Fonseca, Daniele Porello, João Paulo A. Almeida, and Tiago Prince Sales. Ufo: Unified foundational ontology. Applied Ontology, 17(1):167–210, 2022.

[14] Giancarlo Guizzardi, Gerd Wagner, Ricardo de Almeida Falbo, Renata SS Guizzardi, and João Paulo A Almeida. Towards ontological foundations for the conceptual modeling of events. In International Conference on Conceptual Modeling, pages 327–341. Springer, 2013.

[15] Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language, 2013. [Accessed 05-03-2025].

[16] INCOSE. INCOSE systems engineering handbook. John Wiley & Sons, 2023.

[17] Harry Jones. Common cause failures and ultra reliability. In 42nd International Conference on Environmental Systems, page 3602, 2012.

[18] Bernhard Kaiser, Peter Liggesmeyer, and Oliver Mäckel. A new component concept for fault trees. In Peter A. Lindsay and Anthony Cant, editors, Safety Critical Systems and Software 2003, Eigth Australian Workshop on Safety-Related Programmable Systems, (SCS2003), Canberra, ACT, Australia, 9-10 October 2003, volume 33 of CRPIT, pages 37–46. Australian Computer Society, 2003.

[19] Aref Majdara and Toshio Wakabayashi. A new approach for computer-aided fault tree generation. In 2009 3rd Annual IEEE Systems Conference, pages 308–312. IEEE, 2009.

[20] Xiangzhen Meng, Bo Jing, Shenglong Wang, Jinxin Pan, Yifeng Huang, and Xiaoxuan Jiao. Fault knowledge graph construction and platform development for aircraft PHM. Sensors, 2024.

[21] Faïda Mhenni, Nga Nguyen, and Jean-Yves Choley. Automatic fault tree generation from SysML system models. In AIM2014, 2014.

[22] P. P. Negri, V. E. S. Souza, A. L. D. C. Leal, R. D. A. Falbo, and G. Guizzardi. Towards an ontology of goal-oriented requirements. In Proceedings of the 20th Workshop on Requirements Engineering, 2017.

[23] Stefano M. Nicoletti, E. Moritz Hahn, Mattia Fumagalli, Giancarlo Guizzardi, and Mariëlle Stoelinga. Watchdog: an ontology-aware risk assessment approach via object-oriented disruption graphs. In John Krogstie, Stefanie Rinderle-Ma, Gerti Kappel, and Henderik A. Proper, editors, Advanced Information Systems Engineering, pages 314–331, Cham, 2025. Springer Nature Switzerland.

[24] Stefano M. Nicoletti, Milan Lopuhaä-Zwakenberg, E. Moritz Hahn, and Mariëlle Stoelinga. Pfl: A probabilistic logic for fault trees. In Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker, editors, Formal Methods, pages 199–221, Cham, 2023. Springer International Publishing.

[25] Manzi Aimé Ntagengerwa, Georgiana Caltais, and Mariëlle Stoelinga. Fault tree synthesis from knowledge graphs. In 2025 IEEE Annual Reliability and Maintainability Symposium - Europe (RAMS-Europe), pages 1–7, 2025.

[26] Ítalo Oliveira, Stefano M Nicoletti, Mattia Fumagalli, Gal Engelberg, and Giancarlo Guizzardi. Toward an ontology-based modeling for risk management. In 18th International Workshop on Value Modelling and Business Ontologies, VMBO 2025, 2025.

[27] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In Isabel Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Mike Uschold, and Lora M. Aroyo, editors, The Semantic Web - ISWC 2006, pages 30–43, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[28] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. ACM Transactions on Database Systems (TODS), 34(3):1–45, 2009.

[29] RDF Working Group. Resource Description Framework (RDF), February 2014.

[30] Enno J.J. Ruijters and Mariëlle I.A. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. Computer science review, 2015.

[31] Jaime Salas and Aidan Hogan. Semantics and canonicalisation of sparql 1.1. Semantic Web Journal, 13(5):829–893, 2022.

[32] Tiago Prince Sales, Fernanda Baião, Giancarlo Guizzardi, João Paulo A. Almeida, Nicola Guarino, and John Mylopoulos. The common ontology of value and risk. In Juan C. Trujillo, Karen C. Davis, Xiaoyong Du, Zhanhuai Li, Tok Wang Ling, Guoliang Li, and Mong Li Lee, editors, Conceptual Modeling, pages 121–135, Cham, 2018. Springer International Publishing.

[33] Li Shen, Hao Tang, Lixiang Wang, Junliang Cai, and Xing Cui. A fault knowledge graph creation method and application based on fault tree analysis and failure mode, effects and criticality analysis. In 2023 IEEE 3rd ICIBA, 2023.

[34] Michael Stamatelatos, William Vesely, Joanne Dugan, Joseph Fragola, Joseph Minarick, and Jan Railsback. Fault tree handbook with aerospace applications, 2002.

[35] Allan Venceslau, Raphaela Lima, Luiz Affonso Guedes, and Ivanovitch Silva. Ontology for computer-aided fault tree synthesis. In Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), pages 1–4, 2014.

[36] William E Vesely, Francine F Goldberg, Norman H Roberts, and David F Haasl. Fault tree handbook. Technical report, Nuclear Regulatory Commission, Washington, DC (USA). Office of Nuclear Regulatory Research, 1981.

[37] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. Commun. ACM, 2014.

[38] Zhaoguang Xu, Yanzhong Dang, Peter Munro, and Yuhang Wang. A data-driven approach for constructing the component-failure mode matrix for fmea. Journal of Intelligent Manufacturing, 31(1):249–265, 2020.

[39] Lingfeng Zhong, Jia Wu, Qian Li, Hao Peng, and Xindong Wu. A comprehensive survey on automatic knowledge graph construction. ACM Comput. Surv., 56(4), November 2023.

Algorithm 1 The CIG to FT transformation function τ

$\mathcal{L} : \text{Gate} \cup \{\varepsilon\}$
$P \subseteq \mathcal{P}(\mathcal{L})$ ▷ Encountered FT gates (recurs. guard)

function BEs($\langle \text{BE}, G, T, I \rangle$: FaultTree)
  return BE
end function

function Gates($\langle \text{BE}, G, T, I \rangle$: FaultTree)
  return G
end function

function I($\langle \text{BE}, G, T, I \rangle$: FaultTree)
  return I
end function

function $\tau(\Gamma : \mathcal{G}, f : \mathbb{F})$
  FT $\leftarrow$ new FaultTree
  $\tau'(\Gamma, \emptyset, f, \varepsilon, \text{FT})$
  return FT
end function

function $\tau'(\Gamma : \mathcal{G}, P, f : \mathscr{C}, \gamma : \mathcal{L}, \text{FT} : \text{FaultTree})$
  if $\text{or}_f \in P$ **then**
    Generate basic event be_f and place it under γ such that

$$\text{be}_f \in \text{BEs(FT)};\ \text{be}_f \in I(\gamma)$$

    return
  end if
  Generate OR gate or_f such that

$$\text{or}_f \in \text{Gates(FT)}$$

  $P' \leftarrow P \cup \{\text{or}_f\}$
  if $\gamma = \varepsilon$ **then**
    or_f becomes the top-level event, such that

$$\text{Gates(FT)} = \{\text{or}_f\}$$

  else
    Place or_f under γ such that

$$\text{or}_f \in I(\gamma)$$

  end if
  Generate basic event be_f and place it under or_f such that

$$\text{be}_f \in \text{BEs(FT)};\ \text{be}_f \in I(\text{or}_f)$$

  for all $r \in \psi_\Gamma(f)$ **do** ▷ Prerequisite resources for f
    Generate AND gate ${}_r\text{and}_f$ and place it under or_f such that

$${}_r\text{and}_f \in \text{Gates(FT)};\ {}_r\text{and}_f \in I(\text{or}_f)$$

    for all $k \in \mathscr{D}^A(f, r)$ **do** ▷ Channel output capabilities that provide f with r
      Generate OR gate ${}_k\text{or}_f$ and place it under ${}_r\text{and}_f$ such that

$${}_k\text{or}_f \in \text{Gates(FT)};\ {}_k\text{or}_f \in I({}_r\text{and}_f)$$

      Generate basic event be_k and place it under ${}_k\text{or}_f$ such that

$$\text{be}_k \in \text{BEs(FT)};\ \text{be}_k \in I({}_k\text{or}_f)$$

      Obtain the channel's input capability $h = \mathscr{D}^B_\Gamma(f, r)$ ▷ Channel input capability that provides k with r
      Generate OR gate ${}_h\text{or}_k$ and place it under ${}_k\text{or}_f$ such that

$${}_h\text{or}_k \in \text{Gates(FT)};\ {}_h\text{or}_k \in I({}_k\text{or}_f)$$

      Generate basic event be_h and place it under ${}_h\text{or}_k$ such that

$$\text{be}_h \in \text{BEs(FT)};\ \text{be}_h \in I({}_h\text{or}_k)$$

      for all $g \in \mathscr{D}^C(h, r)$ **do** ▷ Component capabilities that provide h with r
        Generate OR gate ${}_g\text{or}_h$ and place it under ${}_h\text{or}_k$ such that

$${}_g\text{or}_h \in \text{Gates(FT)};\ {}_g\text{or}_h \in I({}_h\text{or}_k)$$

        Generate basic event be_g and place it under ${}_g\text{or}_h$ such that

$$\text{be}_g \in \text{BEs(FT)};\ \text{be}_g \in I({}_g\text{or}_h)$$

        $\tau'(\Gamma, P', g, {}_g\text{or}_h, \text{FT})$
      end for
    end for
  end for
end function