System Engineering Concepts Modeled via SuperHyperGraphs: Entity–Relationship, Service Dependency, Change Impact, and Workflow Structures

Takaaki Fujita [1] *

[1] Independent Researcher, Shinjuku, Shinjuku-ku, Tokyo, Japan. Takaaki.fujita060@gmail.com

Abstract

Graph theory provides a foundational framework for modeling pairwise relationships through vertices and edges [1, 2]. Hypergraphs extend this paradigm by allowing edges to connect any subset of vertices, capturing higher-order interactions [3], and SuperHyperGraphs introduce hierarchical structure via iterated powerset constructions [4, 5]. In systems engineering, specialized graph models—Entity–Relationship Graphs for database schema design, Service Dependency Graphs for incident impact analysis, Change Impact Graphs for assessing configuration-change risks, and ITIL Workflow Graphs for process automation—are widely used. In this paper, we show how each of these models can be formulated and enriched as hypergraphs and n-layer SuperHyperGraphs, thereby enabling multi-way relationships and multi-scale hierarchical analysis.

Keywords: Superhypergraph, Hypergraph, Call Graphs, State Transition Graphs, Resource Allocation Graphs

1 Introduction

1.1 From Graphs to SuperHyperGraphs

Standard graphs capture only pairwise relationships by connecting two vertices with an edge [2, 6]. However, many real-world systems—such as group communications or multi-party collaborations—require modeling interactions among more than two entities at once. Hypergraphs address this need by allowing each *hyperedge* to join an arbitrary nonempty set of vertices in a single relation [7–9]. To represent nested or multi-scale structures (for example, communities within communities), *SuperHyperGraphs* go further: they iteratively apply the powerset operator to the vertex set, creating successive layers of vertices and hyperedges and yielding a hierarchical, multi-layer network [10–14].

1.2 Graph Models in Systems Engineering

In systems engineering [15–17], four specialized graph models are particularly important:

- **Entity–Relationship Graphs:** Represent database schemas with entities as nodes and relationships as edges, supporting normalization and query planning [18–20]. A related concept is the Entity–Relationship diagram [21–24].
- **Service Dependency Graphs:** Depict how IT services depend on one another, supporting impact analysis during incidents and problem management [25–27].
- **Change Impact Graphs:** Show which configuration items and services are affected by updates or releases, guiding risk assessment and approval workflows [28].
- **Workflow Graphs:** Model ITIL process steps and decision points as nodes and edges, promoting consistency and automation in operations [29–33].

1.3 Overview of Contributions

In this paper, we demonstrate how each of these models can be systematically formulated and extended into HyperGraphs and n-SuperHyperGraphs. This hierarchical and multi-relational framework facilitates the representation of complex interactions and dependencies, enabling multi-way analysis and scalable abstraction. These extensions potentially enhance the analytical capabilities for handling intricate concepts in systems engineering, such as layered dependencies, cross-functional workflows, and dynamic service interactions.

2 Preliminaries

We first establish notation and definitions used throughout. Unless noted otherwise, all graphs are finite, simple, and undirected. For detailed discussions of specific constructs, the reader is referred to standard references.

2.1 SuperHyperGraph

A *hypergraph* extends a traditional graph by permitting each *hyperedge* to connect any nonempty subset of vertices at once [3, 9, 34, 35]. To introduce hierarchy, a *SuperHyperGraph* applies the powerset operation repeatedly to the base vertex set, creating successive layers of vertices and hyperedges [5, 12–14, 36–38].

Definition 2.1 (Base Set)**.** A *base set* S is the underlying domain from which all constructions arise:

$$S = \{\, x \mid x \text{ belongs to the specified universe} \}.$$

Every element of $\mathcal{P}(S)$ or $\mathcal{P}_n(S)$ is drawn from S.

Definition 2.2 (Powerset)**.** The *powerset* of S, denoted $\mathcal{P}(S)$, is the collection of all subsets of S, including the empty set:

$$\mathcal{P}(S) = \{\, A \mid A \subseteq S \}.$$

Definition 2.3 (Hypergraph)**.** [3, 7] A *hypergraph* $H = (V(H), E(H))$ consists of

- a finite vertex set $V(H)$,
- a finite family $E(H)$ of nonempty subsets of $V(H)$, called *hyperedges*.

Example 2.4 (Co-authorship Hypergraph)**.** Let $V_0 = \{A, B, C, D, E\}$ be a set of researchers. Define the hyperedges to be the author lists of three papers:

$$E(H) = \big\{\{A, B, C\},\ \{B, D\},\ \{C, D, E\}\big\}.$$

Then the hypergraph

$$H = \big(V_0,\ E(H)\big)$$

models the fact that paper 1 was written by A, B, C, paper 2 by B, D, and paper 3 by C, D, E, capturing multi-way collaborations in a single relation.

Definition 2.5 (n-th Powerset)**.** [39–41] Define

$$\mathcal{P}^0(X) = X, \quad \mathcal{P}^{k+1}(X) = \mathcal{P}\big(\mathcal{P}^k(X)\big), \quad k \geq 0.$$

Then the n-th powerset is $P_n(X) = \mathcal{P}^n(X)$, and its nonempty version $P_n^*(X)$ excludes the empty set.

Definition 2.6 (n-SuperHyperGraph)**.** [42, 43] Let V_0 be a finite base set and define $\mathcal{P}^k(V_0)$ iteratively as above. An *n-SuperHyperGraph* is a pair

$$\mathrm{SuHyG}^{(n)} = (V, E), \quad V, E \subseteq \mathcal{P}^n(V_0),$$

where elements of V are called *n-supervertices* and elements of E *n-superedges*.

Example 2.7 (2-SuperHyperGraph of Publications)**.** Starting from the same base set $V_0 = \{A, B, C, D, E\}$, form the 1st powerset $\mathcal{P}^1(V_0)$, whose elements include all nonempty co-author sets as above. Now let

$$V = \big\{\{A, B, C\},\ \{B, D\},\ \{C, D, E\}\big\} \subseteq \mathcal{P}^1(V_0),$$

and define two conference sessions as superedges in $\mathcal{P}^2(V_0)$:

$$E = \big\{\, \{\{A, B, C\}, \{C, D, E\}\},\ \{\{B, D\}, \{C, D, E\}\}\big\}.$$

Then the 2-SuperHyperGraph

$$\mathrm{SuHyG}^{(2)} = \big(V,\ E\big)$$

represents a two-level hierarchy: at level 1, individual papers; at level 2, grouping of papers into sessions (e.g. Session 1 includes papers by $\{A, B, C\}$ and $\{C, D, E\}$).

3 Entity–Relationship Graphs and Their Extensions

Entity–Relationship Graphs model database schemas, with entities as nodes and relationships as edges, facilitating normalization and efficient query execution planning [18,44]. We extend this concept using hypergraphs and superhypergraphs.

Definition 3.1 (Entity–Relationship Graph)**.** An *Entity–Relationship Graph* is a quadruple

$$G = (V,\ E,\ \tau_V,\ \tau_E),$$

where

- V is a finite set of *entities*,
- $E \subseteq V \times V$ is a finite set of *relationships*,
- $\tau_V : V \to \mathcal{T}_V$ assigns to each entity $v \in V$ an *entity type* $\tau_V(v)$ from a finite set $\mathcal{T}_V$,
- $\tau_E : E \to \mathcal{T}_E$ assigns to each relationship $e = (u \to v) \in E$ a *relationship type* $\tau_E(e)$ from a finite set $\mathcal{T}_E$.

Example 3.2. Let $\mathcal{T}_V = \{\text{Person}, \text{Project}\}$ and $\mathcal{T}_E = \{\text{worksOn}, \text{manages}\}$. Then an Entity–Relationship Graph

$$G = \big(\{p_1, p_2, \pi_1\},\ \{(p_1 \to \pi_1), (p_2 \to \pi_1), (p_1 \to p_2)\},\ \tau_V,\ \tau_E\big)$$

might satisfy

$$\tau_V(p_1) = \tau_V(p_2) = \text{Person}, \quad \tau_V(\pi_1) = \text{Project},$$
$$\tau_E(p_1 \to \pi_1) = \text{worksOn}, \quad \tau_E(p_2 \to \pi_1) = \text{worksOn}, \quad \tau_E(p_1 \to p_2) = \text{manages}.$$

This models two people p_1, p_2 collaborating on project π_1, with p_1 managing p_2.

Definition 3.3 (Entity–Relationship HyperGraph)**.** An *Entity–Relationship HyperGraph* is a quadruple

$$H = \big(V,\ \mathcal{E},\ \tau_V,\ \tau_E\big),$$

where

- V is a finite set of *entities*,
- $\mathcal{E} \subseteq \mathcal{P}(V) \setminus \{\emptyset\}$ is a finite set of nonempty subsets of V, called *hyperedges*,
- $\tau_V : V \to \mathcal{T}_V$ assigns to each entity $v \in V$ an *entity type* $\tau_V(v)$ from a finite set $\mathcal{T}_V$,
- $\tau_E : \mathcal{E} \to \mathcal{T}_R$ assigns to each hyperedge $e \in \mathcal{E}$ a *relationship type* $\tau_E(e)$ from a finite set $\mathcal{T}_R$.

Example 3.4 (Retail Transaction Entity–Relationship HyperGraph)**.** Let

$$V = \{\, c_1,\ c_2,\ p_1,\ p_2,\ s_1\},$$

where $\mathcal{T}_V = \{\mathsf{Customer}, \mathsf{Product}, \mathsf{Store}\}$ and the entity-type map τ_V is given by

$$\tau_V(c_i) = \mathsf{Customer}, \quad \tau_V(p_j) = \mathsf{Product}, \quad \tau_V(s_1) = \mathsf{Store}.$$

Define two purchase events as hyperedges:

$$e_1 = \{\, c_1,\ p_1,\ s_1\}, \quad e_2 = \{\, c_2,\ p_2,\ s_1\},$$

and let

$$\mathcal{E} = \{\, e_1,\ e_2\}, \qquad \mathcal{T}_R = \{\mathsf{Purchase}\}.$$

Assign the relationship-type map τ_E by

$$\tau_E(e_1) = \tau_E(e_2) = \mathsf{Purchase}.$$

Then

$$H = \big(V,\ \mathcal{E},\ \tau_V,\ \tau_E\big)$$

is an Entity–Relationship HyperGraph modeling retail transactions: each hyperedge e_i connects one customer, one product, and the store where the purchase occurred.

Theorem 3.5 (Generalization and Hypergraph Character)**.** *Let*

$$G = (V,\, E_G,\, \tau_V,\, \tau_E)$$

be an Entity–Relationship Graph as in Definition 3.1. Define

$$\mathcal{E} \;=\; \big\{\{u, v\} \subseteq V \mid (u \to v) \in E_G\big\}, \qquad \tau'_E\big(\{u, v\}\big) \;=\; \tau_E\big(u \to v\big).$$

Then the structure

$$H = \big(V,\, \mathcal{E},\, \tau_V,\, \tau'_E\big)$$

satisfies:

1. *H is an Entity–Relationship HyperGraph that* generalizes *the original graph G by encoding each binary edge as a size-2 hyperedge with matching type labels;*
2. *Ignoring types, $(V, \mathcal{E})$ meets the axioms of a (pure) hypergraph.*

Proof. **(1) Construction and Generalization.** Since $E_G \subseteq V \times V$, for every directed relationship $(u \to v) \in E_G$ the unordered set $\{u, v\}$ is nonempty and lies in $\mathcal{P}(V)$. Thus $\mathcal{E}$ is indeed a collection of nonempty subsets of V. By defining

$$\tau'_E\big(\{u, v\}\big) = \tau_E(u \to v),$$

we ensure that each binary relationship in G is represented as a hyperedge of cardinality 2 in H, with identical relationship-type labeling. All entity-type assignments remain unchanged via τ_V. Hence H encodes exactly the same schema information as G, proving that every Entity–Relationship Graph embeds into this class of HyperGraphs.

(2) Verification of Hypergraph Axioms. By Definition 2.3, a hypergraph $H' = (V', E')$ requires V' to be finite and $E' \subseteq \mathcal{P}(V') \setminus \{\emptyset\}$. In our case, $V' = V$ is finite by hypothesis, and $\mathcal{E} \subseteq \mathcal{P}(V) \setminus \{\emptyset\}$ by construction. Therefore, $(V, \mathcal{E})$ satisfies exactly the definition of a hypergraph. Forgetting the type-labels τ_V, τ'_E leaves an (untyped) hypergraph structure on V, as required. □

Definition 3.6 (Entity–Relationship n-SuperHyperGraph)**.** Let

$$H_0 = \big(V_0,\, \mathcal{E}_0,\, \tau_V,\, \tau_E\big)$$

be an Entity–Relationship HyperGraph as in Definition 3.3. For any integer $n \geq 1$, define the *nesting map*

$$N^1(A) = \{A\}, \qquad N^{k+1}(A) = \{\, N^k(A)\,\}, \quad A \subseteq \mathcal{P}^m(V_0),$$

so that $N^k \colon \mathcal{P}^m(V_0) \to \mathcal{P}^{m+k}(V_0)$. Then set

$$V^{(n)} = \big\{\, N^n(\{v\}) \mid v \in V_0\big\} \subseteq \mathcal{P}^n(V_0), \quad \mathcal{E}^{(n)} = \big\{\, N^{n-1}(e) \mid e \in \mathcal{E}_0\big\} \subseteq \mathcal{P}^n(V_0).$$

Define type-functions by

$$\tau_V^{(n)}\big(N^n(\{v\})\big) = \tau_V(v), \qquad \tau_E^{(n)}\big(N^{n-1}(e)\big) = \tau_E(e).$$

The tuple

$$H^{(n)} = \big(V^{(n)},\, \mathcal{E}^{(n)},\, \tau_V^{(n)},\, \tau_E^{(n)}\big)$$

is called the *Entity–Relationship n-SuperHyperGraph* of H_0.

Example 3.7 (IT System Development Entity–Relationship 2-SuperHyperGraph)**.** Consider a simple system-engineering scenario with the following base entities:

$$V_0 = \{\,\mathsf{Feature1},\, \mathsf{Feature2},\, \mathsf{TeamA},\, \mathsf{ToolX},\, \mathsf{ServiceX}\},$$

with entity types

$$\tau_V(\mathsf{Feature1}) = \tau_V(\mathsf{Feature2}) = \mathsf{Feature}, \quad \tau_V(\mathsf{TeamA}) = \mathsf{Team},$$

$$\tau_V(\mathsf{ToolX}) = \mathsf{Tool}, \quad \tau_V(\mathsf{ServiceX}) = \mathsf{Service}.$$

Define two development events as hyperedges in the Entity–Relationship HyperGraph H_0:

$$e_1 = \{\mathsf{Feature1}, \mathsf{TeamA}, \mathsf{ToolX}, \mathsf{ServiceX}\},$$

$$e_2 = \{\mathsf{Feature2}, \mathsf{TeamA}, \mathsf{ToolX}, \mathsf{ServiceX}\},$$

and set

$$\mathcal{E}_0 = \{e_1, e_2\}$$

with relationship type

$$\tau_E(e_1) = \tau_E(e_2) = \mathsf{DevelopmentEvent}.$$

Hence

$$H_0 = \big(V_0,\ \mathcal{E}_0,\ \tau_V,\ \tau_E\big)$$

is the base Entity–Relationship HyperGraph.

Applying Definition 3.6 with $n = 2$, we form

$$V^{(2)} = \big\{\, N^2(\{v\}) \mid v \in V_0 \big\}$$

$$= \big\{\{\{v\}\} \;\big|\; v \in V_0\big\} \subseteq \mathcal{P}^2(V_0),$$

$$\mathcal{E}^{(2)} = \big\{\, N^1(e) \mid e \in \mathcal{E}_0 \big\}$$

$$= \big\{\{e_1\},\ \{e_2\}\big\} \subseteq \mathcal{P}^2(V_0),$$

with $\tau_V^{(2)}(N^2(\{v\})) = \tau_V(v)$ and $\tau_E^{(2)}(N^1(e_i)) = \tau_E(e_i)$ for $i = 1, 2$. Thus

$$H^{(2)} = \big(V^{(2)},\ \mathcal{E}^{(2)},\ \tau_V^{(2)},\ \tau_E^{(2)}\big)$$

is the Entity–Relationship 2-SuperHyperGraph. Concretely, at layer 1 we have the original development-event hyperedges e_1, e_2; at layer 2 each is lifted to a nested "superevent" $\{e_i\}$, preserving all type information.

Theorem 3.8. *Let H_0 and $H^{(n)}$ be as above. Then:*

1. *$H^{(n)}$ is an n-SuperHyperGraph with $V^{(n)}, \mathcal{E}^{(n)} \subseteq \mathcal{P}^n(V_0)$;*
2. *The maps $\varphi_V : v \mapsto N^n(\{v\})$ and $\varphi_E : e \mapsto N^{n-1}(e)$ embed H_0 into $H^{(n)}$, preserving both entity-type and relationship-type assignments.*

Proof. **(1) SuperHyperGraph Structure.** By construction each $N^n(\{v\})$ lies in $\mathcal{P}^n(V_0)$, so $V^{(n)} \subseteq \mathcal{P}^n(V_0)$. Likewise each $N^{n-1}(e) \subseteq \mathcal{P}^n(V_0)$, so $\mathcal{E}^{(n)} \subseteq \mathcal{P}^n(V_0)$. Hence $(V^{(n)}, \mathcal{E}^{(n)})$ satisfies the definition of an n-SuperHyperGraph [42, 43].

(2) Embedding and Type Preservation. Define

$$\varphi_V : V_0 \to V^{(n)}, \quad \varphi_V(v) = N^n(\{v\}), \qquad \varphi_E : \mathcal{E}_0 \to \mathcal{E}^{(n)}, \quad \varphi_E(e) = N^{n-1}(e).$$

Both φ_V and φ_E are bijections onto their images by the invertibility of the nesting construction. Moreover, for each $v \in V_0$,

$$\tau_V^{(n)}\big(\varphi_V(v)\big) = \tau_V^{(n)}\big(N^n(\{v\})\big) = \tau_V(v),$$

and for each $e \in \mathcal{E}_0$,

$$\tau_E^{(n)}\big(\varphi_E(e)\big) = \tau_E^{(n)}\big(N^{n-1}(e)\big) = \tau_E(e).$$

Thus all type-labels are preserved and H_0 is isomorphically embedded in $H^{(n)}$. This completes the proof. □

4 Service Dependency Graphs and Their Extensions

Dependency Graphs represent relationships where nodes depend on others, often used in task scheduling, build systems, and software engineering (cf. [45–48]). Service Dependency Graph depicts the service provision relationships and constraints among IT services, used for impact analysis in incident and problem management [27]. We extend this concept using hypergraphs and superhypergraphs.

Definition 4.1 (Service Dependency Graph)**.** [27] A *Service Dependency Graph* is a directed graph

$$G = (V, E),$$

where

- V is a finite set of *services*, each $v \in V$ represents an individual IT service or microservice;
- $E \subseteq V \times V$ is a finite set of *dependency edges*, where $(u, v) \in E$ indicates that service u depends on service v.

Example 4.2. Suppose we have three services $\{S_1, S_2, S_3\}$ with dependencies:

$$E = \{(S_1, S_2),\ (S_2, S_3)\}.$$

Then S_1 depends on S_2, and S_2 depends on S_3, giving the directed graph

$$G = \big(\{S_1, S_2, S_3\}, \{(S_1, S_2), (S_2, S_3)\}\big).$$

This reflects a sequential dependency chain $S_1 \to S_2 \to S_3$.

Definition 4.3 (Service Dependency HyperGraph)**.** Let

$$G = (V, E_G)$$

be a Service Dependency Graph as in Definition 4.1, where V is a finite set of services and $E_G \subseteq V \times V$ records directed "depends-on" edges. Define for each $u \in V$

$$D(u) \ = \ \{\, v \in V \mid (u, v) \in E_G \},$$

and let

$$\mathcal{E} = \big\{\ e_u \mid u \in V,\ D(u) \neq \varnothing\ \big\}, \quad e_u = \{\, u \} \ \cup \ D(u).$$

Then the *Service Dependency HyperGraph* is the pair

$$H = \big(V,\ \mathcal{E}\big),$$

where each hyperedge $e_u \in \mathcal{E}$ collects a service u together with all the services it depends on.

Example 4.4 (E-commerce Service Dependency HyperGraph)**.** Consider an e-commerce platform composed of the following services:

$$V = \{\mathsf{Gateway},\ \mathsf{Auth},\ \mathsf{User},\ \mathsf{Order},\ \mathsf{Inventory}\}.$$

At runtime, the directed dependencies are recorded as

$$E_G = \big\{(\mathsf{Gateway}, \mathsf{Auth}),\ (\mathsf{Gateway}, \mathsf{User}),\ (\mathsf{User}, \mathsf{Order}),\ (\mathsf{User}, \mathsf{Inventory}),\ (\mathsf{Order}, \mathsf{Inventory})\big\}.$$

For each service $u \in V$, let

$$D(u) = \{\, v \in V \mid (u, v) \in E_G \}.$$

Then we obtain the nonempty dependency sets

$$D(\mathsf{Gateway}) = \{\mathsf{Auth}, \mathsf{User}\}, \quad D(\mathsf{User}) = \{\mathsf{Order}, \mathsf{Inventory}\}, \quad D(\mathsf{Order}) = \{\mathsf{Inventory}\},$$

and $D(\mathsf{Auth}) = D(\mathsf{Inventory}) = \varnothing$. Form the hyperedges

$$e_{\mathsf{Gateway}} = \{\mathsf{Gateway}\} \cup D(\mathsf{Gateway}) = \{\mathsf{Gateway}, \mathsf{Auth}, \mathsf{User}\},$$

$$e_{\mathsf{User}} = \{\mathsf{User}\} \cup D(\mathsf{User}) = \{\mathsf{User}, \mathsf{Order}, \mathsf{Inventory}\},$$
$$e_{\mathsf{Order}} = \{\mathsf{Order}\} \cup D(\mathsf{Order}) = \{\mathsf{Order}, \mathsf{Inventory}\}.$$

Setting

$$\mathcal{E} = \{\, e_{\mathsf{Gateway}},\, e_{\mathsf{User}},\, e_{\mathsf{Order}}\},$$

we obtain the Service Dependency HyperGraph

$$H = \big(V,\ \mathcal{E}\big).$$

This hypergraph captures multi-service dependency "clusters": $\{\mathsf{Gateway}, \mathsf{Auth}, \mathsf{User}\}$ for request routing, $\{\mathsf{User}, \mathsf{Order}, \mathsf{Inventory}\}$ for order placement, and $\{\mathsf{Order}, \mathsf{Inventory}\}$ for stock reservation.

Theorem 4.5. *Let $G = (V, E_G)$ and $H = (V, \mathcal{E})$ be as above. Then:*

1. *H is a (pure) hypergraph with $\mathcal{E} \subseteq \mathcal{P}(V) \setminus \{\emptyset\}$.*
2. *H* generalizes *G in that the directed dependency edges of G can be recovered from the membership relations in $\mathcal{E}$.*

Proof. **(1) Hypergraph Axioms.** By construction V is finite and nonempty. Each hyperedge $e_u = \{u\} \cup D(u)$ satisfies $\emptyset \neq e_u \subseteq V$, so $\mathcal{E} \subseteq \mathcal{P}(V) \setminus \{\emptyset\}$. Hence $(V, \mathcal{E})$ fulfills the definition of a hypergraph.

(2) Recovery of Directed Edges. To see that H encodes G without loss, observe that for each original dependency $(u, v) \in E_G$, by definition $v \in D(u)$ and therefore $v \in e_u$. Conversely, if $v \in e_u \setminus \{u\}$ for some hyperedge e_u, then $(u, v) \in E_G$. Thus the set of ordered pairs

$$\{\, (u, v) \in V \times V \mid u \neq v,\ v \in e_u\}$$

coincides exactly with E_G. This shows G is recovered from H, proving that the Service Dependency HyperGraph indeed generalizes the original Service Dependency Graph. □

Definition 4.6 (Service Dependency n-SuperHyperGraph)**.** Let

$$H_0 = (V_0,\ \mathcal{E}_0)$$

be the Service Dependency HyperGraph of Definition 4.3, where $\mathcal{E}_0 = \{\, e_u = \{u\} \cup D(u) \mid u \in V_0\} \subseteq \mathcal{P}(V_0) \setminus \{\emptyset\}$. For any integer $n \geq 1$, define the *nesting operator*

$$N^1(A) = \{A\}, \qquad N^{k+1}(A) = \{\, N^k(A)\,\}, \quad A \subseteq \mathcal{P}^m(V_0),$$

so that $N^k(A) \in \mathcal{P}^{m+k}(V_0)$. Then set

$$V^{(n)} = \big\{\, N^n(\{v\}) \ \big|\ v \in V_0\big\} \ \subseteq\ \mathcal{P}^n(V_0), \quad \mathcal{E}^{(n)} = \big\{\, N^{n-1}(e) \ \big|\ e \in \mathcal{E}_0\big\} \ \subseteq\ \mathcal{P}^n(V_0).$$

Define

$$H^{(n)} = \big(V^{(n)},\, \mathcal{E}^{(n)}\big).$$

We call $H^{(n)}$ the *Service Dependency n-SuperHyperGraph* of H_0.

Example 4.7 (E-commerce Service Dependency 2-SuperHyperGraph)**.** Consider an e-commerce platform whose microservices are

$$V_0 = \{\mathsf{Gateway},\ \mathsf{Auth},\ \mathsf{User},\ \mathsf{Order},\ \mathsf{Inventory},\ \mathsf{Payment}\}.$$

At runtime the "depends-on" relations form the Service Dependency HyperGraph

$$H_0 = (V_0, \mathcal{E}_0), \quad \mathcal{E}_0 = \big\{e_{\mathsf{Gateway}}, e_{\mathsf{User}}, e_{\mathsf{Order}}, e_{\mathsf{Payment}}\big\},$$

where

$$e_{\mathsf{Gateway}} = \{\mathsf{Gateway}, \mathsf{Auth}, \mathsf{User}\}, \quad e_{\mathsf{User}} = \{\mathsf{User}, \mathsf{Order}, \mathsf{Inventory}\},$$
$$e_{\mathsf{Order}} = \{\mathsf{Order}, \mathsf{Payment}, \mathsf{Inventory}\}, \quad e_{\mathsf{Payment}} = \{\mathsf{Payment}, \mathsf{Inventory}\}.$$

Applying Definition 4.6 with $n = 2$, we form

$$V^{(2)} = \left\{ N^2(\{v\}) \mid v \in V_0 \right\} = \left\{ \{\{v\}\} \;\middle|\; v \in V_0 \right\} \subseteq \mathcal{P}^2(V_0),$$

$$\mathcal{E}^{(2)} = \left\{ N^1(e) \mid e \in \mathcal{E}_0 \right\} = \left\{ \{e_{\mathsf{Gateway}}\}, \{e_{\mathsf{User}}\}, \{e_{\mathsf{Order}}\}, \{e_{\mathsf{Payment}}\} \right\} \subseteq \mathcal{P}^2(V_0).$$

Thus the Service Dependency 2-SuperHyperGraph is

$$H^{(2)} = \left(V^{(2)}, \mathcal{E}^{(2)}\right),$$

which at layer 1 comprises the original dependency clusters e_*, and at layer 2 wraps each cluster as a nested hyperedge, preserving all service-dependency information in a bona fide 2-SuperHyperGraph.

Theorem 4.8. *With notation as above, for each $n \geq 1$:*

1. *$H^{(n)} = (V^{(n)}, \mathcal{E}^{(n)})$ is an n-SuperHyperGraph, i.e. $V^{(n)}, \mathcal{E}^{(n)} \subseteq \mathcal{P}^n(V_0)$.*
2. *There are natural injections $\varphi_V : V_0 \hookrightarrow V^{(n)},\ v \mapsto N^n(\{v\})$ and $\varphi_E : \mathcal{E}_0 \hookrightarrow \mathcal{E}^{(n)},\ e \mapsto N^{n-1}(e)$, making $H^{(n)}$ a* generalization *of the original Service Dependency HyperGraph H_0.*

Proof. **(1) SuperHyperGraph Structure.** By definition, each $N^n(\{v\})$ lies in the nth iterated powerset $\mathcal{P}^n(V_0)$, so $V^{(n)} \subseteq \mathcal{P}^n(V_0)$. Similarly, for each hyperedge $e \in \mathcal{E}_0$,

$$N^{n-1}(e) \;\subseteq\; \mathcal{P}^n(V_0),$$

and hence $\mathcal{E}^{(n)} \subseteq \mathcal{P}^n(V_0)$. These two inclusions verify that $(V^{(n)}, \mathcal{E}^{(n)})$ satisfies the definition of an n-SuperHyperGraph [43].

(2) Embedding and Generalization. Define

$$\varphi_V : V_0 \to V^{(n)}, \quad v \mapsto N^n(\{v\}), \qquad \varphi_E : \mathcal{E}_0 \to \mathcal{E}^{(n)}, \quad e \mapsto N^{n-1}(e).$$

Each $\varphi_V(v)$ and $\varphi_E(e)$ is nonempty by construction, and the nesting operator N^k is injective on subsets of the appropriate level, so φ_V and φ_E are injections.

Moreover, the original hypergraph H_0 can be recovered by the inverse "flattening" maps:

$$\psi_V\left(N^n(\{v\})\right) = v, \qquad \psi_E\left(N^{n-1}(e)\right) = e,$$

which satisfy $\psi_V \circ \varphi_V = \mathrm{id}_{V_0}$ and $\psi_E \circ \varphi_E = \mathrm{id}_{\mathcal{E}_0}$. Thus $H^{(n)}$ strictly extends H_0 while preserving all original service-dependency relations, proving that $H^{(n)}$ is indeed a generalization of the Service Dependency HyperGraph and an n-SuperHyperGraph. □

5 Change impact graphs and Their Extensions

Change Management is a structured process for controlling and implementing modifications in IT systems with minimal disruption and risk [49, 50]. Change Impact refers to the assessment of how a proposed change affects dependent components, services, or processes within a system [51–53]. Change Impact Graph visualizes configuration items and services affected by configuration changes or release packages, supporting risk assessment and approval workflows [28]. We extend this concept using hypergraphs and superhypergraphs.

Definition 5.1 (Change Impact Graph)**.** [28] Let S be a software system, and let $\mathcal{F}$ be the finite set of all its functions. Suppose we have a version-control history that records, for each function $f \in \mathcal{F}$, the sequence of its concrete definitions over time. Fix a time t (the "current" version) and a *period of interest* $[t_b, t_e]$ in the past.

Let $G_t(f) = (V, E)$ be the directed call-graph of function f at time t, defined by

$$V = \{\, g \in \mathcal{F} \mid g \text{ is reachable from } f \text{ by a sequence of calls at time } t \},\quad E = \{\, (u \to v) \mid u, v \in V,\ u \text{ calls } v \}.$$

Define a marking $\theta : V \to \{\mathsf{unaffected}, \mathsf{changed}, \mathsf{affected}\}$ as follows:

1. For each $v \in V$, set

$$\theta(v) = \begin{cases} \mathsf{changed}, & \text{if } v \text{ 's definition was added or modified at some time } t' \in [t_b, t_e], \\ \mathsf{unaffected}, & \text{otherwise.} \end{cases}$$

2. Repeat until no further changes occur: for any $v \in V$ with $\theta(v) = \mathsf{unaffected}$, if there exists $(v \to w) \in E$ with $\theta(w) \in \{\mathsf{changed}, \mathsf{affected}\}$, then set $\theta(v) \leftarrow \mathsf{affected}$.

The resulting *Change Impact Graph* of f at time t over $[t_b, t_e]$ is the tuple

$$\mathrm{CIG}_t\big(f; [t_b, t_e]\big) = \big(V,\ E,\ \theta\big).$$

Nodes marked changed are those whose definitions changed during $[t_b, t_e]$; nodes marked affected are those transitively impacted by such changes; and unaffected nodes remain purely unchanged.

Definition 5.2 (Change Impact HyperGraph)**.** Let

$$\mathrm{CIG}_t(f; [t_b, t_e]) = (V,\ E_G,\ \theta)$$

be the Change Impact Graph of Definition 5.1. For each vertex $v \in V$, define its *dependency set*

$$D(v) \ = \ \{\, u \in V \mid (u \to v) \in E_G \},$$

and set

$$e_v \ = \ \{\, v \} \ \cup \ D(v).$$

Let

$$\mathcal{E} = \{\, e_v \mid v \in V,\ D(v) \neq \emptyset \}.$$

The *Change Impact HyperGraph* is the triple

$$\widehat{\mathrm{CIG}}_t(f; [t_b, t_e]) = \big(V,\ \mathcal{E},\ \theta\big),$$

where each hyperedge e_v collects a function v together with all functions that call v directly.

Example 5.3 (Change Impact HyperGraph in a Microservice Application)**.** Consider a microservice-based backend with the following functions:

$$V = \{\mathsf{Auth},\ \mathsf{User},\ \mathsf{Order},\ \mathsf{Inventory}\},$$

and directed call-graph at time t:

$$E_G = \big\{(\mathsf{User} \to \mathsf{Auth}),\ (\mathsf{Order} \to \mathsf{User}),\ (\mathsf{Inventory} \to \mathsf{User})\big\}.$$

Suppose that during the release period $[t_b, t_e]$, only the Auth function was modified. We first mark

$$\theta(\mathsf{Auth}) = \mathsf{changed}, \quad \theta(\mathsf{User}) = \theta(\mathsf{Order}) = \theta(\mathsf{Inventory}) = \mathsf{unaffected}.$$

Propagating effects transitively, we update:

$$\theta(\mathsf{User}) \ \mapsto \ \mathsf{affected}, \quad \theta(\mathsf{Order}) \ = \ \theta(\mathsf{Inventory}) \ = \ \mathsf{affected}.$$

Next, compute each dependency set

$$D(v) = \{\, u \in V \mid (u \to v) \in E_G \},$$

giving

$$D(\mathsf{Auth}) = \{\mathsf{User}\}, \quad D(\mathsf{User}) = \{\mathsf{Order},\ \mathsf{Inventory}\}, \quad D(\mathsf{Order}) = D(\mathsf{Inventory}) = \varnothing.$$

Form the nonempty hyperedges

$$e_{\mathsf{Auth}} = \{\mathsf{Auth}\} \cup D(\mathsf{Auth}) = \{\mathsf{Auth}, \mathsf{User}\}, \quad e_{\mathsf{User}} = \{\mathsf{User}\} \cup D(\mathsf{User}) = \{\mathsf{User}, \mathsf{Order}, \mathsf{Inventory}\}.$$

Then the Change Impact HyperGraph is

$$\widehat{\mathrm{CIG}}_t = \big(V,\ \{\, e_{\mathsf{Auth}},\ e_{\mathsf{User}}\},\ \theta\big),$$

where each hyperedge collects a function together with all direct callers, and θ records which functions were changed or affected.

Theorem 5.4. *With notation as above:*

1. *The underlying pair* $(V, \mathcal{E})$ *is a (pure) hypergraph, i.e.* $\mathcal{E} \subseteq \mathcal{P}(V) \setminus \{\emptyset\}$.
2. $\widehat{\mathrm{CIG}}_t(f; [t_b, t_e])$ generalizes *the original Change Impact Graph* $\mathrm{CIG}_t(f; [t_b, t_e])$*: one recovers* θ *unchanged and*
$$E_G = \big\{(u \to v) \in V \times V \mid u \in e_v \setminus \{v\}\big\}.$$

Proof. **(1) Hypergraph Axioms.** By Definition 5.1, V is finite. Each $e_v = \{v\} \cup D(v)$ is nonempty and a subset of V; hence $\mathcal{E} \subseteq \mathcal{P}(V) \setminus \{\emptyset\}$, so $(V, \mathcal{E})$ satisfies the definition of a hypergraph [3, 7].

(2) Recovery of the Original Graph and Marking. By construction, for any $(u \to v) \in E_G$ we have $u \in D(v) \subseteq e_v$, and conversely if $u \in e_v \setminus \{v\}$ then $(u \to v) \in E_G$. Thus the set of directed edges E_G is exactly recovered from the membership relation in $\mathcal{E}$. The marking function $\theta : V \to \{\mathsf{unaffected}, \mathsf{changed}, \mathsf{affected}\}$ remains unchanged, so the hypergraph with marking faithfully generalizes the original Change Impact Graph. □

Definition 5.5 (Change Impact n-SuperHyperGraph)**.** Let
$$\widehat{\mathrm{CIG}}_t\big(f; [t_b, t_e]\big) = \big(V_0,\ \mathcal{E}_0,\ \theta\big)$$
be the Change Impact HyperGraph of Definition 5.2, where $\mathcal{E}_0 = \{\, e_v = \{v\} \cup D(v) \mid v \in V_0\}$. For any integer $n \geq 1$, define the *nesting operator*
$$N^1(A) = \{A\}, \qquad N^{k+1}(A) = \big\{\, N^k(A)\big\}, \quad A \subseteq \mathcal{P}^m(V_0),$$
so that $N^k(A) \in \mathcal{P}^{m+k}(V_0)$. Then set
$$V^{(n)} = \big\{\, N^n(\{v\}) \ \big|\ v \in V_0\big\} \ \subseteq\ \mathcal{P}^n(V_0), \qquad \mathcal{E}^{(n)} = \big\{\, N^{n-1}(e) \ \big|\ e \in \mathcal{E}_0\big\} \ \subseteq\ \mathcal{P}^n(V_0),$$
and extend the marking by
$$\theta^{(n)}\big(N^n(\{v\})\big) = \theta(v) \quad \text{for each } v \in V_0.$$
The tuple
$$\widehat{\mathrm{CIG}}_t^{\,(n)}\big(f; [t_b, t_e]\big) = \big(V^{(n)},\ \mathcal{E}^{(n)},\ \theta^{(n)}\big)$$
is called the *Change Impact n-SuperHyperGraph* of $\widehat{\mathrm{CIG}}_t(f; [t_b, t_e])$.

Example 5.6 (Change Impact 2-SuperHyperGraph in a Microservice Application)**.** Starting from the Change Impact HyperGraph of Example 5.3, let
$$V_0 = \{\mathsf{Auth}, \mathsf{User}, \mathsf{Order}, \mathsf{Inventory}\},$$
$$\mathcal{E}_0 = \{\, e_{\mathsf{Auth}},\ e_{\mathsf{User}}\},$$
$$\theta(\mathsf{Auth}) = \mathsf{changed},$$
$$\theta(\mathsf{User}) = \theta(\mathsf{Order}) = \theta(\mathsf{Inventory}) = \mathsf{affected}.$$
By Definition 5.5, for $n = 2$ we form
$$V^{(2)} = \big\{\, N^2(\{v\}) \mid v \in V_0\big\}$$
$$= \big\{\{\{v\}\} \ \big|\ v \in V_0\big\} \ \subseteq\ \mathcal{P}^2(V_0),$$
$$\mathcal{E}^{(2)} = \big\{\, N^1(e) \mid e \in \mathcal{E}_0\big\} = \big\{\{e_{\mathsf{Auth}}\},\ \{e_{\mathsf{User}}\}\big\} \ \subseteq\ \mathcal{P}^2(V_0),$$
and extend the marking by
$$\theta^{(2)}\big(N^2(\{v\})\big) = \theta(v), \qquad v \in V_0.$$
Thus the Change Impact 2-SuperHyperGraph is
$$\widehat{\mathrm{CIG}}_t^{\,(2)} = \big(V^{(2)},\ \mathcal{E}^{(2)},\ \theta^{(2)}\big).$$
Concretely, at layer 1 we have the original hyperedges $\{e_{\mathsf{Auth}}\}$ and $\{e_{\mathsf{User}}\}$, each labeling which functions are directly impacted; at layer 2 each such set is lifted to a nested "superevent" $\{\{e_v\}\}$, preserving the changed/affected marking on the corresponding 2-supervertices.

Theorem 5.7. *With notation as above, for each $n \geq 1$:*

1. $\widehat{\mathrm{CIG}}_t^{(n)}\big(f; [t_b, t_e]\big)$ *is an n-SuperHyperGraph, i.e.* $V^{(n)}, \mathcal{E}^{(n)} \subseteq \mathcal{P}^n(V_0)$.
2. *There are natural injections* $\varphi_V : V_0 \hookrightarrow V^{(n)},\ v \mapsto N^n(\{v\})$ *and* $\varphi_E : \mathcal{E}_0 \hookrightarrow \mathcal{E}^{(n)},\ e \mapsto N^{n-1}(e)$, *satisfying* $\theta^{(n)} \circ \varphi_V = \theta$. *Hence* $\widehat{\mathrm{CIG}}_t^{(n)}$ *generalizes the original Change Impact HyperGraph* $\widehat{\mathrm{CIG}}_t$.

Proof. **(1) SuperHyperGraph Structure.** By construction each $N^n(\{v\})$ lies in the nth iterated powerset $\mathcal{P}^n(V_0)$, so $V^{(n)} \subseteq \mathcal{P}^n(V_0)$. Similarly, for each $e \in \mathcal{E}_0$, $N^{n-1}(e) \subseteq \mathcal{P}^n(V_0)$, hence $\mathcal{E}^{(n)} \subseteq \mathcal{P}^n(V_0)$. Therefore $\big(V^{(n)}, \mathcal{E}^{(n)}\big)$ satisfies the definition of an n-SuperHyperGraph.

(2) Embedding and Generalization. Define

$$\varphi_V(v) = N^n(\{v\}), \quad \varphi_E(e) = N^{n-1}(e).$$

Since the nesting operator N^k is injective at each level, both φ_V and φ_E are injections. Moreover, for each $v \in V_0$,

$$\theta^{(n)}\big(\varphi_V(v)\big) = \theta^{(n)}\big(N^n(\{v\})\big) = \theta(v),$$

so the marking is preserved. Thus the original Change Impact HyperGraph $\widehat{\mathrm{CIG}}_t$ embeds isomorphically into $\widehat{\mathrm{CIG}}_t^{(n)}$, showing that the latter indeed generalizes the former while forming a bona fide n-SuperHyperGraph. □

6 Workflow Graphs and Their Extensions

Workflow Graph defines ITIL process steps (cf. [54–56]) and decision branches—as nodes and edges—for incident, problem, and change management, aiding operational standardization and automation [29, 33]. We extend this concept using hypergraphs and superhypergraphs.

Definition 6.1 (Workflow Graph)**.** A *workflow graph* is a tuple

$$WG = \big(N, T, TS, TE, C, F\big),$$

where

- N is a finite set of *nodes*;
- $T \subseteq N$ is a finite set of *tasks*;
- $TS \subseteq T$ is the (nonempty) set of *start tasks*;
- $TE \subseteq T$ is the (nonempty) set of *end tasks*;
- $C \subseteq N$ is a finite set of *choice/merge coordinators*;
- $N = T \cup C$ with $T \cap C = \emptyset$;
- $F \subseteq N \times N$ is the *control-flow relation*, a set of directed edges.

Definition 6.2 (Workflow HyperGraph)**.** Let

$$WG = \big(N, T, TS, TE, C, F\big)$$

be a Workflow Graph as in Definition 6.1, where $F \subseteq N \times N$ is the directed control-flow relation. For each node $n \in N$, define its *successor set*

$$\mathrm{Succ}(n) = \{\, m \in N \mid (n, m) \in F \}.$$

Then set

$$\mathcal{E} = \big\{\, e_n \mid n \in N,\ \mathrm{Succ}(n) \neq \emptyset \big\}, \quad e_n = \{n\} \cup \mathrm{Succ}(n).$$

The *Workflow HyperGraph* is the pair

$$H = \big(N,\ \mathcal{E}\big),$$

where each hyperedge e_n collects a node n together with all of its immediate successors in the workflow.

Example 6.3 (ITIL Incident Management Workflow HyperGraph)**.** Consider a simplified ITIL incident management process with:

$$N = \{\mathsf{DetectIncident},\ \mathsf{LogIncident},\ \mathsf{CategorizeSeverity},$$
$$\mathsf{AssignEngineer},\ \mathsf{InvestigateCause},\ \mathsf{ImplementFix},\ \mathsf{CloseIncident}\},$$

start task DetectIncident, end task CloseIncident, and choice/merge coordinator CategorizeSeverity. The control-flow relation $F \subseteq N \times N$ is

$$F = \big\{(\mathsf{DetectIncident}, \mathsf{LogIncident}),\ (\mathsf{LogIncident}, \mathsf{CategorizeSeverity}),$$
$$(\mathsf{CategorizeSeverity}, \mathsf{AssignEngineer}),\ (\mathsf{AssignEngineer}, \mathsf{InvestigateCause}),$$
$$(\mathsf{InvestigateCause}, \mathsf{ImplementFix}),\ (\mathsf{ImplementFix}, \mathsf{CloseIncident})\big\}.$$

For each $n \in N$ with successors, define

$$\mathrm{Succ}(n) = \{\, m \in N \mid (n, m) \in F\},$$
$$e_n = \{\, n\} \cup \mathrm{Succ}(n).$$

Thus the nonempty hyperedges are

$$e_{\mathsf{DetectIncident}} = \{\mathsf{DetectIncident}, \mathsf{LogIncident}\},$$
$$e_{\mathsf{LogIncident}} = \{\mathsf{LogIncident}, \mathsf{CategorizeSeverity}\},$$
$$e_{\mathsf{CategorizeSeverity}} = \{\mathsf{CategorizeSeverity}, \mathsf{AssignEngineer}\},$$
$$e_{\mathsf{AssignEngineer}} = \{\mathsf{AssignEngineer}, \mathsf{InvestigateCause}\},$$
$$e_{\mathsf{InvestigateCause}} = \{\mathsf{InvestigateCause}, \mathsf{ImplementFix}\},$$
$$e_{\mathsf{ImplementFix}} = \{\mathsf{ImplementFix}, \mathsf{CloseIncident}\}.$$

Setting

$$\mathcal{E} = \big\{\, e_{\mathsf{DetectIncident}},\ e_{\mathsf{LogIncident}},\ e_{\mathsf{CategorizeSeverity}},$$
$$e_{\mathsf{AssignEngineer}},\ e_{\mathsf{InvestigateCause}},\ e_{\mathsf{ImplementFix}}\big\},$$

the resulting Workflow HyperGraph is

$$H = \big(N,\ \mathcal{E}\big),$$

which captures each workflow step together with all of its immediate successors in the incident management process.

Theorem 6.4. *Let $WG = (N, T, TS, TE, C, F)$ be a Workflow Graph and $H = (N, \mathcal{E})$ its associated Workflow HyperGraph. Then:*

1. *$(N, \mathcal{E})$ is a (pure) hypergraph, i.e. $\mathcal{E} \subseteq \mathcal{P}(N) \setminus \{\emptyset\}$.*
2. *H* generalizes *WG in that the original control-flow edges F can be recovered by*

$$F \ = \ \big\{\, (n, m) \in N \times N \mid n \neq m,\ m \in e_n\big\}.$$

Proof. **(1) Hypergraph Axioms.** By Definition 6.1, N is finite. For each $n \in N$ with $\mathrm{Succ}(n) \neq \emptyset$, the set

$$e_n = \{n\} \cup \mathrm{Succ}(n)$$

is nonempty and a subset of N. Hence $\mathcal{E} \subseteq \mathcal{P}(N) \setminus \{\emptyset\}$, so $(N, \mathcal{E})$ satisfies the definition of a hypergraph [7].

(2) Recovery of Control-Flow. By construction, for any $(n, m) \in F$ we have $m \in \mathrm{Succ}(n) \subseteq e_n$, and conversely if $m \in e_n \setminus \{n\}$, then $(n, m) \in F$. Therefore the set of directed edges $\{(n, m) \mid m \in e_n \setminus \{n\}\}$ coincides exactly with F. This shows that the original Workflow Graph is fully encoded in and recoverable from the Workflow HyperGraph, confirming that H generalizes WG. □

Definition 6.5 (Workflow n-SuperHyperGraph)**.** Let

$$H_0 = \big(N_0,\ \mathcal{E}_0\big)$$

be the Workflow HyperGraph of Definition 6.2, where $\mathcal{E}_0 = \{\, e_x = \{x\} \cup \mathrm{Succ}(x) \mid x \in N_0,\ \mathrm{Succ}(x) \neq \emptyset\}$. For any integer $n \geq 1$, define the *nesting operator*

$$N^1(A) = \{A\}, \qquad N^{k+1}(A) = \{\, N^k(A)\,\}, \quad A \subseteq \mathcal{P}^m(N_0),$$

so that $N^k(A) \in \mathcal{P}^{m+k}(N_0)$. Then set

$$V^{(n)} = \big\{\, N^n(\{x\}) \ \big|\ x \in N_0\big\} \ \subseteq\ \mathcal{P}^n(N_0), \qquad \mathcal{E}^{(n)} = \big\{\, N^{n-1}(e) \ \big|\ e \in \mathcal{E}_0\big\} \ \subseteq\ \mathcal{P}^n(N_0).$$

The *Workflow n-SuperHyperGraph* is the pair

$$H^{(n)} = \big(V^{(n)},\ \mathcal{E}^{(n)}\big).$$

Example 6.6 (ITIL Incident Management Workflow 2-SuperHyperGraph)**.** Starting from the Workflow HyperGraph $H_0 = (N_0, \mathcal{E}_0)$ in Example 6.3, where

$$N_0 = \{\mathsf{DetectIncident},\ \mathsf{LogIncident},\ \mathsf{CategorizeSeverity},$$
$$\mathsf{AssignEngineer},\ \mathsf{InvestigateCause},\ \mathsf{ImplementFix}\},$$

and

$$\mathcal{E}_0 = \big\{e_{\mathsf{DetectIncident}}, e_{\mathsf{LogIncident}}, e_{\mathsf{CategorizeSeverity}},$$
$$e_{\mathsf{AssignEngineer}}, e_{\mathsf{InvestigateCause}}, e_{\mathsf{ImplementFix}}\big\}$$

with each $e_x = \{x\} \cup \mathrm{Succ}(x) \subseteq N_0$. For $n = 2$, apply the nesting operator N^k to obtain:

$$V^{(2)} = \big\{\, N^2(\{x\}) \ \big|\ x \in N_0\big\} = \big\{\{\{x\}\} \ \big|\ x \in N_0\big\} \ \subseteq\ \mathcal{P}^2(N_0),$$

$$\mathcal{E}^{(2)} = \big\{\, N^1(e) \ \big|\ e \in \mathcal{E}_0\big\} = \big\{\{\, e_{\mathsf{DetectIncident}}\,\}, \{\, e_{\mathsf{LogIncident}}\,\}, \ldots, \{\, e_{\mathsf{ImplementFix}}\,\}\big\} \ \subseteq\ \mathcal{P}^2(N_0).$$

Hence the Workflow 2-SuperHyperGraph is

$$H^{(2)} = \big(V^{(2)},\ \mathcal{E}^{(2)}\big).$$

Concretely, layer 1 comprises the original hyperedges e_x each collecting a step and its successors, while layer 2 wraps each such cluster into a nested hyperedge $\{e_x\}$, preserving the exact workflow structure within a bona fide 2-SuperHyperGraph.

Theorem 6.7. *With notation as above, for every $n \geq 1$:*

1. *$(V^{(n)}, \mathcal{E}^{(n)})$ is an n-SuperHyperGraph, i.e. $V^{(n)}, \mathcal{E}^{(n)} \subseteq \mathcal{P}^n(N_0)$.*
2. *There are natural injections $\varphi_N : N_0 \hookrightarrow V^{(n)},\ x \mapsto N^n(\{x\})$ and $\varphi_E : \mathcal{E}_0 \hookrightarrow \mathcal{E}^{(n)},\ e \mapsto N^{n-1}(e)$, recovering the original Workflow HyperGraph H_0 by "flattening": $\psi_N(N^n(\{x\})) = x$ and $\psi_E(N^{n-1}(e)) = e$.*

Proof. **(1) SuperHyperGraph Structure.** By construction, for each $x \in N_0$ we have $N^n(\{x\}) \in \mathcal{P}^n(N_0)$, so $V^{(n)} \subseteq \mathcal{P}^n(N_0)$. Likewise, for each $e \in \mathcal{E}_0$, $N^{n-1}(e) \subseteq \mathcal{P}^n(N_0)$, hence $\mathcal{E}^{(n)} \subseteq \mathcal{P}^n(N_0)$. Therefore $(V^{(n)}, \mathcal{E}^{(n)})$ meets the definition of an n-SuperHyperGraph [43].

(2) Embedding and Flattening. Define

$$\varphi_N(x) = N^n(\{x\}), \quad \varphi_E(e) = N^{n-1}(e).$$

The nesting operator N^k is injective at each level, so φ_N and φ_E are injections. Define inverse "flattening" maps

$$\psi_N\big(N^n(\{x\})\big) = x, \qquad \psi_E\big(N^{n-1}(e)\big) = e,$$

which satisfy $\psi_N \circ \varphi_N = \mathrm{id}_{N_0}$ and $\psi_E \circ \varphi_E = \mathrm{id}_{\mathcal{E}_0}$. Hence H_0 is isomorphically embedded in $H^{(n)}$, showing that the Workflow n-SuperHyperGraph generalizes the original Workflow HyperGraph while forming a bona fide n-SuperHyperGraph. □

7 Conclusion and Future Works

In this paper, we demonstrated how classical models such as call graphs, state transition graphs, and resource allocation graphs can be reformulated and extended into the frameworks of hypergraphs and n-SuperHyperGraphs. This extension enables the representation of multi-way interactions and supports hierarchical analysis across multiple levels of abstraction.

As future work, we aim to investigate concrete algorithmic implementations and practical applications of these structures. Furthermore, we plan to explore enhanced models incorporating uncertainty and vagueness by extending the current framework using Fuzzy Sets [57, 58], HyperFuzzy Sets [59, 60], Bipolar Fuzzy Sets [61–63], Intuitionistic Fuzzy Sets [64, 65], and Plithogenic Sets [66, 67].

Funding

This study did not receive any financial or external support from organizations or individuals.

Acknowledgments

We extend our sincere gratitude to everyone who provided insights, inspiration, and assistance throughout this research. We particularly thank our readers for their interest and acknowledge the authors of the cited works for laying the foundation that made our study possible. We also appreciate the support from individuals and institutions that provided the resources and infrastructure needed to produce and share this paper. Finally, we are grateful to all those who supported us in various ways during this project.

Data Availability

This research is purely theoretical, involving no data collection or analysis. We encourage future researchers to pursue empirical investigations to further develop and validate the concepts introduced here.

Ethical Approval

As this research is entirely theoretical in nature and does not involve human participants or animal subjects, no ethical approval is required.

Conflicts of Interest

The authors confirm that there are no conflicts of interest related to the research or its publication.

Disclaimer

This work presents theoretical concepts that have not yet undergone practical testing or validation. Future researchers are encouraged to apply and assess these ideas in empirical contexts. While every effort has been made to ensure accuracy and appropriate referencing, unintentional errors or omissions may still exist. Readers are advised to verify referenced materials on their own. The views and conclusions expressed here are the authors' own and do not necessarily reflect those of their affiliated organizations.

References

[1] Reinhard Diestel. *Graph theory*. Springer (print edition); Reinhard Diestel (eBooks), 2024.

[2] Jonathan L Gross, Jay Yellen, and Mark Anderson. *Graph theory and its applications*. Chapman and Hall/CRC, 2018.

[3] Claude Berge. *Hypergraphs: combinatorics of finite sets*, volume 45. Elsevier, 1984.

[4] Florentin Smarandache. *Introduction to the n-SuperHyperGraph-the most general form of graph today*. Infinite Study, 2022.

[5] Mohammad Hamidi, Florentin Smarandache, and Mohadeseh Taghinezhad. *Decision Making Based on Valued Fuzzy Superhypergraphs*. Infinite Study, 2023.

[6] Reinhard Diestel. Graduate texts in mathematics: Graph theory.

[7] Alain Bretto. Hypergraph theory. *An introduction. Mathematical Engineering. Cham: Springer*, 1, 2013.

[8] Derun Cai, Moxian Song, Chenxi Sun, Baofeng Zhang, Shenda Hong, and Hongyan Li. Hypergraph structure learning for hypergraph neural networks. In *IJCAI*, pages 1923–1929, 2022.

[9] Yue Gao, Zizhao Zhang, Haojie Lin, Xibin Zhao, Shaoyi Du, and Changqing Zou. Hypergraph learning: Methods and practices. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(5):2548–2566, 2020.

[10] Mohammad Hamidi, Florentin Smarandache, and Elham Davneshvar. Spectrum of superhypergraphs via flows. *Journal of Mathematics*, 2022(1):9158912, 2022.

[11] Mohammad Hamidi and Mohadeseh Taghinezhad. *Application of Superhypergraphs-Based Domination Number in Real World*. Infinite Study, 2023.

[12] Takaaki Fujita and Florentin Smarandache. A concise study of some superhypergraph classes. *Neutrosophic Sets and Systems*, 77:548–593, 2024.

[13] N. B. Nalawade, M. S. Bapat, S. G. Jakkewad, G. A. Dhanorkar, and D. J. Bhosale. Structural properties of zero-divisor hypergraph and superhypergraph over $\mathbb{Z}_n$: Girth and helly property. *Panamerican Mathematical Journal*, 35(4S):485, 2025.

[14] Mohammed Alqahtani. Intuitionistic fuzzy quasi-supergraph integration for social network decision making. *International Journal of Analysis and Applications*, 23:137–137, 2025.

[15] Benjamin S Blanchard, Wolter J Fabrycky, and Walter J Fabrycky. *Systems engineering and analysis*, volume 4. Prentice hall Englewood Cliffs, NJ, 1990.

[16] John G Proakis, Masoud Salehi, Ning Zhou, and Xiaofeng Li. *Communication systems engineering*, volume 2. Prentice Hall New Jersey, 1994.

[17] Norman S Nise. *Control systems engineering*. John wiley & sons, 2019.

[18] Philipp Skavantzos and Sebastian Link. Entity/relationship graphs: Principled design, modeling, and data integrity management of graph databases. *Proceedings of the ACM on Management of Data*, 3(1):1–26, 2025.

[19] Soumen Chakrabarti and Alekh Agarwal. Learning parameters in entity relationship graphs from ranking preferences. In *Knowledge Discovery in Databases: PKDD 2006: 10th European Conference on Principles and Practice of Knowledge Discovery in Databases Berlin, Germany, September 18-22, 2006 Proceedings 10*, pages 91–102. Springer, 2006.

[20] Minsuk Kahng, Sangkeun Lee, and Sang-goo Lee. Ranking objects by following paths in entity-relationship graphs. In *Proceedings of the 4th workshop on Workshop for Ph. D. students in information & knowledge management*, pages 11–18, 2011.

[21] Peter Pin-Shan Chen. The entity-relationship model-toward a unified view of data. *ACM transactions on database systems (TODS)*, 1(1):9–36, 1976.

[22] Toby J Teorey, Dongqing Yang, and James P Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys (CSUR)*, 18(2):197–222, 1986.

[23] Ramez Elmasri, J Weeldreyer, and A Hevner. The category concept: An extension to the entity-relationship model. *Data & Knowledge Engineering*, 1(1):75–116, 1985.

[24] Pramuditha Suraweera and Antonija Mitrovic. An intelligent tutoring system for entity relationship modelling. *International Journal of Artificial Intelligence in Education*, 14(3-4):375–417, 2004.

[25] Weijie Hong, Yong Yang, Junqi Wu, Dongdong Shangguan, Yuanhao Lai, Qiang Bai, and Ying Li. Nicsdg: A non-intrusive approach to constructing concise service dependency graphs for microservice systems. In *2024 IEEE 35th International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 79–84. IEEE, 2024.

[26] Edgars Gaidels and Marite Kirikova. Service dependency graph analysis in microservice architecture. In *Perspectives in Business Informatics Research: 19th International Conference on Business Informatics Research, BIR 2020, Vienna, Austria, September 21–23, 2020, Proceedings 19*, pages 128–139. Springer, 2020.

[27] Patrick Harris, Mia Gortney, Amr S Abdelfattah, Tomas Cerny, and Pablo Rivas. Designing a system-centered view to microservices using service dependency graphs: Elaborating on 2d and 3d visualization. In *2024 4th International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pages 01–07. IEEE, 2024.

[28] Daniel M German, Ahmed E Hassan, and Gregorio Robles. Change impact graphs: Determining the impact of prior codechanges. *Information and Software Technology*, 51(10):1394–1408, 2009.

[29] Wil MP van der Aalst, Alexander Hirnschall, and HMW Verbeek. An alternative way to analyze workflow graphs. In *Advanced Information Systems Engineering: 14th International Conference, CAiSE 2002 Toronto, Canada, May 27–31, 2002 Proceedings 14*, pages 535–552. Springer, 2002.

[30] Johann Eder, Wolfgang Gruber, and Horst Pichler. Transforming workflow graphs. In *Interoperability of Enterprise Software and Applications*, pages 203–214. Springer, 2006.

[31] Emanuele Santos, Lauro Lins, James P Ahrens, Juliana Freire, and Cláudio T Silva. A first study on clustering collections of workflow graphs. In *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop, IPAW 2008, Salt Lake City, UT, USA, June 17-18, 2008. Revised Selected Papers 2*, pages 160–173. Springer, 2008.

[32] Minsuk Chang, Ben Lafreniere, Juho Kim, George Fitzmaurice, and Tovi Grossman. Workflow graphs: A computational model of collective task strategies for 3d design software. In *Graphics Interface 2020*, 2020.

[33] Thomas S Heinze, Wolfram Amme, and Simon Moser. A restructuring method for ws-bpel business processes based on extended workflow graphs. In *International Conference on Business Process Management*, pages 211–228. Springer, 2009.

[34] Takaaki Fujita and Prem Kumar Singh. Hyperfuzzy graph and hyperfuzzy hypergraph. *Journal of Neutrosophic and Fuzzy Systems (JNFS)*, 10(01):01–13, 2025.

[35] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3558–3565, 2019.

[36] Takaaki Fujita. Exploration of graph classes and concepts for superhypergraphs and n-th power mathematical structures. *Advancing Uncertain Combinatorics through Graphization, Hyperization, and Uncertainization: Fuzzy, Neutrosophic, Soft, Rough, and Beyond*, 3(4):512, 2025.

[37] Takaaki Fujita. Review of some superhypergraph classes: Directed, bidirected, soft, and rough. *Advancing Uncertain Combinatorics through Graphization, Hyperization, and Uncertainization: Fuzzy, Neutrosophic, Soft, Rough, and Beyond (Second Volume)*, 2024.

[38] Takaaki Fujita. An introduction and reexamination of molecular hypergraph and molecular n-superhypergraph. *Asian Journal of Physical and Chemical Sciences*, 13(3):1–38, 2025.

[39] Florentin Smarandache. Foundation of superhyperstructure & neutrosophic superhyperstructure. *Neutrosophic Sets and Systems*, 63(1):21, 2024.

[40] Florentin Smarandache. *SuperHyperFunction, SuperHyperStructure, Neutrosophic SuperHyperFunction and Neutrosophic SuperHyperStructure: Current understanding and future directions*. Infinite Study, 2023.

[41] Florentin Smarandache. Superhyperstructure & neutrosophic superhyperstructure, 2024. Accessed: 2024-12-01.

[42] Takaaki Fujita. Review of probabilistic hypergraph and probabilistic superhypergraph. *Spectrum of Operational Research*, 3(1):319–338, 2026.

[43] Florentin Smarandache. *Extension of HyperGraph to n-SuperHyperGraph and to Plithogenic n-SuperHyperGraph, and Extension of HyperAlgebra to n-ary (Classical-/Neutro-/Anti-) HyperAlgebra*. Infinite Study, 2020.

[44] Hyewon Jeon and Jay-Yoon Lee. Graphcheck: Multi-path fact-checking with entity-relationship graphs. *arXiv preprint arXiv:2502.20785*, 2025.

[45] Sebastian Rötzer, Sebastian Schweigert-Recksiek, Dominik Thoma, and Markus Zimmermann. Attribute dependency graphs: modelling cause and effect in systems design. *Design Science*, 8:e27, 2022.

[46] Željko Agić, Alexander Koller, and Stephan Oepen. Semantic dependency graph parsing using tree approximations. In *Proceedings of the 11th International Conference on Computational Semantics*, pages 217–227, 2015.

[47] Winston Ewert. The dependency graph of life. *BIO-Complexity*, 2018, 2018.

[48] Chadi Kari, Alexander Russell, and Narasimha Shashidhar. Work-competitive scheduling on task dependency graphs. *Parallel Processing Letters*, 25(02):1550001, 2015.

[49] Shalinka Jayatilleke and Richard Lai. A systematic review of requirements change management. *Information and Software Technology*, 93:163–185, 2018.

[50] Stephen G Eick, Todd L Graves, Alan F Karr, J Steve Marron, and Audris Mockus. Does code decay? assessing the evidence from change management data. *IEEE transactions on software engineering*, 27(1):1–12, 2001.

[51] Neha Rungta, Suzette Person, and Joshua Branchaud. A change impact analysis to characterize evolving program behaviors. In *2012 28th IEEE international conference on software maintenance (ICSM)*, pages 109–118. IEEE, 2012.

[52] Barbara G Ryder and Frank Tip. Change impact analysis for object-oriented programs. In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 46–53, 2001.

[53] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara G Ryder, and Ophelia Chesley. Chianti: a tool for change impact analysis of java programs. In *Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 432–448, 2004.

[54] Kevin Holland. An example itil-based model for effective service integration and management. *Whitepaper. Axelos Ltd*, 2015.

[55] Rudy Yandri, Ditdit Nugeraha Utama, Amalia Zahra, et al. Evaluation model for the implementation of information technology service management using fuzzy itil. *Procedia computer science*, 157:290–297, 2019.

[56] Olano Garces Luisa Vivian, Olano Garces Lidia Alessandra, Casildo Bedón Nancy Esther, Levano Rodriguez Danny, and Soria Quijaite Juan Jesús. Prioritization of incident management process using itil-fuzzy for informatics development sector of private universities. *International Journal of Fuzzy Logic and Intelligent Systems*, 25(1):37–54, 2025.

[57] Lotfi A Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.

[58] Hongxing Li and Vincent C Yen. *Fuzzy sets and fuzzy decision-making*. CRC press, 1995.

[59] Young Bae Jun, Kul Hur, and Kyoung Ja Lee. Hyperfuzzy subalgebras of bck/bci-algebras. *Annals of Fuzzy Mathematics and Informatics*, 2017.

[60] Jayanta Ghosh and Tapas Kumar Samanta. Hyperfuzzy sets and hyperfuzzy group. *Int. J. Adv. Sci. Technol*, 41:27–37, 2012.

[61] Wen-Ran Zhang. Bipolar fuzzy sets and relations: a computational framework for cognitive modeling and multiagent decision analysis. *NAFIPS/IFIS/NASA '94. Proceedings of the First International Joint Conference of The North American Fuzzy Information Processing Society Biannual Conference. The Industrial Fuzzy Control and Intellige*, pages 305–309, 1994.

[62] Wen-Ran Zhang. Bipolar fuzzy sets. 1997.

[63] Muhammad Akram. Bipolar fuzzy graphs. *Information sciences*, 181(24):5548–5564, 2011.

[64] Krassimir Atanassov and George Gargov. Elements of intuitionistic fuzzy logic. part i. *Fuzzy sets and systems*, 95(1):39–52, 1998.

[65] Krassimir T Atanassov. *On intuitionistic fuzzy sets theory*, volume 283. Springer, 2012.

[66] Muhammad Azeem, Humera Rashid, Muhammad Kamran Jamil, Selma Gütmen, and Erfan Babaee Tirkolaee. Plithogenic fuzzy graph: A study of fundamental properties and potential applications. *Journal of Dynamics and Games*, pages 0–0, 2024.

[67] Nivetha Martin. Plithogenic swara-topsis decision making on food processing methods with different normalization techniques. *Advances in Decision Making*, 69, 2022.