# Enhancing Expressivity, Modularity and Rigour of Graphical Data Modelling with Fragmenta

NUNO AMÁLIO, Nottingham Trent University, UK

Graphical data (or conceptual) modelling, software engineering's most popular application of modelling, needs an uplift. Graphical data models easily become unwieldy, even for trivial medium-sized systems. Practitioners long for improved modelling primitives and mechanisms that improve modelling, namely, its expressiveness, the way it is experienced and its appeal. These are the drivers of Fragmenta, a formal data- and meta-modelling framework with advanced means to separate concerns. Fragmenta's novelties include: (i) multilevel modelling through vertical refinement, (ii) improved horizontal decomposition and composition of fragments through proxies, (iii) *virtuals* for lighter inheritance, and (iv) graphical model constraints. Fragmenta is the first data modelling framework with a mathematical foundation and a supporting implementation, combining both vertical and horizontal means to separate concerns.

CCS Concepts: • **Software and its engineering → Design languages**.

Additional Key Words and Phrases: Model-driven development, modelling, data-modelling, scalability, metamodelling, graphs, Fragmenta, multilevel modelling, model refinement, graphical constraints, OO inheritance

## 1 INTRODUCTION

Graphical data models easily become large and unwieldy. The more complex the problem the worse it gets. When models become cluttered, in the case of data models this may happen even in simple systems, our capacity to handle them degrades considerably [46]. Such models must be kept tidy with manageable complexity; this requires means to break things apart properly to separate concerns [54, 61]. The effectiveness of the mechanisms that help managing model complexity dictate how scalable a particular language is. Because mainstream graphical languages, such as UML and entity relationship diagrams, lack advanced complexity management means, their scalability is limited [49]. This paper presents Fragmenta, the formal data- and meta- modelling graphical framework that provides novel means of expressiveness, abstraction and decomposition.

Fragmenta's class modelling[1] articulates concerns through *vertical* and *horizontal refinement*. Vertical refinement (or layering) is about abstraction; it allows abstract layers to be refined or elaborated into more concrete layers. Horizontal refinement, about division or separation, decomposes class models into fragments along *proxies*, which are the joints holding the separate fragments (smaller class models) together, enabling composition; a Fragmenta model is a collection of fragments with a coherent meaning as a whole.

Fragmenta's multilevel modelling based on vertical refinement offers an architecture based on two strata: type (or meta) and instance. The type stratum can be further divided into layers related through a chain of vertical refinements, following principles and ideas of data refinement [35]. The instance stratum is indivisible. Type and instance strata are related through instantiation (or typing); type (or meta) layers or models are related through refinement.

Defined mathematically upon algebraic graphs and their morphisms, Fragmenta provides a foundation for modular and abstract class modelling. It was specified in Z [37, 60, 64] using the CZT type-checker [45, 47], verified in Isabelle [63], and implemented in Haskell [15, 36, 44][2].

[1] Class modelling, which evolved from database design and entity relationship modelling [19], is used for both data- and meta-modelling. The latter, Fragmenta's major use so far, describes the abstract syntax of graphical languages.
[2] Available from https://github.com/namalio/Fragmenta.

Author's address: Nuno Amálio, Nottingham Trent University, Nottingham, UK, nuno.rodriguesamalio@ntu.ac.uk.

### 1.1 Research Questions

The paper addresses the following research questions (RQs):

- *RQ1*: Can the complexity of large data models be mastered through division (or separation) to ultimately facilitate comprehension, articulation and mechanisation?[3]
- *RQ2*: Can class-modelling accommodate vertical abstraction and multi-level modelling?
- *RQ3*: Can graphical data modelling be improved to enhance expressiveness?

### 1.2 Contributions

The paper's contributions are as follows:

- Fragmenta, initiated in [7] and applied in [11], is equipped here with *layering* or *vertical refinement*. To the author's knowledge, Fragmenta is the first formal data- and meta- modelling framework that supports both horizontal and vertical refinement to complement separation with enhanced abstraction.
- Fragmenta's multilevel modelling based on an architecture made-up of two strata, type and instance, with the former possibly further divided into layers through vertical refinement, and allowing extra-typings between models within the same layer, is, to the author's knowledge, novel. Fragmenta pioneers the application of refinement, inspired by data refinement [35], in formal and graphical multilevel data- and meta-modelling, refreshing a body of work focused mostly on instantiation.
- The *proxy* primitive, a notational novelty of [7], is enhanced here with fragment composition (or resolution), based on the newly introduced notion of graph subsumption, providing a precisely defined composition mechanism.
- Several notational novelties of Fragmenta, namely: (i) *virtual nodes* (or *virtuals*), which provide lighter inheritance with weaker lineage (or ancestry), supporting union types; (ii) derived, value-constraint and path edges, which express model constraints graphically.
- The tool, which attests to the realisability of Fragmenta, enables experimentation; it checks the healthiness of Fragmenta models, and the correctness of typings and refinements; it deemed adequate all layered models and small examples presented in the paper.

### 1.3 Outline

The remainder of this paper starts with a primer on Fragmenta (section 2), to introduce Fragmenta's main primitives. This is followed by Fragmenta's mathematical definition which upgrades and re-defines [7] (section 3). Fragmenta is then further illustrated through three examples: (i) an e-commerce system (section 4), (ii) the abstract syntax of statecharts [33] (section 5), (iii) and a SysML profile for cyber-physical systems (section 6). The evaluation (section 7) appraises Fragmenta against related approaches. The remaining sections discuss the paper's results and achievements (section 8) and related work (section 9), and draw the conclusions (section 10). Appendix A provides Fragmenta's detailed mathematical definitions.

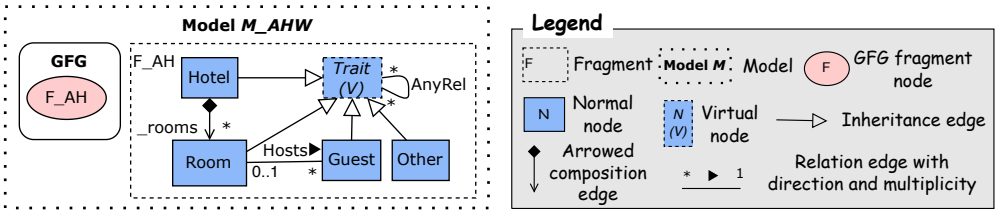



Fig. 1. Abstract hotel world model made up of one sole fragment.

## 2 A PRIMER ON FRAGMENTA

A two-layered model illustrates Fragmenta. The abstract layer (Fig. 1) describes hotels in general; the concrete layer (Fig. 2) describes a hotel for pets. Figure 2 refines Fig. 1, called *vertical refinement* herein: something more concrete is defined from something more abstract.

The model of Fig. 1 comprises one sole fragment, F_AH, as per the *Global Fragment Graph* (GFG). F_AH illustrates several modelling primitives of Fragmenta:

- All rectangles are nodes representing concepts from the domain of discourse. The dashed rectangle named Trait and labelled with "(V)" is a *virtual*, a Fragmenta novelty to represent inheritable features, providing a weaker notion of inheritance parenthood and enabling multiple inheritance. Virtuals are both like and unlike abstract nodes (e.g. Pet in Fig. 2), the equivalent of abstract classes in the classical object-oriented (OO) paradigm; like because they do not have direct instances; unlike because abstract nodes set an inheritance lineage and only allow single inheritance in Fragmenta[4]. The remaining nodes are self-descriptive; Other represents any other relevant concept, apart from Hotel, Room and Guest.
- The solid plain lines (without diamonds or triangles) are *relation* edges, which say that one node is meaningfully related to another. In Fig. 1, Hosts says that guests stay in rooms (arrow above line indicates direction), and AnyRel says that Traits may be related.
- The hollow-headed arrows establish *inheritance* or *is-a* relations between nodes — Other, Guest, Hotel and Room are Traits, hence AnyRel also applies to them.
- Compositions (lines with black diamond at the end) establish *part-of* relations with the implication that parts are not shared; Hotel_rooms[5] says that Rooms belong to Hotels.
- Relation and composition edges have multiplicities at the ends — e.g. Hosts is many ($*$) to optional ($0\,..\,1$) —, and have a direction indicated by either the edge's arrow, or symbols ◄ and ► sitting above an undirected edge. Undirected edges have multiplicities of 1, unless otherwise stated. Arrowed edges only require the target multiplicity (e.g. edge Hotel_rooms); the source multiplicity is assumed to be either many (relations) or 1 (compositions).

Figure 2 portrays M_PW, a model of an hotel for pets. Figure 3 maps the elements of M_PW onto M_AHW to say how the former complies to the latter. The GFG of Fig. 2 says how the model's fragments transitively *reference* each other,

[3]Complexity tackled through division (or separation) is the basis of modularity in computer science and software engineering [54]; modularity tends to benefit comprehension and articulation [61].
[4]Virtuals lack an actual existence and embody a weaker notion of parenthood being virtual and not actual parents. A node may have: at most one strong (a normal or abstract node) and many virtual parents.
[5]Names of directed edges starting with an underscore ('_') make use of a convention to shorten names; the final name is obtained by prefixing the given name with the name of the source node — here, final name is Hotel_rooms.

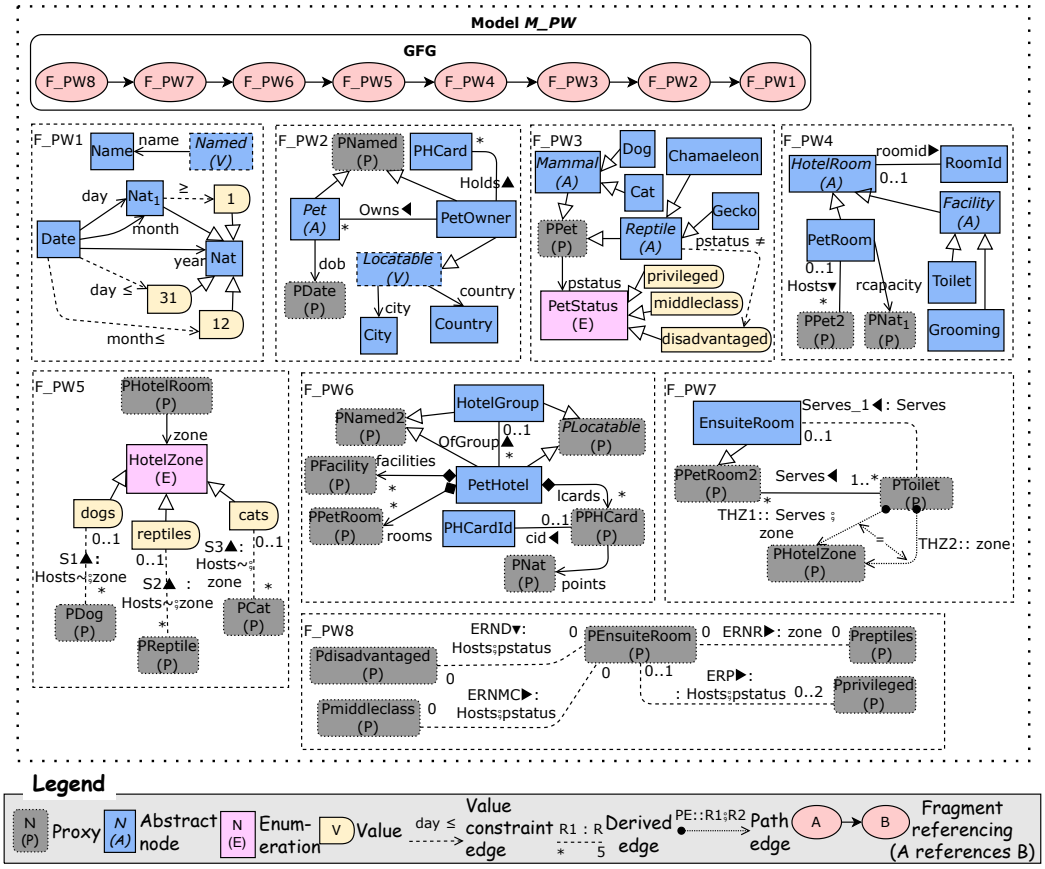



Fig. 2. Pets world fragmented model

building one upon another. M_PW comprises eight fragments which divide the modelling discourse through *horizontal refinement* into: base concepts (F_PW1), pets and their owners (F_PW2), pet kinds and their status (F_PW3), rooms (F_PW4), hotel zones (F_PW5), the pet hotel (F_PW6), ensuite rooms (F_PW7) and room constraints (F_PW8).

The fragments of M_PW (Fig. 2), which illustrate other primitives of Fragmenta, are as follows:

- F_PW1 illustrates *value constraint edges* (VCEs), dashed arrows, a Fragmenta novelty. VCEs constrain the values of a node or edge according to an operator, such as =, to some value. In F_PW1: (i) $\texttt{Nat}_1$, which specialises Nat (all natural numbers), is constrained to be made of positive numbers through operator ≥, and (ii) Date's edges day and month, which are $\texttt{Nat}_1$s, are restricted to be less or equal to 31 and 12, respectively.
- F_PW2 illustrates *proxies*, rounded rectangles with dotted lines labelled with "(P)". PNamed and PDate represent Named and Date from F_PW1, allowed because in the GFG F_PW2 references F_PW1. F_PW2 introduces abstract node *Pet*, which sets a classical inheritance lineage, whereas the virtual Locatable represents something inheritable with virtual parenthood to say that PetOwners have a city and country.

- F_PW3 introduces *enumeration* PetStatus, which defines the status of pets as either privileged, middleclass or disadvantaged. Pets are further subdivided, through inheritance, into Mammals, either Dogs or Cats, or Reptiles, either Chamaeleon or Gecko; the VCE states that reptiles must not be disadvantaged.
- F_PW5 illustrates *derived edges* (dashed lines), a Fragmenta novelty to describe constraints through multiplicities and path expressions, to say that: dogs, reptiles and cats must be hosted in their respective zones (the path expression starts from PDog and takes the inverse of Hosts to reach PetRoom, then it takes zone to reach dogs). In F_PW7 and F_PW8 the derived edges state that: (i) ensuite rooms are served by one toilet, (ii) privileged pets stay in ensuite rooms only, (iii) there are no ensuite rooms in the reptile zone, and (iv) disadvantaged or middle-class pets cannot stay in ensuite rooms.
- F_PW6 further illustrates proxies to show how they can be used to bring over information from other fragments in order to define new model concepts.
- F_PW7 illustrates *path edges* (pointed lines connected with an extra double arrow indicating an operator, either = or ⊆), another Fragmenta novelty to define constraints on edge paths through commuting, to say that zones of served room and toilet must be the same[6].

Elements of *M_PW* (Fig. 2, concrete) are mapped onto their *M_APW* (Fig. 1, abstract) counter-parts in Fig. 3; this mapping defines morphism *ms*. *M_PW* refines *M_APW* via *ms* as per the formula $(M\_PW, ms) \sqsupset^M M\_APW$, explained in the next section.

The graph of Fig. 4, an instance of Fig. 2, depicts a pets hotel with facilities and guests. The typing (or instance-of) compliance which exists between Figs. 4 and 2 can be written concisely as $G\_PW1 \ni^M M\_PW$ using the definitions of the next section; this compliance is valid as confirmed by Fragmenta's tool. A mutation of Fig. 4, say *G_PW2*, where (i) the rooms allocated to dog K and cat *G* are swapped, (ii) the capacity of room PR3 is changed from 20 to 0, and (iii) the month of L's birth is changed from 11 to 13, would be an invalid instance of Fig. 2, as: (i) ERND derived edge of fragment F_PW8 would be breached as disadvantaged K is staying in ensuite room PR1; (ii) derived edges S1 and S3 of fragment F_PW5 would be breached as K would be staying in a dog zone and G in a cat zone; (iii) the 0 capacity of room PR3 would breach the value-constraint edge of node $\mathtt{Nat}_1$ in fragment F_PW1; and (iv) month 13 of L's birth would breach the value constraint edge associated with relation edge month in fragment F_PW1. These issues would be flagged by the Fragmenta tool as reasons for not establishing that $G\_PW2 \ni^M M\_PW$.

## 3 THE FRAGMENTA THEORY

Fragmenta is formalised using algebraic graph theory, redefining and upgrading [7]. The foundations of the Fragmenta theory lie in plain graphs (section 3.1). From then the theory is enriched incrementally with structural graphs (SGs) in section 3.2, built on top of graphs, to represent class-like diagrams, widely used in software engineering, fragments (section 3.3), built on on top of SGs, global fragment graphs (GFGs, section 3.4), and models (section 3.5). The text refers to appendix A which contains all of Fragmenta's mathematical definitions that refer to theorems proved in Isabelle [63][7]. All the mathematics was specified in Z [5] and implemented in Haskell; all Fragmenta examples give herein were tested in the implementation[8].

[6]Here, each path has a different expression: from Toilet we take Serve and then zone to reach HotelZone, or we just take zone — the overall constraint states that both paths must lead to the same zone.
[7]https://isabelle.in.tum.de/
[8]Fragmenta's Z specification, Isabelle theory and Haskell implementation are all available from https://github.com/namalio/Fragmenta.

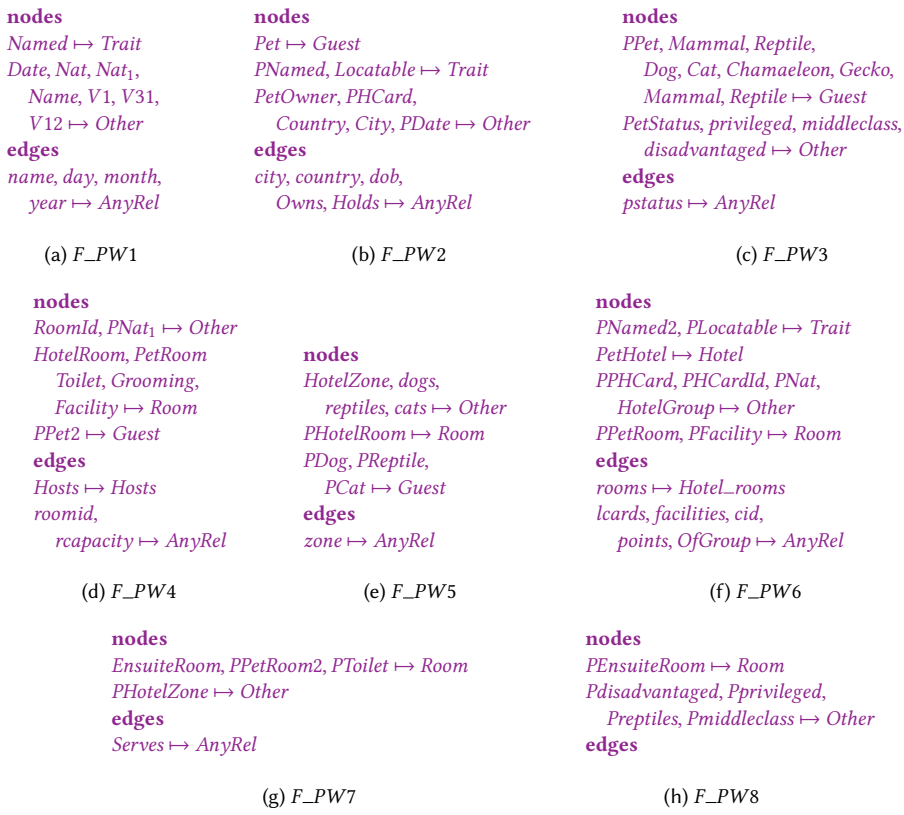



Fig. 3. Mappings for each fragment of the concrete model into the abstract model.

The sequel uses several operators related to sets (def. 1), relations (def. 2) and functions (def. 4). An abbreviated function application convention is used throughout — e.g. $Ns_G$ means $Ns\,G$ or $Ns(G)$. Certain infix binary operators are either sub- or super-scripted with a label indicating the structure they operate upon — e.g. $\cup_G$ is graph union (G refers to graphs).

### 3.1 Graphs and their morphisms

Sets $V$ and $E$ represent all possible graph nodes and edges (def. 5). A graph $G$ (Fig. 5a), a member of set $Gr$ (def. 7), written $G \in Gr$ or $G : Gr$, is a quadruple $G = (N, A, s, t)$, made up of sets $N \subseteq V$ and $A \subseteq E$ of nodes and edges (or arcs), respectively, and total functions $s, t : A \to N$ which give the source and target nodes of edges, respectively. Projection functions $Ns$, $Es$, $src$ and $tgt$ extract the individual components of a graph, $Els$ gives a pair formed by the graph's nodes and edges, and $\varnothing_G$ gives the empty graph (def. 7).

The following handle graphs (def. 8): (i) $adjacent(G, v_1, v_2)$ says whether node $v_1$ is adjacent to $v_2$ in graph $G$; (ii) $adjacent_E(G, e_1, e_2)$ says whether edge $e_1$ is adjacent to $e_2$; (iii) $G \bowtie_{Es} es$ restricts $G$ to set of edges $es$; (iv) $G \multimap vs$ gives incident edges of nodes $vs$; (i) $G \bullet\!\!\to\!\!\bullet\ vs$ gives the edges connecting the nodes $vs$; (vi) $G^{\leftrightarrow}$ builds a mathematical relation between the nodes connected in $G$; (vii) $G_1 \cup_G G_2$ gives union of $G_1$ and $G_2$; (viii) $G \odot s$ subsumes $G$ as per replacement

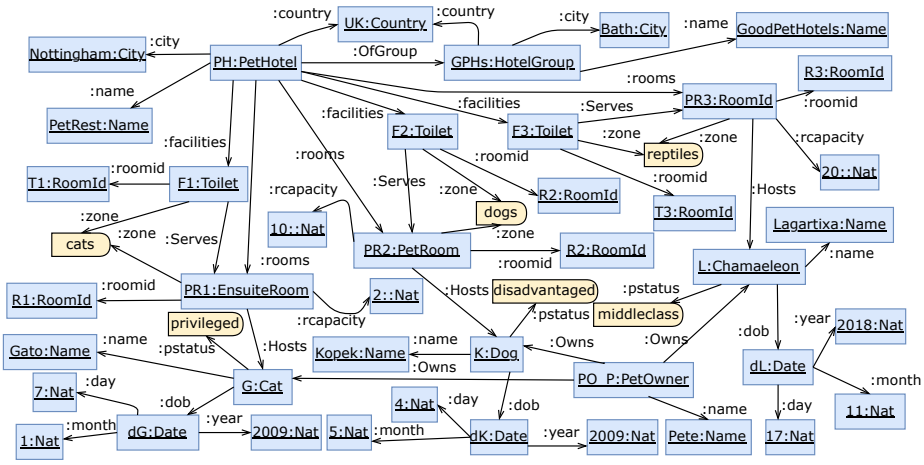
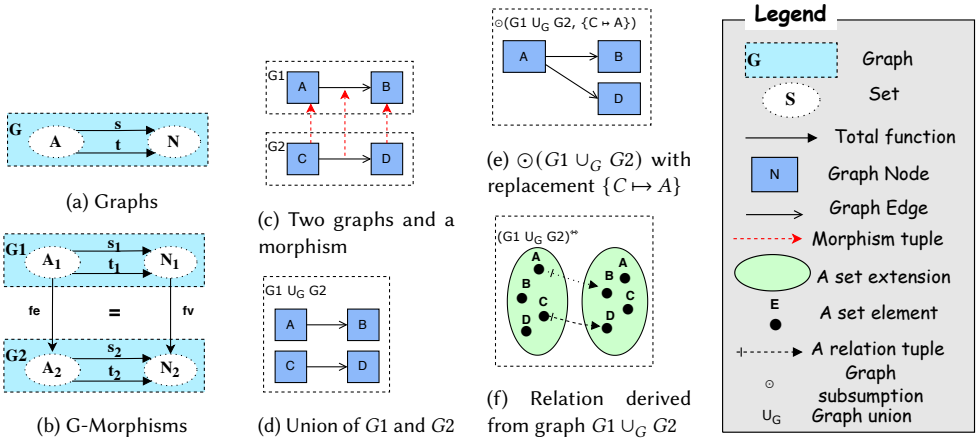



Fig. 4. An instance of the pets world fragmented model of Fig. 2.




(a) Graphs

(b) G-Morphisms

(c) Two graphs and a morphism

(d) Union of $G1$ and $G2$

(e) $\odot(G1 \cup_G G2)$ with replacement $\{C \mapsto A\}$

(f) Relation derived from graph $G1 \cup_G G2$

Fig. 5. Graphs and their morphisms

mapping $s$; (ix) $\oslash\ G$ checks whether $G$ is acyclic; and (x) $\boxminus(G_1, G_2)$ checks disjointness of $G1$ and $G_2$. Disjoint graphs G1 and G2 of Fig. 5c, that is $\boxminus(G1, G2)$, are joined in Fig. 5d as $G1 \cup_G G2$; Fig. 5e pictures $G1 \cup_G G2 \odot \{C \mapsto A\}$ a subsumption of Fig. 5d's graph with $C$ subsumed by $A$; Venn diagram of Fig. 5f portrays $(G1 \cup_G G2)^{\leftrightarrow}$, the relation derived from Fig. 5d. Laws of def. 8 establish proved facts about these definitions — e.g. $G \cup_G \varnothing_G = G$ and $G \bowtie_{Es} \varnothing = \varnothing_G$.

A graph (or a G-) morphism $f : G_1 \longrightarrow G_2$ (Fig. 5b), a member of set $GrM$ (def. 9), defines a pair of functions $f = (fv, fe)$, where $fv : Ns_{G1} \longrightarrow Ns_{G2}$ and $fe : Es_{G1} \longrightarrow Es_{G2}$ map G1's nodes and edges onto G2's, respectively, preserving the source and target functions — as depicted in Fig. 5b, where $fv \circ s_1 = s_2 \circ fe$ and $fv \circ t_1 = t_2 \circ fe$. Projection functions $f_V$ and $f_E$ extract the mappings for nodes and edges of a morphism, respectively (def. 9); functions *domg*, and *codg* yield the domain and codomain of a graph morphism, respectively (def. 9). Graph morphisms $f : G1 \longrightarrow G2$ and $g : G2 \longrightarrow G3$ can

be composed: $g \circ f = (f_V\ g \circ f_V\ f, f_E\ g \circ f_E\ f)$ (def. 10). Figure 5c pictures a graph morphism. Graph morphisms are partial when the mapping functions are partial rather than total (def. 9)[9].

A graph with typing, a member of set *GrwT* (def. 11), is a pair $GwT = (G, m)$ made up of a graph $G : Gr$ and a morphism $m : GrM$, such that $domg\ \ m =_p Els\ G$ (def. 11); such graphs have an empty element ($\varnothing_{Gwt}$) and a union operator ($\cup_{GwT}$) (def. 11). Graphs with extra typing, set *GrwET* (def. 12), are another type of graph: a pair $GwET = (GwT, m)$, where $GwT : GrwT$ and $m$ is a partial morphism, providing extra typing, a concept elaborated in the sequel.

### 3.2 SGs

SGs capture graphs typical of object-oriented (OO) class modelling notations, such as UML class diagrams, which typically include: (i) relations with multiplicities, (ii) inheritance, (iii) and part-of (or composition) relations. SGs embody many of Fragmenta's novelties, namely: (i) proxies (a representative of another node), (ii) virtuals (an inheritable characteristic passed with virtual or soft parenthood allowing multiple inheritance), (iii) value-constraint edges (a way to restrict the values of a node or edge through an operator, such as =), (iv) derived edges (multiplicity constraints on edge paths), and (v) path constraints (commuting constraints on edge paths). SGs can represent templates (or generics) using a combination of virtuals and derived edges (see below).

*3.2.1 Node and Edge Types.* SGs support a diversity of nodes and edges. Set *SGNT* (def. 13), of SG's node types, include: normal (*nnrml*), abstract (*nabst*), proxy (*nprxy*), enumeration (*nenum*), value (*nval*) and virtual (*nvirt*). Set *SGET* (def. 13), of edge types, includes: inheritance (*einh*), composition (*ecomp*), relation (*erel*), derived (*eder*), value constraint (*evcnt*) and path (*epath*). Compositions and relations are either bi-directional (*dbi*) or one-way (*duni*) (def. 13).

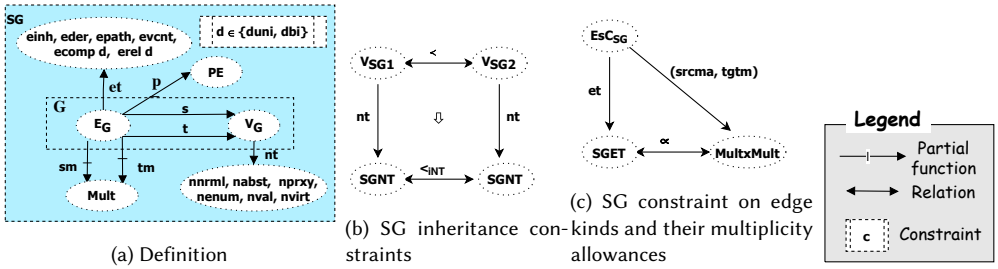
Ordering relation $\prec_{NT}$ (def. 13) stipulates the inheritance restrictions of node types. Orderings $\leq_{rNT}$ and $\leq_{ET}$ (def. 13) stipulate the node and edge types which are refinement-relatable, respectively.

*3.2.2 Multiplicities.* Multiplicity values (set *MultVal* of def. 14) are ordered through $\leq_{mv}$ (e.g. $1 \leq_{mv} *$, 1 less or equal than many). Ordering $\leq_{\mathcal{M}c}$ establishes whether one multiplicity compound (a member of set *MultC* of def. 14) is within another (e.g. $* \leq_{\mathcal{M}c} 0\,..\,*$, and $1 \leq_{\mathcal{M}c} 1\,..\,*$ ). Relation $\propto$ (def. 14) indicates the allowed multiplicities of edge types; for instance: two-way relations can have any multiplicity, and one-way compositions may have source multiplicities of 1 or 0 . . 1.

*3.2.3 Path Expressions (PEs).* PEs (def. 15) describe edge traversals as edge-based paths. They are part of derived and path edges, and describe extra multiplicity and commuting constraints, respectively. PEs are made up of path expression atoms (PEAs), either: a particular edge or its reverse (e.g. from edge $e$ we get PEAs $e$ and $e^{-1}$). A path expression *PE*, defined recursively, can be either a PEA, a PEA restricted on the domain or range to a particular node through $\lhd_{PE}$ or $\rhd_{PE}$, respectively, and one PE followed by another through $\fatsemi_{PE}$. Edges $e_1$ and $e_2$ may result in the PEs: $e_1$, $e_1^{-1}$, and $e_1 \fatsemi_{PE} e_2$.

*3.2.4 Base SGs.* An SG (depicted in Fig. 6a), a member of $SGr_0$, the set of base SGs (def. 17), is a tuple ($G$, *nt*, *et*, *sm*, *tm*, *pe*, *ds*) comprising: (a) graph $G : Gr$ (see above, def. 7); (b) colouring functions $nt : Ns\ G \rightarrow SGNT$ and $et : Es\ G \rightarrow SGET$, giving kinds of nodes and edges, respectively; (c) partial functions $sm, tm : Es\ G \rightarrow Mult$, assigning multiplicities to source and target of edges, respectively; (d) partial function $p : Es\ G \nrightarrow PE$, giving PE of either a derived or path edge; and (e) relation $d : Es\ G \leftrightarrow Es\ G$, giving dependencies between path edges. Figures 7a to 7g give examples of SGs. Functions *gr*,

[9] A partial morphism can be built from graph of Fig. 5d into $G1$ of Fig. 5c by mapping C, D and their connecting edge onto $G1$'s A, B and their connecting edge, respectively — A and B and the edge that connects them would not be mapped.




(a) Definition (b) SG inheritance constraints (c) SG constraint on edge kinds and their multiplicity allowances

Fig. 6. Structural graph definition

*nty*, *ety*, *srcm*, *tgtm*, *pe* and *ds* extract the different SG components — for instance, $gr : SGr_0 \longrightarrow Gr$ yields SG's graph. Graph functions *Ns*, *Es*, *src* and *tgt* (section 3.1, def. 7) also apply to SGs (def. 17) — for instance, for any $SG : SGr_0$, $Ns\ SG$ is same as $Ns(gr\ SG)$—, $\varnothing_{SG}$ is the empty SG (def. 17), and hence: $gr\ \varnothing_{SG} = \varnothing_G$, $nty\ \varnothing_{SG} = \varnothing$ and $Ns\ \varnothing_{SG} = \varnothing$.

Functions that extract information from SGs include (def. 17): (i) *NsTy SG nks* gets all nodes of *SG* of set of kinds *nks*; (ii) *EsTy SG eks* gets all edges of set of kinds *eks*. These are used to define relevant sets, such as $NsP = NsTy\ SG\ \{nprxy\}$ which yields proxies, similarly for *NsEther* (kinds virtual, abstract and enumeration). Sets *EsA*, *EsD*, *EsM* and *EsI* of all association (includes composition and relation), derived, multiplicity (includes association and derived), and inheritance edges, respectively, are defined from *EsTy* — e.g. $EsI\ SG = EsTy\ SG\ \{einh\}$.

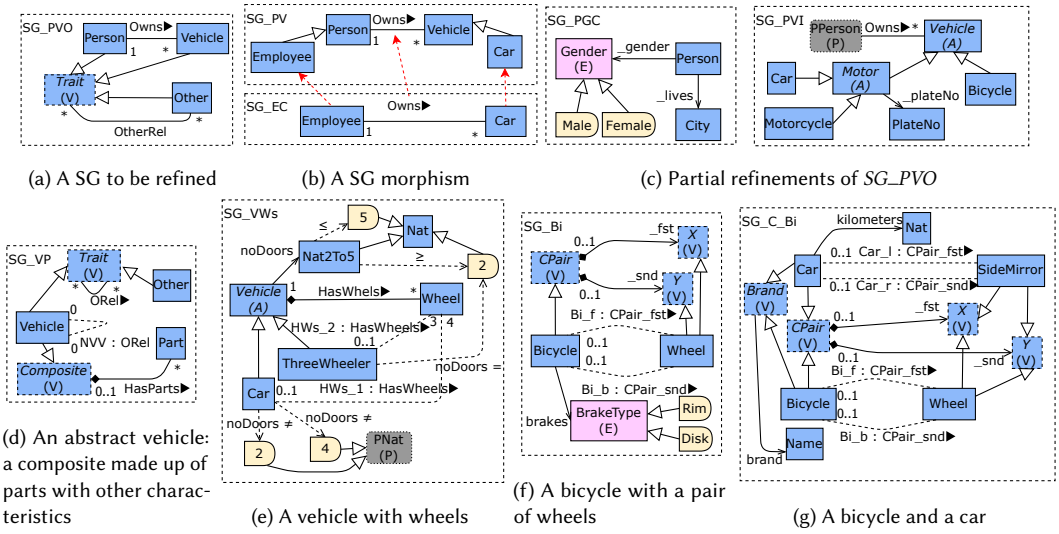
Inheritance hierarchies are defined from inheritance edges. The inheritance graph is obtained through operator $\Uparrow : SGr_0 \longrightarrow Gr$ (def. 17) — $\Uparrow\ SG = gr\ SG \bowtie_{Es} EsI\ SG$. The inheritance relation, $\prec : SGr_0 \longrightarrow V \leftrightarrow V$ (def. 17), obtained from the inheritance graph — $\prec\ SG = (\Uparrow\ SG)^{\leftrightarrow}$ —, captures parenthood. In Fig. 7a, $Person \prec Trait$, $Other \prec Trait$ and $Vehicle \prec Trait$.

*3.2.5 Partial and Total SGs.* SGs either stand alone, or are part of something larger. They are *partial* when part of a fragmented whole, *total* when monolithic or self-contained. Set *SGr* of partial SGs (def. 19) requires that: (i) multiplicity edges have source and target multiplicities, (ii) relation *ds* (dependencies) between path edges is anti-reflexive, (iii) function *pe* gives path edges of derived and path edges only, (iv) paths must be made of SG's association edges only, (v) edges comply to the multiplicity restrictions of their types (portrayed in Fig. 6c), (vi) the inheritance is well-formed, implying that the inheritance relation ($\prec$ SG) complies with the required inheritance restrictions associated with the types of the related nodes (portrayed in Fig. 6b)[10], and the inheritance is acyclic.

Total SGs, set *TSGr* (def. 20), are partial SGs requiring that: (i) ethereal nodes (enumeration, abstract and virtual) are inherited; (ii) derived and path edges are well-formed (the nodes linked by the derived edges are consistent with the path expression and the encapsulated path expression is valid); and (iii) the inheritance hierarchy without virtual nodes is a tree (apart from virtual nodes, the inheritance is single, meaning that a node has at most one non-virtual or strong parent).

SGs of Fig. 7 illustrate different aspects of SGs: inheritance, virtuals and the *virtual traits* pattern (Figs. 7a and 7d); SG morphisms (Fig. 7b); enumerations, proxies and abstract nodes (Fig. 7c); compositions (Figs. 7d to 7g); derived and value constraint edges (Figs. 7d to 7g); the *templates* pattern where a pair generic is instantiated through a combination of virtual nodes and derived edges (Figs. 7f and 7g); and the *flexible edges* pattern (Fig. 7d): a unary (or self) edge on a virtual allowing multiple configurations, here constrained to preclude unary edges on Vehicle.

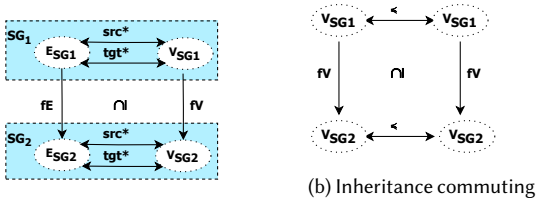
[10] $\forall\ v_1, v_2 \bullet v_1\ (\prec\ SG)\ v_2 \Rightarrow (nty\ SG)\ v_1\ \prec_{NT}\ (nty\ SG)\ v_2$.




(a) A SG to be refined (b) A SG morphism (c) Partial refinements of *SG_PVO*

(d) An abstract vehicle: a composite made up of parts with other characteristics (e) A vehicle with wheels (f) A bicycle with a pair of wheels (g) A bicycle and a car

Fig. 7. Examples of structural graphs (SGs), illustrating: virtuals and the virtual traits pattern (a and d), SG morphisms (b), enumerations, proxies and abstract nodes (c), compositions (d to g), derived and value constraint edges (d to g), and the templates pattern (f and g)




(a) Commuting of $src^*$ and $tgt^*$ (b) Inheritance commuting (relation $\preceq$)

Fig. 8. SG-morphisms

Operator $\preceq$ (def. 18), the reflexive and transitive closure of the base inheritance relation — $\preceq\ SG = (\prec\ SG)^*$ — defines a partial order. In SG_PV of Fig. 7b: *Employee* $\preceq$ *Person* $\preceq$ *Person*. Functions $src^*, tgt^* : E \leftrightarrow V$ (def. 18) extend the graph functions *src* and *tgt* of def. 7 to give the source and target relations between edges and vertices — in an inheritance setting edges may possess more than one source and target nodes as descendants inherit the association edges of their ancestors. In SG_PV of Fig. 7b, Owns is an edge of both Employee and Car — $(Owns, Employee) \in src^*_{SG\_PV}$ and $(Owns, Car) \in tgt^*_{SG\_PV}$.

SGs are joined either through union ($\cup_{SG}$ , def. 21), or subsumption along proxies ($\odot^{SG}$, def. 21). Figure 7e's subsumption would eliminate PNat and redirect its incoming inheritance edges to Nat.

*3.2.6 SG morphisms.* These cater for the specificities of SGs. Because $src^*$ and $tgt^*$ yield relations, the equality commuting (Fig. 5b) is replaced by subset commuting (Fig. 8a). A SG morphism (def. 22, Fig. 8a) $m = (fv, fe)$ between $SG_1, SG_2 : SGr$, written $m \in SG_1 \rightarrow_{SG} SG_2$, where $fv \in Ns\,SG_1 \rightarrow Ns\,SG_2$ and $fe \in EsA\,SG_1 \rightarrow EsA\,SG_2$ (includes association and excludes inheritance edges), requires that:

$$fv \circ src^*\ SG_1 \subseteq src^*\ SG_2 \circ fe \wedge fv \circ tgt^*\ SG_1 \subseteq tgt^*\ SG_2 \circ fe \wedge fv \circ \preceq_{SG_1} \subseteq \preceq_{SG_2} \circ fv$$

Above, the first two conjuncts establish the commuting and the third inheritance-monotonicity (depicted in Fig. 8b). Partial SG morphisms, written $m \in SG_1 \nrightarrow_{SG} SG_2$, are also allowed (def 22).

SGs of Figs. 7a and 7b are related through morphisms: $m_1 : SG\_PV \rightarrow_{SG} SG\_PVO$, $m_2 : SG\_EC \rightarrow_{SG} SG\_PVO$, $m_3 : SG\_EC \rightarrow_{SG} SG\_PV$ (depicted in Fig. 7b), $m_4 : SG\_PV \rightarrow_{SG} SG\_EC$:

$$m_1 = (\{Person, Employee \mapsto Person, Vehicle, Car \mapsto Vehicle\}, \{Owns \mapsto Owns\})$$
$$m_2 = (\{Employee \mapsto Person, Car \mapsto Vehicle\}, \{Owns \mapsto Owns\})$$
$$m_3 = (\{Employee \mapsto Employee, Car \mapsto Car\}, \{Owns \mapsto Owns\})$$
$$m_4 = (\{Person, Employee \mapsto Employee, Vehicle, Car \mapsto Car\}, \{Owns \mapsto Owns\})$$

SGs of Figs. 7c and 7a are related through $m_5 : SG\_PGC \rightarrow_{SG} SG\_PVO$, $m_6 : SG\_PVI \rightarrow_{SG} SG\_PVO$, the partial $m_7 : SG\_PVI \nrightarrow_{SG} SG\_PVO$ and $m_8 : SG\_PGC \cup_{SG} SG\_PVI \rightarrow_{SG} SG\_PVO$:

$$m_5 = (\{Person \mapsto Person, Gender, City, Male, Female \mapsto Other\},$$
$$\{Person\_gender, Person\_lives \mapsto OtherRel\})$$
$$m_6 = (\{PPerson \mapsto Person, Vehicle, Motor, Bicycle, Motorcycle \mapsto Vehicle, PlateNo \mapsto Other\},$$
$$\{Owns \mapsto Owns, Motor\_plateNo \mapsto OtherRel\})$$
$$m_7 = (\{PPerson \mapsto Person\}, \{Owns \mapsto Owns\})$$
$$m_8 = m_5 \cup_{GM} m_6$$

Finally, SGs of Figs. 7d to 7g are related through $m_9 : SG\_VWs \rightarrow_{SG} SG\_VP$, $m_{10} : SG\_Bi \rightarrow_{SG} SG\_VP$, $m_{11} : SG\_C\_Bi \rightarrow_{SG} SG\_VP$:

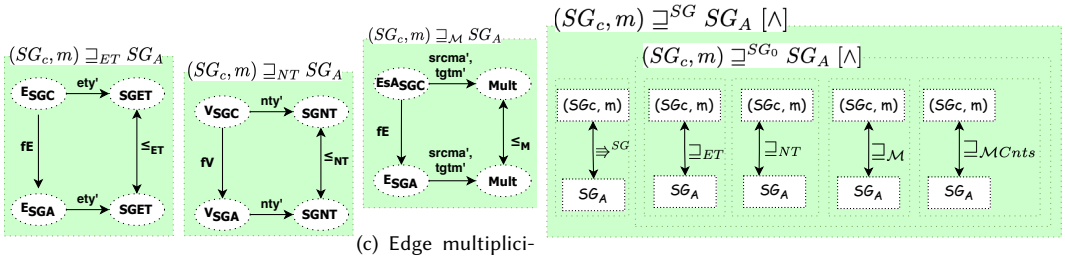
$$m_9 = (\{Vehicle \mapsto Vehicle, Car \mapsto Vehicle, ThreeWheeler \mapsto Vehicle, Wheel \mapsto Part\},$$
$$\{HasWheels \mapsto HasParts\})$$
$$m_{10} = (\{CPair \mapsto Composite, Wheel \mapsto Part, Bicycle \mapsto Vehicle, X \mapsto Part, Y \mapsto Part\},$$
$$\{CPair\_fst \mapsto HasParts, CPair\_snd \mapsto HasParts\})$$
$$m_{11} = (\{CPair \mapsto Composite, Wheel \mapsto Part, Bicycle \mapsto Vehicle, Car \mapsto Vehicle,$$
$$SideMirror \mapsto Part, X \mapsto Part, Y \mapsto Part\}, \{CPair\_fst \mapsto HasParts, CPair\_snd \mapsto HasParts\})$$

The Fragmenta tool deemed the validity of all ten morphisms above.

*3.2.7 Refinement.* SG vertical refinement, Fragmenta's multilevel modelling underpinning, is built on top of SG morphisms, used to relate elements of refined (or concrete) and abstract SGs. Fragmenta's refinement principles are: (i) the abstract model sets a base model which refinements are required to comply to; (ii) refinements add detail and are allowed to restrict the abstract model. A subset relation underpins Fragmenta refinement; the abstract SG defines a multitude of information structures and their relations, of which refinements are subsets. This can be stated mathematically; given a function $\mathcal{M}$, which yields the meaning of a SG as the set of all possible SG instances, and concrete and abstract SGs, $SG_c$ and $SG_a$, respectively, and a morphism $f$, then $SG_c$ refines $SG_a$ whenever $\mathcal{M} \circ f(SG_c) \subseteq \mathcal{M}(SG_a)$.

The two facets of Fragmenta models, decomposed (or fragmented) and composed, yield two refinement notions: *partial* and *total*. Partial refinement applies to fragments or partial SGs. Total refinement applies to monolithic or composed models with one sole fragment.

A concrete $SG_c : SGr$ partially refines a more abstract $SG_a : SGr$ via a morphism $m : GrM$, written $(SG_c, m) \sqsupseteq^{SG} SG_a$ (def. 23), whenever the conditions depicted in Fig. 9 (summarised in Fig. 9d) are met: (i) $m$ is a valid morphism —




(a) Edge types (b) Node types (c) Edge multiplicities (d) Partial SG Refinement

Fig. 9. Partial refinement of SGs ($\sqsupseteq_{\mathcal{M}Cnts}$ is not described graphically).

$m \in SG_c \rightarrow_{SG} SG_a$ or $(SG_c, m) \Rrightarrow^{SG} SG_a$[11]; (ii) $SG_c$ edges are refinement-compatible with their corresponding $SG_a$ edges as per $\leq_{ET}$ of def. 13 (Fig. 9a); (iii) $SG_c$ nodes are refinement-compatible with their $SG_a$ counter-parts as per $\leq_{rNT}$ of def. 13 (Fig. 9b); and (iv) multiplicities of $SG_c$'s association edges comply with $SG_a$'s to ensure that refined multiplicities (a) can only be narrowed (Fig. 9c), and (b) comply to the multiplicities of $SG_a$'s derived edges.

Total refinement (def. 26), written $(SG_c, m) \sqsupset^{SG} SG_a$, requires that: (i) $m$ is a valid SG morphism, (ii) the conditions of partial refinement are met — $(SGc, m) \sqsupseteq^{SG_0} SGa$[12] —, (iii) $SG_a$'s association edges are refined adequately in $SG_c$ — $(SG_c, m) \sqsupset_{AEs} SG_a$ —, and (iv) all normal nodes are refined — $m \sqsupset_{ANNs} SG_a$. Partial refinement is weaker than total refinement: $(SG_c, m) \sqsupset^{SG} SG_a \Rightarrow (SG_c, m) \sqsupseteq^{SG} SG_a$.

The following refinement results were confirmed in Fragmenta's tool:

- In *SG_PVO* (Fig. 7a), and *SG_EC* and *SG_PV* (Fig. 7b), given morphisms $m_1$, $m_2$, $m_3$ and $m_4$ defined above: $(SG\_PV, m_1) \sqsupseteq^{SG} SG\_PVO$ and $(SG\_EC, m_2) \sqsupseteq^{SG} SG\_PVO$ — both partial refinements only because normal node `Other` is not mapped to —, $(SG\_EC, m_3) \sqsupseteq^{SG} SG\_PVO$ — only partial because normal nodes `Person` and `Vehicle` are not mapped to (it would be total if these nodes were abstract in *SG_PV*) —, and $(SG\_PV, m_4) \sqsupset^{SG} SG\_EC$.
- *SG_PGC* and *SG_PVI* (Fig. 7c) partially refine *SG_PVO* (Fig. 7a): $(SG\_PGC, m_5) \sqsupseteq^{SG} SG\_PVO$ and $(SG\_PVI, m_6) \sqsupseteq^{SG} SG\_PVO$, but their union (Fig. 7c) is a total refinement: $(SG\_PGC \cup_{SG} SG\_PVI, m_7) \sqsupset^{SG} SG\_PVO$.
- In Figs. 7d to 7g, we have that $(SG\_VWs, m_8)$, $(SG\_Bi, m_9)$, $(SG\_C\_Bi, m_{10}) \sqsupset^{SG} SG\_VP$. SG_C_Bi (Fig. 7g) depicts a valid refinement of SG_VP (Fig. 7d), adding an association between `Bicycle` and `Car` would invalidate the refinement.

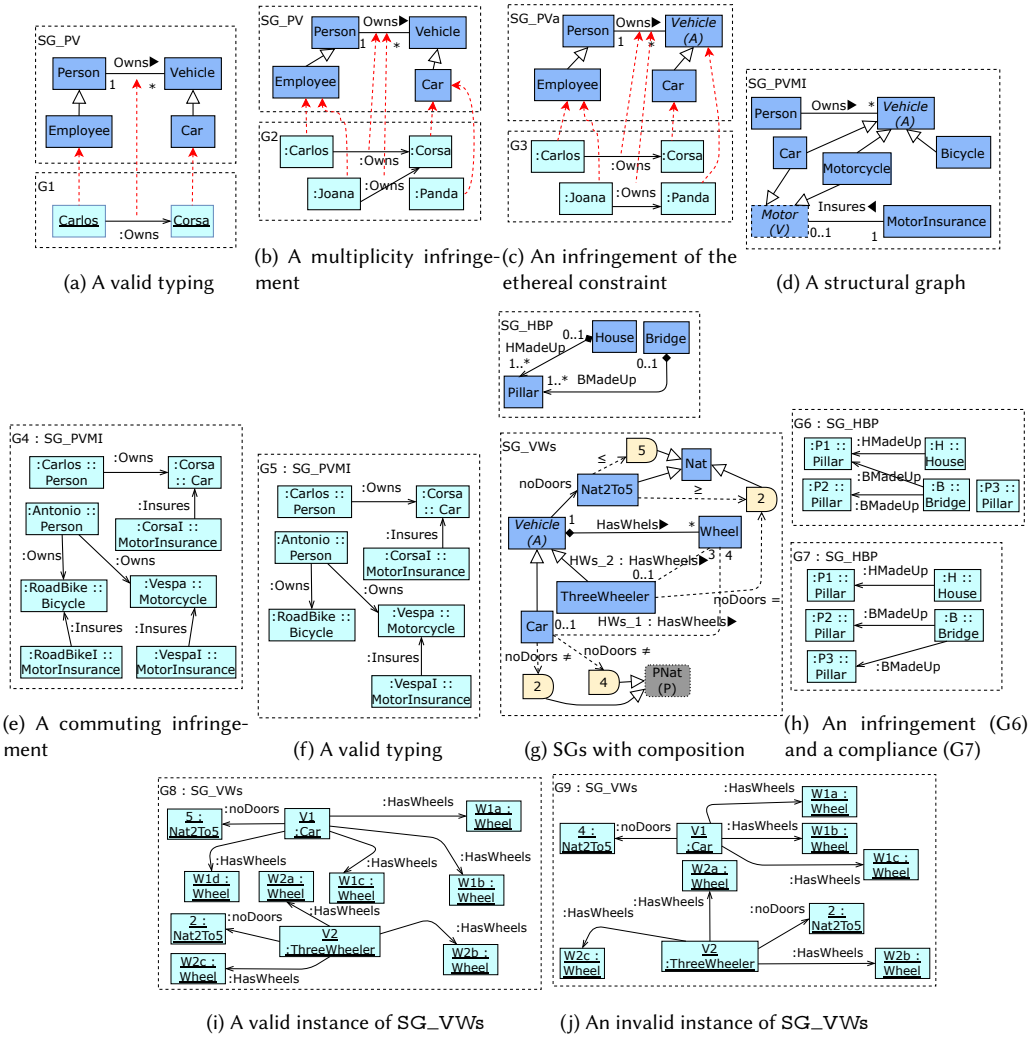
*3.2.8 Typing.* Concerned with how instances conform to their types, typing, like refinement, builds upon graph morphisms to map instances to their types. Figures 10a to 10c provide examples of graphs typed by SGs.

A morphism $m : GrM$ from a graph $G : Gr$ to a $SG : SGr$, written $m \in G \rightarrow_{G2SG} SG$, is a pair of functions $m = (fv, fe)$ such that (def. 28): (i) $fv \in Ns_G \rightarrow Ns_{SG}$, (ii) $fe \in Es_G \rightarrow EsA_{SG}$ (only SG association edges are allowed as edge types), (iii) $fv \circ src_G \subseteq src^*{}_{SG} \circ fe$, and (iv) $fv \circ tgt_G \subseteq tgt^*{}_{SG} \circ fe$. The last two equations recast base graph commuting (depicted in Fig. 5b) into subset commuting to account for inheritance in the SG. All G-to-SG morphisms of Fig. 10 are valid (confirmed by the Fragmenta tool), except Fig. 10e, which breaches commuting; not all of them are well-typed.

A graph with typing $GwT : GrwT$ (def. 11), a pair made-up of graph $G : Gr$ and graph morphism $m : GrM$, complies with its type $SG : SGr$, written $GwT \ni^{SG} SG$ (def. 30), provided: (i) $m \in G \rightarrow_{G2SG} SG$ (or $GwT \Rrightarrow^{GwT} SG$); (ii) $GwT$

[11] Relation $\Rrightarrow^{SG}$ (def. 22) says that two SGs are related through a morphism: $(SG_c, m) \Rrightarrow^{SG} SG_a \Leftrightarrow m \in SG_c \rightarrow_{SG} SG_a$

[12] Relation $\sqsupseteq^{SG_0}$ (def. 23) captures the conditions associated with partial refinement (see above).




(a) A valid typing (b) A multiplicity infringement (c) An infringement of the ethereal constraint (d) A structural graph

(e) A commuting infringement (f) A valid typing (g) SGs with composition (h) An infringement (G6) and a compliance (G7)

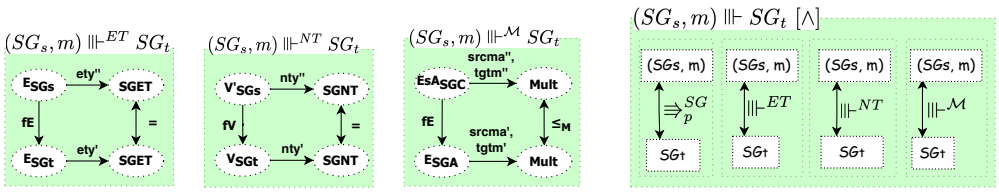
(i) A valid instance of SG_VWs (j) An invalid instance of SG_VWs

Fig. 10. Graphs typed by SGs

complies with the multiplicity constraints of its type (GwT $\ni_{\mathcal{M}}$ SG); (iii) *GwT* does not contain instances of ethereal nodes (either abstract, virtual or enumeration), written *GwT* $\ni_{FI}$ *SG*; and (iv) *GwT* lacks parts shared among instances of the same compound (*GwT* $\ni_{PNS}$ *SG*).

Typed graphs of Fig. 10 have the following correctness results (checked by FRAGMENTA's tool):

- *G*1 in Fig. 10a is well-typed. Fig. 10b's *G*2 breaches multiplicity twice: (i) `Corsa` has two owners, (ii) `Panda` has none. *G*3 in Fig. 10c includes an instance of an abstract SG node.
- *G*4 in Fig. 10e is a mistyping of Fig. 10d — commuting is infringed, bicycles cannot have motor insurances. Figure 10f's *G*5, on the other hand, is a valid instance.




(a) Edge types (b) Node types (c) Edge multiplicities (d) Overall extra typing compatibility

Fig. 11. Extra typing compatibility at the level of SGs.

- In Fig. 10h, G6 is a mistype of SG_HBP (Fig. 10g) — composition is infringed as node P1 is shared by compounds H and B. G7, on the other hand, is well-typed.
- Figure 10i is a valid type of SG_VWs of Fig. 10g, unlike Fig. 10j which infringes: the multiplicity constraint of the derived edge (a car must have four wheels) and the value constraint edge of Car (a car must not have 4 doors).

*3.2.9 Extra Typing.* Certain situations call for extra typing, where a portion of some graph is required to comply with a portion of another graph. An extra typing could be built by considering that SG_PVMI (Fig.10d) would have extra typing into SG_PV (Fig.10a) by mapping the nodes Car and Person and edge Owns into Car, Employee and Owns, respectively, effectively saying that instances of Person in SG_PVMI are instances of Employee in SG_PV.

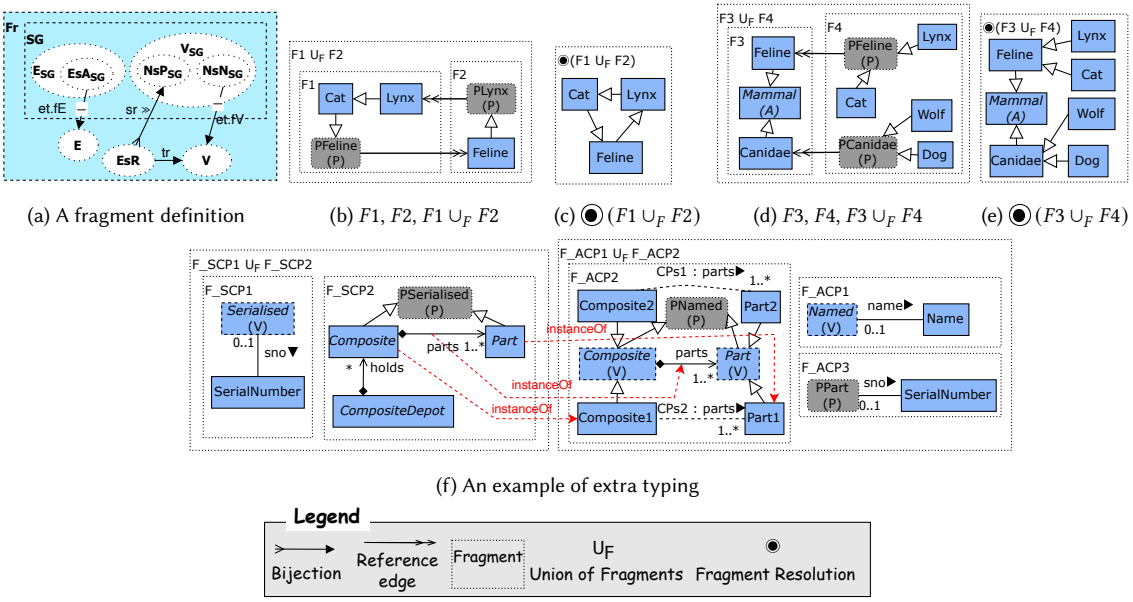
This is handled by extra type compliance and the associated notion of extra-type compatibility. Given SGs $SG_1, SG_2 : SGr$, $SG_1$ is extra-type compatible with $SG_2$ via the morphism $m$, written $(SG_1, m) \dashv\vdash^{SG} SG_2$ (def 31), provided several conditions, described in Fig. 11 (summarised in Fig. 11d), are met: (i) $m$ is a partial morphism from $SG_1$ to $SG_2$, (ii) the multiplicities of $SG_1$'s mapped edges comply to the multiplicities of corresponding $SG_2$'s edges (source multiplicities can only be narrowed) (Fig. 11c), (iii) the types of $SG_1$'s mapped nodes and edges are the same as their $SG_2$ counterparts (Figs. 11a and 11b). The next section develops extra typing further.

### 3.3 Fragments

Fragments provide referencing, either intra- or inter-fragments, by stipulating the nodes proxies refer to (or represent), called referents. In addition, fragments may stipulate the extra typing of certain elements which may be allowed to have types in another graph outside the stipulated metamodel.

A fragment (Fig. 12a, def. 32) is a quintuple $F = (SG, esr, sr, tr, et)$, comprising: (i) a $SG : SGr$ (def. 17), (ii) a set of reference edges, $esr \subseteq E$, disjoint from the edges of the inner SG — $esr \cap Es\,SG = \varnothing$ —, (iii) functions $sr : esr \rightarrowtail\!\!\!\!\twoheadrightarrow NsP\,SG$ (bijective) and (iv) $tr : esr \rightarrow V$, which give source and target of reference edges, respectively, where $tr$ gives nodes proxies refer to, and (v) extra-typing partial morphism $et$, where $domg\,et \subseteq_p (NsN\,SG, EsA\,SG)$. Figures 12b to 12f illustrate fragments, with referencing depicted as double-arrows; in Fig. 12b, proxy PFeline refers to its homonym.

Projection functions $fSG$, $EsR$, $srcR$, $tgtR$ and $fet$ (def. 32) extract the inner elements of a fragment — e.g. $fSG(SG, esr, sr, tr, et) = SG$. Other fragment functions include (def. 32): (i) $fEs\,F$ yields all edges of fragment $F$ (all of SG plus reference edges), (ii) $fLNs\,F$ gives local nodes of $F$ (those of embedded SG), (iii) $fNs_F$ gives all nodes involved in $F$ (local plus foreign nodes), and (iv) $srcF$ and $tgtF$ give the source and target functions which cater for all edges, including reference edges. In Fig. 12b, $FLNs\,F1$ yields all nodes included in the fragment; $FNs\,F_1$ also yields *Feline* of $F_2$. Function $\overset{G}{\leftrightsquigarrow}$ (def. 32) builds




(a) A fragment definition (b) $F1$, $F2$, $F1 \cup_F F2$ (c) ◉$(F1 \cup_F F2)$ (d) $F3$, $F4$, $F3 \cup_F F4$ (e) ◉$(F3 \cup_F F4)$

(f) An example of extra typing

Fig. 12. Fragment-related drawings depicting the mathematical definition (a), and several illustrations, namely: a circular fragmentation (b) and its resolution (c), an healthy fragmentation (d) with its inheritance-acyclic resolution (e), two fragment collections related with an extra typing (f)

the references graph made-up of proxies connected to their referents, the basis of $\rightsquigarrow$ (def. 32), a function that gives the referent of a given proxy — in Fig. 12d, $\rightsquigarrow_{F_3 \cup_F F4}$ $PFeline = Feline$.

Fragment union $\cup_F$ (def. 33), illustrated in Figs. 12b and 12d with fragments $F_1 \cup_F F_2$ and $F_3 \cup_F F_4$, respectively, joins the contents of fragments. Fragment resolution ◉ (def. 34), founded on graph subsumption (defined for graphs in section 3.1 and SGs in section 3.2), replaces proxies by their referents. Figure 12c gives ◉$(F_1 \cup_F F_2)$ which resolves the fragment of Fig. 12b; Fig. 12e depicts ◉$(F_3 \cup_F F_4)$ which resolves the fragment of Fig. 12d.

Fragments are grounded on the following definitions:

- Set $Fr_a$ (def. 35), of fragments with acyclic references. All examples of Fig. 12 are reference-acyclic.
- Set of partial fragments $Fr$ includes all reference-acyclic fragments whose resolved encapsulated $SGs$ are partial, which implies that inheritance graphs of resolved SGs are acyclic (def. 35). In Fig. 12b, although $F_1 \in Fr$ and $F_2 \in Fr$, we have that $F_1 \cup_F F_2 \notin Fr$ — ◉$(F_1 \cup_F F_2)$ (depicted in Fig. 12c) has an inheritance cycle.
- Set of total fragments $TFr$ includes all acyclic fragments whose internally resolved SGs are total (def. 36). The individual component fragments $F_3$ and $F_4$ of Fig. 12d are partial; $F_3 \cup_F F_4$ is total because (i) all proxy references are local and (ii) the resolved SG (Fig. 12e) is total.

An important law proved in [7], says that the union of two partial fragments is well-formed provided: (i) the individual fragments are well-formed, (ii) mutually disjoint and (iii) the references between them only go one way (def. 35). In Fig. 12b, the two individual fragments are well-formed, but not compliant to the union law as they reference each other. The two fragments of Fig. 12d are both well-formed and compliant to the union law.

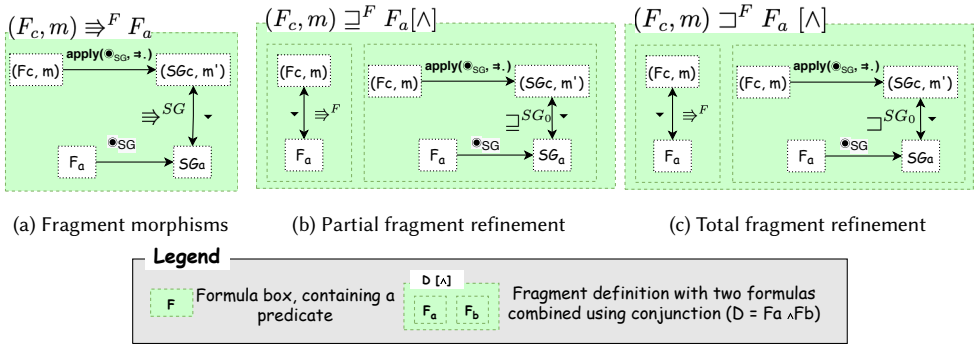



Fig. 13. Definitions of morphisms and refinements for fragments

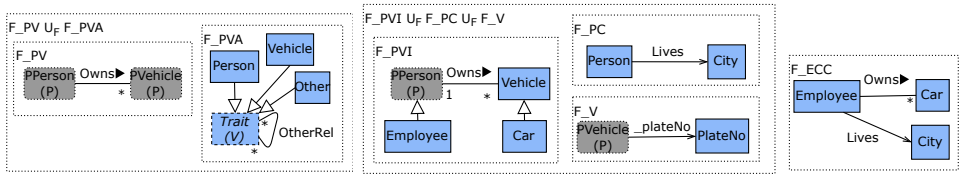



Fig. 14. Fragments and their possible refinements

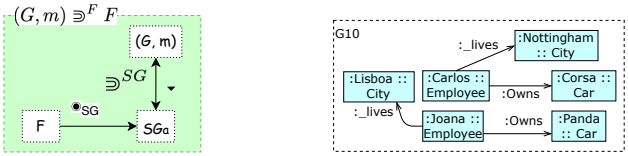
*3.3.1 Morphisms and Refinement.* Figure 13 depicts the definitions of morphisms and refinements for fragments. A morphism $m : GrM$ between $F_c, F_a : Fr$, written $m \in F_c \rightarrow_F F_a$ or $(F_c, m) \Rrightarrow^F F_a$ (Fig. 13a), is equivalent to a morphism between the resolved SGs encapsulated in the fragments. Fragment refinement is built on top of morphisms: a fragment $F_c$ partially refines a fragment $F_a$ through a morphism $m$, written $(F_c, m) \sqsupseteq^F F_a$, if the SGs resulting from resolving the fragments are in a partial refinement relation through the resolved morphism $m$ (def. 38, depicted in formula box of Fig. 13b). Total fragment refinement (symbol $\sqsupset^F$, def. 39) requires the fragments to be total and the resolved SGs to be in a total refinement relation as depicted in Fig. 13c.

The following morphisms apply to Fig. 14: $m_1 : F\_PVI \rightarrow_F F\_PV$, $m_2 : F\_PC \rightarrow_F F\_PVA$, $m_3 : F\_V \rightarrow_F F\_PVA$, $m_4 : F\_PVI \cup_F F\_PC \cup_F F\_V \rightarrow_F F\_PV \cup_F F\_PVA$ and $m_5 : F\_ECC \rightarrow_F F\_PV \cup_F F\_PVA$, which are defined as:

$m_1 = (\{PPerson, Employee \mapsto PPerson, Car, Vehicle \mapsto PVehicle\}, \{Owns \mapsto Owns\})$
$m_2 = (\{Person \mapsto Person, City \mapsto Other\}, \{Lives \mapsto OtherRel\}$
$m_3 = (\{PVehicle \mapsto Vehicle, PlateNo \mapsto Other\}, \{Vehicle\_plateNo \mapsto OtherRel\})$
$m_4 = m_1 \cup_{GM} m_2 \cup_{GM} m_3$
$m_5 = (\{Employee \mapsto Person, Car \mapsto Vehicle, City \mapsto Other\}, \{Owns \mapsto Owns, Lives \mapsto OtherRel\})$

The following was verified in the Fragmenta tool:

- $(F\_PVI, m_1) \sqsupseteq^F F\_PV$
- $(F\_PC, m_2) \sqsupseteq^F F\_PVA$ and $(F\_V, m_3) \sqsupseteq^F F\_PVA$
- $(F\_PVI \cup_F F\_PC \cup_F F\_V, m_4) \sqsupset^F F\_PV \cup_F F\_PVA$
- $(F\_ECC, m_5) \sqsupset^F F\_PV \cup_F F\_PVA$




(a) Typing of a graph with typing by a Fragment (b) An instance of fragments $F\_PVI \cup_F F\_PC$ of Fig. 14b and $F\_ECC$ of Fig. 14c

Fig. 15. Fragments as Types: definition (a) and an illustration (b).

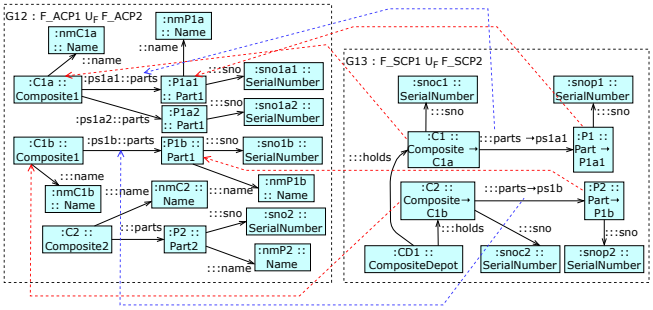
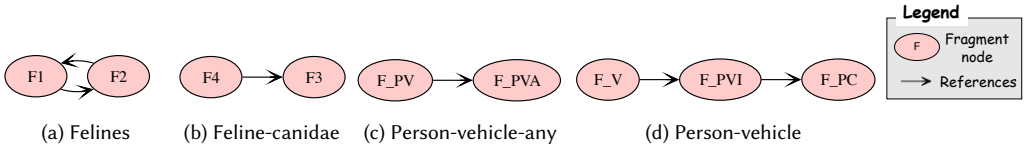



Fig. 16. Graph instances and extra typing.

*3.3.2 Fragments as Types.* SG typing (section 3.2.8) is extended to fragments. A graph with typing $GwT : GrwT$ (def. 11), a pair made-up of a graph $G : Gr$ and graph morphism $m : GrM$, complies with a fragment $F : Fr$, written $GwT \circledcirc^F F$, provided $GwT$ complies with the $SG$ resulting from resolving $F$ as depicted in Fig. 15a (def 40). Figure 15b depicts a graph with typing $G10$ which complies with fragments $F\_PVI \cup_F F\_PC$ of Fig. 14b and $F\_ECC$ of Fig. 14c — $G10 \circledcirc^F F\_PVI \cup_F F\_PC$ and $G10 \circledcirc^F F\_ECC$, respectively.

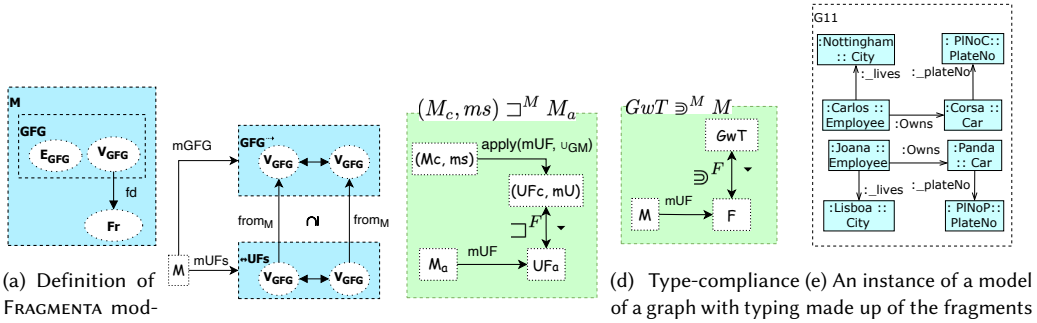
*3.3.3 Extra Typing.* Extra typing relies on a fragment's extra-type morphism, obtainable with function *fet*. A fragment $F_s : Fr$ is extra-type compatible with a fragment $F_t : Fr$, written $F_s \dashv\vdash^F F_t$, provided the SGs of the resolved fragments are extra-type compatible via $F_s$'s extra-type morphism (def. 41). This provides the basis for extra-typing compliance, which stipulates the conditions under which some graph with extra typing $Gwet : GrwET$ (def. 12), typed by a fragment $F_s : F_r$ (def. 32), is extra-typed by a graph with typing $Gwt : GrwT$ (def. 11), typed by a fragment $F_t : F_r$, written $(Gwet, F_s) \Vdash^F (Gwt, F_t)$, as per def. 41, which is the case provided: (i) $Gwet$ is typed by $F_s$ — $Gwet^{Gw} \circledcirc^F F_s$ —, (ii) $Gwt$ is typed by $F_t$ — $GwT \circledcirc^F F_t$ —, (iii) the domain of $Gwet$'s extra typing morphism is consistent with $F_s$'s extra typing morphism, (iv) $Gwet$'s extra typing morphism is a partial morphism from $Gwet$ to $Gwt$, (v) $F_s$ is extra-type compatible with $F_t$ — $F_s \dashv\vdash^F F_t$.

Extra typing compatibility can be checked for the example of Fig. 12f, to establish that $F\_SCP2 \dashv\vdash^F F\_ACP2$ and $F\_SCP1 \cup_F F\_SCP2 \dashv\vdash^F F\_ACP1 \cup_F F\_ACP2$. It is also possible to check the overall extra typing compliance of Fig. 16 where: $(G13, F\_SCP1 \cup_F F\_SCP2) \Vdash^F (G12, F\_ACP1 \cup_F F\_ACP2)$. These checks were carried out in the Fragmenta tool.




(a) Felines (b) Feline-canidae (c) Person-vehicle-any (d) Person-vehicle

Fig. 17. Global Fragment Graphs (GFG)




(a) Definition of Fragmenta models (b) Predicate ⇑ (c) Model refinement (d) Type-compliance of a graph with typing to a model (e) An instance of a model made up of the fragments of Fig. 14b

Fig. 18. Fragmenta Models, their refinement and typing

### 3.4 Global Fragment Graphs (GFGs)

GFGs say how fragments reference one another. A fragment $A$ references fragment $B$ when a proxy of $A$ refers to either a node from $B$, or from the fragments referenced by $B$ as per transitivity. GFGs dictate the permitted referencing flows of local fragments.

A GFG is a graph, $GFG : Gr$ (def. 42), where nodes represent fragments, and edges indicate the referencing flows. Set $GFGr$ (def. 42) of all well-formed GFGs includes those GFGs which, after the removal of the identity (or self) edges, are acyclic.

Relation $\overset{*}{\rightarrow}$ (def. 42) gives all fragments referenced by a particular fragment; it is defined as the transitive closure of the relation obtained from the GFG: $GFG \overset{*}{\rightarrow} = (GFG^{\leftrightarrow})^{+}$

Figure 17 gives examples of GFGs:

- Figure 17a captures the felines example of Fig. 12b; it depicts a malformed GFG with a disallowed cycle. Figure 17b, a well-formed GFG, captures the example of Fig. 12d.
- Figures 17c and 17d portray the fragment-referencing of Figs. 14a and 14b, respectively.

### 3.5 Models

Models hold their constituting fragments, usually inter-related to be meaningful as a whole. Depicted in Fig. 18a, a model is a pair $(GFG, fd)$, comprising a $GFG : GFGr$ (def. 42) and a function $fd : Ns_{GFG} \rightarrow Fr$ which maps nodes representing fragments to their definitions (def. 43). The set of base models, $Mdl_0$, comprises all models with valid GFGs and mutually disjoint fragments (def. 43). Function $mUFs$ builds the union of all the fragments of a model, and $from$ gives the fragment provenance of any model node (def. 43). Predicate $\Uparrow$ (def. 43) says whether the referencing embedded in the model's fragments complies with the model's GFG by resorting to function $from$ and relation $\overset{*}{\rightarrow}$, depicted as

commuting in Fig. 18b (the commuting equation is shortened to an equation with quantifiers in def. 43). Set *Mdl* of well-formed models (def. 43) includes all those base models (set $Mdl_0$) whose union of fragments is a total fragment and whose fragments' referencing complies with the model's GFG (as per $\Uparrow$). Finally, operator $\circledbullet^M$ (def. 43) gives the fragment resulting from resolving the union of the fragments of a model.

Many of the examples explored so far can now be framed as models:

- The abstract hotel world model made-up of Fig. 1 is well-formed.
- The pets world model of Fig. 2 is well-formed. Saying in the GFG that fragment `F_PW1` references `F_PW2` instead, would make the model malformed — the GFG would misrepresent the underlying referencing.
- A model made-up of the GFG of Fig. 17a and fragments `F1` and `F2` of Fig. 12b would be malformed, as the GFG contains a cycle. Removing the cycle in the GFG would not resolve this issue as the GFG would diverge from the underlying referencing.
- A model made-up of the GFG of Fig. 17b and fragments `F3` and `F4` of Fig. 12d would be well-formed. Making `F3` reference `F4` in the GFG would misrepresent the referencing flow.
- The model with GFG of Fig. 17c with fragments of Fig. 14a is well-formed. Likewise, for the model with GFG of Fig. 17d and fragments of Fig. 14b.

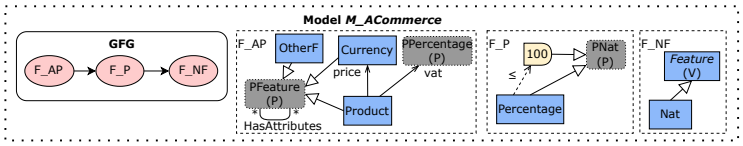
*3.5.1 Morphisms and Refinement.* Model morphisms are defined from fragment morphisms through the union of morphism. A model morphism $m$ betweem models $M_s, M_t : Mdl$, written $m \in M_s \rightarrow_M M_t$, maps elements from the union of fragments of $M_s$ onto the union of fragments of $M_t$ (def. 44). Because model morphisms are usually a collection of morphisms defined for each fragment of the source model, we say that a model $M_s : Mdl$ and a set of morphisms $ms : \mathbf{P}\, GrM$, are morphism-related to another model $M_t : Mdl$, written $(M_s, ms) \Rrightarrow^M M_t$, when the union of the morphisms in $ms$ is a model morphism from $M_s$ to $M_t$ as per def. 44. Model refinement is defined similarly in terms of total fragment refinement (def. 45, pictured in Fig. 18c); a model $M_c : Mdl$ refines another model $M_a : Mdl$ via the morphisms $ms : \mathbf{P}\, GrM$, written $(M_c, ms) \sqsupset^M M_a$, if the union of fragments of $M_c$ refines the union of fragments of $M_a$ via the union of morphisms of $ms - (mUFs\, M_c, \bigcup_{GM} ms) \sqsupset^F mUFs\, M_a$.

The refinement examples explored so far can now be framed as models:

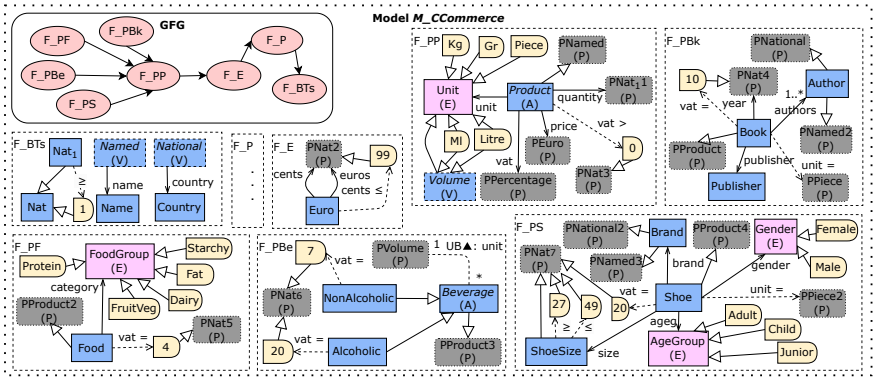
- The pets world concrete model of Fig. 2 is a model refinement of the abstract hotel model of Fig. 1 through the morphisms of Fig. 3.
- The model made-up of the GFG of Fig. 17d and the fragments of Fig. 14b is a refinement of the model made-up of the GFG of Fig. 17c and the fragments of Fig. 14a, through morphism $m_4$ of section 3.3.1.

*3.5.2 Models as Types.* Fragment typing is extended to model typing through the union of fragments. A graph with typing $GwT : GrwT$ (def. 11) complies with a model $M : Mdl$ — written $GwT \Ralph^M M$ — provided the $GwT$ complies with the union of the model's fragments depicted in Fig. 18d (def. 46). The graph of Fig. 4 is an instance of (or is type-compliant with) the pets world concrete model of Fig. 2. The graph of Fig. 18e is an instance of the person-vehicle model made-up of the GFG of Fig. 17d and the fragments of Fig. 14b.

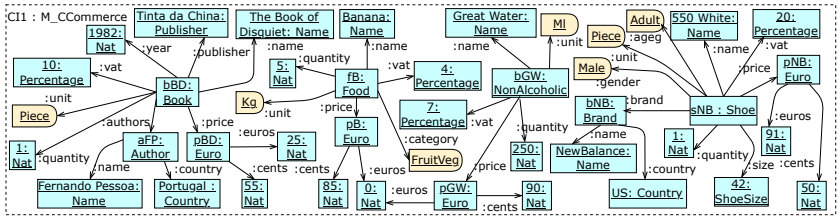
Extra-typing works in a similar fashion: extra-typing at the level of models reduces to extra-typing at the level of fragments by reducing models to the union of their fragments (def. 47). Two models, *M_ACP* and *M_SCP*, can be built from the fragments of Fig. 12f, where $M\_SCP \dashv\vdash_{\mathcal{M}} M\_ACP$. For the two graphs of Fig. 12f, we have that $(G10, M\_SCP) \Vdash^M (G11, M\_ACP)$.



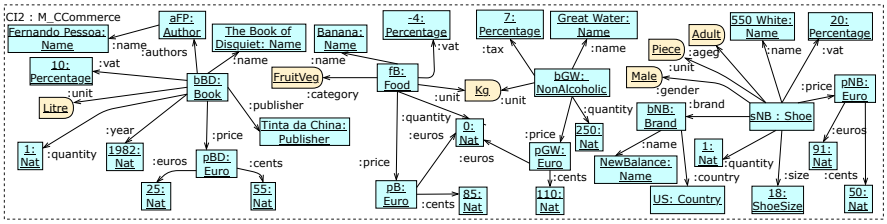

(a) Abstract model




(b) Concrete model (F_P is the fragment with same name from (a))




(c) A valid instance of M_CCommerce




(d) An invalid instance M_CCommerce

Fig. 19. The three layers of the simple e-commerce world.

## 4 A BENCHMARK LAYERING PROBLEM

Fragmenta is illustrated with an e-commerce problem [13, 20, 21], a benchmark in the area of multi-level modelling . The uppermost level of abstraction describes a *meta* world made up of product types and their characteristics. At a more concrete level of abstraction, we have actual product types, namely books, food, beverages and shoes. The instance level includes actual books, food and beverage items, and shoes which are on sale. This three-layered architecture discussed in the literature [1, 13, 20, 21] is expressed in Fragmenta in Fig. 19, highlighting Fragmenta's layering strata: type (or

meta) and instance. The type stratum includes the abstract model of Fig. 19a (the meta-layer) and its more concrete refinement of Fig. 19b (the type layer). The instance stratum includes Figs. 19c and 19d. The models that make the type stratum are divided in fragments which build upon each other as per their respective GFGs.

The abstract model layer (Fig. 19a) introduces three fragments: (i) F_NF says that Nat (natural numbers) is a Feature, which is a virtual; (ii) F_P defines Percentages; (iii) F_AP says that Products have a *price* in some currency and a value added tax (*vat*) Percentage, and are Features, which may have several attributes (*virtual traits* pattern), and introduces OtherF to represent a feature other than Nat and Product.

The concrete model layer (Fig. 19b), with its six fragments, is as follows:

- Fragment F_BTs introduces nodes Nat, Nat_1, defined from Nat to represent the strictly positive natural numbers, Name and Country, and the virtuals Named and National. In terms of the mapping to Fig. 19a, all Nat-related nodes are mapped to Nat, Named and National to Feature and all others to OtherF.
- Fragment F_P refers to the homonymous of Fig. 19a (same fragment is used in both layers). In the mapping, Percentage is mapped to itself, and the remaining nodes to Nat.
- Fragment F_E defines the Euro currency. In the mapping, Euro is mapped to Currency, and the remaining nodes to Nat.
- Fragment F_PP defines the abstract Product, made-up of a *vat* Percentage, a Euro *price*, and a *unit*, either kilograms, grams, pieces, millilitres and litres, and the Volume virtual a disjoint union of millilitres and litres. In the mapping, Product is mapped to its homonymous, Unit and its values to OtherF, Volume to Feature, and the proxies are mapped as their referents.
- Fragment F_PB introduces Books (a product), which have a *year*, *authors*, *publisher*, a 10%vat and sold as pieces. In the mapping, Book is mapped to Product, and Publisher and Author to OtherF.
- Fragment F_PF introduces Food, categorised according to the FoodGroup enumeration as either fruit or vegetable, starchy food (such as potatoes and cereals), dairy, fat (such as olive oil) and protein (such as meat, fish and beans), and with a vat of 4% and sold as pieces. In the mapping, Food is mapped to Product, and FoodGroup and its values to OtherF.
- Fragment F_Be introduces the abstract Beverage, a product sold in volume units, sub-divided into Alcoholic with a vat of 20/% and NonAlcoholic with a vat of 7%. In the mapping, Beverage, Alcoholic and NonAlcoholic are mapped to Product.
- Fragment F_PS introduces Shoe, a product with a *brand*, a vat of 20/%, a shoe *size*, a targeted age group and gender. In the mapping, Shoe is mapped to Product, and AgeGroup and its values, and Gender and its values to OtherF.
- The refinement to the abstract layer has been checked in the Fragmenta tool.

Figures 19c and 19d, both instances of Fig. 19b, illustrate the instance layer. They both define the book bBD, the food item fB, the non-alcoholic beverage bGW, and the shoe sNB. Figure 19c is a valid instance. Figure 19d breaches several metamodel constraints: (i) bBD's unit is litres, (ii) author aFP is countryless, (iii) fB's vat is negative, (iv) fB and bGW have both kilogram units, (v) sNB's size 18 is below the minimum allowed, and (vi) the price of bGW is 0 euros and 118 cents, which breaches the euro cents constraint.

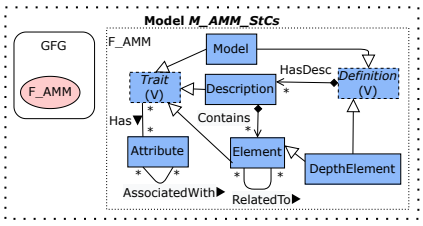



Fig. 20. Abstract layer of the FRAGMENTA metamodel of statecharts.

## 5 A LAYERED DEFINITION OF STATECHARTS

The metamodel of statecharts (StCs) [33] is modelled in FRAGMENTA. The sequel presents abstract and concrete layers of StCs' metamodel, illustrated and exercised through instances.

### 5.1 The Abstract Layer

The abstract metamodel (Fig. 20) is made up of a single fragment:

- Definitions (virtuals) are defined in terms of Descriptions, which contain Elements. Models and DepthElements are Definitions.
- Elements are related to other elements (edge RelatedTo) to capture horizontality. DepthElements (Elements with depth), captures verticality or depth.
- Models, Elements and Descriptions are Traits (a virtual node) which may have Attributes that may be inter-related.

### 5.2 The Concrete Layer

The concrete metamodel (Fig. 21) is made-up of the following fragments inter-related as per GFG:

- F_MM1: Statechart definitions (virtual StCDef) contain descriptions (StCDesc), made-up of Elements (abstract). A StCModel is a StCDef. StC models and descriptions are Named, a virtual. In the mapping into Fig. 20, StCModel is mapped to Model, StCDesc to Description, StCDef to Definition, Element to Element, Named to Trait, Name to Attribute, HasDesc to HasDesc, Contains to Contains and Named_name to Has.
- F_MM2: Abstract States are Elements, sub-divided into StartStates, EndStates, HistoryStates, and MutableStates, which are named and inherit StCDef to allow depth. In terms of Fig. 20, all states are mapped to Element, except MutableState which is mapped to DepthElement; proxies are mapped like their referents.
- F_MM3 constrains StC descriptions through derived edges to say that descriptions contain one StartState, and may contain one EndState and one HistoryState.
- F_MM4: Transitions are elements made up of one source and one target States. They may have an Event, a Guard and an Action. Virtual WExp represents something made up of an expression, such as Events, Guards and Actions. With respect to the mapping into Fig. 20, Transition maps to Element, and all other non-proxies to Attribute.
- F_MM5 constrains the way transitions are related to start, end and history states. Transitions must not have init nodes as targets, and end nodes as sources; start states must be the source state of at least one transition;

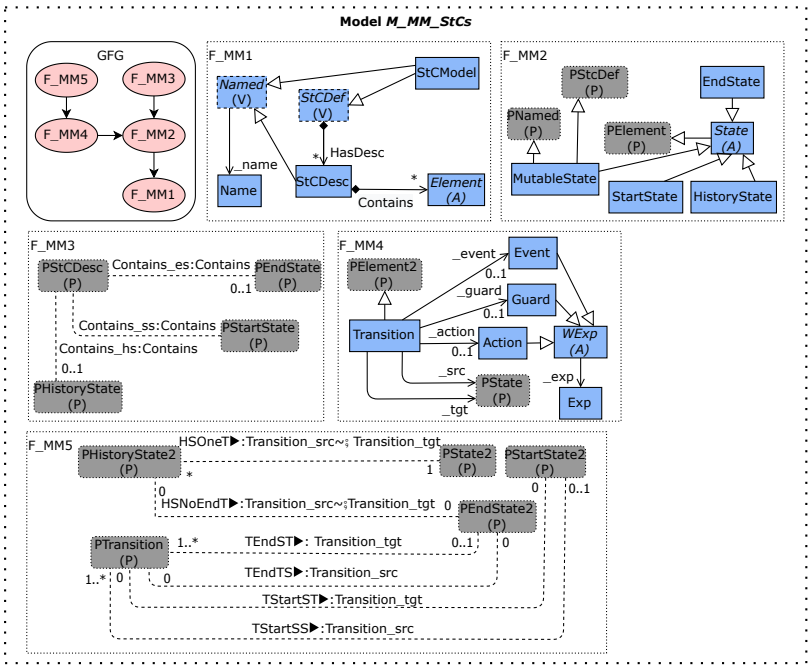


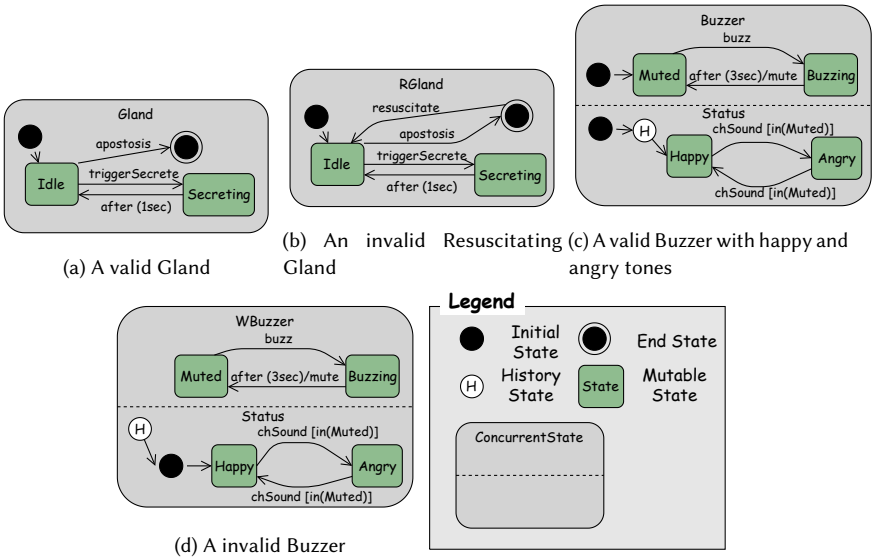

Fig. 21. Fragmenta model capturing the concrete layer of StCs' metamodel.

end states must be targets of at least one transition; history states must have one transition into one other state, which must not be an end state.

### 5.3 Illustrations

The StCs of Fig. 22 were compliance-checked against the metamodel (Fig. 21) by Fragmenta's tool:

- Figure 22a complies with the metamodel. Glands, initially idle, secrete hormones when triggered by the environment, and die upon receiving the apostosis (or cell-death) signal.
- Figure 22b describes a resuscitating gland, an invalid metamodel instance that breaches a constraint of F_MM5 which rules out end states with outgoing transitions.
- Figure 22c, a valid StC with one definition and two descriptions, describes a buzzer with two different tones. The top description says that an initially muted buzzer goes from Muted to Buzzing upon event buzz, and back after 3 seconds triggering mute. In the bottom description: (i) the buzzer emits a happy tone (state Happy), changeable to an angry one (state Angry) and back upon chSound, contingent on the buzzer being Muted; (ii) the history state H marks the most recently visited state (Happy by default).
- Figure 22d, a variant of Fig. 22c, breaches several constraints: (i) the top description lacks an initial state (see F_MM3); (ii) the bottom description violates two F_MM5 rules: history states must have incoming transitions, and initial states must not have incoming transitions. In Fig. 22c, removing the transition from the history state, would breach the constraint of derived edge HSOneT, saying that history states must have one outgoing transition.




(a) A valid Gland

(b) An invalid Resuscitating Gland

(c) A valid Buzzer with happy and angry tones

(d) A invalid Buzzer

Fig. 22. Example Statecharts

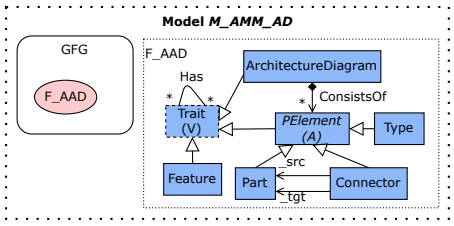



Fig. 23. Fragmenta metamodel of the abstract architecture diagrams of the Into-SysML profile.

## 6 INTO-SYSML: A SYSML PROFILE FOR CYBER-PHYSICAL SYSTEMS

Fragmenta is applied to *INTO-SysML*, a SysML [23, 52] profile for cyber-physical systems [10, 11]. The metamodel, previously described using an early version of Fragmenta, is described herein as a Fragmenta layered model. The abstract layer (section 6.1) captures general architectural models; it is refined by two concrete layers: architecture structure diagrams (section 6.2) and connection diagrams (section 6.3). Section 6.4 illustrates INTO-SysML with a water tanks system.

### 6.1 The Abstract Layer

The abstract metamodel (Fig. 23) comprises a single fragment, F_AAD; it is as follows:

- An ArchitectureDiagram consists of primitive elements — abstract node PElement.
- A PElement can either be Part, a Connector or a Type. Connectors join Parts through edges Connector_src and Connector_tgt.

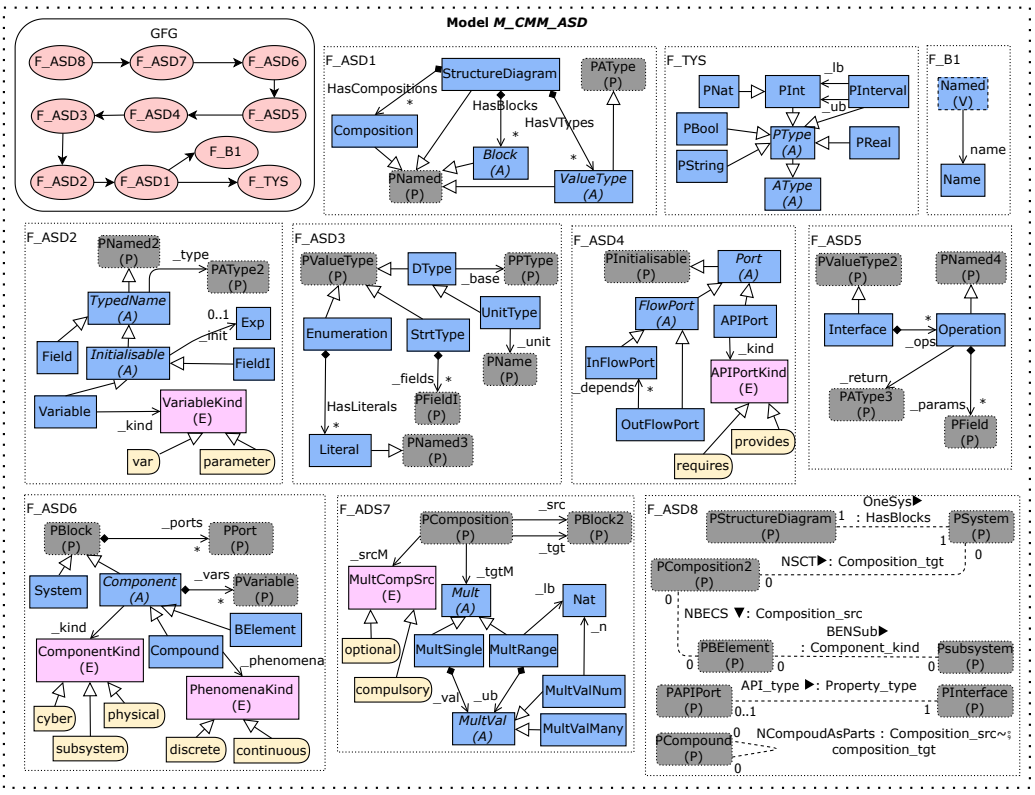



Fig. 24. Fragmenta metamodel of architecture structure diagrams (ASDs) of the Into-SysML profile.

- Virtual `Trait` represents any model element. A feature represent an extra aspect of the model. `Traits` are inter-relatable through edge `Has` (virtual traits pattern).

### 6.2 The Concrete Metamodel of Architecture Structure Diagrams (ASDs)

Figure 24 presents INTO-SysML's concrete metamodel of architecture structure diagrams (ASDs), a specialisation of SysML's block-definition diagrams. The GFG says how the model's 10 fragments are inter-related. The fragments are as follows:

- `F_B1`: `Named`, a virtual, represents something with a `Name`. In the refinement mapping to the abstract model of Fig. 23, `Named` is mapped to `Trait`, and `Name` to `Feature`.
- `F_TYS`: Primitive types (`PTypes`) include integers (`PInt`), of which natural numbers (`PNat`) are a subset, intervals (`PInterval`) made-up of integer lower- and upper-bounds, reals, booleans and strings. A `PType` is a `AType`, which includes primitive and value-types. In the refinement mapping, all nodes are mapped to `Type`.
- `F_ASD1`: a `StructureDiagram` comprises `Block` and `ValueType` (both abstract), and `Composition`. `ValueTypes` are types — proxy `PAType` refers to `AType` of `F_TYS`. In the mapping: `StructureDiagram` is mapped to `ArchitectureDiagram`, `Block` to `Part`, and `ValueType` to `Type` — proxies are mapped like their referents.

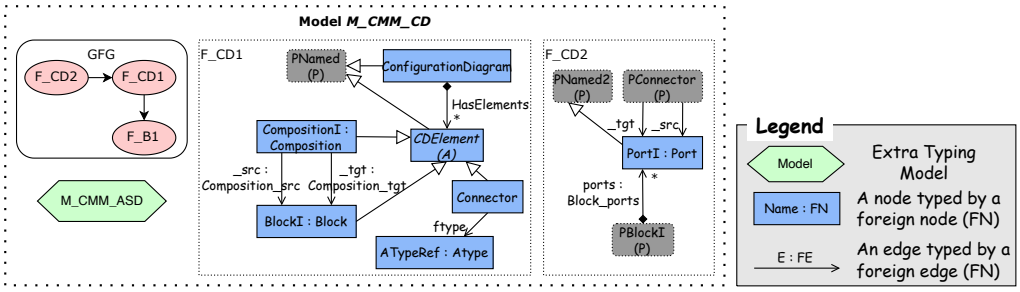
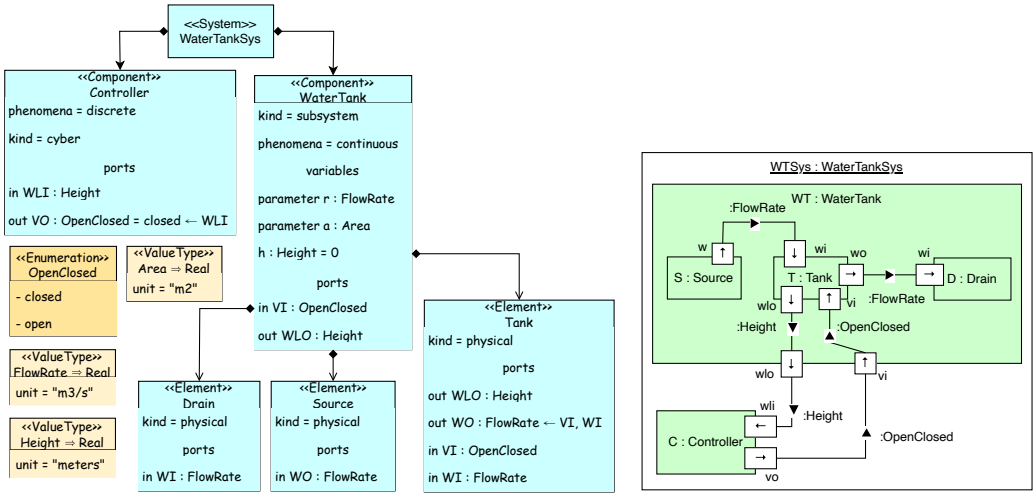



Fig. 25. FRAGMENTA metamodel of connection diagrams (CDs) of the Into-SysML profile.

- F_ASD2: A Field is a TypedName (abstract), which has a type. Initialisable (abstract), a TypedName, may have an initialisation expression. FieldIs and Variables are Initialisables; Variables may either be normal or parameters. In the mapping, Exp, VariableKind, var and parameter are mapped to Feature, TypedName and Initialisable to Trait and the two field nodes to Part.
- F_ASD3: Enumerations are value types made up of named Literals. Structural types (StrtType), also value types, are made up of initialisable fields. Derived types (DType), also value types, have a base, a primitive type. UnitTypes (such as $^{\circ}C$ or centigrades), which are DTypes, have a name of a unit. In the mapping, StrtType, DType, UnitType, Enumeration are mapped to Type, literals to Feature.
- F_ASD4: Ports (abstract) are initialisable and divided into FlowPorts and APIPorts. APIPorts' two kinds of abstract programming interfaces (APIs) are requires and provides. FlowPorts are further divided into in- and out-flow ports, where the latter have dependencies with the former. In the mapping, ports are mapped to Part, APIPortKind, requires and provides to Feature.
- F_ASD5: Interfaces are value types made up of Operations, which have a return type and parameters. In the mapping, Interface is mapped to Type, Operation to Feature.
- F_ASD6: A block is either a System or a Component (abstract), either of kind cyber, subsystem or physical, and sub-divided into Compounds or base elements (BElement). Compounds may handle either discrete or continuous phenomena. In the mapping, all Block nodes are mapped to Part, ComponentKind and PhenomenaKind and their children to Feature.
- F_ASD7: Compositions have a source and a target block with an optional source multiplicity, either optional or compulsory, and a target multiplicity. Multiplicities (Mult) can either be single (MultSingle) or range (MultRange). A MultSingle holds a multiplicity value, MultVal, either a single natural number (MultValNum) or just many (MultValMany, written $*$). A MultRange has a natural number lower bound (edge to Nat) and a MultVal upper bound. In the mapping, all multiplicity-related nodes are mapped to Feature.
- F_ASD8: The following constraints are described as derived edges: (i) a ASD must have one system block; (ii) systems may not be parts in compositions; (iii) BElements may neither be compounds nor subsystems; (iv) an APIPort's type must be interface; and (vi) compounds may not have other compounds as parts.

### 6.3 The Concrete Metamodel of CDs

Figure 25 presents the concrete metamodel of CDs, which specialise SysML's internal block diagrams. CDs' metamodel imports ASDs' metamodel of Fig. 24, presented above, to provide the extra ASD types that certain CD elements require.




(a) ASD (b) CD

Fig. 27. Into-SysML model of the water tanks system from [11].

CDs' metamodel is made-up of three fragments (see GFG), one of which is F_B1, shared with ASDs; the other two fragments are as follows:

- F_CD1: ConfigurationDiagrams comprise a collection of CDElements, an instance of either composition (CompositionI) or block (BlockI), or a Connector. Extra type constraints state that BlockIs and CompositionIs are instances of ASDs' Block and Composition (Fig. 24), respectively. A CompositionI has a source and a target BlockIs (extra-typed as edges CompositionI_src and CompositionI_tgt). A Connector has a flowing type, represented by ATypeRef, extra-typed as AType. In the refinement mapping to Fig. 23, ConfigurationDiagram is mapped to ArchitectureDiagram, CDElement to PElement, CompositionI and Connector to Connector, BlockI to Part and ATypeRef to Type.
- F_CD2: Port instances (PortI) (port instances) are extra-typed as Port. Block instances are made up of PortIs; edge ports is extra-typed as Block_Ports — PortIs of a CD block correspond to ports of a ASD block. A Connector has a source and a target PortI. In the refinement mapping, PortI is mapped to Part.

### 6.4 Illustration

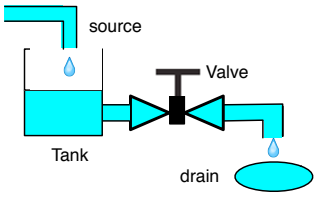



Fig. 26. Water Tanks system.

In the system of Fig. 26, a source of water fills a tank. A valve controls the outflow of water, which flows into the drain when the valve is open. A software controller opens or closes the valve depending on the tank's water level. The Into-SysML model of this simple cyber-physical system (Fig. 27) comprises one ASD (Fig. 27a) and one CD (Fig. 27b).

In the ASD (Fig. 27a), the system block comprises: Controller, a cyber component, and WaterTank, a subsystem. Controller includes: (i) input port WLI of type Height (in meters) to get a water level input from WaterTank; and (ii) valve output VO of type OpenClosed (an enumeration) to instruct the WaterTank to either

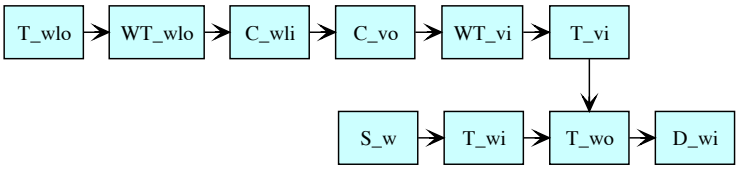



Fig. 28. Port dependency graph extracted from the Into-SysML model of water tanks (Fig. 27)

open or close the valve. WaterTank's three variables are: (i) FlowRate parameter $r$ measures the inner flow of water in cubic meters per second, (ii) parameter $a$ gives the tank's Area in square meters, and (iii) variable $h$ holds the height (or level) of water in the tank. WaterTank has two ports: (i) input port VI takes the valve's instruction, and (ii) port WLO outputs the water's height. WaterTank comprises three components: (i) Drain with its input port WI (incoming water flow); (ii) Source with its output port WO (outgoing water flow); and (iii) Tank with its input ports VI (valve's state) and WI (incoming water flow), and output ports WO (outgoing water flow) and WLO (water level).

The CD (Fig. 27b) connects the ASD block instances through port instances as per Fig. 26. System WTSys (instance of WaterTankSys) is composed of WT and C, instances of WaterTank and Controller respectively. WT is sub-divided into S, T and D. Connectors link port instances — for example, C's output port vo (instance of Controller port VO) is linked to WT input port vi (instance of WaterTank input port VI).

The algebraic loop check subjacent to such models, done in CSP in [11], can be performed within the Fragmenta tool. Figure 28 presents the port dependency graph extracted from Fig. 27; within the Fragmenta tool it can be established that the graph is acyclic — hence, algebraic loops are absent in the model. In the ASD (Fig. 27a), making port Tank.WLO dependent on port Tank.VI would introduce an algebraic loop — an edge from T_vi to T_wlo would be added to Fig. 28.

## 7 EVALUATION

The sequel introduces evaluation questions, dissected from perspectives (section 7.1). Fragmenta is compared against a selection of competitors using a sample of models (section 7.2). Finally, criteria and results are presented for the two perspectives, horizontal (section 7.3) and vertical (section 7.4) separation of concerns, before presenting the overall results (section 7.5).

### 7.1 Evaluation Question (EQ)

The following question drives the evaluation:

**EQ1.** Is Fragmenta's modelling approach supported by its novelties better than competing approaches with respect to horizontal and vertical decomposition, formality and graphical expressiveness?

This question is dissected through the prism of two perspectives: *horizontal* and *vertical separation of concerns*. The former deals with articulation and covers graphical expressiveness, horizontal decomposition and horizontality's formality; the latter with layering which concerns abstraction, refinement and typing (or instantiation).

### 7.2 Competitors and Evaluation Sample

Table 1 presents Fragmenta's evaluation competitors. Because UML is a benchmark, three different flavours are considered: (i) monolith UML (MUML), a traditional usage without partitioning, (ii) partitioned UML (PUML), an

| Approach | Description |
|---|---|
| Monolith UML (MUML) | A traditional, non-partitioned UML usage. |
| Partitioned UML (PUML) | Informal partitioning for UML class models (from [25, 43]). |
| Package UML (PkgUML) | An approach based on UML with package merge [24, 53]. |
| MergeUML | Sabetzadeh et al [58, 59]'s UML model merging approach. |
| MetaDepth | The multi-level modelling approach based on potencies of [20, 21]. |

Table 1. Approaches compared against Fragmenta in the evaluation.

| Model | Problem | Description |
|---|---|---|
| Abstract Hotel | Hotel – Pet hotel | Abstract hotel layered model (Fig. 1, section. 2). |
| Pet Hotel | Hotel – Pet hotel | Concrete pet hotel layered model (Fig. 2, section. 2). |
| Pet hotel instance | Hotel – Pet hotel | Pet hotel instances of layered model (Fig. 4, section. 2). |
| Abstract commerce | Commerce | Abstract e-commerce layered model (Fig. 19a, section 4). |
| Concrete commerce | Commerce | Concrete e-commerce layered model (Fig. 19b, section 4). |
| Commerce instances | Commerce | Instances of the e-commerce model (Figs. 19c and 19d, section 4). |
| Abstract statecharts | Statecharts | Abstract statecharts layered model (Fig. 20, section 5.1). |
| Concrete statecharts | Statecharts | Concrete statecharts layered model (Fig. 21,section 5.2). |
| Statechart instances | Statecharts | Instances of the statecharts metamodel (Fig. 22, section 5.3). |
| Abstract Into-SysML | Into-SysML | Abstract Into-SysML's architecture diagrams (Fig. 23, section 6.1). |
| Architecture structure diagrams (ASDs) | Into-SysML | ASDs (Fig. 24), part of the Into-SysML metamodel (section 6.2). |
| Connection diagrams (CDs) | Into-SysML | CDs (Fig. 25), part of the Into-SysML metamodel (section 6.3). |
| Into-SysML instances | Into-SysML | ASD and CD instances (Fig. 27), illustrating Into-SysML (section 6.4). |

Table 2. Sample of models used in the evaluation.

informal approach from the literature (e.g. [25, 43]), and (iii) UML with packaging (PkgUML), investigated in [24, 53]. Two approaches were selected because they are novel with respect to horizontal and vertical separation of concerns: UML with model merge (MergeUML) of Sabetzadeh et al [58, 59], and MetaDepth [20, 21], respectively.

The sample of models that sets the basis of the comparison includes the four layered models presented herein (table 2).

### 7.3 Horizontal separation of concerns (HSoC): criteria and results

The sequel presents HSoC's orthogonal factors (sections 7.3.1 to 7.3.4), the HSoC overall score formula (section 7.3.5), and the overall results (section 7.3.5).

*7.3.1 Transmission Effectiveness (TE).* Founded on the limits of human information processing [46], TE gauges how information is conveyed. A good TE means effective decomposition and good usability. A bad TE entails information overload, unsatisfactory decomposition and impoverished usability.

Information load (IL), TE's base measure, accounts for amount of information; it is based on summing the weights assigned to model elements as per Table 3a, which gives element types and their IL weights. Individual elements weigh 1; additional representations weigh 5 due to the cognitive overhead involved in handling them (e.g. separate diagram); ILs of textual descriptions are estimated as follows: 6 for simple constraints and 9 for complex constraints[13]. This

[13]These are estimated as they are not precisely specified herein. Typically, they correspond to VCL's value edge constraints (simple constraints) and derived and path edges (complex constraints).

| | |
|---|---|
| Node contour | 1 |
| Edge and direction | 1 |
| Name | 1 |
| Multiplicity | 1 |
| Supplemental information | 1 |
| Extra representation | 5 |
| Simple textual constraint | 6 |
| Complex textual constraint | 9 |

(a) Element types and their information load weights

| | |
|---|---|
| Excellent | 5 |
| Good | 4 |
| Satisfactory | 3 |
| Unsatisfactory | 2 |
| Poor | 1 |

(b) Evaluation's five-point scale

| | |
|---|---|
| Excellent (5) | 0 . . 55 |
| Good (4) | 56 . . 80 |
| Satisfactory (3) | 81 . . 105 |
| Unsatisfactory (2) | 106 . . 130 |
| Poor (1) | > 130 |

(c) Transmission effectiveness scores and corresponding information load intervals

Table 3. Information load weights, the five-point scale, and transmission effectiveness scores

| | |
|---|---|
| Excellent (5) | Almost everything described graphically, despite possible minor limitations ($GER \geq .95$). The model's TE is at least good ($TE \geq 4$). |
| Good (4) | Most description needs addressed graphically ($.8 \leq GER < .95$). The model's TE is at least good ($TE \geq 4$). |
| Satisfactory (3) | Despite linguistic limitations, majority of what is needed is described graphically ($.65 \leq GER < .80$) and the model's TE is at least satisfactory ($TE \geq 3$), or most description needs aggressed graphically ($GER > .80$), but model's TE is below satisfactory ($TE < 3$). |
| Unsatisfactory (2) | Linguistic limitations prevent the graphical expression of many pertinent properties ($.5 \leq GER \leq .65$). |
| Poor (1) | The graphical linguistic power is too limited ($GER < .5$). |

Table 4. Criteria of expressiveness (E).

provides the basis for: *maximum* ($IL^{\bullet}$) and *mean* ($\overline{IL}$) diagram IL, *total* model IL ($IL^{\Sigma}$) and IL per fragment ($IL/F$). $IL^{\bullet}$ gives a fragmented model's maximum IL, $\overline{IL}$ is the mean, $IL^{\Sigma}$ is the overall model's IL (sum of individual ILs), including other diagrams describing how fragments are interrelated, and $IL/F$ gives the overall model's IL score per fragment ($IL^{\Sigma}$ divided by number of fragments of the model).

Table 3c says how TE rankings, on the evaluation's five-point scale (table 3b), are derived from IL intervals. A model's TE can either be *excellent* ($IL \leq 55$), *good* ($56 \leq IL \leq 80$), *satisfactory* ($81 \leq IL \leq 105$), *unsatisfactory* ($106 \leq IL \leq 130$), or *poor* ($IL \geq 130$). Measures maximum TE ($TE^{\bullet}$), average TE ($\overline{TE}$) and TE per fragment ($TE/F$) are derived from $IL^{\bullet}$, $\overline{IL}$, and $IL/F$, respectively, by applying the function described by table 3c.

The $TE$ score is the mean of the three individual TEs:

$$TE = \frac{TE^{\bullet} + \overline{TE} + TE/F}{3}$$

A model's size is characterised with respect to $IL^{\Sigma}$ as either *small* ($IL^{\Sigma} < 90$), medium ($90 \leq IL^{\Sigma} < 200$) or large ($IL^{\Sigma} \geq 200$).

*7.3.2 Expressiveness (E).* About graphical descriptive capability, E is related to articulation, a tenet of HSoC[14]. A good $E$ indicates capacity to express what is needed succinctly. A bad $E$ is a sign of verbosity, awkwardness or inability to express graphically certain required things.

[14]The graphical focus is due to Fragmenta's graphical nature.

| | |
|---|---|
| Excellent (5) | The decomposition of the model is effective and mechanically composable. DC is formal and tool-supported. |
| Good (4) | The decomposition of the model is effective and mechanically composable. DC is formally defined, but lacks tool-support. |
| Satisfactory (3) | DC mechanisms are either flawed or unconvincing. The model is small ($IL^{\Sigma} < 90$), which diminishes need for DC. |
| Unsatisfactory (2) | DC mechanisms are either flawed or unconvincing. Either they are not properly defined or unconvincing and the model is at least medium-sized ($IL^{\Sigma} \geq 90$), or a satisfactory definition is missing altogether and the model is medium-sized ($90 \leq IL^{\Sigma} < 200$). |
| Poor (1) | Not possible to make a much needed mechanical decomposition due to the lack of means and the model is large ($IL^{\Sigma} >= 200$). |

Table 5. Criteria of decomposition and composition (DC)

| | |
|---|---|
| Excellent (5) | HWF, formally defined and tool-supported, ensures excellent healthiness. All healthiness checks, pertinent to the evaluation presented here, are covered. |
| Good (4) | HWF, formally defined and tool-supported, ensures good healthiness. The few unsupported pertinent checks are negligible as model is small ($IL^{\Sigma} < 90$). |
| Satisfactory (3) | Healthiness guarantees are only satisfactory. Existing HWF is not proper (unrigorous and unconvincing), without tooling and resulting model is small ($IL^{\Sigma} < 90$). Or HWF is proper but the unsupported pertinent checks are important as model is not small ($IL^{\Sigma} \geq 90$). |
| Unsatisfactory (2) | Healthiness guarantees are dubious. HWF is either not proper (unrigorous, unconvincing, and without tooling) and model is medium-sized ($90 \leq IL^{\Sigma} < 200$), or it is unacceptable and model is small ($IL^{\Sigma} < 90$). |
| Poor (1) | Healthiness guarantees are unreliable. Acceptable HWF is missing altogether and model is not small ($IL^{\Sigma} \geq 90$), or existing HWF is not proper (unrigorous and unconvincing) and model is large ($IL^{\Sigma} \geq 200$). |

Table 6. Criteria of horizontal well-formedness (HWF)

Measure *graphical expressiveness ratio* (GER), which supports appraisals of E, is calculated as:

$$GER = \frac{IL^{\Sigma} - TIL}{IL^{\Sigma}}$$

where *TIL* (textual IL) is the total IL of the estimated textual descriptions.

Table 4 sets out E's criteria on table 3b's five-point scale, using objective measures *GER* and *TE*.

*7.3.3 Decomposition and Composition (DC).* Decomposition breaks things apart, composition puts the parts together. A good *D*C indicates capacity to break down soundly and compose mechanically. A bad DC entails inadequate DC means. Table 5 sets out DC's criteria, which uses the measure $IL^{\Sigma}$.

*7.3.4 Horizontal well-formedness (HWF).* A well defined and, ideally, formal notion of well-formedness paves the way for rigorous consistency and adequacy checking. A good *HWF* entails soundness, formality and effective consistency-checking. A bad *HWF* signals an inadequate healthiness notion that may be imprecise, sketchy or limited. Table 6 sets out the criteria for HWF using $IL^{\Sigma}$.

*7.3.5 Score.* *HSoC*'s score is a weighted sum calculated from the factors:

$$HSoC = TE \times .34 + (E + DC + HWF) \times .22$$

TE gets a weight of 34%; *E*, *DC* and *HWF* get 22% each. The higher TE weight is due to TE's superior objectivity, based on a measure of diagram element counts, which captures the goodness of a decomposition, a tenet of HSoC.

*7.3.6 Results.* Table 7 gives the results for HSoC; it includes columns for: number of fragments (# *Fs*), ILs for maximum ($IL^{\bullet}$), average ($\overline{IL}$) and overall ($IL^{\Sigma}$), IL per fragment ($IL/F$), maximum ($TE^{\bullet}$) and mean ($\overline{TE}$) TE, TE per fragment ($TE/F$), overall TE (*TE*), *E*, *DC*, *HWF* and overall *HSoC*.

The rationale for the HSoC evaluation of the hotel world model (Fig. 1) is as follows:

- In Fragmenta (Fig. 1), ILs of nodes `Hotel`, `Room`, `Guest` and `Other` cost 2 (one box, plus one name), `Trait` costs 3 (2 plus supplemental kind), inheritance arrows cost 1 (line with arrow), all remaining edges cost 4. In `F_AH`, $IL = 27$, also the model's $IL^{\bullet}$ and $\overline{IL}$ as there is only one fragment. $IL^{\Sigma} = 34 - 27$ plus cost of GFG (5) and its node (2). $IL/F = 34$. *TE* is excellent — $TE^{\bullet} = \overline{TE} = TE/F = 5$. *E*, *DC* and *HWF* are all excellent: everything is expressed neatly, the only fragment is formally composable, and well-formedness is tool-checked, respectively. Overall *HSoC* is excellent.
- A trivial MUML translation of Fig. 1 would make `Trait` abstract. Virtuals differ from abstract classes, which imply a stronger *is-a* semantics and single inheritance, but this is negligible in this case — expressing `AnyRel` of Fig. 1 with individual associations would be a more onerous alternative. Therefore, MUML's $IL^{\bullet}$ and $\overline{IL}$ is 27, as Fragmenta, but with a lower $IL^{\Sigma}$ of 27 (no extra package diagram), giving a $IL/F$ of 27. *TE* is excellent. *E* is excellent as everything is expressed graphically and the TE is good. *DC* is satisfactory — only one fragment without a composable underpinning. *HWF* is unsatisfactory (2) as its definition is mostly informal, not overly significant in this small model. Overall *HSoC* is good ($3.9 \approx 4$).
- The other approaches vary only slightly from Fragmenta or MUML. MergeUML trails closely behind Fragmenta with an *HSoC* of $4.78 \approx 5$, ranked 4 in *HWF* due to limitations at the level of decomposition healthiness[15] and tool-supported formalisation[16], not significant in this model. MetaDepth's *HSoC* is good ($4.12 \approx 4$), ranked satisfactory in *HWF* — its formalisation is too focused on layering, neglecting relevant aspects, such as multiplicities.

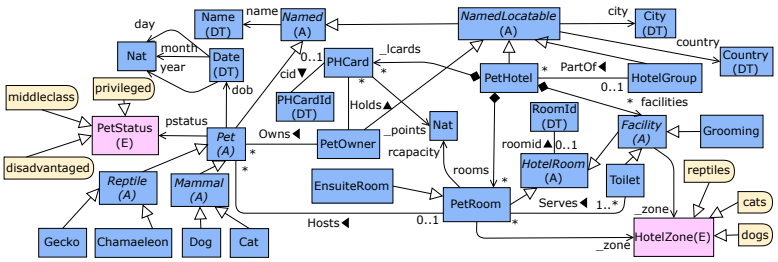
The rationale for table 7's evaluation of pet hotel (section 2), expressed in Fragmenta in Fig. 2, and monolith and partitioned UML in Fig. 29, is as follows:

- ILs of Fragmenta fragments $\texttt{F\_PW}_i$ of Fig. 2, where $i \in 1..8$, are, respectively, 40, 37, 42, 35, 51, 55, 24 and 46. Hence: $IL^{\bullet} = 55$ ($TE^{\bullet} = 5$), $\overline{IL} \approx 41$ ($\overline{TE} = 5$), $IL^{\Sigma} = 365$ (sum of all fragment ILs, plus extra diagram, plus IL of GFG), $IL/F = 365/8 \approx 46$ ($TE/F = 5$); overall TE is excellent. *E*, *DC* and *HWF* are excellent (5) as: (i) everything is expressed including constraints, (ii) mechanical composition and decomposition is effective, and (iii) the model's healthiness is checked thoroughly with a tool. Overall, *HSoC* is excellent (= 5).
- MUML's sole fragment (Fig. 29a) has a $IL^{\bullet} = \overline{IL} = 237$, but $IL^{\Sigma} = 342$ to account for the constraints expressed textually. Hence, TE is poor — $TE^{\bullet} = \overline{TE} = TE/F = 1$. *E* is unsatisfactory — *GER* = .7 and *TE* = 1, reflecting

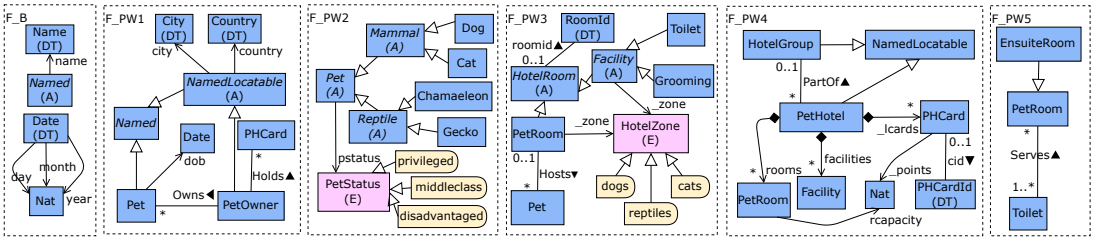
[15] Unlike Fragmenta, Merge UML lacks a notion of a well-formed fragment to ensure horizontal healthiness and correct compositions.
[16] Merge UML's tool is unable to check whether the source multiplicity of Fig. 1's composition is either 1 or 0..1 only, and whether multiplicities are well-formed as multiplicities are separate and external from the core graph-based representation.

| Problem | Approach | # *Fs* | $IL^{\bullet}$ | $\overline{IL}$ | $IL^{\Sigma}$ | $IL/F$ | $TE^{\bullet}$ | $\overline{TE}$ | $TE/F$ | $TE$ | $E$ | $DC$ | $HWF$ | **HSoC** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Hotel** | Fragmenta | 1 | 27 | 27 | 34 | 34 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5.00 |
| **World** | MUML | 1 | 27 | 27 | 27 | 27 | 5 | 5 | 5 | 5 | 5 | 3 | 2 | 3.90 |
| | PUML | 1 | 27 | 27 | 27 | 27 | 5 | 5 | 5 | 5 | 5 | 3 | 2 | 3.90 |
| | PkgUML | 1 | 27 | 27 | 34 | 34 | 5 | 5 | 5 | 5 | 5 | 3 | 2 | 3.90 |
| | MergeUML | 1 | 27 | 27 | 27 | 27 | 5 | 5 | 5 | 5 | 5 | 5 | 4 | 4.78 |
| | MetaDepth | 1 | 27 | 27 | 27 | 27 | 5 | 5 | 5 | 5 | 5 | 3 | 3 | 4.12 |
| **Pets** | Fragmenta | 8 | 55 | 41 | 365 | 46 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5.00 |
| **World** | MUML | 1 | 237 | 237 | 342 | 342 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1.22 |
| | PUML | 6 | 52 | 36 | 320 | 53 | 5 | 5 | 5 | 5 | 3 | 1 | 1 | 2.80 |
| | PkgUML | 6 | 52 | 36 | 342 | 57 | 5 | 5 | 4 | 5 | 3 | 2 | 1 | 3.02 |
| | MergeUML | 6 | 52 | 36 | 350 | 58 | 5 | 5 | 4 | 5 | 3 | 5 | 3 | 4.12 |
| | MetaDepth | 1 | 237 | 237 | 342 | 342 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1.22 |
| **Abstract** | Fragmenta | 3 | 25 | 15 | 57 | 19 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5.00 |
| **Commerce** | MUML | 1 | 29 | 29 | 40 | 40 | 5 | 5 | 5 | 5 | 4 | 3 | 2 | 3.68 |
| | PUML | 2 | 31 | 19 | 48 | 24 | 5 | 5 | 5 | 5 | 4 | 3 | 2 | 3.68 |
| | PkgUML | 2 | 31 | 19 | 56 | 28 | 5 | 5 | 5 | 5 | 4 | 3 | 2 | 3.68 |
| | MergeUML | 2 | 31 | 19 | 67 | 34 | 5 | 5 | 5 | 5 | 4 | 5 | 4 | 4.56 |
| | MetaDepth | 1 | 30 | 30 | 41 | 41 | 5 | 5 | 5 | 5 | 4 | 3 | 3 | 3.90 |
| **Concrete** | Fragmenta | 8 | 77 | 38 | 348 | 44 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5.00 |
| **Commerce** | MUML | 1 | 134 | 134 | 226 | 226 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1.22 |
| | PUML | 6 | 50 | 34 | 295 | 49 | 5 | 5 | 5 | 5 | 3 | 1 | 1 | 2.80 |
| | PkgUML | 6 | 50 | 34 | 310 | 52 | 5 | 5 | 5 | 5 | 3 | 2 | 1 | 3.02 |
| | MergeUML | 6 | 50 | 34 | 313 | 52 | 5 | 5 | 5 | 5 | 3 | 5 | 3 | 4.12 |
| | MetaDepth | 1 | 135 | 135 | 227 | 227 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1.22 |
| **Abstract** | Fragmenta | 1 | 42 | 42 | 50 | 50 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5.00 |
| **Statecharts** | MUML | 1 | 45 | 45 | 45 | 45 | 5 | 5 | 5 | 5 | 5 | 3 | 2 | 3.90 |
| | PUML | 1 | 45 | 45 | 45 | 45 | 5 | 5 | 5 | 5 | 5 | 3 | 2 | 3.90 |
| | PkgUML | 1 | 45 | 45 | 52 | 52 | 5 | 5 | 5 | 5 | 5 | 3 | 2 | 3.90 |
| | MergeUML | 1 | 45 | 45 | 45 | 45 | 5 | 5 | 5 | 5 | 5 | 5 | 4 | 4.78 |
| | MetaDepth | 1 | 45 | 45 | 45 | 45 | 5 | 5 | 5 | 5 | 5 | 3 | 3 | 4.12 |
| **Concrete** | Fragmenta | 5 | 58 | 37 | 222 | 44 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5.00 |
| **Statecharts** | MUML | 1 | 86 | 86 | 173 | 173 | 3 | 3 | 1 | 2 | 2 | 2 | 1 | 1.78 |
| | PUML | 3 | 37 | 29 | 164 | 55 | 5 | 5 | 5 | 5 | 2 | 2 | 1 | 2.80 |
| | PkgUML | 3 | 37 | 29 | 184 | 61 | 5 | 5 | 4 | 5 | 2 | 2 | 1 | 2.80 |
| | MergeUML | 3 | 37 | 29 | 187 | 62 | 5 | 5 | 4 | 5 | 2 | 5 | 3 | 3.90 |
| | MetaDepth | 1 | 86 | 86 | 173 | 173 | 3 | 3 | 1 | 2 | 2 | 2 | 2 | 2.00 |
| **Abstract** | Fragmenta | 1 | 34 | 34 | 42 | 42 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5.00 |
| **Into-SysML** | MUML | 1 | 34 | 34 | 34 | 34 | 5 | 5 | 5 | 5 | 5 | 3 | 3 | 4.12 |
| **ADs** | PUML | 1 | 34 | 34 | 34 | 34 | 5 | 5 | 5 | 5 | 5 | 3 | 3 | 4.12 |
| | PkgUML | 1 | 34 | 34 | 41 | 41 | 5 | 5 | 5 | 5 | 5 | 3 | 3 | 4.12 |
| | MergeUML | 1 | 34 | 34 | 34 | 34 | 5 | 5 | 5 | 5 | 5 | 5 | 4 | 4.78 |
| | MetaDepth | 1 | 34 | 34 | 34 | 34 | 5 | 5 | 5 | 5 | 5 | 3 | 3 | 4.12 |
| **Concrete** | Fragmenta | 10 | 69 | 41 | 459 | 46 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5.00 |
| **Into-SysML** | MUML | 1 | 266 | 266 | 314 | 314 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 1.44 |
| **ASDs** | PUML | 9 | 53 | 34 | 354 | 39 | 5 | 5 | 5 | 5 | 4 | 1 | 1 | 3.02 |
| | PkgUML | 9 | 53 | 34 | 385 | 43 | 5 | 5 | 5 | 5 | 4 | 2 | 1 | 3.24 |
| | MergeUML | 9 | 53 | 34 | 410 | 46 | 5 | 5 | 5 | 5 | 4 | 5 | 3 | 4.34 |
| | MetaDepth | 1 | 266 | 266 | 314 | 314 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 1.44 |
| **Concrete** | Fragmenta | 3 | 58 | 37 | 89 | 30 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5.00 |
| **Into-SysML** | MUML | 1 | 58 | 58 | 58 | 58 | 4 | 4 | 4 | 4 | 5 | 3 | 3 | 3.78 |
| **CDs** | PUML | 3 | 38 | 22 | 66 | 22 | 5 | 5 | 5 | 5 | 5 | 3 | 3 | 4.12 |
| | PkgUML | 3 | 38 | 22 | 79 | 26 | 5 | 5 | 5 | 5 | 5 | 3 | 3 | 4.12 |
| | MergeUML | 3 | 38 | 22 | 83 | 28 | 5 | 5 | 5 | 5 | 5 | 5 | 4 | 4.78 |
| | MetaDepth | 1 | 58 | 58 | 58 | 58 | 4 | 4 | 4 | 4 | 5 | 3 | 3 | 3.78 |

Table 7. Assessment of horizontal separation of concerns (HSoC) for the sample's nine models. *TE*, *E*, *DC*, *HWF* and *HSoC* measured on a five-point scale (Table 3b)




(a) Pets world model of Fig. 2 in monolith UML.

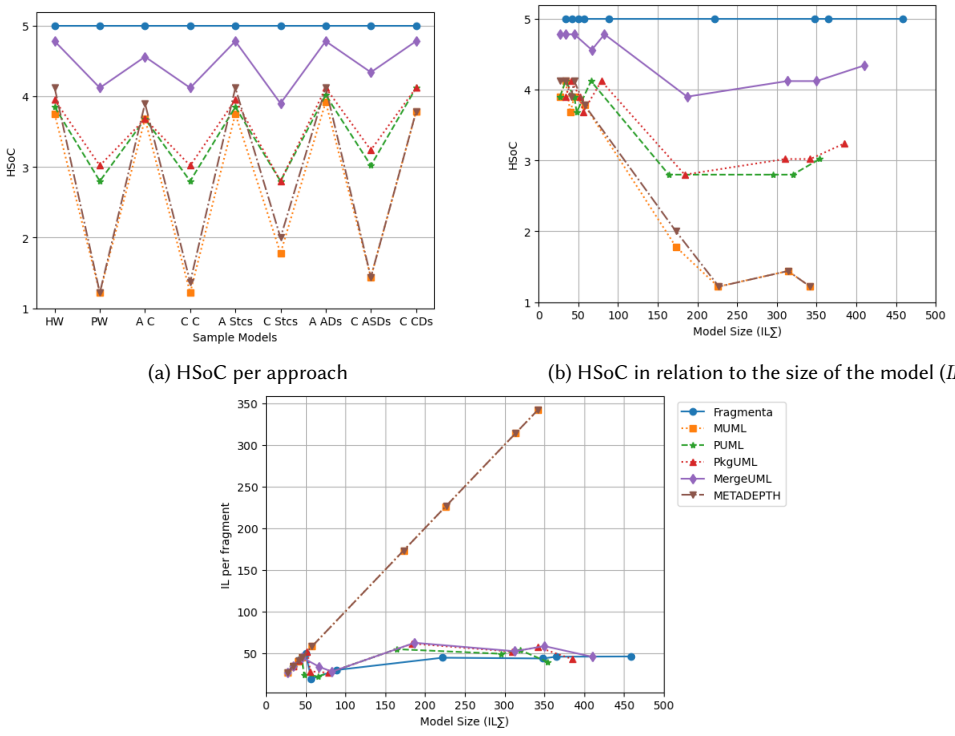



(b) Pets world model of Fig. 2 in partitioned UML.

Fig. 29. Pets-world model of Fig. 2 expressed in monolith and partitioned UML.

UML's inability to express constraints graphically. DC is poor due to lack of decomposition means much needed in this fairly large model. *HWF* is poor as model is large. Overall, *HSoC* is poor ($1.22 \approx 1$).

- In PUML (Fig. 29b), ILs of $\texttt{F\_PW}_i$, where $i \in 1..6$, are, respectively, 19, 39, 48, 52, 45 and 12. Hence: $\mathit{IL}^{\bullet} = 52$ ($\mathit{TE}^{\bullet} = 5$), $\overline{\mathit{IL}} \approx 36$ ($\overline{\mathit{TE}} = 5$), $\mathit{IL}^{\Sigma} = 320$, and $\mathit{IL}/F = 320/6 \approx 53$ ($\mathit{TE}/F = 5$); overall TE is excellent. *E* is satisfactory — *GER* = .57 and *TE* = 5. DC is poor (1) as a much needed mechanical underpinning is absent in a model of this size. *HWF* is poor as the model is fairly large and without a formal underpinning. Overall, *HSoC* is satisfactory ($2.8 \approx 3$).
- A PkgUML model would include a package diagram in addition to Fig. 29b. Hence: $\mathit{IL}^{\bullet} = 52$ ($\mathit{TE}^{\bullet} = 5$), $\overline{\mathit{IL}} \approx 36$ ($\overline{\mathit{TE}} = 5$), $\mathit{IL}^{\Sigma} = 342$, and $\mathit{IL}/F = 342/7 \approx 49$ ($\mathit{TE}/F = 4$); overall TE is excellent. Like PUML, *E* is satisfactory (3). *DC* is unsatisfactory as PkgUML's DC theory is deficient (see section 9) and the model is large. Like PUML, *HWF* is poor. Overall, *HSoC* is satisfactory ($3.02 \approx 3$).
- A MergeUML model would include an *interconnection diagram* relating manifestations of the same element across fragments in addition to Fig. 29b. Hence: $\mathit{IL}^{\bullet} = 52$ ($\mathit{TE}^{\bullet} = 5$), $\overline{\mathit{IL}} \approx 36$ ($\overline{\mathit{TE}} = 5$), $\mathit{IL}^{\Sigma} = 350$, and $\mathit{IL}/F = 350/7 \approx 58$ ($\mathit{TE}/F = 4$); overall TE is excellent. *E* is satisfactory (3), as PUML and PkgUML. *DC* is excellent (5) due to the underlying formal mechanics. *HWF* is satisfactory (3) due to the limitations of the formal underpinning, relevant for a model of this size. Overall, *HSoC* is good ($4.12 \approx 4$).
- A MetaDepth model would be similar to MUML (Fig. 29a). Because horizontal decomposition is not supported, the IL-based TE scores are low: $\mathit{IL}^{\bullet} = 237$ ($\mathit{TE}^{\bullet} = 1$), $\overline{\mathit{IL}} = 237$ ($\overline{\mathit{TE}} = 1$), $\mathit{IL}^{\Sigma} = 342$, and $\mathit{IL}/F = 342$ ($\mathit{TE}/F = 1$); overall TE is poor. Like MUML, *E* is unsatisfactory, and *DC* and *HWF* are poor (the formalisation is excessively focused on vertical layering, leaving crucial aspects out and the model is large). Overall, *HSoC* is poor ($1.22 \approx 1$).




(a) HSoC per approach

(b) HSoC in relation to the size of the model ($IL^{\Sigma}$)

(c) IL per fragment ($IL/F$) in relation to model size ($IL^{\Sigma}$)

Fig. 30. Plots of the evaluation of Horizontal Separation of Concerns (HSoC). Abbreviations: HW = Hotel World (Fig. 1), PW = Pets World (Fig. 2), A StCs = Abstract Statecharts (Fig. 20), C Stcs = concrete statecharts (Fig. 21), A ADs = abstract SysML-based architecture diagrams (Fig. 20), C ASDs = concrete SysML-based architecture structure diagrams (Fig. 24), C CDs = concrete SysML-based connection diagrams (Fig. 25).

The three plots of Fig. 30 provide different perspectives over table 7's data, namely: HSoC scores for each model of the sample (Fig. 30a), HSoCs with respect to a model's size or $IL^{\Sigma}$ (Fig. 30b), and relation between IL per fragment and $IL^{\Sigma}$ (Fig. 30c). The results ($N = 9$) are as follows:

- Fragmenta is consistently excellent (mean 5, standard deviation = 0); MergeUML ($m = 4.56 \approx 5$, $sd = .25$) is also excellent.
- MetaDepth ($m = 2.88 \approx 3$, $sd = 1.36$) and MUML ($m = 2.73 \approx 3$, $sd = 1.27$) are highly dependant on the model's size (Fig 30b) — $TE$ degrades considerably when the size increases.
- Pkg-UML ($m = 3.48$, $sd = .46$) is only slightly better than PUML ($m = 3.41$, $sd = .54$), and both better than MetaDepth and MUML because of their superior $TE$; however, performances still degrade with larger models due to lower $E$, $DC$ and $HWF$ which become more pertinent as models increase in size (Fig 30b).
- Figure 30c shows that all approaches, except MUML and MetaDepth, are able to control $TE$ as the size of the model increases, as $IL/F$, a good TE indicator, remains nearly constant. Hence, the clear differences in overall $HSoC$ are mainly due to mechanics and quality of what is conveyed, as per factors $E$, $DC$, and $HWF$, as discernable from Figs. 30a and 30b.

| | |
|---|---|
| Excellent (5) | Layering concerns articulated adequately, covering vertical abstraction (VA) and instantiation (or typing). Extra typing supported if needed. |
| Good (4) | Most layering concerns articulated adequately, covering both VA and instantiation (or typing). Extra typing needed but unsupported. |
| Satisfactory (3) | Layering concerns articulated partially. Either only instantiation supported and model is small ($IL^{\Sigma} < 90$), or VA supported but unconvincingly and model is at most medium-sized ($IL^{\Sigma} < 200$). |
| Unsatisfactory (2) | Layering concerns articulated partially. Either only instantiation supported and the model is medium-sized ($90 \leq IL^{\Sigma} < 200$), or VA supported unconvincingly and model is large ($IL^{\Sigma} \geq 200$). |
| Poor (1) | Layering concerns only partially articulated. Only instantiation supported. VA unsupported but important as model is large ($IL^{\Sigma} \geq 200$). |

Table 8. Criteria for layering description (LD)

### 7.4 Vertical Separation of Concerns (VSoC): criteria and results

Concerned with vertical organisation, VSoC provides layering: models constitute layers, kept separate and independent but related through element mappings to other layers, analogous to the paper's graph theory: layers are graphs related through morphisms.

The evaluation assesses how the examined approaches fare against Fragmenta, designed to facilitate layering through abstraction (or refinement), typing and extra typing. The sequel presents evaluation factors and criteria, the overall score formula, and VSoC's evaluation results.

*7.4.1 Factors.* VSoC is evaluated on the sample models by appraising each approach on table 3b's 5-point scale with respect to the following factors:

- **Layering description (*LD*).** This ascertains the adequacy of the linguistic and conceptual support for abstraction (construction of abstract layers above), refinement (concrete layers below) and instantiation (or typing). LD's criteria, set out in table 8, uses the measure $IL^{\Sigma}$.
- **Vertical formality and checking (VFC).** This assesses the formal underpinning of the vertical machinery and its capacity to check both the healthiness of layered models, and the validity of model instances with respect to type models according to table 9's criteria.

*7.4.2 Overall Score.* VSoC's overall score is a weighted sum with 40% for LD and 60% for VFC :

$$VSoC = LD \times .4 + VFC \times .6$$

VFC's 60% is higher than LD's 40% to reflect the higher importance attributted to formality and mechanical verification.

*7.4.3 Results.* Figure 31 summarises the results. Table of Fig. 31a provides the average scores for the four layered models (pet world, e-commerce, statecharts and Into-SysML as per table 2), plotted in Fig. 31b. A summary of VSoC's evaluation is as follows:

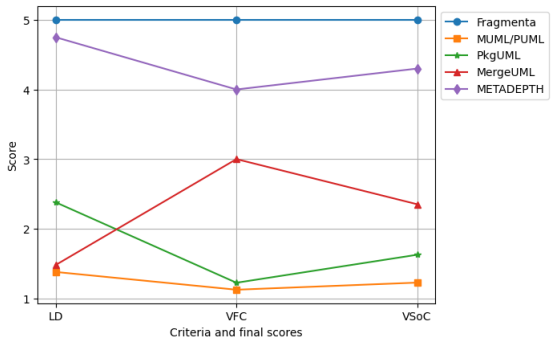
- Fragmenta's *LD* and *VFC* are excellent across the four models. Layering is fully and adequately articulated, including the extra typing. Refinements, extra typing and instantiations are checked using Fragmenta's tool. The overall score is excellent (**VSoC** = 5).

| | |
|---|---|
| Excellent (5) | Layering's formal underpinning, supporting instantiation and vertical abstraction (I & VA), is sound and tool-supported enabling thorough checking of the: (i) healthiness of a model's vertical layering, and (ii) validity of object (or instance) models with respect to any type of model higher up in the vertical hierarchy. A few pertinent properties ($\leq 5$) may be left unchecked provided the main model is small ($IL^{\Sigma} < 90$). |
| Good (4) | Layering's formal I & VA underpinning is tool-supported, checking both vertical healthiness and instance validity. A few pertinent properties ($\leq 5$) are left unchecked and the main model is at least medium-sized ($IL^{\Sigma} \geq 90$). |
| Satisfactory (3) | Layering's formal underpinning is tool-supported, but checks instantiation only. Only two vertical layers supported: instance and model. Several pertinent properties or aspects are unchecked. |
| Unsatisfactory (2) | Layering's semi-formal linguistics for I & VA enable manual instance validity checking only, manageable as main model is small ($IL^{\Sigma} < 90$). |
| Poor (1) | Layering is semi-formal, if supported at all. Manual instance validity checks are difficult to manage as main model is at least medium-sized ($IL^{\Sigma} \geq 90$). |

Table 9. Criteria of VSoC Formality and Checking (FC)

| Approach | *LD* | *VFC* | **VSoC** |
|---|---|---|---|
| Fragmenta | 5.0 | 5.0 | **5.00** |
| MUML | 1.4 | 1.1 | **1.23** |
| PUML | 1.4 | 1.1 | **1.23** |
| PkgUML | 2.4 | 1.1 | **1.63** |
| MergeUML | 1.4 | 3.0 | **2.35** |
| MetaDepth | 4.8 | 4.0 | **4.30** |

(a) Assessment of VSoC



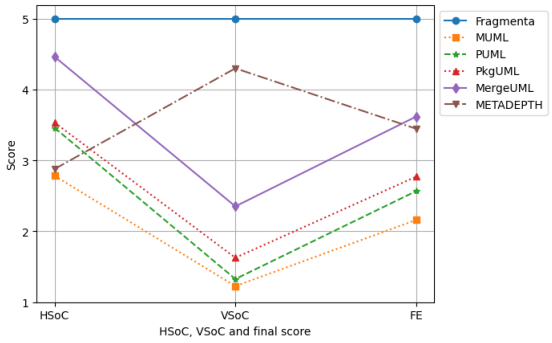

(b) Evaluation scores for VSoC

Fig. 31. Evaluation results of vertical separation of concerns (VSoC). Variables: *LD* = layering description, *VFC* = vertical formality and checking, *VSoC* = overall VSoC. Criteria assessed on a 5-point scale: 1 = poor, 2 = unsatisfactory, 3 = satisfactory, 4 = good and 5 = excellent.

- MUML and PUML's LD scores oscillate between poor and unsatisfactory with an average of 1.4 ($\approx 1$) as vertical abstraction is unsupported and the linguistics of instantiation is semi-formal. *VFC* is also deemed poor on average ($= 1.1 \approx 1$) as layering checks are manual. Overall, score is poor (**VSoC** $= 1.23 \approx 1$).
- PkgUML gets an average score of unsatisfactory ($2.8 \approx 3$) in *LD*, better than PUML and MUML due to its support for package refinement. Like MUML and PUML, its *VFC* average score is poor ($= 1.1 \approx 1$). Overall score is unsatisfactory (**VSoC** $= 1.93 \approx 2$).
- MergeUML is deemed unsatisfactory in *LD*, as PUML and MUML for the same reasons. *VFC*'s average score of satisfactory ($= 3$) is due to the formal and tool-supported instantiation checks. Overall score is unsatisfactory (**VSoC** $= 2.35 \approx 2$).
- METADEPTH's *LD*, excellent in all layering examples except Into-SysML connection diagrams as extra-typing is unsupported, yields an average score of excellent ($= 4.8 \approx 5$); the vertical abstraction scheme based on typing tends to be more succinct than Fragmenta's refinement. *VFC* is good ($4.3 \approx 4$) as: (i) there is a formal

| Approach | *HSoC* | *VSoC* | **FES** |
|---|---|---|---|
| FRAGMENTA | 5.00 | 5.00 | 5.00 |
| MUML | 2.78 | 1.23 | 2.16 |
| PUML | 3.46 | 1.23 | 2.57 |
| PkgUML | 3.53 | 1.63 | 2.77 |
| MergeUML | 4.46 | 2.35 | 3.62 |
| METADEPTH | 2.88 | 4.30 | 3.45 |

(a) Evaluation perspectives and final evaluation score




(b) Plot

Fig. 32. Overall evaluation highlighting the two perspectives, HSoC and VSoC, and final evaluation score (FES). All evaluations assessed on a 5-point scale with values 1 (poor), 2 (unsatisfactory), 3 (satisfactory), 4 (good) and 5 (excellent). Abbreviations: HSoC = Horizontal separation of concerns; VSoC = Vertical separation of concerns; FE = final evaluation.

underpinning based on graph morphisms with a (ii) notion of well-formedness which is tool-supported, but (iii) instantiation checking, excessively focussed on vertical abstraction and potencies, is limited as many relevant aspects are left out. Overall score is good (**VSoC** = 4.3 ≈ 4).

### 7.5 Overall Evaluation

*FE*, the overall evaluation score, is a weighted sum obtained from *HSoC* and *VSoC*:

$$FE = HSoC \times .6 + VSoC \times .4$$

*HSoC*'s 60%, higher than *VSoC*'s 40%, is due to *HSoC*'s higher prevalence, as it concerns what the models express and the fragmentations involved in controlling complexity, which are major modelling concerns in the current practice; *VSoC* is about relations between models at different levels of abstraction, important to control complexity, and for abstraction and comprehensibility, but perhaps less important in the current practice.

Figure 32 presents the overall evaluation accounting for *HSoC* and *VSoC*, and *FE*. Table of Fig. 32a gives the evaluation results, plotted in Fig. 32b, which are as follows:

- FRAGMENTA, the top-scorer, is appraised consistently as excellent (*FE* = 5).
- MergeUML [58, 59], the runner-up, was good overall (*FE* = 3.78 ≈ 4). Excellent in HSoC, but only satisfactory in *VSoC*.
- METADEPTH [21, 21] was also good overall (*FE* = 3.57 ≈ 4). It was excellent in *VSoC*, but only satisfactory in *HSoC*.
- PUML (*FE* = 2.69 ≈ 3) and PkgUML (*FE* = 2.86 ≈ 3) were both satisfactory overall, whereas MUML was unsatisfactory (*FE* = 2.28 ≈ 2).

### 7.6 Evaluation Questions Revisited

An analysis of the evaluation questions under the light of the results highlights that FRAGMENTA answers EQ1 (Is FRAGMENTA's modelling approach supported by its novelties better than competing approaches with respect to horizontal

and vertical decomposition, formality and graphical expressiveness?) positively as it was consistently and clearly better than its competitors overall.

An in-depth analysis of the comparative benefits of Fragmenta's novelties highlights that:

- Proxies, a key HSoC ingredient, underpin the key decomposition theorem from [7], a pillar upon which Fragmenta's well-formedness notion for fragmented models is built. Fragmenta's main HSoC competitor, MergeUML, lacks this primitive, hence, it does not guarantee the well-formedness of compositions of fragmented models. This is one of the reasons why Fragmenta's results in HWF – $mean = 5, sd = 0$— were better than MergeUML's — $mean = 3.56 \approx 4, sd = .53$ —, especially for large models.
- Virtuals, the light inheritance packs, proved to be an interesting feature which provided only a negligible advantage in the evaluation as the other approaches could circumvent the lack of virtuals in most cases with either abstract classes or more definitions, with only a negligible increase in the underlying linguistics.
- Fragmenta's graphical constraints, provided by derived, value constraint and path edges, provided an advantage. Fragmenta's HSoC's expressiveness results — $mean = 5, sd = 0$ — were better than those of the runner-up MergeUML — $mean = 4, sd = 1.12$.
- Fragmenta's VSoC novelties also provided an advantage. In LD, Fragmenta — $mean = 5, sd = 0$ — was slightly better than the runner-up METADEPTH — $mean = 4.75, sd = 0.5$ — mainly due to Fragmenta's extra typing. In formality and checking, Fragmenta — $mean = 5, sd = 0$ — was better than METADEPTH — $mean = 4, sd = 0$.

### 7.7 Threats to Validity

The sequel discusses limitations and biases that may affect the reliability, integrity and generalisability of the evaluation results given herein.

*7.7.1 Internal Validity.* Considerations on whether extraneous factors could have interfered with the evaluation results, raise the question of the adequacy of the sample of approaches Fragmenta was compared against, which could be biased somehow. The selection rationale considered the need for: (i) a representation for the current practice which would set the benchmark, and (ii) approaches that innovate and compete with Fragmenta. The current practice is represented by three UML flavours: (i) traditional monolith UML, the way UML is mostly taught and practiced, (ii) partioned UML, a dialect used in the advanced UML literature, based on ad hoc fragmentation, and (iii) a more modern UML with linguistic support for fragmentation through packages. The innovative approaches included Sabetzadeh et al [58, 59] and METADEPTH [20, 21], both appraised by the author as the most advanced competing approaches in HSoC and VSoC, respectively.

*7.7.2 Construct Validity.* Because the evaluation was designed and undertaken by the author who himself appraises the merits of Fragmenta, his work, the measurements could be prone to bias, especially if the evaluation criteria is overly subjective. This issue was addressed by objectifying the evaluation criteria, which provides only a little margin of subjectivity, in some cases none at all; this diminishes the possibility of evaluator disagreements considerably but not entirely. Another source of bias could lie in the weights that underpin the scores of HSoC, VSoC and overall. Criteria and weights underpinning evaluation scores are all based on the judgments of the author, hence, they have a certain inherent subjectivity; they have been made explicit, for the sake of rigour and transparency, to facilitate posterior scientific scrutiny.

It is important to acknowledge the limitations and assumptions of the evaluation's measurements. In HSoC, a prominent factor is TE, based on notions of information load and limits of human information processing [46], and notions of modularity and separation of concerns and how decomposition favours comprehensibility [61]. TE is HSoC's highest-weighted factor, 34%, because it is the most objective and the one reflecting closely what HSoC is about. The results suggest that the TE measurements make sense — good decompositions leading to high TE scores and bad ones leading to low ones. However, this is perhaps not as accurate as a user controlled experiment in which users say what is and what is not working for them.

Another criteria whose assumptions need acknowledging is HSoC's expressiveness (E) and its graphical expressiveness focus. It is in graphical expressiveness that notable novelties of Fragmenta lie; this graphical emphasis is grounded on the cognitive benefits of pictorial representations [42]. However, it is not always the case that the graphics are better [42]. Hence, the results of the evaluation, with its graphical expressiveness focus aimed at appraising Fragmenta's graphical novelties, should not be interpreted as better expressiveness in general nor better usability; a proper appraisal of these qualities would require another study.

*7.7.3 External Validity.* The inadequacy of the models used in the evaluation could put the generalisability of the results into question. Apart from the model of section 2, developed for the purpose of this paper, all other models originate from elsewhere and have a certain inherent degree of external validity; the model of section 4 is derived from a benchmark example of multi-level modelling with added complexity, the model of section 5 is simply a model of statecharts [33], a popular graphical language, and the model of section 6 was initiated in the context of an European Union research project. The evaluation results are based on these four models.

## 8 DISCUSSION

### 8.1 Fragmenta's Graph-based Underpinning

Fragmenta captures the richness of data modelling as structural graphs (SGs), which build upon simple graphs. SGs support multiplicities and inheritance; graph colouring accounts for diversity. SGs' revised morphisms — between SGs (vertical refinement) and from graphs to SGs (typing) — cater for inheritance by allowing edges to have more than one source and target nodes — an edge can be the source of a particular node and all its descendants. Fragments builds up on SGs by adding referencing; they are composed and reduced to SGs through subsumption.

### 8.2 Fragments, GFGs and Models

Fragments, defined from SGs, provide confinement but allow intra- and inter-node connections through proxies. Fragments deal with matter at the local level; proxies are local referencing artifices to connect to nodes from the overall model; the referencing aids the model's logic by providing the provenance of a proxy and is crucial for mechanical composition. GFGs (Global Fragment Graphs) take a more global stance, saying how the different fragments of a model reference each other; GFG nodes represent fragments, edges indicate that one fragment references another. A model is a GFG together with an indexed set of fragments. Instances are represented in Fragmenta as graphs with typing; the paper provided examples where instances take the form of statecharts (section 5), Into-SysML models (section 6), and products of an e-commerce system (section 4).

### 8.3 Virtuals

Virtuals embody a novel lighter inheritance scheme. Like *abstract* nodes, they lack a direct existence as they have no direct instances and represent something inheritable; unlike their abstract counter-parts, virtuals put aside strong lineage or ancestry, in favour of weaker parenthood to overcome the limitations of the prevalent single inheritance — in Fragmenta, a node may inherit from at most one abstract node, but may inherit from many virtuals. Virtuals embody the idea that a node may only have one strong parent, but may have traits in common with weaker parents. Metaphorically, virtuals are like cultural traits possessed and shared by members of a culture. Virtuals can also represent ad-hoc groupings of unrelated nodes as disjoint unions. Virtuals were key in the paper's main Fragmenta illustrations (sections 2 and sections 4 to 6) to represent union types and anything inheritable, allowing a range of classes to inherit from a pool of inheritable things. In Fig. 2, `Named`, widely inherited, represents something with a name; in Fig. 19b `Volume` is a disjoint union.

### 8.4 Graphical Constraints

Together, derived, constraint and path edges, make up Fragmenta's support for graphical constraints, a noteworthy novelty. Derived edges express extra multiplicities; in Fig. 2, derived edge of fragment `F_PW5` specialises edge `Serves` — pet rooms have at least one toilet, but ensuite rooms have exactly one. Constraint-value edges constrain the values of either nodes or association edges; in Fig. 2, fragment `F_PW1` says that: the values of $\texttt{Nat}_1$ are greater or equal than 1, and edge `day` of node `Date` takes natural number values less or equal than 31. Path edges describe diagram commutings; in Fig. 2, fragment `F_PW7` says that a toilet's zone must be the same as the zone of the rooms it services (the two paths must commute).

### 8.5 Patterns

Virtuals and derived edges give rise to interesting patterns. The paper identified three Fragmenta patterns: (i) *virtual traits*, (ii) *flexible relations* and (iii) *templates*. *Virtual traits* makes relations inheritable by a number of classes through a virtual; the relation may be a manifestation of *flexible relations* — a unary association on a virtual allowing for a variety of configurations —, as is the case in Fig. 1 and the other three major illustrations (sections 4 to 6). These two patterns usually concern abstract layered modelling to prescribe a structure for refinements, but *flexible relations* is applicable in other settings as well.

*Templates* represent generic things, instantiable in a model. This relies on virtuals to represent the generic parts, inheritance for instantiation, and derived edges for refinement of inherited relations, as illustrated in Figs. 7f and 7g, where the virtual `CPair` represents a pair made up of two generic components, which are instantiated differently.

### 8.6 Horizontal and Vertical Refinement

Horizontal refinement gives rise to partitionings (decompositions, or fragmentations). It was introduced in [7], but enhanced here with *fragment resolution* (or composition) to stitch fragments mechanically along proxies through subsumption. It inherits from [7] the restrictions of proxies, namely: (a) proxies may not inherit, and (b) fragment references must be one-way only.

Vertical refinement, introduced here, complements horizontal refinement. It provides a novel vertical *layering* inspired by data refinement [35] based on graph notions of morphisms and commuting. A model refines another if: (i) there is a valid morphism from the concrete model to its abstract counter-part, and (ii) the concrete model satisfies certain

compliance conditions related to multiplicities, and node and edge types. Vertical refinement follows principles and ideas of data refinement, rather than being directly derived from it. *Narrowing* is such an example; refinements narrow down the denoted set of states of their abstract counter-parts, which is reflected in the rules: (i) total refinements should not exclude things from the abstract model (only partial refinements may do so); and (ii) refinement multiplicities are either narrowed or remain the same. Fragmenta's vertical refinement affects state spaces, differing from general data refinement which has a behavioural dimension.

### 8.7 Multilevel modelling

Through refinement, distinguished from typing in Fragmenta's upgrade presented here, Fragmenta puts forward a multilevel modelling architecture based on two strata: *type* and *instance*. The type (or meta) stratum may be further divided into layers; the instance stratum is indivisible. Type layers represent models at different levels of abstraction, from abstract to concrete related through the transitive refinement relation, a morphism with special characteristics focused on elaboration. Type and instance strata are related through typing, also defined as a morphism. Fragmenta's multi-level modelling novelty relies on refinement's ability to relate models vertically, underpinned by the transitive refinement relation, an alternative to the overused instantiation relation as a means to relate models [1, 2]. In addition, Fragmenta allows partial extra typings: models of the type stratum may have some of its elements typed by another model.

The pets world example of section 2, together with e-commerce, statecharts and Into-SysML of sections 4 to 6 illustrate multilevel modelling. The abstract metamodel coveys an essential structure that must be complied with by concrete metamodels, providing vertical separation of concerns. Fragmenta's typing allows graphs typed by fragmented metamodels, as shown with the statechart examples of section 5, whose correctness can be checked with Fragmenta's tool. Fragmenta's extra typing allows type elements to be instances of elements in another model, as illustrated in Into-SysML model of section 6, where elements of CDs have extra types in the model of ASDs.

Multilevel modelling serves several useful purposes: (i) *stepwise development*, where the abstract model acts as the precursor to the concrete model; (ii) *separation of concerns*, as abstract and concrete models may be carefully articulated to focus on different things; (iii) *abstraction*, as the abstract model provides an abstraction which may be useful for reasoning, verification and explanation.

To ease the description of morphism mappings which underpin typing and refinement, the morphism definitions inherited from [7] were improved to abstain from mapping inheritance edges.

### 8.8 Separation and Composition

Separation (or decomposition) constitutes the means through which complexity is managed horizontally. Composition is grounded on two major mechanisms: (i) *fragment union*, which joins fragments without altering their contents, and (ii) *fragment resolution*, which replaces proxies by their referents. Both are used to produce the overall model composition, a monolith: (i) the model's fragments are joined using union, and then (ii) the proxies of the union fragment are resolved.

The composition theorem, proved in [7], says that the composition of a model's fragments is well-formed provided fragments are individually well-formed — or, if a fragmented model is well-formed then its composed monolith is well-formed also. This is relevant to Fragmenta, which relies on fragment composition or resolution to reduce fragments to SGs. The theorem imposes two restrictions: (i) the fragment's referencing is one-way only, and (ii) proxies must not inherit. In practice, these restrictions are not severely limiting,

### 8.9 Method, Formalisation and Proofs

Fragmenta's mathematics was developed using a combination of notations and tools. The theory was specified in Z [37, 60, 64] with the support of the CZT type-checker [45, 47], to ensure consistency with respect to names and types. Z, a high-level mathematical language based on the Zermelo-Fraenkel set theory, supports sets, functions, relations and predicate logic. The mathematics presented here was derived from, and is very close to, the Z specification [5]. The Isabelle proof assistant [63][17] was used to prove the theory's theorems or laws (see appendix A). Z provided high-level mathematical expressivity, with support for consistency and well-formedness checking; Isabelle was mainly used to prove theorems. Formal specification in Z and proof of theorems in Isabelle contributed to the quality of the work presented here.

### 8.10 Graphical Expressiveness

Fragmenta emphasises graphical modelling, maximising graphical expressivity to avoid extra textual constraints. The evaluation (section 7) confirmed Fragmenta's cutting-edge graphical expressivity. This is positive and follows from studies that endorsed the benefits of pictorial representations [42], which have been influential in computer science [33, 34]. However, graphical is not necessarily better, hence, Fragmenta's superior graphical expressivity does not necessarily entail neither superior expressivity in general, nor improved usability. However, [6] highlighted benefits of graphical software modelling, namely in reading and comprehensibility.

The different Fragmenta constraints express things differently. Derived edges enhance classical association-multiplicity constraints. Value-constraint edges express graphically what is known as relational or boolean expressions involving operators (e.g. $2 \leq 3$). Path constraints express the commuting constraints used in graph and category theories.

Fragmenta makes some expressivity gains through virtuals, which go beyond classical inheritance to factor more commonality than what is usually done, and proxies as means to express decomposition joints. Most gains, however, come from constraint edges (derived, value-constraint and path) which extend the reach of data modelling's graphical expressivity.

### 8.11 Fragmenta Tool

The Fragmenta tool is an Haskell [15, 36, 44] implementation of the Fragmenta theory presented herein[18]. It enlivened Fragmenta and increased the reliability of the paper's results. The tool can: (i) check the well-formedness of graphs, SGs, fragments, GFGs and models; (ii) resolve fragments and models using subsumption; (iii) check the correctness of typings and refinements, including partial extra typings; and (iv) produce graphs processable by the Graphviz tools [27, 31, 32][19]. The tool enabled experimentation and contributed significantly to Fragmenta's quality; it checked all Fragmenta models and examples presented herein.

## 9 RELATED WORK

Fragmenta, initiated in [7], upgraded and redefined here, is a mathematical theory of a graphical data- and meta-modelling framework that tackles scalability and graphical expressivity. Metamodelling, regarded as important for model-driven development [2], is based on graph typing, founded on the graph morphism notion. Scalability, underpinned by

[17] https://isabelle.in.tum.de/
[18] The Haskell code is available at https://github.com/namalio/Fragmenta/.
[19] https://graphviz.org/

ideas related to separation of concerns [54, 61], is tackled through horizontal and vertical refinement, both mechanisms of modularity and abstraction, respectively. Fragmenta's application to a SysML profile for cyber-physical systems in [10, 11] is revisited in section 6; Fragmenta's new features generate a metamodel redesign where: (i) an abstract model captures the commonalities of two different diagram types, (ii) constraints are expressed graphically (they were expressed textually in [10, 11]) and (iii) deals with the profile's extra typing needs, whereby elements of a connection diagram are instances of elements of an architecture structure diagram, not dealt with in [10, 11].

The work presented here extends and improves [7] in the following ways:

- Fragmenta is augmented with vertical refinement inspired by data refinement [35] to enable the construction of layered data models in a sound manner.
- Abstract expressiveness is enhanced with virtual nodes to represent things with an inherent degree of variability (e.g. virtual traits patterns discussed in section 8.5).
- Refinement is distinguished from instantiation (or typing), both based on graph morphisms. Refinement, concerned with elaboration, checks whether a concrete model is an acceptable elaboration of its abstract counter-part. Typing concerns model instantiation; it checks whether instances comply with the restrictions imposed by their types.
- To enhance both vertical and horizontal refinement, models may have extra typing relations with other models within the same layer.
- Proxies, introduced in [7] and inspired by VCL's *references* [6, 8, 9], are enhanced with *fragment resolution*, a composition mechanism which replaces proxies by their referents that is theoretically underpinned by pushouts of category theory [26, 41], and is here defined as graph subsumption — simpler than classical solutions which refine pushouts as quotients. This entailed a redefinition of Fragmenta's higher-level constructions [7] — refinement and typings at the level of models and fragments are reduced to SGs via resolution.
- Horizontal refinement is considerably improved: (i) global fragments graphs (GFGs) are simplified to support `references` edges only, allowing both top-down and bottom-up design; (ii) revised morphisms preclude inheritance from the mappings ([7] required self-inheritance dummy edges); (iii) the cluster layer of [7] was removed in favour of a simpler two-layered architecture based on fragments and GFGs.
- Virtuals, inspired by disjoint unions and *soft parents* [4], enhance inheritance to express disjoint unions and ancestry-free inheritable properties.
- The emphasis on graphical expressiveness is inspired by [42] and the inspiration that it induced in computer science [33, 34], and VCL's positive empirical results [6].
- Derived edges specialise edge multiplicity constraints. General properties may be stated as relation edges and specialised with derived edges, involving edge paths, a composition of one or more edges. The pets example of section 2 says that, in general, toilets are shared by several pet rooms (expressed as relation edge), except ensuite rooms which have one single toilet which is not shared (expressed as derived edge). As discussed in section 8.5, virtuals together with derived edges give rise to interesting patterns enabling the expression of templates.
- As illustrated in the paper, value-constraint and path edges provide further means to express constraints graphically. Value-constraint edges are inspired by VCL's *derived edges* [6, 8, 9], and path edges, which, like derived edges, also affect whole edge paths, by the commuting notion, used widely in graph and category theories.

- The Haskell tool, which implements and supports the theory presented here, verifies the well-formedness of models, the correctness of typings and refinements, and produces model compositions. It verified all the paper's examples.

The sequel analyses related work in graph-based horizontal refinement, complexity management of graphical data modelling, multilevel modelling, and UML modularity and composition.

### 9.1 Graph-based theories of Horizontal Refinement

The underpinning of Jurack and Taentzer [39, 40] is, like Fragmenta, graph-based. Its two-layer structure resembles Fragmenta; their local level based on graphs with inheritance and containment (IC-graphs) is similar to Fragmenta's SGs, but uses import and export interfaces to enable composition; their global level is akin to Fragmenta's GFGs; their typing foundation is graph morphisms as Fragmenta. As acknowledged by [40], inheritance cycles (a malformation) may arise when parts are composed, avoided in Fragmenta through the proved union composition law [7], which ensures preservation of inheritance well formedness provided some constraints are met. Despite the similarities and common foundation, Fragmenta goes further than [39, 40] with respect to: (i) support for both fragmentation and layering (or horizontal and vertical refinement, respectively), (ii) fragment composition through fragment union and resolution (pushout-based composition of [39, 40] is not actually specialised and defined), (iii) edge multiplicities, (iv) novel concepts, such as proxies, virtual nodes, graphical constraint edges edges, and (v) an implementation.

### 9.2 Complexity Management of Graphical Data Modelling

Graphical data modelling, rooted in seminal works of database design such as entity relationship modelling [19], was taken-up by class modelling within the object-oriented paradigm [14] and subsequently applied to metamodeling to describe the abstract syntax of graphical languages. Metamodelling, Fragmenta's main application so far, is exemplified here with the metamodels of statecharts (section 5) and the Into-SysML profile (section 6).

Complexity management of data models, researched for over four decades [28, 34, 48–50], tackles the scalability issues arising in large data models [28, 49] caused by the limits in the human processing of information [46]. The proposed solutions to this problem reduce the amount of information, typically through abstraction or separation, which correspond to vertical or horizontal refinement, respectively.

Feldman and Miller [28]'s predominately vertical approach for entity-relationship diagrams (ERD) can be used in either a bottom-up or top-down fashion, making use of both abstraction and decomposition. Teory et al [62] elaborates [28] through a bottom-up method to identify abstractions. Moody [48, 49] changes the direction pursued until then towards horizontal refinement; a context data model indicates the main subject areas or fragments; each subject area has its own data model; proxies (*foreign entities* in [48, 49]) establish cross-subject relations.

These works are essentially methods with limited semi-formal linguistic support, stipulating how things are broken apart (decomposition), but eluding composition. Fragmenta, on the other hand, is mathematically defined with an underpinning theory of vertical refinement, decomposition through proxies, and composition through fragment union and resolution. Moody [48, 49]'s linguistic construct, a possible precursor of Fragmenta's proxy, is not accompanied by a mathematical study or implementation which delves into the construct's properties and intricacies, like is done in [7] and continued here; Fragmenta supports layering as [28], left out of [48, 49]. Moody's context data model, absent in [28], is akin to Fragmenta's GFG.

### 9.3 Multilevel Modelling

Metamodelling values multilevel modelling [1, 2, 12, 20, 22], seen as underpinning the relation between models and metamodels. Fragmenta opens up a new path in this area. Layering based on typing, which emphasises inheritance of explicitly stated properties — concrete layers inherit class properties of their abstract counter-parts —, gives way to refinement, which emphasises abstract modelling and the establishment of a structure or organisation which concrete layers must comply to. It is not explicitly-stated properties that are inherited — in the e-commerce example of section 4, such properties where inherited (or shared) through fragment reuse —, but some fundamental architecture capturing some essence of the modelled problem.

Stratification, the layered organisation of models, is a characteristic of multilevel modelling approaches. Two distinct models can be discerned, *stratified* and *meshed*, corresponding, respectively, to the *strict* and *loose* approaches to metamodelling of [12]. Fragmenta and MetaDepth [20, 21] are stratified — layers are separate descriptions related through morphisms. DeepTelos [38] and ML2 [29] are meshed — layering is embedded in a single description as relations within the same model, which implies absence of actual separation. The stratified model is more true to layering and separation of concerns. The meshed model can be criticised for being essentially uni-level modelling, as it relies on intra-model relations; the meshed model's alleged superior flexibility comes at the cost of added complexity and reduced conceptual clarity.

Multilevel modelling approaches relate layers differently. MetaDepth, DeepTelos and ML2 rely on instantiation. Fragmenta resorts to refinement in addition to instantiation, used to relate type and instance strata and for extra-typing. The sole reliance on instantiation entails a dual interpretation of meta-classes; all approaches except Fragmenta follow the *clabject* model [1, 3, 12] — certain elements have a dual facet: classes for levels below, and objects or instances for levels above.

Instantiation depth control is a notable concern. Potency-based modelling [1, 3] relies on a *potency* or *level* attribute, a natural number prescribing instantiation depth — 0 means that a clabject is an instance, hence, not instantiatable. Potency-based MetaDepth [20, 21] relates levels through graph morphisms, which indicate the distance between the related levels (adjacent levels have a distance of 1). Fragmenta does not control instantiation depth; like MetaDepth, it relates layers through graph morphisms, but allows adjacent layers only; classes retain their class semantics irrespective of their level and may be instantiated provided they are not *ethereal* (neither abstract nor virtual); objects can only inhabit the instance stratum which cannot be further instantiated. DeepTelos and ML2 are rather unclear about the meta-levels involved. DeepTelos uses the concept of most general instances to define hierarchies of what are essentially types (or classes); ML2 also relies on type hierarchies; both DeepTelos and ML2, like Fragmenta, rely on chains of related elements; in DeepTelos and ML2, chains are grounded on the instantiation relation, whereas Fragmenta uses refinement. Instantiation depth control is, therefore, either based on *potencies* or *chained*.

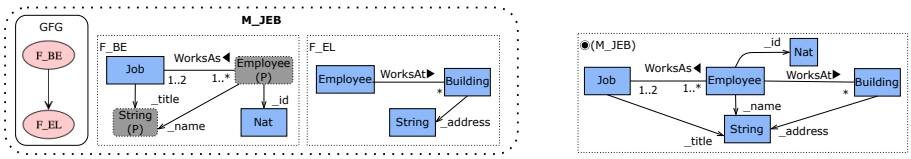
Depth may be achieved in either a *transitive* or *mixed* fashion. Fragmenta is transitive as it relies on the transitivity of both refinement and morphism composition — elements from non-adjacent layers may only be indirectly and transitively related. MetaDepth, ML2 and DeepTelo are mixed as they provide direct, and indirect and transitive means.

All analysed approaches have a mathematical underpinning. Fragmenta and MetaDepth are based on algebraic graph theory, ML2 and DeepTelos on predicate logic.

Table 10 contrasts the analysed approaches with respect to stratification, the supported layering types (either refinement or instantiation), instantiation control, depth and formalism. With respect to separation as a means to

| Research Work | Stratification | Inter-level relations | Instantiation control | Depth | Formalism |
|---|---|---|---|---|---|
| Fragmenta | stratified | instantiation and refinement | chained | transitive | graphs |
| MetaDepth [20, 21] | stratified | instantiation | potencies | mixed | graphs |
| DeepTelos [38] | meshed | instantiation | chained | mixed | predicate logic |
| ML2 [29] | meshed | instantiation | chained | mixed | predicate logic |

Table 10. Classification of multilevel modelling approaches, according to stratification (*stratified* or *meshed*), layering (*instantiation*, or *instantiation and refinement*), instantiation depth control (*potencies* or *chained*), depth (*transitive* or *mixed*) and formalism (*graphs* or *predicate-logic*).




(a) Model Employee

(b) Fragment composition resulting from (a)

Fig. 33. The Employee example from [24] in Fragmenta.

control complexity, a tenet principle of this paper, both graph-based approaches, Fragmenta and MetaDepth, are better — their stratification ensures layer separation.

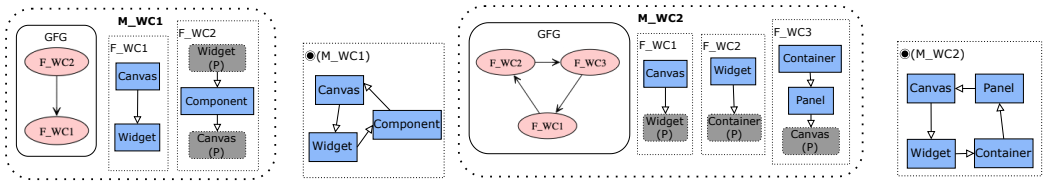
### 9.4 UML Modularity and Composition

Catalysis [25] divides a UML class model into subject areas, effectively fragments; it shares Fragmenta's philosophy: different things can be said about something in different contexts. Catalysis relies on informal mechanisms of decomposition and composition, unlike Fragmenta.

UML groups model elements in *packages* [30, 53], akin to Fragmenta's fragments but may contain both classes and collaborations. Larman [43] uses packages to separate subject areas. Two UML packaging mechanisms are relevant: *imports* and *merges*. Importing provides access to elements of imported packages [53]; merging builds bigger packages from smaller ones [24, 53]. Figure 33 renders in Fragmenta the UML package example of [24, p. 446].

UML package importing resembles Fragmenta's referencing. When a fragment $A$ references another, say $B$, it means that $A$ may contain proxies referring to elements of either $B$, or of other fragment that $B$ references, as per transitivity. In Fragmenta, elements are imported through proxies; in UML, importing brings elements into some context (or namespace). In Fig. 33a, proxies String and Employee in fragment F_BE materialise imports from F_EL. Fragmenta allows both a top-down and a bottom-up style of referencing and importing; models of sections 2 and 5 follow a traditional bottom-up style — elements are defined before the proxies that reference them —, whereas Fig. 33a follows a more top-down style — proxies are introduced first.

UML package merging resembles Fragmenta's fragment resolution (or composition), which, unlike package merging, is mathematically defined, grounded on graph subsumption defined herein. Figure 33b shows the composition of Fig. 33a's two fragments. Dingel et al [24] reports issues with package merging, proposing formal solutions which lack an accompanying implementation.




(a) Model WC1 (b) Resolution of WC1 (c) Model WC2 (d) Resolution of WC2

Fig. 34. Example models from [59] in Fragmenta, both invalid, and their malformed fragment resolutions

UML templates [53], another UML modularity and composition mechanism, provide flexible parametrisation means akin to Fragmenta's template patterns obtained through virtuals, inheritance and derived edges as shown in Figs. 7f and 7g and discussed in section 8.5. Unlike Fragmenta, UML templates lack a formal underpinning.

### 9.5 Model Merging

Category theory's colimit, proposed as a foundation for model merging [18], is at the heart of of [57], which relies on *interconnection diagrams* to relate separate models[20]. Fragmenta follows the same underpinning, but integrates proxies within the modelling, avoiding the separate representation.

Sabetzadeh et al [58, 59] builds up on [57] to detect inconsistencies when fragments are composed, such as the ones from section 3.3 (Figs. 12b and 12c). Figure 34 renders in Fragmenta models from [58, 59]. While [58, 59] spots the inheritance cycles of Fig. 34 from the merged model, Fragmenta does so locally without building the overall composition; fragment F_WC2 of Fig. 34a violates the constraint which prevents proxies from inheriting; the inheritance cycle resulting from model of Fig. 34c (see Fig. 34d) is caught through the GFG which includes a disallowed cycle. To avoid the manual *interconnection diagrams*, the separate representation relating fragments of [57–59], Nejati et al [51] resorts to an heuristic matcher to automate the interconnections. Fragmenta, illustrated in Fig. 34, constructs fragmented models through proxies, which precludes both manually built *interconnection diagrams* and complicated matching algorithms — only the proxy's referent needs to be provided. Like [51, 57–59], Fragmenta supports a single mode of inheritance, but provides *virtuals* to overcome its restrictions. Model composition mechanics of [51, 57–59] is pair-wise; the overall composition is built incrementally with big models requiring many smaller compositions. Fragmenta builds the overall composition in two steps: the different fragments are composed using union composition, and the overall composition is obtained from the union fragment through resolution. The optimisations of Rubin and Chechik [55] to tackle the inefficiencies of pair-wise composition are unable to improve the NP complexity of the matching algorithms.

## 10 CONCLUSIONS

Graphical data (or conceptual) modelling, software engineering's most widely used form of modelling, is here given an uplift. The field's roots can be traced back to Chen's work on entity-relationship modelling in the 1970s [19], extended to embrace the ideas of object-oriented (OO) modelling in the 1980s [14, 16, 17, 56] and subsequently applied to metamodelling, widely used to define the abstract syntax of graphical languages, in the 1990s [12]. The paper proposes a substantiated solution to the problem of complexity management, researched for four decades. Mainstream OO modelling's core concepts were established in the early 1990s without any substantial primitives added ever since. This

[20]The models from [18, 57], not far from Fig. 33, require an extra representation to relate elements from different fragments.

paper uplifts OO modelling through an upgrade to the novel Fragmenta class modelling framework, initially proposed in [7], following the vein of innovative OO and graphical modelling research discussed in section 9.

Section 2 presents and motivates Fragmenta with a layered model, involving an abstract representation of hotels in general and a concrete realisation (a refinement) as a more complex hotel for pets. This illustrates Fragmenta's major novelties, namely, vertical refinement as the basis of the layering, horizontal refinement to manage the complexity of the pets world model with proxies acting as modelling entities and composition joints, virtuals to express intricate inheritance relations, and graphical constraints to state graphically what is usually stated textually.

Section 3 defines Fragmenta mathematically by uplifting [7]. The theory is built incrementally, starting from graphs, and then on to structural graphs, the different kinds of morphisms, fragments, GFGs and models. This includes the major horizontal refinement improvement of *fragment resolution* (or composition), the new vertical refinement inspired by data refinement [35], which enables multilevel modelling, and the partial extra typings, another multilevel modelling ingredient.

Sections 4 to 6 illustrate Fragmenta with: (i) a layered model of an e-commerce application, similar to a benchmark model of multilevel modelling (section 4), (ii) a statecharts metamodel (section 5), and (iii) a metamodel of a SysML profile for cyber-physical systems (section 6). These illustrations use a two-layered metamodel, highlighting vertical refinement's capacity to articulate essence and detail. The several fragmented models, with its panoply of detailed properties expressed visually, showcase Fragmenta's graphical expressiveness and horizontal refinement.

Section 7 evaluates Fragmenta to appraise its adequacy and effectiveness. It concludes that Fragmenta, supported by its novelties, is overall better than the compared approaches, a prominent Fragmenta advantage being graphical expressiveness.

Section 8 discusses the results of the paper, highlighting characteristics of Fragmenta and its novelties, and the way it was developed. Section 9 discusses related work from different related areas of research.

Fragmenta is exercised here with four major case studies, which although not excessively large, are sufficiently large and interesting to highlight its capabilities, which proved advantageous in a comparison to related approaches in section 7's evaluation. This supports Fragmenta's suitability for large-scale modelling. The paper's main contributions are as follows:

- Fragmenta is the first graphical and formal data modelling framework supporting both vertical (layering) and horizontal (decomposition or fragmentation) refinement.
- Fragmenta's novel layered architecture based on two strata, type and instance, where the type stratum may be further divided into layers through vertical refinement. In addition, models may have extra typings within the same layer. Vertical refinement is a novel application of ideas linked to formal data refinement, hitherto scarcely explored in the areas of data- (or conceptual-) and meta-modelling.
- Fragment composition (or resolution) formally defined using graph subsumption enhances and substantiates the novel proxy. To the author's knowledge, no other work gave this much depth to the proxy concept, key to horizontal refinement and fragmented modelling.
- The notational novelties of (i) virtuals, which enhance inheritance, and (ii) derived, value-constraint and path edges, which enhance graphical constraint modelling.
- Fragmenta's Haskell implementation[21] attests to Fragmenta's realisability, enables experimentation, and contributed considerably to the quality of the work presented here.

[21] available from https://github.com/namalio/Fragmenta

| RQ | Contributions |
|---|---|
| RQ1 | Fragmenta is the first formal data-modelling framework supporting both vertical and horizontal refinement, which are mechanisms of abstraction and modularity, respectively. |
| RQ2 | Fragmenta proposes a layered architecture based on two strata, type and instance; the type stratum may be further divided into layers through vertical refinement; extra typings say that elements from one model are typed by elements of another. |
| RQ3 | The paper introduces several novelties: (i) fragment composition along proxies, defined as graph subsumption, improves horizontal refinement, (ii) virtuals make inheritance more agile, and (iii) derived, value-constraint and path edges enhance graphical constraint expressivity. |
| RQ1, RQ2, RQ3 | Fragmenta's adequacy with respect to large scale modelling is exercised with four case studies, which although not excessively large, are sufficiently large and interesting to highlight Fragmenta's capabilities which proved advantageous in an evaluation against related approaches (section 7). One case study is a benchmark problem of multilevel modelling; two case studies involve industrial languages, namely, statecharts, a UML notation, and the Into-SysML profile, which involves two SysML notations. |
| RQ1, RQ2, RQ3 | Fragmenta's Haskell implementation is a minor contribution which attests to Fragmenta's realisability; it enables experimentation with the framework and is a contributor to the quality of the work presented here. All layered models and examples presented in the paper were checked using the tool. |

Table 11. Summary of the paper's contributions per research question (RQ).

Table 11 summarises the paper's contributions for each research question raised in section 1.1.

## ACKNOWLEDGMENTS

I thank the anonymous reviewers for their suggestions, remarks and criticism; this constituted valuable feedback which contributed to a better article.

## REFERENCES

[1] Colin Aktinson and Thomas Kühne. 2001. The Essence of Multilevel Metamodeling. In *UML 2001 (LNCS, Vol. 2185)*. Springer.
[2] Colin Aktinson and Thomas Kühne. 2003. Model-driven development: a metamodeling foundation. *IEEE Software* 20, 5 (2003), 36–41.
[3] Colin Aktinson and Thomas Kühne. 2008. Reducing accidental complexity in domain models. *Software and Systems Modeling* 7, 3 (2008), 36–41.
[4] Nuno Amálio. 2019. Sound and Relaxed Behavioural Inheritance. In *From Astrophysics to Unconventional Computation: Essays Presented to Susan Stepney on the Occasion of her 60th Birthday (Emergence, Complexity and Computation, Vol. 35)*, Andrew Adamatzky and Vivien Kendon (Eds.). Springer, 255–298.
[5] Nuno Amálio. 2021. *Z Specification of Fragmenta.* Technical Report. Available at https://bit.ly/3EDR5vK.
[6] Nuno Amálio, Lionel Briand, and Pierre Kelsen. 2019. An Experimental Scrutiny of Visual Design Modelling: VCL up against UML+OCL. *Empirical Software Engineering* 25, 2 (2019), 1205–1258.
[7] Nuno Amálio, Juan de Lara, and Esther Guerra. 2015. FRAGMENTA: A Theory of Fragmentation for MDE. In *MODELS 2015*. IEEE, 106–115.
[8] Nuno Amálio and Christian Glodt. 2015. A tool for visual and formal modelling of software designs. *Science of Computer Programming* 98, Part 1 (2015), 52 – 79.
[9] Nuno Amálio, Pierre Kelsen, Qin Ma, and Christian Glodt. 2010. Using VCL as an Aspect-Oriented Approach to Requirements Modelling. *TAOSD* VII (2010), 151–199.
[10] Nuno Amálio, Richard Payne, Ana Cavalcanti, Etienne Brosse, and Jim Woodcock. 2015. *Foundations of the SysML profile for CPS modelling*. Technical Report D2.1a. INTO-CPS project.
[11] Nuno Amálio, Richard Payne, Ana Cavalcanti, and Jim Woodcock. 2016. Checking SysML Models for Co-Simulation. In *ICFEM 2016 (LNCS, Vol. 10009)*. Springer.
[12] Colin Atkinson. 1997. Meta-modeling for distributed object environments. In *EDOC '97*. IEEE, 90–101.
[13] Colin Atkinson and Thomas Kühne. 2008. Reducing accidental complexity in domain models. *Software and Systems Modeling* 7, 3 (2008), 345–359.
[14] S. C. Bailin. 1989. An object-oriented requirements specifications method. *Commun. ACM* 32, 5 (1989), 608–623.

[15] Richard Bird. 2014. *Thinking Functionally with Haskell.* Cambridge University Press.

[16] Grady Booch. 1986. Object-Oriented Development. *IEEE Transactions on Software Engineering* SE-12, 2 (1986), 211–221.

[17] Grady Booch. 1994. *Object-oriented analysis and design with applications.* Addison-Wesley.

[18] Greg Brunet, Marsha Chechik, Steve Easterbrook, Shiva Nejati, Nan Niu, and Mehrdad Sabetzadeh. 2006. A Manifesto for Model Merging. In *GaMMa '06: International workshop on Global integrated model management.* 5–12.

[19] Peter Pin-Shan Chen. 1976. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems* 1, 1 (1976), 9–36.

[20] Juan de Lara and Esther Guerra. 2010. Deep Meta-Modelling with MetaDepth. In *TOOLS (LNCS, Vol. 6141).* Springer, 1–20.

[21] Juan de Lara and Esther Guerra. 2021. Language family engineering with product lines of multi-level models. *Formal Aspects of Computing* 33 (2021), 1173–1208.

[22] Juan de Lara, Esther Guerra, and Jesús Sánchez Cuadrado. 2014. When and How to use Multi-Level Modelling. *ACM Transactions on Software Engineering and Methodology* 24, 2 (2014), 1–46.

[23] Lenny Delligatti. 2014. *SysML distilled: a brief guide to the systems modelling language.* Addison-Wesley.

[24] J. Dingel, Z. Diskin, and A. Zito. 2008. Understanding and improving UML package merge. *Software & Systems Modeling* 7 (2008), 443–467.

[25] Desmond F. D'Souza and Alan Cameron Wills. 1999. *Objects, Components, and Frameworks with UML: The Catalysis Approach.* Addison-Wesley.

[26] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. 2006. *Fundamentals of Algebraic Graph Transformation.* Springer.

[27] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. 2003. Graphviz and dynagraph – static and dynamic graph drawing tools. In *GRAPH DRAWING SOFTWARE.* Springer.

[28] P. Feldman and D. Miller. 1986. Entity model clustering: structuring a data model by abstraction. *Computer Journal* 29, 4 (1986), 348–360.

[29] Claudenir M. Fonseca, João Paulo A. Almeida, Giancarlo Guizzardi, and Victorio A. Carvalho. 2018. Multi-level Conceptual Modeling: From a Formal Theory to a Well-Founded Language. In *ER 2018 (LNCS, Vol. 11157).* Springer, 409–423.

[30] Martin Fowler and Kendall Scott. 2004. *UML distilled* (2nd ed.). Addison-Wesley.

[31] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem phong Vo. 1993. A Technique for Drawing Directed Graphs. *IEEE Transactions on Software Engineering* 19, 3 (1993), 214–230.

[32] Emden R. Gansner and Stephen C. North. 2000. An open graph visualization system and its applications to software engineering. *Software, Practice & Experience* 30, 11 (2000), 1203–1233.

[33] David Harel. 1987. STATECHARTS: a visual formalism for complex systems. *Science of Computer Programming* 8 (1987), 231–274.

[34] David Harel. 1988. On visual formalisms. *Commun. ACM* 31, 5 (1988), 514 – 530.

[35] Jifeng He, Anthony Hoare, and Jeff W. Sanders. 1986. Data Refinement Refined. In *ESOP'86 (LNCS, Vol. 213).* Springer, 187–196. https://doi.org/10.1007/3-540-16442-1_14

[36] Paul Hudak, John Hughes, Simon Peyton Jones, and Philip Wadler. 2007. A History of Haskell: Being Lazy With Class. In *HOPL III.* ACM, 12–1–12–55.

[37] ISO. 2002. Information Technology—Z Formal Specification Notation—Syntax, Type System and Semantics. ISO/IEC 13568:2002, Int. Standard.

[38] Manfred A. Jeusfeld and Bernd Neumayr. 2016. DeepTelos: Multi-level Modeling with Most General Instances. In *ER 2016 (LNCS, Vol. 9974).* Springer, 198–211.

[39] Stefan Jurack and Gabriele Taentzer. 2010. A Component Concept for Typed Graphs with Inheritance and Containment Structures. In *ICGT (LNCS, Vol. 6372).* Springer, 187–202.

[40] Stefan Jurack and Gabriele Taentzer. 2012. Transformation of Typed Composite Graphs with Inheritance and Containment Structures. *Fundam. Inform.* 118, 1-2 (2012), 97–134.

[41] Saunders Mac Lane. 1971. *Categories for the Working Mathematician.* Springer.

[42] Jill H. Larkin and Herbert A. Simon. 1987. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science* 11 (1987), 65–99.

[43] Craig Larman. 2005. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* Pearson Education.

[44] Miran Lipovača. 2011. *Learn You a Haskell for Great Good!* No Starch Press.

[45] Petra Malik and Mark Utting. 2005. CZT: A Framework for Z Tools. In *ZB 2005 (LNCS, Vol. 3455).* Springer.

[46] George A. Miller. 1956. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *Psychological Review* 63, 2 (1956), 81—97.

[47] Tim Miller, Leo Freitas, Petra Malik, and Mark Utting. 2005. CZT Support for Z Extensions. In *IFM 2005 (LNCS).*

[48] Daniel Moody. 1991. A Practical Methodology for the Representation of Large Data Models. In *Australian Database and Information Systems Conference.*

[49] Daniel Moody. 1997. A multi-level architecture for representing enterprise data models. In *Conceptual Modeling — ER*, Vol. 1331. 184–197.

[50] Daniel Moody. 2002. Complexity effects on end user understanding of data models: an experimental comparison of large data model representation methods, In ECIS. *ECIS*, 391–405.

[51] Shiva Nejati, Mehrdad Sabetzadeh, Marsha Chechik, Steve Easterbrook, and Pamela Zave. 2007. Matching and Merging of Statecharts Specifications. In *ICSE'07.* IEEE, 54–64.

[52] OMG. 2012. *OMG Systems Modeling Language, version 1.3.* Technical Report. OMG.

[53] OMG. 2017. *OMG Unified Modeling Language.* Technical Report Version 2.5.1. OMG. https://www.omg.org/spec/UML/2.5.1/PDF.

[54] David Parnas. 1972. On the criteria to be used in decomposing systems into modules. *Commun. ACM* 15, 12 (1972), 1053–1058.

[55] Julia Rubin and Marsha Chechik. 2013. N-Way Model Merging. In *ESEC/FSE'13*. ACM, 301 – 311.
[56] James Rumbaugh, Michel Blaha, William Permerlani, Frederick Eddy, and William Lorenson. 1991. *Object-oriented modelling and design*. Prentice-Hall.
[57] Mehrdad Sabetzadeh and Steve Easterbrook. 2006. View merging in the presence of incompleteness and inconsistency. *Requirements Engineering* 11 (2006), 174–193.
[58] Mehrdad Sabetzadeh, Shiva Nejati, Steve Easterbrook, and Marsha Chechik. 2008. Global Consistency Checking of Distributed Models with TReMer+. In *ICSE'08*. IEEE, 815–818.
[59] Mehrdad Sabetzadeh, Shiva Nejati, Sotirios Liaskos, Steve Easterbrook, and Marsha Chechik. 2007. Consistency Checking of Conceptual Models via Model Merging. In *RE'07*. IEEE, 221–230.
[60] J. M. Spivey. 1998. *The Z Notation: A Reference Manual*. Prentice-Hall.
[61] Peri Tarr, Harold Ossher, William Harrison, and Jr Stanley M. Sutton. 1999. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *ICSE'99*. IEEE, 107–119.
[62] Toby J. Teorey, Guangping Wei, Deborah L. Bolton, and John A. Koenig. 1989. ER model clustering as an aid for user communication and documentation in database design. *CACM* 32, 8 (1989), 975–987.
[63] Markus Wenzel Tobias Nipkow, Lawrence C. Paulson. 2020. *A Proof Assistant for Higher-Order Logic*. Springer.
[64] J. Woodcock and J. Davies. 1996. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall.

## A AUXILIARY DEFINITIONS

Definition 1 (Sets). *Set membership* $\in$, *set containment* $\subseteq$, *powersets* $\mathbf{P}$ *and the empty set* $\varnothing$ *are defined as:*

$$x \in S \Leftrightarrow \exists\, y : S \bullet x = y \quad S \subseteq T \Leftrightarrow \forall\, x : S \bullet x \in T \quad \mathbf{P}\, X = \{S \mid S \subseteq X\} \quad \varnothing X = \{x \mid \neg x \in X\}$$

*Set operations union* $\cup$, *intersection* $\cap$ *and subtraction* $\setminus$ *are defined as:*

$$A \cup B = \{x \mid x \in A \lor x \in B\} \quad A \cap B = \{x \mid x \in A \land x \in B\} \quad A \setminus B = \{x \mid x \in A \land x \notin B\}$$

*Non-empty powersets* $\mathbf{P}_1$, *and optionality (either a singleton or the empty set) are defined as:*

$$\mathbf{P}_1\, X = \{S \mid S \in \mathbf{P}\, X \land S \neq \varnothing\} \quad \mathbf{opt}\, X = \{s : \mathbf{P}\, X \mid (\exists\, x : X \bullet s = \{x\}) \lor s = \varnothing\}$$

□

Definition 2 (Relations). *Operator* $\leftrightarrow$ *denotes a relation (a set of pairs) between two given sets;* id *builds the identity relation over a given set; and operator* $^\sim$ *inverts a given relation:*

$$X \leftrightarrow Y = \{(x, y) \mid x \in X \land y \in Y\} \quad \mathrm{id}\, X = \{(x, x) \mid x \in X\} \quad r^\sim = \{(y, x) \mid (x, y) \in r\}$$

*Domain and range of a relation may be obtained using functions* dom *and* ran, *respectively:*

$$\mathrm{dom}\, r = \{x \mid \exists\, y : \mathrm{ran}\, r \bullet (x, y) \in r\} \quad \mathrm{ran}\, r = \{y \mid \exists\, x : \mathrm{dom}\, r \bullet (x, y) \in r\}$$

*Operators for domain* ($\lhd$) *and range* ($\rhd$) *restriction, domain* ($\lhd\!\!\!-$) *and range* ($\rhd\!\!\!-$) *subtraction, and relational image* ($(\!|\ |\!)$) *are defined as:*

$$S \lhd r = \{r' \mid r' \subseteq r \land \mathrm{dom}\, r' \subseteq S\} \quad r \rhd S = \{r' \mid r' \subseteq r \land \mathrm{ran}\, r' \subseteq S\}$$
$$S \lhd\!\!\!- r = (\mathrm{dom}\, r \setminus S) \lhd r \quad r \rhd\!\!\!- S = r \rhd (\mathrm{ran}\, r \setminus S) \quad r\,(\!|\, S \,|\!) = \mathrm{ran}(S \lhd r)$$

*Operator* $\fatsemi$ *composes two relations;* $\circ$ *does the backward composition of relations;* $^+$ *and* $^*$ *give, respectively, the transitive and the reflexive-transitive closure of a relation* $X \leftrightarrow X$, *and operator* $\oplus$ *denotes the relational overriding of a relation within*

*another::*

$$r \fatsemi s = \{(x, z) \mid \exists\, y : \operatorname{ran} r \bullet (x, y) \in r \land (y, z) \in s\} \quad g \circ f = f \fatsemi g$$
$$r^+ = \begin{cases} r & \mathit{if}\ r \fatsemi r \subseteq r \\ r \fatsemi r & \mathit{otherwise} \end{cases} \quad r^* = r^+ \cup \operatorname{id}(\operatorname{dom} r) \quad r \oplus s = (\operatorname{dom} s) \lhd r \cup s$$

*Relations* acyclic, *injective and* antireflexive *are built from generic sets $X$ and $Y$:*

$$\operatorname{acyclic} X = \{r \mid r \in X \leftrightarrow X \land r^+ \cap \operatorname{id} X = \varnothing\} \quad \operatorname{injrel} X\, Y = \{r \mid r \in X \leftrightarrow Y \land r^\sim \in Y \pfun X\}$$
$$\operatorname{antireflexive} X = \{r \mid r \in X \leftrightarrow X \land r \cap \operatorname{id} X = \varnothing\}$$

*Above, $\cap$ is set intersection and $\varnothing$ is the empty set (def. 1).* □

Definition 3 (Functions). *Partial ($\pfun$), total ($\fun$), injective ($\rightarrowtail$), surjective ($\twoheadrightarrow$) and bijective ($\bijection$) functions are defined from relations (def. 2):*

$$X \pfun Y = \{f \mid f \in X \leftrightarrow Y \land \forall\, x : X ; y_1, y_2 : Y \mid (x, y_1) \in f \land (x, y_2) \in f \bullet y_1 = y_2\}$$
$$X \fun Y = \{f \mid f \in X \pfun Y \land \operatorname{dom} f = X\} \quad X \pinj Y = \{f \mid f \in X \pfun Y \land f^\sim \in Y \pfun X\}$$
$$X \rightarrowtail Y = \{f \mid f \in X \fun Y \land f^\sim \in Y \pfun X\} \quad X \twoheadrightarrow Y = \{f \mid f \in X \fun Y \land \operatorname{ran} f = Y\}$$
$$X \bijection Y = \{f \mid f \in X \rightarrowtail Y \land f \in X \twoheadrightarrow Y\}$$

*Trees are those acyclic relations (see def. 2) over some set $X$ which are partial functions:*

$$\operatorname{tree} X = \{r \mid r \in X \leftrightarrow X \land r \in \operatorname{acyclic} \land r \in X \pfun X\}$$

□

Definition 4 (Generic Functions and Predicates). *The generic functions defined below are as follows: (i)* apply *does a pairwise function application, (ii)* the *takes the value of an optional set, (iii)* ⊡ *totalises a partial function within a set, and (iv)* $^\oplus$ *is the closure of relation composition with respect to relation override, which stretches a relation by getting only the end points of a transitive closure:*

$$\operatorname{apply}[X, Y, Z, W] : ((X \fun Z) \times (Y \fun W)) \fun (X \times Y) \fun (Z \times W) \quad \operatorname{the} : \operatorname{opt}[X] \pfun X$$
$$\operatorname{apply}(f, g)(x, y) = (f\, x, g\, y) \quad \operatorname{the}\{x\} = x$$
$$\_ \boxdot \_ : (X \fun X) \times \mathbf{P}\, X) \fun (X \pfun X) \quad \_^\oplus : (X \leftrightarrow X) \fun (X \leftrightarrow X)$$
$$f \boxdot A = (\operatorname{id} A) \oplus f$$
$$r^\oplus = \begin{cases} r & \mathit{if}\ (r \fatsemi r) = r \\ r \oplus (r \fatsemi r)^\oplus & \mathit{otherwise} \end{cases}$$

*Above, $\oplus$ is relation override and $\fatsemi$ is relational composition (def. 2).*

*Predicates $=_p$ and $\subseteq_p$ are pairwise equality and set containment:*

$$(xs, ys) =_p (zs, ws) \Leftrightarrow xs = zs \land ys = ws \quad (xs, ys) \subseteq_p (zs, ws) \Leftrightarrow xs \subseteq zs \land ys \subseteq ws$$

**Laws.** *The following laws were proved in Isabelle:*

$$\vdash f \boxdot \varnothing = f \quad \vdash \varnothing \boxdot s = (\operatorname{id} s) \quad \vdash f \boxdot (A \cup B) = f \boxdot A \cup f \boxdot B$$
$$r \in X \pfun Y \vdash r^\oplus \in X \pfun Y \quad \vdash \mathit{ran}(f \boxdot s) = s \setminus \operatorname{dom} f \cup \operatorname{ran} f$$

□

Definition 5 (Vertices and Edges). *Disjoint sets $V$ and $E$ represent all possible nodes and edges of graphs, respectively.*

□

Definition 6 (Numeric Nodes). *Special nodes designate the numeric sets, namely:*

$$nNatS : V \quad nIntS : V \quad nRealS : V$$

*These denote (from left to right) the natural numbers, the integers and the reals, respectively.*

*There are also functions which extract either the integer or real number embedded in a node:*

$$\mathrm{to}\mathbb{Z} : V \to \mathrm{opt}[\mathbb{Z}] \quad \mathrm{to}\mathbb{R} : V \to \mathrm{opt}[\mathbb{R}]$$

$\mathrm{to}\mathbb{Z}$ *and* $\mathrm{to}\mathbb{R}$ *above yield an optional* (opt, *def. 1), which yields a singleton with the extracted number if there is a number embedded in the node, or the empty set otherwise.* □

Definition 7 (Graphs). *A graph $G = (N, A, s, t)$ consists of sets $N \subseteq V$ and $A \subseteq E$ of nodes and edges respectively, and source and target functions $s, t : A \to N$.*

*The set of graphs $Gr$, such that $G : Gr$, is defined as:*

$$Gr = \{(N, A, s, t) \mid N \in \mathbf{P}\, V \land A \in \mathbf{P}\, E \land s \in A \to N \land t \in A \to N\}$$

*Functions Ns, Es, src and tgt extract the different components of a graph:*

$$Ns\,(N, A, s, t) = N \quad Es\,(N, A, s, t) = A \quad src\,(N, A, s, t) = s \quad tgt\,(N, A, s, t) = t$$

*Function Els yields a pair with the given graph's nodes and edges; $\varnothing_G$ yields the empty graph:*

$$Els\,G = (Ns\,G, Es\,G) \quad \varnothing_G = (\varnothing, \varnothing, \varnothing, \varnothing)$$

□

Definition 8 (Graph Functions). *Several functions (below) operate upon a given graph $G$: (i) $EsId$ yields $G$'s identity edges, (ii) $rNs\,G\,es$ gives $G$'s nodes connected by set of edges $es$, (iii) $G \bowtie_{Es} es$ restricts $G$ to set of edges $es$, (iv) $G \bowtie_{Ns} vs$ restricts $G$ to set of nodes $vs$, and (v) $G \ominus_{Ns} vs$ removes set of nodes $vs$ from $G$.*

$$EsId : Gr \to \mathbf{P}\, E \quad rNs : Gr \to \mathbf{P}\, E \quad \bowtie_{Es} : (Gr \times \mathbf{P}\, E) \to Gr \quad \bowtie_{Ns}, \ominus_{Ns} : (Gr \times \mathbf{P}\, V) \to Gr$$
$$EsId_G = \{e : Es_G \mid src_G\, e = tgt_G\, e\} \quad rNs\,G\,es = ran(es \lhd src_G) \cup ran(es \lhd tgt_G)$$
$$G \bowtie_{Es} es = (rNs_G\, es, Es_G \cap es, es \lhd src_G, es \lhd tgt_G)$$
$$G \bowtie_{Ns} vs = (Ns_G \cap vs, G \bullet\!\!\leftrightarrow\!\!\bullet vs, (G \bullet\!\!\leftrightarrow\!\!\bullet vs) \lhd src_G, (G \bullet\!\!\leftrightarrow\!\!\bullet vs) \lhd tgt_G)$$
$$G \ominus_{Ns} vs = (Ns_G \setminus vs, Es_G \setminus (G \circ\!\!\rightarrow\!\!\circ vs), (G \circ\!\!\rightarrow\!\!\circ vs) \lhd\!\!\!\cdot\, src_G, (G \circ\!\!\rightarrow\!\!\circ vs) \lhd\!\!\!\cdot\, tgt_G)$$

*Above, $\lhd$ and $\lhd\!\!\!\cdot$ are relational domain restriction and subtraction, respectively (def. 2).*

*Notion of adjacency is captured by two functions: (i) $adjacent(G, v_1, v_2)$ says whether nodes $v_1$, $v_2$ are adjacent in $G$, (ii) $adjacent_E(G, e_1, e_2)$ says whether edges $e_1$, $e_2$ are adjacent in $G$.*

$$adjacent\_ : \mathbf{P}(Gr \times V \times V) \quad adjacent_E\_ : \mathbf{P}(Gr \times E \times E)$$
$$adjacent(G, v_1, v_2) \Leftrightarrow \exists\, e : Es_G \bullet src_G\, e = v_1 \land tgt_G\, e = v_2 \quad adjacent_E(G, e_1, e_2) \Leftrightarrow src_G\, e_1 = tgt_G\, e_2$$

*Other functions explore notions of connectedness and incidence: (i) $G \circ\!\!\rightarrowtail\!\!\circ\ vs$ yields G's edges incident to nodes vs (edges with the nodes as either source or target), and (ii) $G \bullet\!\!\leftrightarrow\!\!\bullet\ ns$ yields G's edges connecting set of nodes vs (edges with the nodes as both source and target).*

$$\_ \circ\!\!\rightarrowtail\!\!\circ\ \_,\ \_ \bullet\!\!\leftrightarrow\!\!\bullet\ \_ : Gr \times \mathbf{P}\ V \to \mathbf{P}$$
$$G \circ\!\!\rightarrowtail\!\!\circ\ vs = src_G^{\sim}(\!|vs|\!) \cup tgt_G^{\sim}(\!|vs|\!) \quad G \bullet\!\!\leftrightarrow\!\!\bullet\ vs = src_G^{\sim}(\!|vs|\!) \cap tgt_G^{\sim}(\!|vs|\!)$$

*Several functions build new structures from graphs: (i) $G^{\rightleftarrows}$ gives the graph that takes G and inverts its edges: (ii) $G^{\leftrightarrow}$ builds G's node adjacency relation, (iii) $G^{\leftrightarrow E}$, builds G's edge adjacency relation, and (iv) $G_1 \cup_G G_2$ constructs the union of graphs $G_1$ and $G_2$.*

$$\_^{\rightleftarrows} : Gr \to Gr \quad \_^{\leftrightarrow} : Gr \to V \leftrightarrow V \quad \_^{\leftrightarrow E} : Gr \to E \leftrightarrow E \quad \_ \cup_G \_ : Gr \times Gr \to Gr$$
$$G^{\rightleftarrows} = (Ns_G, Es_G, tgt_G, src_G)$$
$$G^{\leftrightarrow} = \{(v_1, v_2) \mid adjacent(G, v_1, v_2)\} \quad G^{\leftrightarrow E} = \{(e_1, e_2) \mid adjacent_E(G, e_1, e_2)\}$$
$$G_1 \cup_G G_2 = (Ns_{G_1} \cup Ns_{G_2}, Es_{G_1} \cup Es_{G_2}, src_{G_1} \cup src_{G_2}, tgt_{G_1} \cup tgt_{G_2})$$

*Above, graph union is defined as the set union (symbol ∪, def. 1) of the graph's components.*

*Graph subsumption (symbol ⊙) replaces nodes according to a replacement mapping:*

$$\_ \odot \_ : Gr \times (V \leftrightarrow V) \nrightarrow Gr$$
$$G \odot s = \begin{cases} (Ns_G \setminus \operatorname{dom} s \cup \operatorname{ran} s, Es_G, (s \boxdot Ns_G) \circ src_G, (s \boxdot Ns_G) \circ tgt_G) & \text{if } s \in Ns_G \nrightarrow Ns_G \\ G & \text{otherwise} \end{cases}$$

*Above, the substitution mapping s is a partial function covering the nodes of given graph G. The subsumed graph includes the graph's nodes minus subsumed nodes, the edges remain the same, and the source and target functions are totalised within node identity through function ⊡ (def. 4) — subsumed nodes are substituted, and the rest remains the same.*

*The following predicates capture useful conditions. ⊗ says whether a graph is acyclic, ⊟ says whether two graphs are disjoint (no nodes or edges in common), and ⊞ generalises ⊟ for any number of graphs:*

$$\otimes\ \_ : \mathbf{P}\ Gr \qquad \boxminus_{Es}, \boxminus\ \_ : \mathbf{P}(Gr \times Gr) \qquad \boxplus\ \_ : \mathbf{P}(I \nrightarrow Gr)$$
$$\otimes\ G \Leftrightarrow G^{\leftrightarrow E} \in \text{acyclic} \quad \boxminus_{Es}(G_1, G_2) \Leftrightarrow Es\ G_1 \cap Es\ G_2 = \varnothing$$
$$\boxminus(G_1, G_2) \Leftrightarrow Ns\ G_1 \cap Ns\ G_2 = \varnothing \wedge \boxminus_{Es}(G_1, G_2) \quad \boxplus\ Gs[I] \Leftrightarrow \forall\, i, j : \operatorname{dom}\ Gs \mid i \neq j \bullet \boxminus(Gs\ i, Gs\ j)$$

*Above, set I is any indexing set. ⊗ checks whether the graph's edge relation is acyclic (def. 2).*

**Laws.** *The following laws were proved in Isabelle:*

$$\boxminus(G_1, G_2) \vdash \boxminus_{Es}(G_1, G_2) \quad G_1, G_2 \in Gr \vdash G_1 \cup_G G_2 \in Gr \quad G \in Gr \vdash G \cup_G \varnothing_G = G$$
$$G_1, G_2 \in Gr\ ; \boxminus_{Es}(G_1, G_2) \vdash G_1 \cup_G G_2 = G_2 \cup_G G_1 \quad G_1, G_2 \in Gr \vdash \boxminus(G_1, G_2) = \boxminus(G_2, G_1)$$
$$G_1, G_2 \in Gr \vdash \boxminus_{Es}(G_1, G_2) = \boxminus_{Es}(G_2, G_1)$$
$$G_1, G_2 \in Gr\ ; \boxminus_{Es}(G_1, G_2) \vdash adjacent(G_1 \cup_G G_2, x, y) \Leftrightarrow adjacent(G_1, x, y) \vee adjacent(G_2 x, y)$$
$$G_1, G_2 \in Gr\ ; \boxminus_{Es}(G_1, G_2) \vdash (G_1 \cup_G G_2)^{\leftrightarrow} = G_1^{\leftrightarrow} \cup_G G_2^{\leftrightarrow}$$
$$G_1, G_2 \in Gr\ ; \boxminus_{Es}(G_1, G_2) \vdash rNs\ (G_1 \cup_G G_2)\ es = rNs\ G_1\ es \cup rNs\ G_2\ es \quad G \in Gr \vdash rNs\ G\ es \subseteq Ns\ G$$
$$G_1, G_2 \in Gr\ ; \boxminus(G_1, G_2) \vdash rNs\ G_1\ es_1 \cap rNs\ G_2\ es_2 = \varnothing$$
$$G \in Gr \vdash G \bowtie_{Es} es \in Gr \quad G_1, G_2 \in Gr\ ; \boxminus(G_1, G_2) \vdash \boxminus(G_1 \bowtie_{Es} es, G_2 \bowtie_{Es} es)$$
$$\vdash \varnothing_G \bowtie_{Es} es = \varnothing_G \quad G \in Gr \vdash G \bowtie_{Es} \varnothing = \varnothing_G \quad G \in Gr \vdash (G \bowtie_{Es} es) \in Gr$$
$$G_1, G_2 \in Gr\ ; \boxminus_{Es}(G_1, G_2) \vdash (G_1 \cup_G G_2) \bowtie_{Es} es = G_1 \bowtie_{Es} es \cup_G G_2 \bowtie_{Es} es$$
$$G \in Gr\ ; es_b \cap Es_G = \varnothing \vdash G \bowtie_{Es} (es_a \cup es_b) = G \bowtie_{Es} es_a$$
$$G \in Gr\ ; s \in rNs\ G\ es \nrightarrow rNs\ G\ es \vdash (G \odot s) \bowtie_{Es} es = (G \bowtie_{Es} es) \odot s \quad \vdash \varnothing_G \circ\!\!\rightarrow\!\!\circ\ ns = \varnothing$$
$$G \in Gr \vdash G \circ\!\!\rightarrow\!\!\circ\ Ns_G = Es_G \quad G \in Gr \vdash G \circ\!\!\rightarrow\!\!\circ\ \varnothing = \varnothing \quad G \in Gr \vdash (G \circ\!\!\rightarrow\!\!\circ\ ns) \subseteq Es_G$$
$$G_1, G_2 \in Gr\ ; \boxminus_{Es}(G_1, G_2) \vdash (G_1 \cup_G G_2) \circ\!\!\rightarrow\!\!\circ\ vs = G_1 \circ\!\!\rightarrow\!\!\circ\ vs \cup G_2 \circ\!\!\rightarrow\!\!\circ\ vs$$
$$G \in Gr\ ; vs \cap Ns_G = \varnothing \vdash G \circ\!\!\rightarrow\!\!\circ\ vs = \varnothing \quad \vdash \varnothing_G \bullet\!\!\leftrightarrow\!\!\bullet\ ns = \varnothing$$
$$G \in Gr \vdash G \bullet\!\!\leftrightarrow\!\!\bullet\ Ns_G = Es_G \quad G \in Gr \vdash G \bullet\!\!\leftrightarrow\!\!\bullet\ \varnothing = \varnothing \quad G \in Gr \vdash (G \bullet\!\!\leftrightarrow\!\!\bullet\ ns) \subseteq Es_G$$
$$G_1, G_2 \in Gr\ ; \boxminus_{Es}(G_1, G_2) \vdash (G_1 \cup_G G_2) \bullet\!\!\leftrightarrow\!\!\bullet\ vs = G_1 \bullet\!\!\leftrightarrow\!\!\bullet\ vs \cup G_2 \bullet\!\!\leftrightarrow\!\!\bullet\ vs \quad \vdash \varnothing_G^{\rightleftarrows} = \varnothing_G$$
$$G \in Gr \vdash (G^{\rightleftarrows})^{\rightleftarrows} = G \quad G \in Gr \vdash (G^{\rightleftarrows})^{\leftrightarrow} = (G^{\leftrightarrow})^{\sim} \quad G_1, G_2 \in Gr \vdash (G_1 \cup_G G_2)^{\rightleftarrows} = G_1^{\rightleftarrows} \cup_G G_2^{\rightleftarrows}$$
$$G \in Gr \vdash G \odot \varnothing = G \quad G \in Gr \vdash G \odot \mathrm{id}(Ns_G) = G \quad \vdash \varnothing_G \odot s = \varnothing_G \quad G \in Gr \vdash (G \odot s) \in Gr$$
$$G \in Gr \vdash G \bowtie_{Ns} \varnothing = \varnothing_G \quad \vdash \varnothing_G \bowtie_{Ns} ns = \varnothing_G \quad G \in Gr \vdash (G \bowtie_{Ns} ns) \in Gr \quad G \in Gr \vdash G \ominus_{Ns} \varnothing = G$$
$$\vdash \varnothing_G \ominus_{Ns} ns = \varnothing_G \quad G \in Gr \vdash (G \ominus_{Ns} ns) \in Gr \quad G \in Gr \vdash G \ominus_{Ns} ns_a \cup ns_b = G \ominus_{Ns} ns_a \ominus_{Ns} ns_b$$
$$G_1, G_2 \in Gr\ ; \boxminus_{Es}(G_1, G_2) \vdash G_1 \cup_G G_2 \ominus_{Ns} ns = G_1 \ominus_{Ns} ns \cup_G G_2 \ominus_{Ns} ns$$
$$\boxminus_{Es}(G_1, G_2) \vdash \boxminus_{Es}(G_1 \ominus_{Ns} ns_1, G_2 \ominus_{Ns} ns_2) \quad G_1, G_2 \in Gr\ ; \boxminus(G_1, G_2) \vdash \boxtimes(G_1 \cup_G G_2) \Leftrightarrow \boxtimes\ G_1 \wedge \boxtimes\ G_2$$

□

Definition 9 (Graph Morphisms). *Graph morphisms map elements of one graph onto another. A total graph morphism $f$ between graphs $G_1, G_2 : Gr$ (see def. 7), written $f : G_1 \rightarrow_G G_2$, is a pair of total functions (see def. 4) $f = (f_V, f_E)$, $f_V : Ns_{G1} \rightarrow Ns_{G2}$ and $f_E : Es_{G1} \rightarrow Es_{G2}$, which map nodes and edges, respectively, and must preserve the source and target functions of edges: $fv \circ src_{G_1} = src_{G_2} \circ fe$ and $fv \circ tgt_{G_1} = tgt_{G_2} \circ fe$.*

*In partial graph morphisms between graphs $G_1, G_2 : Gr$, written $f : G_1 \nrightarrow_G G_2$, the morphism functions may be partial (see def. 4): $f_V : Ns_{G1} \nrightarrow Ns_{G2}$ and $f_E : Es_{G1} \nrightarrow Es_{G2}$. The commuting equations change to: $fv \circ ((\mathrm{dom}\ f_E) \lhd src_{G_1}) = src_{G_2} \circ fe$ and $fv \circ ((\mathrm{dom}\ f_E) \lhd tgt_{G_1}) = tgt_{G_2} \circ fe$; here, functions source and target are restricted to the edges mapped by morphism $f$ only.*

*Set GrM defines all possible graph morphisms:*

$$GrM = \{(fv, fe) \mid fv \in V \nrightarrow V \wedge fe \in E \nrightarrow E\}$$

*Sets of of total, $G_1 \rightarrow_G G_2$ and partial $G_1 \nrightarrow_G G_2$ morphisms, such that $G_1 \rightarrow_G G_2 \subseteq GrM$ and $G_1 \nrightarrow_G G_2 \subseteq GrM$, are defined as:*

$$G_1 \rightarrow_G G_2 = \{(fv, fe) \mid fv \in Ns_{G_1} \rightarrow Ns_{G_2} \wedge fe \in Es_{G_1} \rightarrow Es_{G_2} \\ \wedge fv \circ src_{G_1} = src_{G_2} \circ fe \wedge fv \circ tgt_{G_1} = tgt_{G_2} \circ fe\}$$
$$G_1 \nrightarrow_G G_2 = \{(fv, fe) \mid fv \in Ns_{G_1} \nrightarrow Ns_{G_2} \wedge fe \in Es_{G_1} \nrightarrow Es_{G_2} \\ \wedge fv \circ ((\mathrm{dom}\, fe) \lhd src_{G_1}) = src_{G_2} \circ fe \wedge fv \circ ((\mathrm{dom}\, fe) \lhd tgt_{G_1}) = tgt_{G_2} \circ fe\}$$

*Above, the equations involving function composition (symbol $\circ$) ensure diagram commutativity.*

*Several relevant functions operating on graph morphisms are defined below. Projection functions $f_V$ and $f_E$ extract the two components of a graph morphism. Functions* gid *and $\varnothing_{GM}$ yield a graph's identity and empty morphisms, respectively. Functions domg, and codg yield the domain and co-domain of a morphism, respectively.*

$$f_V : GrM \rightarrow V \nrightarrow V \quad f_E : GrM \rightarrow E \nrightarrow E \quad \mathrm{gid} : Gr \rightarrow GrM \quad \varnothing_{GM} : GrM$$
$$f_V(fv, fe) = fv \quad f_E(fv, fe) = fe \quad \mathrm{gid} = (id\, Ns\, G, id\, Es\, G) \quad \varnothing_{GM} = (\{\}, \{\})$$
$$domg\ m = (\mathrm{dom}(fV\ m), \mathrm{dom}(fE\ m)) \quad codg\ m = (\mathrm{ran}(fV\ m), \mathrm{ran}(fE\ m))$$

**Laws.** *The following laws were proved in Isabelle:*

$$\vdash gid\ \varnothing_G = \varnothing_{GM} \quad G \in Gr \vdash gid\, G \in G \rightarrow_G G \quad \vdash domg\ \varnothing_{GM} =_p Els\ \varnothing_G \quad \vdash codg\ \varnothing_{GM} =_p Els\ \varnothing_G$$
$$f \in G_1 \rightarrow_G G_2 \vdash domg\ f =_p Els\, G1 \quad f \in G_1 \rightarrow_G G_2 \vdash codg\ f \subseteq_p Els\, G2$$
$$f \in G_1 \nrightarrow_G G_2 \vdash domg\ f \subseteq_p Els\, G1 \quad f \in G_1 \nrightarrow_G G_2 \vdash codg\ f \subseteq_p Els\, G2$$
$$f \in G_1 \rightarrow_G G_2 \vdash f \in G_1 \nrightarrow_G G_2$$

□

Definition 10 (Composition of Graph Morphisms). *The functional composition of graph morphisms $f : G_1 \rightarrow_G G_2$ and $g : G_2 \rightarrow_G G_3$, where $G_{i \in \{1,3\}} : Gr$, is defined as:*

$$g \circ_G f = ((f_V\ g) \circ (f_V\ f), (f_E\ g) \circ (f_E\ f))$$

*Operators union composition $\cup_{GM}$ and generalised union composition $\bigcup_{GM}$ are defined as:*

$$\_ \cup_{GM} \_ : GrM \times GrM \rightarrow GrM \quad \bigcup_{GM} \_ : \mathbf{P}\, GrM \rightarrow GrM$$
$$f \cup_{GM} g = (fV\, f \cup fV\, g, fE\, f \cup fE\, g)$$
$$\bigcup_{GM} \varnothing = \varnothing_{GM}$$
$$\bigcup_{GM} (\{f\} \cup fs) = f \cup_{GM} (\bigcup_{GM} fs)$$

**Laws.** *The following laws were proved in Isabelle:*

$$f \in G_1 \rightarrow_G G_2\ ;\ g \in G_2 \rightarrow_G G_3 \vdash g \circ_G f \in G1 \rightarrow_G G_3 \quad \vdash (h \circ_G g) \circ_G f = h \circ_G (g \circ_G f)$$
$$f \in G_1 \rightarrow_G G_2\ ;\ g \in G_2 \rightarrow_G G_3\ ;\ \boxplus\langle G_1, G_2, G_3\rangle \vdash f \cup_{GM} g \in (G_1 \cup_{GM} G_2) \rightarrow_G (G_3 \cup_{GM} G_4)$$
$$f \in G_1 \rightarrow_G G_2 \vdash f \circ_G gid_{G_1} = f \quad f \in G_1 \rightarrow_G G_2 \vdash gid_{G_2} \circ_G f = f$$

□

Definition 11 (Graphs with Typing). *A graph with typing is a pair $GwT = (G, t)$, made-up of a graph $G : Gr$ and a morphism $t : GrM$, such that $domg\ t =_p Els\, G$ (the morphism's domain is G's nodes and edges). The set of such graphs $GrwT$, such that $GwT : GrwT$, is defined as:*

$$GrwT = \{(G, t) \mid G \in Gr \land t \in GrM \land domg\ t =_p Els\, G\}$$

*The following projection functions extract the elements of such graphs:*

$$(G, t)^G = G \quad (G, t)^T = t$$

*The empty element ($\varnothing_{Gwt}$) and union composition ($\cup_{GwT}$) are defined as follows:*

$$\varnothing_{Gwt} = (\varnothing_G, \varnothing_{GM}) \quad (G_1, t_1) \cup_{GwT} (G_2, t_2) = (G_1 \cup_G G_2, t_1 \cup_{GM} t_2)$$

□

Definition 12 (Graphs with Extra Typing). *A graph with extra typing is a pair $GwET = (G, et)$, made-up of a graph with typing $G : GrwT$ (def. 11) and a partial morphism $et : GrM$, such that $domg\ et \subseteq_p Els\, G$ (the morphism's domain is a subset of G's nodes and edges). The set of such graphs $GrwET$, such that $GwET : GrwET$, is defined as:*

$$GrwET = \{(Gwt, et) \mid Gwt \in GrwT \land et \in GrM \land domg\ et \subseteq_p Els\, Gwt\}$$

*The following projection functions extract the elements of such graphs:*

$$(Gwt, et)^{Gw} = G \quad (Gwt, et)^{ET} = et \quad (Gwt, et)^G = Gwt^G \quad (Gwt, et)^T = Gwt^T$$

*The empty element ($\varnothing_{GwET}$) and union composition ($\cup_{GwET}$) are defined as:*

$$\varnothing_{GwET} = (\varnothing_{Gwt}, \varnothing_{GM}) \quad (G_1, et_1) \cup_{GwET} (G_2, et_2) = (G_1 \cup_{GwT} G_2, et_1 \cup_{GM} et_2)$$

□

Definition 13 (SG Node and Edge Types). *SG's node types (set SGNT) are: normal, abstract, proxy, enumeration, value, and virtual. SG's edge types (set SGET) are: inheritance, composition (or containment), relation, derived, path and value constraint. Composition and relation edges can be either bi-directional or uni-directional.*

$$SGNT = \{nnrml, nabst, nprxy, nenum, nval, nvirt\}$$
$$SGET = \{et \mid et \in \{einh, eder, epath, evcnt\}$$
$$\quad \lor\ et \in \{ae\ d \mid ae \in \{ecomp, erel\} \land d \in \{dbi, duni\}\}\}$$

*$\prec_{NT}$, an ordering relation on SGNT, indicates the node types that are inheritance-relatable:*

$$nt_1 \prec_{NT} nt_2 \Leftrightarrow nt_1 \neq nprxy \land nt_2 \neq nval$$
$$\quad \land\ nt_1 = nabst \Rightarrow nt_2 \in \{nabst, nvirt, nprxy\} \land (nt_1 = nenum \Rightarrow nt_2 \in \{nvirt, nprxy\})$$

*Above, $\prec_{NT}$ stipulates that (i) proxies must not inherit, (ii) values must not be inherited, (iii) abstract nodes may inherit from abstract, virtual and proxy nodes only, (iv) enumerations may only inherit virtuals or proxies.*

*$\leq_{rNT}$ and $\leq_{ET}$ are ordering relations on sets SGNT and SGET, respectively, indicating node and edge types that can be refinement-related, respectively:*

$$nt_1 \leq_{rNT} nt_2 \Leftrightarrow nt_1 = nt_2 \vee nt_1 = nprxy \vee nt_1 = nnrml \wedge nt_2 = nprxy$$
$$\vee\ nt_2 \in \{nabst, nvirt\} \wedge nt_1 \in \{nnrml, nvirt, nabst\} \vee nt_2 = nnrml$$
$$et_1 =_{ET} et_2 \Leftrightarrow et_1 = et_2$$
$$\vee\ (\forall d_1, d_2 : SGED \bullet et_1 = erel\ d_1 \wedge et_2 = erel\ d_2 \vee et_1 = ecomp\ d_1 \wedge et_2 = ecomp\ d_2)$$
$$et_1 \leq_{ET} et_2 \Leftrightarrow \neg\{einh, eder, epath, evcnt\} \subseteq \{et_1, et_2\} \wedge (et_1 =_{ET} et_2$$
$$\vee\ et_2 = erel\ dbi \wedge (et_1 = ecomp\ dbi \vee et_1 = ecomp\ duni) \vee et_2 = erel\ duni \wedge et_1 = ecomp\ duni)$$

□

Definition 14. *(Multiplicities) The multiplicity sets include: (i) MultVal (multiplicity values), the disjoint union (symbol ⊎) of natural numbers with symbol ∗ (many); (ii) MultR (multiplicity ranges), pairs of lower and upper bound multiplicities; (iii) MultC (multiplicity compounds), either a single multiplicity value or a range; and (iv) Mult, a non-empty set of multiplicity compounds (a singleton when there is just one multiplicity, a larger set otherwise):*

$$MultVal = \mathbb{N} \uplus \{*\}\quad MultR = \{(lb, ub) \mid lb \in \mathbb{N} \wedge ub \in MultVal \wedge (ub = * \vee (ub \in \mathbb{N} \wedge lb \leq ub))\}$$
$$MultC = MultVal \uplus MultR \quad MultCMany = \{*, (0, *)\}$$
$$Mult = \{ms : \mathbf{P}_1\ MultC \mid \#\ ms > 1 \Rightarrow \forall\ m : ms \bullet m \notin MultCMany\}$$

*Function mlb and mub give lower and upper bounds of a $m \in MultC$, respectively, where $k \in \mathbb{N}$:*

$$mlb, mub : Mult \nrightarrow MultVal$$
$$mlb\ * = 0 \quad mlb\ k = k \quad mlb\ (k, m) = k \quad mub\ m = m \quad mub\ (k, m) = m$$

*Relation $\leq_{mv}$ stipulates an ordering on set MultVal:*

$$m_1 \leq_{mv} m_2 \Leftrightarrow m_2 = * \vee \{m_1, m_2\} \subseteq \mathbb{N} \wedge m_1 \leq m_2$$

*Relation $\wr$ says whether a natural number complies to a multiplicity:*

$$k \wr m \Leftrightarrow mlb\ m \leq_{mv} k \wedge k \leq_{mv} mub\ m$$

*Predicate $\cdots$ says whether some natural number complies with any of the given multiplicities:*

$$k \cdots \{m\} \Leftrightarrow k \wr m \quad k \cdots \{m\} \cup ms \Leftrightarrow k \wr m \vee k \cdots \{m\} \cup ms$$

*Relation $\leq_{\mathcal{M}c}$ stipulates an ordering on MultC to say whether one multiplicity is within another:*

$$mc_1 \leq_{\mathcal{M}c} mc_2 \Leftrightarrow mlb\ mc_2 \leq_{mv} mlb\ mc_1 \wedge mub\ mc_1 \leq_{mv} mub\ mc_2$$

*Relation $\leq_{\mathcal{M}}$ stipulates an ordering on Mult to say whether one multiplicity set is within another:*

$$m_1 \leq_{\mathcal{M}} m_2 \Leftrightarrow \forall\ mc_1 : m_1 \bullet \exists\ mc_2 : m_2 \bullet mc_1 \leq_{\mathcal{M}c} mc_2$$

*The multiplicity sets for many, range and alternative multiplicities, respectively, are:*

$$MultMany = \{ms : Mult \mid \exists\, m_c : MultC \bullet ms = \{m_c\} \land m_c \in MultCMany\}$$
$$MultRange = \{ms : Mult \mid \exists\, m : MultC \bullet ms = \{m\} \land (m \in \mathbb{N} \land m > 1)$$
$$\lor \exists\, lb : \mathbb{N}\,;\, um : MultVal \mid 2 \leq_{mv} um \bullet m = (lb, um)\,\}$$
$$MultEither = \{ms : Mult \mid \#\, ms > 1\}$$

*Predicate $\propto$ describes the multiplicities permitted by the different edge types:*

$$et \propto (m_1, m_2) \Leftrightarrow et = erel\ dbi \lor et = eder \lor et = ecomp\ duni \land m_1 = \{1\}$$
$$\lor\ et = erel\ duni \land m_1 \in MultMany \lor et = ecomp\ dbi \land m_1 \in \{\{1\}, \{(0,1)\}\}$$

*The above says: (i) bi-directional relations and derived edges can have any multiplicity, (ii) source multiplicities of uni-directional containments and relations are* 1 *and* $*$*, respectively, and (iii) source multiplicities of bi-directional containments are either* 1 *or* 0 . . 1. □

Definition 15 (Path Expressions). *Path expressions capture edge paths used in path-based constraint edges (either derived or path). Recursive set of path expressions, PE, is defined from the recursive sets of path expressions atoms and composites, PEA and PEC, respectively:*

$$PEA ::= E \mid E^{-1} \quad PEC ::= PEA \mid V \lhd_{PE} PEA \mid PEA \rhd_{PE} V \quad PE ::= PEC \mid PEC \fatsemi_{PE} PE$$

*A PEA is either a single edge or the reverse of an edge. A PEC is either a PEA, a PEA whose domain (or source) is restricted to a particular node, or a PEA whose range is restricted to a node. A PE is either a single path expression composite, or path expression composite followed by a path expression.*

*Function $ePEA : PEA \to E$, which gives the edge of a PEA, is defined inductively:*

$$ePEA\ e = e \quad ePEA\ e^{-1} = e$$

*Functions $srcPEA : Gr \to PEA \to V$, $srcPEC : Gr \to PEC \to V$ and $srcPE : Gr \to PE \to V$, defined inductively, take a graph G and either a PEA, PEC, or PE, respectively, to give the path's source with respect to G. Given $G : Gr$, $e : E$, $pea : PEA$, $v : V$, $pec : PEC$, and $pe : PE$, the definitions are as follows:*

$$srcPEA\ G\ e = src\ G\ e \quad srcPEA\ G\ e^{-1} = tgt\ G\ e \quad srcPEC\ G\ pea = srcPEA\ G\ pea$$
$$srcPEC\ G\ (v \lhd_{PE} pea) = srcPEA\ pea \quad srcPEC\ G\ (pea \rhd_{PE} v) = srcPEA\ pea$$
$$srcPE\ G\ pec = srcPEC\ G\ pec \quad srcPE\ G\ (pec \fatsemi_{PE} pe) = srcPEC\ G\ pe_1$$

*Functions $tgtPEA : PEA \to E$, $tgtPEC : PEC \to E$, and $tgtPE : PE \to E$ give the target node of a PEA, PEC or PE with respect to a given graph G:*

$$tgtPEA\ G\ e = tgt\ G\ e \quad tgtPEA\ G\ e^{-1} = src\ G\ e \quad tgtPEC\ G\ pea = tgtPEA\ G\ pea$$
$$tgtPEC\ G\ (v \lhd_{PE} pea) = tgtPEA\ pea \quad tgtPEC\ G\ (pea \rhd_{PE} v) = tgtPEA\ pea$$
$$tgtPE\ G\ pec = tgtPEC\ G\ pec \quad tgtPE\ G\ (pec \fatsemi_{PE} pe) = tgtPE\ G\ pe$$

*Functions* $startEA_C : PEC \to PEA$, $startEA : PE \to PEA$, $endEA_C : PEC \to PEA$ *and* $endEA : PE \to PEA$, *defined inductively on sets PEC and PE, give either start (or source) or end (or target) PEAs, respectively:*

$$startEA_C\ pea = pea \quad startEA_C\ (v \lhd_{PE} pea) = pea \quad startEA_C\ (pea \rhd_{PE} v) = pea$$
$$startEA\ pec = startEA_C\ pec \quad startEA\ (pec \fatsemi_{PE} pe) = startEA_C\ pec$$
$$endEA_C\ pea = pea \quad endEA_C\ (v \lhd_{PE} pea) = pea \quad endEA_C\ (pea \rhd_{PE} v) = pea$$
$$endEA\ pec = endEA_C\ pec \quad endEA\ (pec \fatsemi_{PE} pe) = endEA\ pe$$

*Functions* $rsrcPE, rtgtPE : PE \to E$ *reduce a PE to either the source or target edges, respectively; they are defined from startEA and endEA, respectively:*

$$rsrcPE = ePEA \circ startEA \quad rtgtPE = ePEA \circ endEA$$

□

Definition 16 (Information of Value Constraint Edges). *Set SGVCEOP defines the allowed operators of a value-constraint edge: equality (eq), inequality (neq), less or equal than (leq), greater or equal than (geq), less than (lt), and greater than (gt).*

$$SGVCEOP = \{eq, neq, leq, geq, lt, gt\}$$

*VCI holds the information associated with a value-constraint edge: an operator, SGVCEOP above, and an optional edge, which indicates the constrained edge or self (constraint refers to source node):*

$$VCI = SGVCEOP \times opt[E]$$

□

Definition 17 (Structural Graphs). *A structural graph* $SG = (G, nty, ety, sm, tm, p, d, vci)$ *comprises: (i) a graph* $G : Gr$ *(def. 7), (ii) two colouring functions for nodes and edges,* $nt : Ns_G \to SGNT$ *and* $et : Es_G \to SGET$ *(def. 13), (iii) source and target multiplicity functions* $sm, tm : Es_G \nrightarrow Mult$ *(def. 14), (iv) a partial function* $p : Es_G \nrightarrow PE$ *giving the path expression of either derived or path edges (def. 15), (v) a relation* $d : Es_G \leftrightarrow Es_G$ *giving the dependencies between path edges, and (vi) a partial function* $vci : Es_G \nrightarrow VCI$ *(def. 16) giving the information associated with value-constraint edges.*

*Base set* $SGr_0$, *such that* $SG : SGr_0$, *is defined as:*

$$SGr_0 = \{(G, nt, et, sm, tm, p, d, vci) \mid G \in Gr \land nt \in Ns_G \to SGNT \land et \in Es_G \to SGET$$
$$\land\ sm \in Es_G \nrightarrow Mult \land tm \in Es_G \nrightarrow Mult \land p \in Es_G \nrightarrow PE \land d \in Es_G \leftrightarrow Es_G \land vci \in Es_G \nrightarrow VCI\}$$

*Several projection functions extract the eight elements of a* $SG : SGr_0$*:*

$$gr(G, nt, et, sm, tm, p, d, vci) = G \quad nty(G, nt, et, sm, tm, p, d, vci) = nt$$
$$ety(G, nt, et, sm, tm, p, d, vci) = et \quad srcm(G, nt, et, sm, tm, p, d, vci) = sm$$
$$tgtm(G, nt, et, sm, tm, p, d, vci) = tm \quad pe(G, nt, et, sm, tm, p, d, vci) = p$$
$$ds(G, nt, et, sm, tm, p, d, vci) = d \quad vcei(G, nt, et, sm, tm, p, d, vci) = vci$$

*In the empty SG* ($\varnothing_{SG}$) *all its eight components are empty:*

$$\varnothing_{SG} = (\varnothing_G, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing)$$

*Graph functions Ns, Es, src and tgt of def. 7 are considered to be applicable to SGs also. For a* $SG : SGr_0$*,* $Ns\,SG$ *is interpreted to be* $Ns(gr\,SG)$*, and likewise for Es, src and tgt.*

*Function EsTy SG ets gives all edges of SG with edge types ets, NsTy SG nts gives all nodes of SG with node types nts:*

$$EsTy\ SG\ ets = (ety\ SG)^{\sim}(\!|ets|\!) \quad NsTy\ SG\ nts = (nty\ SG)^{\sim}(\!|nts|\!) \quad EsW\ SG = EsTy\ SG\ \{ewander\}$$

*Several functions yield the edges of different types, namely: (i) EsA (association which includes relation and composition), (ii) EsM (multiplicity), (iii) EsI (inheritance), (iv) EsD (derived), (v) EsPa (path), (vi) EsVCnt (value constraint) and (vii) EsPaCnt (path constraint).*

$$EsA\ SG = EsTy\ SG\ \{ae\ d \mid ae \in \{ecomp, erel\} \wedge d \in \{dbi, duni\}\} \quad EsI\ SG = EsTy\ SG\ \{einh\}$$
$$EsD\ SG = EsTy\ SG\ \{eder\} \quad EsPa\ SG = EsTy\ SG\ \{epath\} \quad EsM\ SG = EsA\ SG \cup EsD\ SG$$
$$EsVCnt\ SG = EsTy\ SG\ \{evcnt\} \quad EsPaCnt\ SG = EsD\ SG \cup EsPa\ SG$$

*Likewise for nodes: (i) NsP (proxy), (ii) NsVi (virtual), (iii) NsN (normal), (iv) NsVa (value) and (v) NsEther (ethereal, includes abstract, virtual and enumeration).*

$$NsP\ SG = NsTy\ SG\ \{nprxy\} \quad NsVi\ SG = NsTy\ SG\ \{nvirt\} \quad NsVa\ SG = NsTy\ SG\ \{nval\}$$
$$NsN\ SG = NsTy\ SG\ \{nnrml\} \quad NsEther\ SG = NsTy\ SG\ \{nvirt, nabst, nenum\}$$

*Above,* $^{\sim}$ *is the relation inverse and* $(\!|\ |\!)$ *is the relation image (def. 2).* □

Definition 18 (Inheritance in SGs). *Function* $\Uparrow$ *gives the inheritance graph by restricting the given SG to its inheritance edges. The inheritance relation* $\prec$ *is derived from* $\Uparrow$*:*

$$\Uparrow : SGr_0 \rightarrow Gr \qquad \prec : SGr_0 \rightarrow V \leftrightarrow V$$
$$\Uparrow\ SG = gr\ SG \bowtie_{Es} EsI\ SG \quad \prec_{SG} = (\Uparrow\ SG)^{\leftrightarrow}$$

*The following provide further machinery: (i)* $\preceq$ *is the reflexive and transitive closure of* $\prec$ *(def. 17), (ii)* $src^*$ *and* $tgt^*$ *extend graph functions src and tgt (def. 7) to yield relations* $E \leftrightarrow V$ *between edges and vertices to cater for inheritance, (iii) variants* $src^*_M$ *and* $tgt^*_M$*, used for morphisms, do the same but are restricted to allow association edges only:*

$$\preceq : SGr_0 \rightarrow V \leftrightarrow V \quad src^*_0, src^*, tgt^*_0, tgt^* : SGr_0 \rightarrow E \leftrightarrow V$$
$$\preceq\ SG = (\prec\ SG)^* \quad src^*_0\ SG\ es = es \lhd (srcr\ SG) \quad tgt^*_0\ SG\ es = es \lhd (tgtr\ SG)$$
$$src^*\ SG = src^*_0\ SG\ (EsA\ SG \cup EsPaCnt\ SG) \fatsemi (\preceq\ SG)^{\sim}$$
$$tgt^*\ SG = tgt^*_0\ SG\ (EsA\ SG \cup EsPaCnt\ SG) \fatsemi (\preceq\ SG)^{\sim}$$
$$src^*_M\ SG = src^*_0\ SG\ (EsA\ SG) \fatsemi (\preceq\ SG)^{\sim} \quad tgt^*_M\ SG = tgt^*_0\ SG\ (EsA\ SG) \fatsemi (\preceq\ SG)^{\sim}$$

*Above,* $\fatsemi$ *is relation composition (def. 2). Functions* $src^*$ *and* $tgt^*$*, which depart from* $src^*_0$ *and* $tgt^*_0$*, say that ancestor edges are also edges of descendants affecting association and constraint edges;* $src^*_M$ *and* $tgt^*_M$ *affect association edges only.* □

Definition 19 (Partial SGs). *Function srcma, which totalises function srcm, gives source multiplicities of all multiplicity edges; multiplicities of uni-directional composition and relation edges become* 1 *and* $*$ *(many), respectively:*

$$srcma : SGr_0 \rightarrow E \nrightarrow Mult$$
$$srcma\ SG = (srcm_{SG}) \oplus (EsTy_{SG}\ \{ecomp\ duni\} \times \{\{1\}\}) \oplus (EsTy_{SG}\ \{erel\ duni\} \times \{\{*\}\})$$

*Predicate $\mathcal{M}etysOk$ checks if multiplicity edges comply with the multiplicity restrictions of their edges types using relation $\propto$ (def 14):*

$$\mathcal{M}etysOk\ SG \Leftrightarrow \forall\, e : EsM_{SG} \bullet (ety_{SG}\ e) \propto (srcma_{SG}\ e, tgtm_{SG}\ e)$$

*Predicate* inhOk *says whether an inheritance hierarchy is well-formed. It requires that: (i) inheritance relations comply with the inheritance restrictions of the corresponding node types as captured by $\prec_{NT}$ (def. 13), and (ii) the inheritance graph is acyclic:*

$$\text{inhOk}\ SG \Leftrightarrow (\forall\, v, v' : Ns_{SG} \bullet v \prec_{SG} v' \Rightarrow nty_{SG}\ v \prec_{NT} nty_{SG}\ v') \wedge \oslash \Uparrow_{SG}$$

*Predicate* VCntEsOk *checks the well-formedness of value constraint edges (VCEs): (i) all VCEs have the corresponding information (vcei is a total function), (ii) VCEs must constrain association edges only, and (iii) VCEs have either value or proxy nodes as targets.*

$$\begin{aligned}&\text{VCntEsOk}\ SG \Leftrightarrow vcei_{SG} \in EsVCnt_{SG} \rightarrow VCI \wedge \bigcup(map\ second\ ((\text{ran} \circ vcei)\ SG)) \subseteq EsA_{SG}\\ &\quad \wedge (tgt_{SG})\ (\!|\ EsVCnt_{SG}\ |\!) \subseteq NsVa_{SG} \cup NsP_{SG}\end{aligned}$$

*Set of partial SGs, SGr, is defined from the $SGr_0$ base set:*

$$\begin{aligned}&SGr = \{SG \mid SG \in SGr_0 \wedge \{srcma_{SG}, tgtm_{SG}\} \subseteq EsM_{SG} \rightarrow Mult \wedge ds_{SG} \in (\text{antireflexive}\ EsPa_{SG})\\ &\quad \wedge \text{dom}\ pe_{SG} = EsPaCnt_{SG} \wedge \mathcal{M}etysOk\ SG \wedge \text{inhOk}\ SG \wedge \text{VCntEsOk}\ SG\,\}\end{aligned}$$

*Partial SGs are subject to the following constraints: (i) multiplicity edges (set EsM above) must have source and target multiplicities (second conjunct); (ii) relation ds (dependencies) between path edges is anti-reflexive; (iii) function pe has path constraint edges in its domain only; (iv) edge multiplicities comply with the multiplicity restrictions of their types ($\mathcal{M}etysOk$ above); (i) the SG's inheritance is well-formed (*inhOk *above); (vi) the VCEs are well-formed (*VCntEsOk *above).*

**Laws.** *The following laws were proved in Isabelle:*

$$SG \in SGr \vdash SG \circ\!\!\rightarrow\!\!\circ^*\ ns \subseteq Es_{SG} \qquad SG \in SGr \vdash SG \circ\!\!\rightarrow\!\!\circ^*\ ns_1 \cup ns_2 = SG \circ\!\!\rightarrow\!\!\circ^*\ ns_1 \cup SG \circ\!\!\rightarrow\!\!\circ^*\ ns_2$$
$$SG \in SGr\ ;\ ns_2 \cap Ns_{SG} = \varnothing \vdash SG \circ\!\!\rightarrow\!\!\circ^*\ ns_1 \cup ns_2 = SG \circ\!\!\rightarrow\!\!\circ^*\ ns_1 \qquad \vdash \varnothing_{SG} \in SGr$$

□

Definition 20 (Total SGs). *Total SGs must satisfy certain conditions. Predicate derInhOk checks the underlying commuting of derived edges and respective edge paths:*

$$derInhOk\ SG \Leftrightarrow \forall\, e : EsD_{SG} \bullet (gr\ SG \bowtie_{Es} esPE\ (pe\ SG\ e))^{\leftrightarrow} \fatsemi (\preceq SG) \subseteq (\preceq SG) \fatsemi (gr\ SG \bowtie_{Es} esPE\ (pe\ SG\ e))^{\leftrightarrow}$$

*where srcPE and tgtPE, which get source and target node, respectively, of the associated path edge, come from def. 15.*

*okPEASrc and okPEATgt check whether the source or target, respectively, of a path expression atom (PEA) is valid with respect to the extended source ($src^*$) or target ($tgt^*$) relations, respectively:*

$$okPEASrc(SG, v, e) \Leftrightarrow (e, v) \in {src^*}_{SG} \qquad okPEASrc(SG, v, e^{-1}) \Leftrightarrow (e, v) \in {tgt^*}_{SG}$$
$$okPEATgt(SG, v, e) \Leftrightarrow (e, v) \in {tgt^*}_{SG} \qquad okPEATgt(SG, v, e^{-1}) \Leftrightarrow (e, v) \in {src^*}_{SG}$$

*Functions okPEA, okPEC and okPE check whether a given path expression atom (PEA), compound (PEC), or whole (PE), respectively, is valid with respect to a given SG : SGr:*

$$okPEA(SG, e) \Leftrightarrow e \in Es\ SG \quad okPEA(SG, e^{-1}) \Leftrightarrow e \in Es\ SG$$
$$okPEC(SG, pea) \Leftrightarrow okPEA(SG, pea)$$
$$okPEC(SG, v \lhd_{PE} pea) \Leftrightarrow okPEA(SG, pea) \land okPEASrc(srcr, tgtr, v, pea)$$
$$okPEC(SG, pea \rhd_{PE} e) \Leftrightarrow okPEA(SG, pea) \land okPEATgt(srcr, tgtr, v, pea)$$
$$okPE(SG, pec) \Leftrightarrow okPEC(SG, pec)$$
$$okPE(SG, pec \fatsemi_{PE} pe) \Leftrightarrow okPEC(SG, pec) \land okPE(SG, pe)$$
$$\land\ (tgtPEC\ (SG \bowtie_{Es} EsA_{SG})\ pec \preceq_{SG} srcPE\ (SG \bowtie_{Es} EsA_{SG})\ pe$$
$$\lor\ srcPE\ (SG \bowtie_{Es} EsA_{SG})\ pe \preceq_{SG} tgtPEC\ (SG \bowtie_{Es} EsA_{SG})\ pec)$$

*The equations above are as follows: (i) the first two say that a PEA is valid provided the encapsulated edge belongs to the given SG, (ii) a PEC holding a PEA is valid provided the PEA is valid, (iii) a domain restriction expression is valid provided the PEA is valid and the pair node, edge is valid with respect to the source, (iii) similarly for the range restriction expression, and finally (iv) a sequence of PEs is valid if they are individually valid and if the target of the first expression is inheritance related to the source of the second expression.*

*Predicates* isVCEECnt *and* isVCENCnt *indicate whether a VCE refers to either an edge or the source node, respectively, which is to say either when an edge is provided as part of the edge information or when such an edge is not provided, respectively:*

$$\text{isVCEECnt}(SG, vce) \Leftrightarrow (second \circ vcei_{SG})\ vce \neq \varnothing$$

$$\text{isVCENCnt}(SG, vce) \Leftrightarrow (second \circ vcei_{SG})\ vce = \varnothing$$

*Predicate* commonAncestor *says whether two given nodes have a common ancestor in the given SG:*

$$\text{commonAncestor}\ (SG, n_1, n_2) \Leftrightarrow \exists\ nt : Ns_{SG} \bullet n_1 \preceq_{SG} nt \land n_2 \preceq_{SG} nt$$

*Predicate* EsVCntsOk *indicates whether VCEs are well-formed. There are two cases, either: (i) the VCE constrains an edge, in which case the edge's target node is required to have an ancestor in common with the target of the embedded edge, or (ii) the VCE constrains the source node, in which the case the edge's target node must have an ancestor in common with the edge's source node.*

$$\text{EsVCntsOk}\ SG \Leftrightarrow \forall\ vce : EsVCnt_{SG} \bullet$$
$$\text{isVCEECnt}(SG, vce) \Rightarrow \text{commonAncestor}\ (SG, tgt_{SG}\ ((\text{the} \circ second \circ vcei_{SG})\ vce), tgt_{SG}\ vce)$$
$$\land\ \text{isVCENCnt}(SG, vce) \Rightarrow \text{commonAncestor}\ (SG, src_{SG}\ vce, tgt_{SG}\ vce)$$

*Predicate EsCntsOk checks the conformance of derived edges (predicate derInhOk above), the validity of path expressions of path edges (function okPE, def. 15) and the well-formedness of VCEs in a total setting (predicate* EsVCntsOk *above):*

$$EsCntsOk\ SG \Leftrightarrow derInhOk\ SG \land \forall\ e : EsPaCnt_{SG} \bullet okPE(gr\ SG \bowtie_{Es} EsA_{SG}, src^*{}_{SG}, tgt^*{}_{SG}, pe_{SG}\ e)$$
$$\land\ \text{EsVCntsOk}\ SG$$

*A total SG is a partial SG : SGr (def 19) in which: (i) all ethereal nodes (set NsEther of def 17) are inherited, (ii) constraint edges are valid (predicate EsCntsOk above), and (iii) the inheritance graph without virtuals forms a tree (i.e. virtuals apart,*

*the inheritance must be single). Set TSGr of total SGs is as follows:*

$$TSGr = \{SG : SGr \mid NsEther_{SG} \subseteq \mathrm{ran}(\prec_{SG}) \wedge EsCntsOk_{SG} \wedge (\Uparrow_{SG} \ominus_{Ns} NsV_{SG})^{\leftrightarrow} \in \mathrm{tree}\}$$

□

Definition 21 (Composition of SGs). *The union of SGs ($\cup_{SG}$) is defined as:*

$$SG_1 \cup_{SG} SG_2 = (gr_{SG_1} \cup_G gr_{SG_2}, nty_{SG_1} \cup nty_{SG_2}, ety_{SG_1} \cup ety_{SG_2},\\ srcm_{SG_1} \cup srcm_{SG_2}, tgtm_{SG_1} \cup tgtm_{SG_2}, pe_{SG_1} \cup pe_{SG_2}, ds_{SG_1} \cup ds_{SG_2})$$

*SG subsumption ($\odot^{SG}$) subsumes a SG given a substitution of proxies by other SG nodes:*

$$\_ \odot^{SG} \_ : SGr \times (V \leftrightarrow V) \nrightarrow SGr$$

$$SG \odot^{SG} s = \begin{cases} (gr_{SG} \odot s, (\mathrm{dom}\, s \setminus \mathrm{ran}\, s) \lhd nty_{SG}, ety_{SG}, srcm_{SG}, tgtm_{SG}, pe_{SG}, ds_{SG}) & \text{if } s \in NsP_{SG} \nrightarrow Ns_{SG} \\ SG & \text{otherwise} \end{cases}$$

*This subsumes underlying base graph (def. 8) and removes subsumed nodes from function nty.*

*SG disjointness is defined from graph disjointness as:*

$$\boxminus_{SGs}(SG_1, SG_2) \Leftrightarrow \boxminus(gr\ SG_1, gr\ SG_2)$$

**Laws.** *The following laws were proved in Isabelle:*

$$\vdash \varnothing_{SG} \cup_{SG} SG = SG \quad \vdash \varnothing_{SG} \odot^{SG} s = \varnothing_{SG} \quad SG \in SGr \vdash SG \odot^{SG} \varnothing = SG$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash (SG_1 \cup_{SG} SG_2) \circ\!\!\rightarrowtail\!\!\circ^{*} vs = (SG_1 \circ\!\!\rightarrowtail\!\!\circ^{*} vs) \cup (SG_2 \circ\!\!\rightarrowtail\!\!\circ^{*} vs)$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus(SG_1, SG_2) \vdash EsTys(SG_1 \cup_{SG} SG_2)\ ets = EsTys\ SG_1\ ets \cup EsTys\ SG_2\ ets$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash NsTys(SG_1 \cup_{SG} SG_2)\ nts = NsTys\ SG_1\ nts \cup NsTys\ SG_2\ nts$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash EsA(SG_1 \cup_{SG} SG_2) = EsA\ SG_1 \cup_{SG} EsA\ SG_2$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash EsI(SG_1 \cup_{SG} SG_2) = EsI\ SG_1 \cup_{SG} EsI\ SG_2$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash EsD(SG_1 \cup_{SG} SG_2) = EsD\ SG_1 \cup_{SG} EsD\ SG_2$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash EsW(SG_1 \cup_{SG} SG_2) = EsW\ SG_1 \cup_{SG} EsW\ SG_2$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash EsC(SG_1 \cup_{SG} SG_2) = EsC\ SG_1 \cup_{SG} EsC\ SG_2$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash NsP(SG_1 \cup_{SG} SG_2) = NsP\ SG_1 \cup_{SG} NsP\ SG_2$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash NsO(SG_1 \cup_{SG} SG_2) = NsO\ SG_1 \cup_{SG} NsO\ SG_2$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash NsV(SG_1 \cup_{SG} SG_2) = NsV\ SG_1 \cup_{SG} NsV\ SG_2$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash \boxminus(\Uparrow\ SG_1, \Uparrow\ SG_2)$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash \Uparrow(SG_1 \cup_{SG} SG_2) = \Uparrow\ SG_1 \cup_G \Uparrow\ SG_2$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash \prec(SG_1 \cup_{SG} SG_2) = \prec_{SG_1} \cup \prec_{SG_2}$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash \preceq(SG_1 \cup_{SG} SG_2) = \preceq_{SG_1} \cup \preceq_{SG_2}$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash src^{*}(SG_1 \cup_{SG} SG_2) = src^{*}{}_{SG_1} \cup src^{*}{}_{SG_2}$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash tgt^{*}(SG_1 \cup_{SG} SG_2) = tgt^{*}{}_{SG_1} \cup tgt^{*}{}_{SG_2}$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash \oslash(\Uparrow\ SG_1 \cup_G \Uparrow\ SG_2) \Leftrightarrow \oslash(\Uparrow\ SG_1) \wedge \oslash(\Uparrow\ SG_2)$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash SG_1 \cup_{SG} SG_2 = SG_2 \cup_{SG} SG_1$$
$$SG_1, SG_2 \in SGr\ ;\ \boxminus_{SGs}(SG_1, SG_2) \vdash SG_1 \cup_{SG} SG_2 \in SGr$$

□

Definition 22 (SG Morphisms). *Given $SG_s$, $SG_t$ : SGr, a SG morphism $m : SG_s \rightarrow_{SG} SG_t$ is a pair of functions $m = (fv, fe)$ mapping nodes and edges, respectively. The set of morphisms between the mapped SGs, $SG_s \rightarrow_{SG} SG_t$, is defined as:*

$$SG_s \rightarrow_{SG} SG_t = \{ (fv, fe) \mid fv \in Ns_{SG_s} \rightarrow Ns_{SG_t} \land fe \in EsA_{SG_s} \rightarrow EsA_{SG_t} \\ \land fv \circ src^*_{SG_s} \subseteq src^*_{SG_t} \circ fe \land fv \circ tgt^*_{SG_s} \subseteq tgt^*_{SG_t} \circ fe \land fv \circ \preceq_{SG_s} \subseteq \preceq_{SG_t} \circ fv \}$$

*Only association edges are mapped (set EsA, def. 17, excludes inheritance and constraint edges). Commuting is expressed as subsetting ($\subseteq$ rather than $=$) to cater for inheritance.*

*Partial SG morphisms are also defined:*

$$SG_s \nrightarrow_{SG} SG_t = \{ (fv, fe) \mid fv \in Ns_{SG_s} \nrightarrow Ns_{SG_t} \land fe \in EsA_{SG_s} \nrightarrow EsA_{SG_t} \\ \land fv \circ (\mathrm{dom}\, fe \lhd src^*_{SG_s}) \subseteq src^*_{SG_t} \circ fe \land fv \circ (\mathrm{dom}\, fe \lhd tgt^*_{SG_s}) \subseteq tgt^*_{SG_t} \circ fe \\ \land fv \circ \preceq_{SG_s} \subseteq \preceq_{SG_t} \circ fv \}$$

*Above, functions fv and fe are allowed to be partial; on the left-hand side of the commuting equations, the source and target functions are restricted on the domain to the edges mapped through fe to ensure that only the edges being mapped are considered in the corresponding commuting.*

*Relations $\Rrightarrow^{SG}$ and $\Rrightarrow^{SG}_{p}$ define total or partial conformance through a SG morphism, respectively:*

$$(SG_s, m) \Rrightarrow^{SG} SG_t \Leftrightarrow m \in SG_s \rightarrow_{SG} SG_t \quad (SG_s, m) \Rrightarrow^{SG}_{p} SG_t \Leftrightarrow m \in SG_s \nrightarrow_{SG} SG_t$$

□

Definition 23 (Partial Refinement of SGs). *Partial refinement ($\sqsupseteq^{SG}$) says whether a refined (or concrete) SG refines, through morphism m : GrM, a abstract $SG_a$ : SGr:*

$$(SG_c, m) \sqsupseteq^{SG} SG_a \Leftrightarrow m \in SG_c \rightarrow_{SG} SG_a \land (SG_c, m) \sqsupseteq^{SG_0} SG_a$$

*Partial refinement requires that: (i) m is a valid morphism from $SG_c$ to $SG_a$, and (ii) the satisfaction of the conditions associated with $\sqsupseteq^{SG_0}$.*

*The definition of $\sqsupseteq^{SG_0}$ is as follows:*

$$(SG_c, m) \sqsupseteq^{SG_0} SG_a \Leftrightarrow (SG_c, m) \sqsupseteq_{NT} SG_a \land (SG_c, m) \sqsupseteq_{ET} SG_a \land (SG_c, m) \sqsupseteq_{\mathcal{M}} SG_a \\ \land (SG_c, m) \sqsupseteq_{\mathcal{M}Cnts} SG_a$$

*Above, $\sqsupseteq^{SG_0}$ requires: (i) $SG_c$ complies with $SG_a$'s refinement node- ($\sqsupseteq_{NT}$) and edge-types ($\sqsupseteq_{ET}$) constraints, (ii) $SG_c$ complies with $SG_a$'s multiplicity refinement constraints ($\sqsupseteq_{\mathcal{M}}$), and (iii) $SG_c$ complies with the multiplicity constraints of $SG_a$'s derived edges ($\sqsupseteq_{\mathcal{M}Cnts}$).*

*$\sqsupseteq_{NT}$ requires that the nodes of a $SG_c$ comply to the node-type refinement constraints of the corresponding nodes in $SG_a$ through relation $\leq_{rNT}$ (def. 13):*

$$(SG_c, m) \sqsupseteq_{NT} SG_a \Leftrightarrow \forall\, n : Ns_{SG_c} \bullet nty_{SG_c}\, n \leq_{rNT} (nty_{SG_a} \circ (fV\ m))\, n$$

*$\sqsupseteq_{ET}$ requires that the edges of a $SG_c$ comply to the edge-type refinement constraints of the corresponding edges in $SG_a$ through relation $\leq_{ET}$ (def. 13):*

$$(SG_c, m) \sqsupseteq_{ET} SG_a \Leftrightarrow \forall\, e : EsA_{SG_c} \bullet ety_{SG_c}\, e \leq_{ET} (ety_{SG_a} \circ (fE\ m))\, e$$

*Finally, $\sqsupseteq_{\mathcal{M}}$ requires that $SG_c$'s association edges must comply to the multiplicity refinement constraints of the corresponding edges in $SG_a$ through relation $\leq_{\mathcal{M}}$ (def. 14) to say that refinements can weaken multiplicity constraints:*

$$(SG_c, m) \sqsupseteq_{\mathcal{M}} SG_a \Leftrightarrow \forall\, e : EsA_{SG_c} \bullet srcma_{SG_c}\ e \leq_{\mathcal{M}} (srcma_{SG_a} \circ (fE\ m))\ e \\ \wedge\ tgtm_{SG_c}\ e \leq_{\mathcal{M}} (tgtm_{SG_a} \circ (fE\ m))\ e$$

*Above, function srcma (def. 17) totalises srcm in order to handle uni-directional edges.*

*$\sqsupseteq_{\mathcal{M}Cnts}$ checks that each derived edge e is refined by considering three cases: (i) the multiplicity needs to be checked because e's path expression affects instance edges ie and ie′ (predicate caseMultsOk), (ii) e's path expression affects either source or target node and the multiplicity of the opposing node in the edge is mandatory or at least 1 (predicates caseMandatoryT and caseMandatoryS, respectively):*

$$(SG_c, m) \sqsupseteq_{\mathcal{M}Cnts} SG_a \Leftrightarrow \forall\, e : EsD_{SG_a} \bullet \\ \exists\, peas : PEA\ ;\ peae : PEA \mid peas = startEdg(pe\ SG_a\ e) \wedge peae = endEdg(pe\ SG_a\ e) \bullet \\ \forall\, ie : (fE\ m)^{\sim}\ (\!|\{ePEA\ peas\}|)\ ;ie' : (fE\ m)^{\sim}\ (\!|\{ePEA\ peae\}|) \bullet \\ caseMultsOk(SG_c, m, SG_a, e, peas, peae, ie, ie') \\ \wedge\ (caseMandatoryT(SG_c, m, SG_a, e, peae, ie) \vee caseMandatoryS(SG_c, m, SG_a, e, peas, ie'))$$

*Predicate caseMultsOk checks that the multiplicity constraint of given derived edge e of $SG_a$ is satisfied by start and end instance edges, ie and ie′, respectively, provided they are affected by e's path expression:*

$$caseMultsOk(SG_c, m, SG_a, e, peas, peae, ie, ie') \\ \Leftrightarrow affectedPE(SG_c, m, SG_a, pe\ SG_a\ e, e, ie, ie') \\ \Rightarrow smfPEA\ SG_c\ peas\ ie \leq_{\mathcal{M}} srcma\ SG_a\ e \wedge tmfPEA\ SG_c\ peae\ ie' \leq_{\mathcal{M}} tgtm\ SG_a\ e$$

*The functions above, namely, affectedPE (checks whether e's path expression from $SG_a$ affects instance edges ie and ie′ in $SG_c$) smfPEA and tmfPEA are from def. 24. The constraint says that if the instance edges are affected by the constraint of the derived edge, then they must comply to the constraint.*

*Predicates caseMandatoryT and caseMandatoryS require that either start or end node is affected and the opposing node matches the type node, if the multiplicity constraint on the opposing node is mandatory (at least 0):*

$$caseMandatoryT(SG_c, m, SG_a, e, peae, ie) \\ \Leftrightarrow (affectedPEStart(SG_c, m, SG_a, pe\ SG_a\ e, e, ie) \wedge mks\ 1 \leq_{\mathcal{M}} (tgtm\ SG_a)\ e) \\ \Rightarrow (fV\ m)(tPEA\ SG_c\ peae\ ie) = tgt\ SG_a\ e \\ caseMandatoryS(SG_c, m, SG_a, e, peas, ie) \\ \Leftrightarrow (affectedPEEnd(SG_c, m, SG_a, pe\ SG_a\ e, e, ie) \wedge mks\ 1 \leq_{\mathcal{M}} (srcma\ SG_a)\ e) \\ \Rightarrow (fV\ m)(sPEA\ SG_c\ peas\ ie) = src\ SG_a\ e$$

□

Definition 24 (Refinement of Multiplicity Constraints of Derived Edges in SGs). *In a refinement, concrete SGs are required to comply to the multiplicity constraints imposed by the derived edges of their abstract counter-parts; this involves checking that the constraints imposed upon the paths of the derived edges are satisfied. Several functions support this. The following functions obtain either the source or target function (sPEA and tPEA) or the source or target multiplicity*

*function (smfPEA and tmfPEA) relative to direction of the encapsulated edge (whether it is inverted or not):*

$$sPEA, tPEA : SGr \to PEA \to (E \nrightarrow V) \quad smfPEA, tmfPEA : SGr \to PEA \to (E \nrightarrow Mult)$$
$$sPEA\ SG\ e = src\ SG \quad sPEA\ SG\ e^{-1} = tgt\ SG \quad tPEA\ SG\ e = tgt\ SG \quad tPEA\ SG\ e^{-1} = src\ SG$$
$$smfPEA\ SG\ e = srcma\ SG \quad smfPEA\ SG\ e^{-1} = tgtm\ SG$$
$$tmfPEA\ SG\ e = tgtm\ SG \quad tmfPEA\ SG\ e^{-1} = srcma\ SG$$

*Predicates affectedPEAStart and affectedPEAEnd indicate when an edge (ie) of a concrete SG ($SG_c$) is affected by a particular path edge atom expression (pea of set PEA of def. 15) embedded in an edge (e) of the abstract SG ($SG_a$), at either ends of the path (start or end):*

$$affectedPEAStart(SG_c, m, SG_a, pea, e, ie) \Leftrightarrow ie \in (fE\ m)^{\sim}(\!|\{ePEA\ pea\}|\!)$$
$$\wedge\ (fV\ m)(sPEA\ SG_c\ pea\ ie) = sPEA\ SG_a\ pea\ e$$
$$affectedPEAEnd(SG_c, m, SG_a, pea, e, ie)$$
$$\Leftrightarrow ie \in (fE\ m)^{\sim}(\!|\{ePEA\ pea\}|\!) \wedge (fV\ m)(tPEA\ SG_c\ pea\ ie) = tPEA\ SG_a\ pea\ e$$

*Predicates affectedPECStart and affectedPECEnd indicate when an edge (ie) of a concrete SG ($SG_c$) is affected by a particular path edge composite expression (set PEC of def. 15) embedded in a derived edge (e) of the abstract SG ($SG_a$), at either ends of the path (start or end):*

$$affectedPECStart(SG_c, m, SG_a, eat\ pea, e, ie) \Leftrightarrow affectedPEAStart(SG_c, m, SG_a, pea, e, ie)$$
$$affectedPECStart(SG_c, m, SG_a, v \lhd_{PE} pea, e, ie) \Leftrightarrow affectedPEAStart(SG_c, m, SG_a, pea, e, ie)$$
$$\wedge\ \exists\, v' : (fV\ m)^{\sim}(\!|\{v\}|\!) \bullet (sPEA\ SG_c\ pea\ ie, v') \in \preceq\ SG_c$$
$$affectedPECStart(SG_c, m, SG_a, pea \rhd_{PE} v, e, ie) \Leftrightarrow affectedPEAStart(SG_c, m, SG_a, pea, e, ie)$$
$$\wedge\ \exists\, v' : (fV\ m)^{\sim}(\!|\{v\}|\!) \bullet (tPEA\ SG_c\ pea\ ie, v') \in \preceq\ SG_c$$
$$affectedPECEnd(SG_c, m, SG_a, eat\ pea, e, ie) \Leftrightarrow affectedPEAEnd(SG_c, m, SG_a, pea, e, ie)$$
$$affectedPECEnd(SG_c, m, SG_a, v \lhd_{PE} pea, e, ie)$$
$$\Leftrightarrow affectedPEAEnd(SG_c, m, SG_a, pea, e, ie) \wedge \exists\, v' : (fV\ m)^{\sim}(\!|\{v\}|\!) \bullet (sPEA\ SG_c\ pea\ ie, v') \in \preceq\ SG_c$$
$$affectedPECEnd(SG_c, m, SG_a, pea \rhd_{PE} v, e, ie)$$
$$\Leftrightarrow affectedPEAEnd(SG_c, m, SG_a, pea, e, ie) \wedge \exists\, v' : (fV\ m)^{\sim}(\!|\{v\}|\!) \bullet (tPEA\ SG_c\ pea\ ie, v') \in \preceq\ SG_c$$

*Predicates affectedPEStart and affectedPEEnd indicate when an edge (ie) of a concrete SG ($SG_c$) is affected by a particular path edge expression (set PE of def. 15) embedded in a derived edge (e) of the abstract SG ($SG_a$), at either ends of the path (start or end):*

$$affectedPEStart(SG_c, m, SG_a, pec, e, ie) \Leftrightarrow affectedPECStart(SG_c, m, SG_a, pec, e, ie)$$
$$affectedPEStart(SG_c, m, SG_a, pec \fatsemi_{PE} pe, e, ie) \Leftrightarrow affectedPECStart(SG_c, m, SG_a, pec, e, ie)$$
$$affectedPEEnd(SG_c, m, SG_a, pec, e, ie) \Leftrightarrow affectedPECEnd(SG_c, m, SG_a, pec, e, ie)$$
$$affectedPEEnd(SG_c, m, SG_a, pec \fatsemi_{PE} pe, e, ie) \Leftrightarrow affectedPEEnd(SG_c, m, SG_a, pe, e, ie)$$

*Finally, predicate affectedPE, handles complete path expressions made up of many composites; it indicate when an edge (ie) of a concrete SG ($SG_c$) is affected by a particular path edge expression (set PE of def. 15) embedded in a derived edge (e)*

*of the abstract SG ($SG_a$):*

$$
\begin{aligned}
&\mathit{affectedPE}(SG_c, m, SG_a, pec, e, ie, ie') \\
&\quad \Leftrightarrow ie = ie' \wedge \mathit{affectedPECStart}(SG_c, m, SG_a, pec, e, ie) \wedge \mathit{affectedPECEnd}(SG_c, m, SG_a, pec, e, ie) \\
&\mathit{affectedPE}(SG_c, m, SG_a, pec_1 \fatsemi_{PE} pec_2), e, ie, ie') \\
&\quad \Leftrightarrow \mathit{tPEA}\ SG_c\ (\mathit{endEdgC}\ pec_1)\ ie = \mathit{sPEA}\ SG_c\ (\mathit{startEdgC}\ pec_2)\ ie' \\
&\qquad \wedge \mathit{affectedPECStart}(SG_c, m, SG_a, pec_1, e, ie) \wedge \mathit{affectedPECEnd}(SG_c, m, SG_a, pec_2, e, ie') \\
&\mathit{affectedPE}(SG_c, m, SG_a, pec \fatsemi_{PE} pe, e, ie, ie') \\
&\quad \Leftrightarrow \mathit{affectedPECStart}(SG_c, m, SG_a, pec, e, ie) \\
&\qquad \wedge \exists\, ie'' : E \mid \mathit{tPEA}\ SG_c\ (\mathit{endEdgC}\ pec)\ ie = \mathit{sPEA}\ SG_c\ (\mathit{startEdg}\ pe)\ ie'' \bullet \\
&\qquad\quad \mathit{affectedPE}(SG_c, m, SG_a, pe, e, ie'', ie')
\end{aligned}
$$

*Three cases are considered above: (i) the path expression is made up of a single composite, in which case both instance edges, ie and ie′, must be the same, and the composite must be affected at both start and end; (ii) the path expression is made up of one composite followed by another, in which case both instance edges, ie and ie′ must be connected, the two edges must be affected by the two composites at start and end, respectively; (iii) the path expression is made up of one composite followed by another path expression, in which case the first composite must be affected at the start, the first instance edge ie must be connected to the start edge of the path expression pe by some edge ie″, and the remaining path expression pe must be affected.* □

Definition 25 (Extracting Instance Information). *Several functions extract instance information from typing morphisms. They extract: a set of nodes which are instances of given a set of meta- (or type) nodes (ins), a set of edges which are instances of a given set of meta-edges (ies), an instance graph restricted to the edges of a certain edge-type (igRMEs):*

$$
\begin{aligned}
&\mathit{ins}(m, SG, mns) = (fV\ m)^{\sim}(\!|(\prec SG)^{\sim}(\!|mns|\!)|\!) \qquad \mathit{ies}(m, mes) = (fE\ m)^{\sim}(\!|mes|\!) \\
&\mathit{igRMEs}(GwT, mes) = GwT^{G} \bowtie_{Es} \mathit{iesOf}\,(GwT^{T}, mes)
\end{aligned}
$$

*Above, ins uses SG's inheritance relation; information is derived from the inverse morphisms.* □

Definition 26 (Total Refinement of SGs). *Total refinement establishes a refinement relation between a concrete $SG_c : SGr$ and an abstract $SG_a : SGr$ through morphism $m : GrM$:*

$$(SG_c, m) \sqsupset^{SG}\ SG_a \Leftrightarrow m \in SG_c \rightarrow_{SG} SG_a \wedge (SG_c, m) \sqsupset^{SG_0}\ SG_a$$

*This requires that (in the left-to-right order of the conjuncts): (i) morphism m is valid, and (ii) the conditions of total refinement are met.*

*Relation $\sqsupset^{SG_0}$ is defined as follows:*

$$(SG_c, m) \sqsupset^{SG_0}\ SG_a \Leftrightarrow (SG_c, m) \sqsupseteq^{SG_0}\ SG_a \wedge (SG_c, m) \sqsupset_{AEs}\ SG_a \wedge m \sqsupset_{ANNs} SG_a$$

*This requires that (in the left-to-right order of the conjuncts): (i) partial refinement's conditions are met, and (ii) $SG_a$'s association edges (iii) and normal nodes are adequately refined.*

*Predicate $\sqsupset_{AEs}$ requires that each association edge of $SG_a$ is adequately refined in $SG_c$, which excludes inheritance edges (as they are not mapped to):*

$$(SG_c, m) \sqsupset_{AEs}\ SG_a \Leftrightarrow \forall\, e : EsA_{SG_a} \bullet (SG_a, e)\ \mathrm{OkRefinedIn}\ (SG_c, m)$$

*Relation* OkRefinedIn *checks that the instance relation restricted to a particular meta-edge is not empty, and maps source and target elements.:*

$$
\begin{aligned}
&(SG_a, me)\ \text{OkRefinedIn}\ (SG_c, m) \Leftrightarrow \\
&\quad \exists\, r : V \leftrightarrow V ; s, t : \mathbf{P}\, V \mid r = \preceq_{SG_c} \fatsemi igRMEs((gr\ SG_c, m), SG_a, \{me\})^{\leftrightarrow} \fatsemi (\preceq_{SG_c})^{\sim} \\
&\qquad \land\ s = insOf(m, SG_a, src_{SG_a}(\!|\{me\}|\!)) \setminus (NsEther_{SG_c} \setminus \text{dom}\ r) \\
&\qquad \land\ t = insOf(m, SG_a, tgt_{SG_a}(\!|\{me\}|\!)) \setminus (NsEther_{SG_c} \setminus \text{ran}\ r) \bullet r \in s \leftrightarrow t \land r \neq \varnothing
\end{aligned}
$$

*Predicate* $\sqsupset_{ANNs}$ *requires all normal nodes, or one of their ancestors, to be instantiated:*

$$m \sqsupset_{ANNs} SG_a \Leftrightarrow \forall\, nn : NsTy_{SG_a}\{nnrml\} \bullet (\preceq_{SG_a})(\!|\{nn\}|\!) \cap \text{ran}(fV\ m) \neq \varnothing$$

□

Definition 27 (Multiplicity Checking). *Predicate* $r\mathcal{M}Ok$ *check whether a relation* $r : V \leftrightarrow V$ *over vertices, with a source and target sets* $s, t : \mathbf{P}\, V$*, complies with source and target multiplicities* $m_s, m_t$ *: Mult (def. 14).* $r\mathcal{M}Ok$ *is defined on a case by case basis;* rbounded *checks whether a relation is within multiplicity bounds; below,* $mm \in MultMany$*,* $mr \in MultRange$ *and* $me \in MultEither$ *(def. 14):*

$$
\begin{aligned}
&\text{rbounded}(r, s, m) \Leftrightarrow \forall\, x : s \bullet \#(r(\!|\{x\}|\!))\ \emptyset\ m \\
&\text{eitherbounded}(r, s, ms) \Leftrightarrow \forall\, x : s \bullet \#(r(\!|\{x\}|\!)) \cdots ms \\
&r\mathcal{M}Ok(r, s, t, \{1\}, \{1\}) \Leftrightarrow r \in s \rightarrowtail\!\!\!\!\!\rightarrow t \\
&r\mathcal{M}Ok(r, s, t, \{(0,1)\}, \{1\}) \Leftrightarrow r \in s \rightarrowtail t \\
&r\mathcal{M}Ok(r, s, t, \{1\}, \{(0,1)\}) \Leftrightarrow r^{\sim} \in t \rightarrowtail s \\
&r\mathcal{M}Ok(r, s, t, mm, \{1\}) \Leftrightarrow r \in s \rightarrow t \\
&r\mathcal{M}Ok(r, s, t, \{1\}, mm) \Leftrightarrow r^{\sim} \in t \rightarrow s \\
&r\mathcal{M}Ok(r, s, t, mr, \{1\}) \Leftrightarrow r \in s \rightarrow t \land \text{rbounded}(r^{\sim}, t, \text{the}\ mr) \\
&r\mathcal{M}Ok(r, s, t, \{1\}, mr) \Leftrightarrow r^{\sim} \in t \rightarrow s \land \text{rbounded}(r, s, \text{the}\ mr) \\
&r\mathcal{M}Ok(r, s, t, me, \{1\}) \Leftrightarrow r \in s \rightarrow t \land \text{eitherbounded}(r^{\sim}, t, me) \\
&r\mathcal{M}Ok(r, s, t, \{1\}, me) \Leftrightarrow r^{\sim} \in t \rightarrow s \land \text{eitherbounded}(r, s, me) \\
&r\mathcal{M}Ok(r, s, t, \{(0,1)\}, \{(0,1)\}) \Leftrightarrow r \in s \rightarrowtail\!\!\!\!\!\!\rightarrow t \\
&r\mathcal{M}Ok(r, s, t, mm, \{(0,1)\}) \Leftrightarrow r \in s \twoheadrightarrow t \\
&r\mathcal{M}Ok(r, s, t, \{(0,1)\}, mm) \Leftrightarrow r^{\sim} \in t \twoheadrightarrow s \\
&r\mathcal{M}Ok(r, s, t, mr, \{(0,1)\}) \Leftrightarrow r \in s \twoheadrightarrow t \land \text{rbounded}(r^{\sim}, t, \text{the}\ mr) \\
&r\mathcal{M}Ok(r, s, t, \{(0,1)\}, mr) \Leftrightarrow r^{\sim} \in t \twoheadrightarrow s \land \text{rbounded}(r, s, \text{the}\ mr) \\
&r\mathcal{M}Ok(r, s, t, me, \{(0,1)\}) \Leftrightarrow r \in s \twoheadrightarrow t \land \text{eitherbounded}(r^{\sim}, t, me) \\
&r\mathcal{M}Ok(r, s, t, \{(0,1)\}, me) \Leftrightarrow r^{\sim} \in t \twoheadrightarrow s \land \text{eitherbounded}(r, s, me) \\
&r\mathcal{M}Ok(r, s, t, mm_1, mm_2) \Leftrightarrow r \in s \leftrightarrow t \\
&r\mathcal{M}Ok(r, s, t, mm, mr) \Leftrightarrow r \in s \leftrightarrow t \land \text{rbounded}(r, s, \text{the}\ mr) \\
&r\mathcal{M}Ok(r, s, t, mr, mm) \Leftrightarrow r \in s \leftrightarrow t \land \text{rbounded}(r^{\sim}, t, \text{the}\ mr) \\
&r\mathcal{M}Ok(r, s, t, mm, me) \Leftrightarrow r \in s \leftrightarrow t \land \text{eitherbounded}(r, s, me) \\
&r\mathcal{M}Ok(r, s, t, me, mm) \Leftrightarrow r \in s \leftrightarrow t \land \text{eitherbounded}(r^{\sim}, t, me) \\
&r\mathcal{M}Ok(r, s, t, mr_1, mr_2) \Leftrightarrow r \in s \leftrightarrow t \land \text{rbounded}(r, s, \text{the}\ mr_2) \land \text{rbounded}(r^{\sim}, t, \text{the}\ mr_1) \\
&r\mathcal{M}Ok(r, s, t, me_1, me_2) \Leftrightarrow r \in s \leftrightarrow t \land \text{eitherbounded}(r, s, me_2) \land \text{eitherbounded}(r^{\sim}, t, me_1)
\end{aligned}
$$

□

Definition 28 (Morphisms from Graphs to SGs). *Given $G : Gr$ and $SG : SGr$, a G-SG morphism $m : G \rightarrow_{G2SG} SG$ is a pair of functions $m = (fv, fe)$ mapping nodes and edges respectively. The set of such morphisms is defined as:*

$$G \rightarrow_{G2SG} SG = \{(fv, fe) \mid fv \in Ns_G \rightarrow Ns_{SG} \wedge fe \in Es_G \rightarrow EsA_{SG} \mid fv \circ src_G \subseteq src^*{}_{SG} \circ fe \wedge fv \circ tgt_G \subseteq tgt^*{}_{SG} \circ fe\}$$

*Above, the target edges comprise association edges only (set EsA, def. 17).*

*Relation $\Rrightarrow^{GwT}$ provides an alternative formulation for graphs with typing $GwT : GrwT$ (def. 11):*

$$GwT \Rrightarrow^{GwT} SG \Leftrightarrow GwT^T \in GwT^G \rightarrow_{G2SG} SG$$

□

Definition 29 (Auxiliary Functions of Path Expressions). *Function rPEA builds a relation from a path expression atom through the graph edge-restriction operator $\bowtie_{Es}$ (def.8), where $e \in E$:*

$$rPEA : GrwT \times SGr \times PEA \nrightarrow (V \leftrightarrow V)$$
$$rPEA\,(GwT, SG, e) = (GwT^G \bowtie_{Es} ies\,(GwT^T, \{e\}))^{\leftrightarrow}$$
$$PEA\,(GwT, SG, e^{-1}) = rPEA\,(GwT, SG, e)^{\sim}$$

*Above, functions ins and ies yield the instances of a given node or edge type, respectively (def. 25).*

*Function rPE builds a relation from a given path expression, where $pea \in PEA$:*

$$rPE : GrwT \times SGr \times PE \rightarrow (V \leftrightarrow V)$$
$$rPE\,(GwT, SG, pea) = rPEA\,(GwT, SG, pea)$$
$$rPE\,(GwT, SG, v \lhd_{PE} pea) = ins\,(GwT^T, SG, \{v\}) \lhd rPEA(GwT, SG, pea)$$
$$rPE\,(GwT, SG, pea \rhd_{PE} v) = rPEA\,(GwT, SG, pea) \rhd ins\,(GwT^T, SG, \{v\})$$
$$rPE\,(GwT, SG, pe_1 \fatsemi_{PE} pe_2) = rPE\,(GwT, SG, pe_1) \cup rPE\,(GwT, SG, pe_2)$$

*Function $ape : SGr \rightarrow E \nrightarrow PE$ totalises function pe (gives path expressions of constraint edges, def. 17), to all multiplicity edges:*

$$ape\,SG = \{(e, e) \mid e \in EsM\,SG\} \oplus pe\,SG$$

*The path expression of a multiplicity edge is the edge itself, or the edge's path expression (function pe).*

*Function $multComp : Mult \times Mult \rightarrow Mult$ gives the sequential composition of two multiplicities, where $mm_1, mm_2 : MultMany$, and $m_1, m_2 : Mult$:*

$$multComp(mm_1, mm_2) = \{*\} \qquad multComp(m_1, m_2) = \{0\} \Leftrightarrow m_1 = \{0\} \vee m_2 = \{0\}$$
$$multComp(m_1, \{1\}) = m_1 \qquad multComp(\{1\}, m_2) = m_2$$
$$multComp(\{(0,1)\}, m_2) = \{0\} \cup m_2 \qquad multComp(m_1, \{(0,1)\}) = \{0\} \cup m_1$$

*Functions* $src^*_{PEA} : SGr \times PEA \to E \leftrightarrow V$ *and* $src^*_{PE} : SGr \times PE \to E \leftrightarrow V$ *gives the extended source relation of a path expression:*

$$src^*_{PEA}(SG, e) = src^*{}_{SG} \qquad src^*_{PEA}(SG, e^{-1}) = tgt^*{}_{SG}$$
$$src^*_{PE}(SG, pea) = src^*_{PEA}(SG, pea) \qquad src^*_{PE}(SG, v \lhd_{PE} pea) = src^*_{PEA}(SG, pea) \rhd \{v\}$$
$$src^*_{PE}(SG, pea \rhd_{PE} v) = src^*_{PEA}(SG, pea) \qquad src^*_{PE}(SG, pe_1 \fatsemi_{PE} pe_2) = src^*_{PE}(SG, pe_1)$$

*Functions* $tgt^*_{PEA} : SGr \times PEA \to E \leftrightarrow V$ *and* $tgt^*_{PE} : SGr \times PE \to E \leftrightarrow V$ *gives the extended target relation of a path expression:*

$$tgt^*_{PEA}(SG, e) = tgt^*{}_{SG} \qquad tgt^*_{PEA}(SG, e^{-1}) = src^*{}_{SG}$$
$$tgt^*_{PE}(SG, pea) = tgt^*_{PEA}(SG, pea) \qquad tgt^*_{PE}(SG, v \lhd_{PE} pea) = tgt^*_{PEA}(SG, pea)$$
$$tgt^*_{PE}(SG, pea \rhd_{PE} v)) = tgt^*_{PEA}(SG, pea) \rhd \{v\} \qquad tgt^*_{PE}(SG, pe_1 \fatsemi_{PE} pe_2) = tgt^*_{PE}(SG, pe_2)$$

*Functions* $smPEA : SGr \to PEA \to Mult$ *and* $smPE : SGr \to PE \to Mult$ *give the source multiplicity of path expressions:*

$$smPEA\ SG\ e = srcma\ SG\ e \qquad smPEA\ SG\ e^{-1} = tgtm\ SG\ e$$
$$smPE\ SG\ pea = smPEA\ SG\ pea \qquad smPE\ SG\ (v \lhd_{PE} pea) = smPEA\ SG\ pea$$
$$smPE\ SG\ (pea \rhd_{PE} v) = smPEA\ SG\ pea \qquad smPE\ SG\ (pe_1 \fatsemi_{PE} pe_2) = multComp(smPE\ SG\ pe_1, smPE\ SG\ pe_2)$$

*Functions* $tmPEA : SGr \to PEA \to Mult$ *and* $tmPE : SGr \to PE \to Mult$ *give the target multiplicity of path expressions:*

$$tmPEA\ SG\ e = tgtm\ SG\ e \qquad tmPEA\ SG\ e^{-1} = srcma\ SG\ e$$
$$tmPE\ SG\ pea = tmPEA\ SG\ pea \qquad tmPE\ SG\ (v \lhd_{PE} pea) = tmPEA\ SG\ pea$$
$$tmPE\ SG\ (pea \rhd_{PE} v) = tmPEA\ SG\ pea \qquad tmPE\ SG\ (pe_1 \fatsemi_{PE} pe_2) = multComp(tmPE\ SG\ pe_1, tmPE\ SG\ pe_2)$$

□

Definition 30 (Graphs Typed by Their SG types). *Predicate* $\ni^{SG}$ *captures the necessary compliance between an instance graph* $GwT : GrwT$ *(def. 11) and its* $SG : SGr$ *type:*

$$GwT \ni^{SG} SG \Leftrightarrow GwT \Rrightarrow^{GwT} SG \wedge GwT \ni_{\mathcal{M}} SG \wedge GwT \ni_{FI} SG \wedge GwT \ni_{PNS} SG \wedge GwT \ni_{Cnts} SG$$

*This requires that the GwT instance: (i) is related through a morphism to the SG type (*$\Rrightarrow^{GwT}$*, def. 28), (ii) complies with multiplicity constraints of its type SG (*$\ni_{\mathcal{M}}$ *defined below), (iii) excludes forbidden node instances of its SG type (*$\ni_{FI}$ *below), (iv) has all its compounds not sharing parts (*$\ni_{PNS}$ *below), and (v) satisfies certain constraints imposed by its SG (*$\ni_{Cnts}$ *below).*

*A* $GwT : GrwT$ *instance is multiplicity-compliant with its SG type provided all instances of multiplicity edges (association plus derived) comply to the multiplicities of their edge types as per predicate ME*$\mathcal{M}$*Ok (defined below):*

$$GwT \ni_{\mathcal{M}} SG \Leftrightarrow \forall\ me : EsM_{SG} \bullet (SG, me)\ ME\mathcal{M}Ok\ GwT$$

*Several auxiliary functions are involved in the definition of ME*$\mathcal{M}$*Ok. Function* $rME\mathcal{M}Ok : SGr \times E \times GrwT \to V \leftrightarrow V$ *builds the relation between nodes that is to be subject to a multiplicity check, corresponding the given SG, instance graph GwT and meta-edge me, which is derived from the meta-edge's path expression through function rPE (def. 29).*

$$rME\mathcal{M}Ok(SG, me, GwT) =$$
$$\quad \textbf{let}\ s = ins(GwT^T, SG, src^*{}_{SG}\ (\!|\{me\}|\!)) \bullet \textbf{let}\ t = ins(GwT^T, SG, tgt^*{}_{SG}\ (\!|\{me\}|\!)) \bullet$$
$$\quad s \lhd rPE\ (GwT, SG, ape\ SG\ me) \rhd t$$

*Function* $src^*ME\mathcal{M}Ok : SGr \times E \rightarrow E \leftrightarrow V$ *gives extended source relation specific to a particular meta-edge, derived from function* $src^*_{PE}$ *(gives extended source relation of a path expression, def. 29):*

$$src^*ME\mathcal{M}Ok(SG, me) = src^*_{PE}(SG, ape\ SG\ me) \rhd (src^*{}_{SG}) \llparenthesis \{me\} \rrparenthesis$$

*Function* $tgt^*ME\mathcal{M}Ok : SGr \times E \rightarrow E \leftrightarrow V$ *gives extended target relation specific to a particular meta-edge, derived from function* $tgt^*_{PE}$ *(gives extended target relation of a path expression, def. 29):*

$$tgt^*ME\mathcal{M}Ok(SG, me) = tgt^*_{PE}(SG, ape\ SG\ me) \rhd (tgt^*{}_{SG}) \llparenthesis \{me\} \rrparenthesis$$

*Predicate* $ME\mathcal{M}Ok$ *checks whether edge instances comply to the corresponding multiplicity constraints using the relation obtained from rMEMOk:*

$$\begin{aligned}
&(SG, me, pe)\ ME\mathcal{M}Ok\ GwT \Leftrightarrow \exists\, r : V \leftrightarrow V\ ;\ s, t : \mathbf{P}\ V \mid r = rMEMOk\ (SG, me, GwT) \\
&\quad \land\ s = ins(GwT^T, SG, src^*ME\mathcal{M}Ok\ (SG, me) \llparenthesis \{rsrcPE(ape\ SG\ me)\} \rrparenthesis) \\
&\quad \land\ t = ins(GwT^T, SG, tgt^*ME\mathcal{M}Ok\ (SG, me) \llparenthesis \{rtgtPE(ape\ SG\ me)\} \rrparenthesis) \\
&\quad \bullet\ r\mathcal{M}Ok(r, s, t, smPE\ SG\ (ape\ SG\ me), tmPE\ SG\ (ape\ SG\ me))
\end{aligned}$$

*Above, predicate* $r\mathcal{M}Ok$ *(def. 27) checks if the relation derived from the given path expression (through function rMEMOk, def. 29) with appropriate source and target nodes (derived through function ins, def. 25) complies with both source and target SG multiplicities of the meta-edge or its path expression.*

*Predicate* $\ni_{FI}$ *checks that the instance graph excludes direct instances of SG ethereal nodes (includes types abstract, virtual and enum), which involves taking the inversion of the type morphism:*

$$GwT \ni_{FI} SG \Leftrightarrow ((fV \circ ty)GwT)^{\sim} \llparenthesis NsEther\ SG \rrparenthesis = \varnothing$$

*Predicate* $\ni_{PNS}$ *checks that instances of the same compound do not share parts:*

$$GwT \ni_{PNS} SG \Leftrightarrow \forall\, ec : \{ecomp\ dbi, ecomp\ duni\} \bullet igRMEs(GwT, \{\ ec\ \})^{\leftrightarrow\!\!\rightarrow} \in \mathrm{injrel}$$

*Relation obtained from instance graph restricted to composition edges (function igRMEs) must be injective to disallow sharing of part instances.*

*Finally,* $\ni_{Cnts}$ *checks that the instance complies with certain constraints imposed by the type, namely that the instance satisfies: the type's numeric constraints and the type's value edge constraints:*

$$GwT \ni_{Cnts} SG \Leftrightarrow \mathrm{INumbersOk}\ GwT \land \mathrm{IVCEsOk}(SG, GwT)$$

*Predicate* INumbersOk *checks that the constraints concerning natural numbers, integers and reals are satisfied, which means that the numbers extracted from the nodes marked as instances of some numeric set must indeed be members of the corresponding numeric set:*

$$\begin{aligned}
&\mathrm{INumbersOk}\ GwT \Leftrightarrow \mathrm{I}\mathbb{N}\mathrm{sOk}(SG, GwT) \land \mathrm{I}\mathbb{Z}\mathrm{sOk}(SG, GwT) \land \mathrm{I}\mathbb{R}\mathrm{sOk}(SG, GwT) \\
&\mathrm{I}\mathbb{N}\mathrm{sOk}\ GwT \Leftrightarrow \forall\, nnat : (fV\ (GwT^T))^{\sim} \llparenthesis \{nNatS\} \rrparenthesis \bullet \exists\, n : \mathbb{N} \bullet n \in \mathrm{to}\mathbb{Z}\ nnat \\
&\mathrm{I}\mathbb{Z}\mathrm{sOk}\ GwT \Leftrightarrow \forall\, nint : (fV\ (GwT^T))^{\sim} \llparenthesis \{nIntS\} \rrparenthesis \bullet \exists\, n : \mathbb{Z} \bullet n \in \mathrm{to}\mathbb{Z}\ nint \\
&\mathrm{I}\mathbb{R}\mathrm{sOk}\ GwT \Leftrightarrow \forall\, nreal : (fV\ (GwT^T))^{\sim} \llparenthesis \{nRealS\} \rrparenthesis \bullet \exists\, x : \mathbb{R} \bullet x \in \mathrm{to}\mathbb{R}\ nreal
\end{aligned}$$

*Predicate* IVCEsOk *checks that each VCE satisfies the required constraints, depending on the type of VCE, either edge or node:*

$$\begin{array}{l}\text{IVCEsOk}(SG, GwT) \Leftrightarrow \forall\, vce : EsVCnt_{SG} \bullet \\ \quad \text{isVCEECnt}(SG, vce) \Rightarrow \text{satisfiesVCEECnt}(SG, GwT, vce) \\ \quad \land\ \text{isVCENCnt}(SG, vce) \Rightarrow \text{satisfiesVCENCnt}(SG, GwT, vce)\end{array}$$

*Predicate* satisfiesVCEECnt *checks that the corresponding arithmetic constraint is satisfied for those appropriate edges which are instances of the VCE's embedded edge*

$$\begin{array}{l}\text{satisfiesVCEECnt}(SG, GwT, vce) \Leftrightarrow \\ \quad \forall\, ie : fE\,(GwT^T) \sim (\!|(second \circ vcei_{SG})\ vce|\!) \mid fV\ GwT^T(src\ GwT^G\ ie) = src_{SG}\ vce \bullet \\ \quad\quad \text{satisfiesCnt}(SG, GwT^T, tgt\ GwT^G\ ie, (first \circ vcei_{SG})\ vce, tgt_{SG}\ vce)\end{array}$$

*Predicate* satisfiesVCENCnt *checks that the corresponding constraint is satisfied for each instance of the VCE's source node:*

$$\begin{array}{l}\text{satisfiesVCENCnt}(SG, GwT, vce) \Leftrightarrow \forall\, in : fV\,(GwT^T) \sim (\!|\{src_{SG}\ vce\}|\!) \bullet \\ \quad\quad \text{satisfiesCnt}(SG, GwT^T, in, (first \circ vcei_{SG})\ vce, tgt_{SG}\ vce)\end{array}$$

*A constraint is satisfied for a given* $SG : SGr$*, morphism* $t : GrM$*, instance node source* $ns : V$*, operator* $op : SGVCEOP$ *(def. 16) and type node target* $nt : V$*, provided the two numbers extracted from the nodes satisfy the required constraint derived from the operator op, or the op is an equality and the type mapping of ns matches nt, and ns and nt are values:*

$$\begin{array}{l}\text{satisfiesCnt}(SG, t, ns, op, nt) \Leftrightarrow ns \in \text{dom}(fV\ t) \\ \quad \land\ (\exists\, n_1, n_2 : \mathbb{A} \mid n_1 \in toNum(SG, ns, (fV\ t)ns) \land n_2 \in toNum(SG, nt, nt) \bullet (n_1, n_2) \in rOp\ op \\ \quad\quad \lor\ op = eq \land (fV\ t)ns = nt \land nt \in NsVa)\end{array}$$

*Function toNum extracts integer or real number of given node ns, depending on the numeric type of its type node nt in given SG:*

$$toNum(SG, ns, nt) = \begin{cases} \text{to}\mathbb{Z}\ ns & \text{if } nt \preceq_{SG} nNatS \lor nt \preceq_{SG} nIntS \\ \text{to}\mathbb{R}\ n & \text{if } nt \preceq_{SG} nRealS \\ \varnothing & \text{otherwise} \end{cases}$$

*Finally, function rOP obtains the relation that is required to exist between two numbers for the given operator op (of set SGVCEOP, def 16):*

$$\begin{array}{ll} rOp\ eq = \{n_1, n_2 : \mathbb{A} \mid n_1 = n_2\} & rOp\ neq = \{n_1, n_2 : \mathbb{A} \mid n_1 \neq n_2\} \\ rOp\ leq = \{n_1, n_2 : \mathbb{A} \mid n_1 \leq n_2\} & rOp\ geq = \{n_1, n_2 : \mathbb{A} \mid n_1 \geq n_2\} \\ rOp\ lt = \{n_1, n_2 : \mathbb{A} \mid n_1 < n_2\} & rOp\ gt = \{n_1, n_2 : \mathbb{A} \mid n_1 > n_2\} \end{array}$$

□

Definition 31 (Extra Typing for SGs). *A* $SG_s : SGr$ *is compatible with respect to extra typing with* $SG_t : SGr$ *via morphism m, written* $(SG_s, m) \dashv\vdash^{SG} SG_t$*, whenever the following holds:*

$$(SG_s, m) \dashv\vdash^{SG} SG_t \Leftrightarrow m \in SG_s \twoheadrightarrow_{SG} SG_t \land (SG_s, m) \dashv\vdash_{\mathcal{M}} SG_t \land m \dashv\vdash_{NT} SG_t \land (SG_s, m) \dashv\vdash_{ET} SG_t$$

*This requires that: (i) m is partial morphism from $SG_s$ to $SG_t$ (def. 23), (ii) $SG_s$ and morphism m satisfy the compatibility multiplicity constraints of $SG_t$, (iii) $SG_s$ preserves the types of the mapped nodes via m with respect to $SG_t$, and (iv) $SG_s$ preserves the types of the mapped edges via m with respect to $SG_t$.*

*Compatibility with respect to multiplicities $\dashv\vdash_{\mathcal{M}}$ is defined as:*

$$(SG_s, m) \dashv\vdash_{\mathcal{M}} SG_t \Leftrightarrow \forall (e, e') : fE\ m \bullet (srcma\ SG_s)\ e \leq_{\mathcal{M}} (srcma\ SG_t)\ e' \\ \wedge\ (tgtm\ SG_s)\ e \leq_{\mathcal{M}} (tgtm\ SG_t)\ e'$$

*This requires that the source and target multiplicities of each mapped edge via m from $SG_s$ is less or equal ($\leq_{\mathcal{M}}$, def. 14) than the target multiplicities of $SG_t$ of corresponding target edge.*

*Compatibility with respect to node types $\dashv\vdash_{NT}$ is defined as:*

$$m \dashv\vdash_{NT} SG_t \Leftrightarrow (nty\ SG_t)\ (\!|\ \mathrm{ran}(fV\ m)\ |\!) \subseteq \{nnrml, nabst, nvirt\}$$

*This requires that the mapped $SG_t$ nodes are of types normal, abstract and virtual only.*

*Finally, compatibility with respect to edge types $\dashv\vdash_{ET}$ is defined as:*

$$(SG_s, m) \dashv\vdash_{ET} SG_t \Leftrightarrow \mathrm{dom}(fE\ m) \lhd (ety\ SG_s) = (ety\ SG_t) \circ (fE\ m)$$

*This requires that the types of the mapped edges in $SG_s$ are the same as their $SG_t$ counter-parts; the domain restriction operator ($\lhd$) ensures that only the mapped edges are considered.* □

Definition 32 (Base Fragments). *A fragment $F = (SG, esr, sr, tr, it)$ comprises a $SG : SGr$ (def. 17), a set of references edges $esr \subseteq E$, functions $sr : esr \rightarrowtail\!\!\!\!\!\rightarrow NsP_{SG}$, a bijection from references edges to proxies, and $tr : esr \rightarrow V$, from reference edges to nodes, within fragment or elsewhere, and partial morphism $et : GrM$ (def. 9), indicating the extra type of certain nodes and edges (those with an instance-of constraint). The base set of fragments is defined as:*

$$Fr_0 = \{(SG, esr, sr, tr, et) \mid SG \in SGr_0 \wedge esr \in \mathbf{P}\ E \wedge esr \cap Es\ SG = \varnothing \wedge sr \in esr \rightarrowtail\!\!\!\!\!\rightarrow NsP_{SG} \\ \wedge\ tr \in esr \rightarrow V \wedge et \in GrM \wedge domg\ et \subseteq_p (NsN\ SG, EsA\ SG)\}$$

*Above, et partial morphism maps normal nodes and association edges only (sets NsN and EsA).*

*The following projection functions extract the components of a fragment $F : Fr_0$:*

$$fSG : Fr_0 \rightarrow SGr \quad EsR : Fr_0 \rightarrow \mathbf{P}\ E \quad srcR, tgtR : Fr_0 \rightarrow E \rightarrow V \quad fet : Fr_0 \rightarrow GrM$$
$$fSG(SG, esr, sr, tr, et) = SG \quad EsR(SG, esr, sr, tr, et) = esr \quad srcR(SG, esr, sr, tr, et) = sr$$
$$tgtR(SG, esr, sr, tr, et) = tr \quad fet(SG, esr, sr, tr, et) = et$$

*The empty fragment ($\varnothing_F$) is defined as: $\varnothing_F = (\varnothing_{SG}, \varnothing, \varnothing, \varnothing, \varnothing_{GM})$.*

*Several functions extract information from fragments: (i) fLEs, fEs and FLEsC give local, all (includes both local and reference edges), and connection edges of a fragment, respectively, (ii) fLNs, fRNs, fNs give local, reference and all nodes of a fragment, respectively, (iii) srcF and tgtF are the fragment's source and target functions respectively, which cater for all edges including reference edges:*

$$fLEs, fEs, fEsC : Fr_0 \rightarrow \mathbf{P}\ E \qquad fLNs, fRNs, fNs : Fr_0 \rightarrow \mathbf{P}\ V \qquad srcF, tgtF : Fr_0 \rightarrow E \nrightarrow V$$
$$fLEs = Es \circ fSG \qquad fEs\ F = fLEs\ F \cup EsR\ F \qquad fEsC = EsC \circ fSG$$
$$fLNs = Ns \circ fSG \qquad fRNs = \mathrm{ran} \circ tgtR \qquad fNs\ F = fLNs\ F \cup fRNs\ F$$
$$srcF\ F = (src \circ fSG)\ F \cup srcR\ F \qquad tgtF\ F = (tgt \circ fSG)\ F \cup tgtR\ F$$

*The following functions deal with references: (i) $\overset{G}{\rightsquigarrow}$ builds the references graph, (ii) $\rightsquigarrow$ builds the references partial function from the references graph:*

$$\overset{G}{\rightsquigarrow} : Fr_0 \to Gr \qquad \rightsquigarrow : Fr_0 \to V \leftrightarrow V$$
$$\overset{G}{\rightsquigarrow} F = ((NsP \circ fSG)F \cup fRNs\, F, EsR\, F, srcR\, F, tgtR\, F) \qquad \rightsquigarrow F = (\overset{G}{\rightsquigarrow} F)^{\leftrightarrow}$$

□

Definition 33 (Union of Fragments). *The union of fragments $F_1, F_2 : Fr$, written $F_1 \cup_F F_2$, is the union of the fragment's components:*

$$F_1 \cup_F F_2 = (fSG\, F_1 \cup_{SG} fSG\, F_2, EsR\, F_1 \cup EsR\, F_2, srcR\, F_1 \cup srcR\, F_2, tgtR\, F_1 \cup tgtR\, F_2)$$

*The generalised union $\bigcup_F$ yields the union of a set of fragments; it is defined recursively as:*

$$\bigcup_F : \mathbf{P}\, Fr_0 \to Fr_0 \quad \bigcup_F \{\} = \varnothing_F \quad \bigcup_F(\{F\} \cup Fs) = F \cup_F (\bigcup_F Fs)$$

□

Definition 34 (Fragment Resolution). *Given a fragment $F : Fr_0$, $\textcircled{\bullet}$ yields the resolved fragment in which proxies are substituted by the references through graph subsumption. Several functions are involved: (i) $\rightsquigarrow$ gives subsumptions' substitution by restricting partial function $\rightsquigarrow$ to resolvable proxies with inner references, (ii) $\textcircled{\bullet}^{SG}$ yields resolved SG, (iii) $\textcircled{\bullet}$ gives new reference edges of resolved fragment and actual resolved fragment (iv) the actual resolution is achieved through subsumption ($\odot^{SG}$, def. 21):*

$$\rightsquigarrow : Fr_0 \to V \nrightarrow V \quad \rightsquigarrow F = (\rightsquigarrow F) \rhd (fLNs\, F)$$
$$\textcircled{\bullet}^{SG} : Fr_0 \nrightarrow SGr \qquad rEsR : Fr_0 \nrightarrow \mathbf{P}\, E, \textcircled{\bullet} : Fr_0 \nrightarrow Fr_0$$
$$\textcircled{\bullet}^{SG} F = (fSG\, F) \odot^{SG} (\rightsquigarrow F) \quad rEsR\, F = \mathrm{dom}((srcR\, F) \rhd \mathrm{dom}(\rightsquigarrow F))$$
$$\textcircled{\bullet}\, F = (\textcircled{\bullet}^{SG} F, rEsR\, F, (rEsR\, F) \lhd (srcR\, F), (rEsR\, F) \lhd (tgtR\, F))$$

□

Definition 35 (Partial Fragments). *Set of acyclic fragments $Fr_a$ includes all base fragments (def. 32) with acyclic reference graphs. Partial fragments, set Fr, includes all acyclic fragments with partially resolved SGs:*

$$Fr_a = \{F \mid F \in Fr_0 \wedge \circledcirc(\overset{G}{\rightsquigarrow} F)\} \quad Fr = \{F \mid F \in Fr_a \wedge \textcircled{\bullet}^{SG} F \in SGr\}$$

*The following predicates, used in the laws that follow, say whether: (i) two fragments are mutually disjoint ($\boxminus_F$), (ii) a set of fragments is mutually disjoint ($\boxplus_F$), (iii) the flow of references goes from one fragment to another ($\subseteq^{rs}$), and (iv) the flow of references between two fragments is one-way only ($\models\!\!\Rightarrow$), meaning that one fragment references another but not the other way round:*

$$\boxminus_F(F_1, F_2) \Leftrightarrow fLNs\, F_1 \cap fLNs\, F_2 = \varnothing \wedge fEs\, F_1 \cap fEs\, F_2 = \varnothing$$
$$\boxplus_F\, Fs \Leftrightarrow \forall\, i, j : \mathrm{dom}\, Fs \mid i \neq j \bullet \boxminus_F(Fs\, i, Fs\, j)$$
$$F_1 \subseteq^{rs} F_2 \Leftrightarrow \mathrm{ran}(tgtR\, F_1) \cap fNs\, F_2 \neq \varnothing \quad F_1 \models\!\!\Rightarrow F_2 \Leftrightarrow F_1 \subseteq^{rs} F_2 \wedge \neg(F_2 \subseteq^{rs} F_1)$$

**Laws.**

$$F_1 \in Fr_0 \,;\, F_2 \in Fr_0 \,;\, \boxminus_F(F_1, F_2) \vdash \boxminus_{SGs}(fSG\, F_1, fSG\, F_2) \quad F \in Fr_a \vdash \overset{G}{\leftrightsquigarrow} F \in Gr$$
$$F_1 \in Fr \,;\, F_2 \in Fr \,;\, \boxminus_F(F_1, F_2) \,;\, (F_1 \rightrightarrows F_2 \vee F_2 \rightrightarrows F_1) \vdash F_1 \cup_F F_2 \in Fr$$
$$F_1 \cup_F F_2 \in Fr \,;\, \boxminus_F(F_1, F_2) \,;\, (F_1 \rightrightarrows F_2 \vee F_2 \rightrightarrows F_1) \vdash F_1 \in Fr \wedge F_2 \in Fr$$
$$F \in Fr \vdash \textcircled{\bullet}^{SG} F = (fSG \circ \textcircled{\bullet})\, F \quad \vdash \leftrightsquigarrow F \in ((NsP \circ fSG)\, F) \nrightarrow V \quad F \in Fr \vdash \textcircled{\bullet}\, F \in Fr$$

□

Definition 36 (Total Fragments). *Set TFr includes acyclic fragments with local references only (predicate* refsLocal*) and whose underlying resolved SG is total (set TSGr, def. 20):*

$$\text{refsLocal}\, F \Leftrightarrow fRNs\, F \subseteq fLNs\, F \quad TFr = \{F \mid F \in Fr_a \wedge \text{refsLocal}\, F \wedge \textcircled{\bullet}^{SG} F \in TSGr\}$$

□

Definition 37 (Fragment Morphisms). *A fragment morphism $m : F_s \rightarrow_F F_t$ is a pair of functions $m = (fv, fe)$ mapping nodes and edges respectively. The set of fragment morphisms is defined as:*

$$F_s \rightarrow_F F_t = \{(fv, fe) \mid fv \in fLNs_{F_s} \rightarrow fLNs_{F_t} \wedge fe \in fEsC_{F_s} \rightarrow fEsC_{F_t}$$
$$\wedge\, (\textcircled{\bullet}^{SG} F_s, (fv, fe) \rightrightarrows_\bullet (F_s, F_t)) \Rrightarrow^{SG} \textcircled{\bullet}^{SG} F_t\}$$

*Fragment morphisms, which map the local nodes and connection edges of fragments (functions fLNs and fEsC, def. 35) to exclude reference and inheritance edges, rely on fragment resolution; the morphism's nodes mapping is resolved through function $\rightrightarrows_\bullet$; the mappings are then required to be in the set of SG morphisms of the the two resolved fragments' SGs (obtained through function $\textcircled{\bullet}^{SG}$).*

*Morphism resolution is defined as:*

$$m \rightrightarrows_\bullet (F_s, F_t) = ((\rightsquigarrow^{\oplus}_{F_s} \boxdot fLNs_{F_s})^{\sim} \fatsemi (fV\, m) \fatsemi (\rightsquigarrow^{\oplus}_{F_t} \boxdot fLNs_{F_t}), fE\, m)$$

*Above, the resolution function $\rightsquigarrow$ (def. 34) is stretched by using the relation override closure $^{\oplus}$ (def. 4).*

*The relation between two fragments $F_s, F_t : Fr$ through morphism $m : GrM$ is defined as:*

$$(F_s, m) \Rrightarrow^F F_t \Leftrightarrow m \in F_s \rightarrow_F F_t$$

□

Definition 38 (Partial Fragment Refinement). *A concrete fragment $F_c : Fr$ partially refines an abstract fragment $F_a : Fr$ via a morphism $m : GrM$, from concrete to abstract, written $(F_c, m) \sqsupseteq^F F_a$, whenever the following holds:*

$$(F_c, m) \sqsupseteq^F F_a \Leftrightarrow (F_c, m) \Rrightarrow^F F_a \wedge (\textcircled{\bullet}^{SG} F_c, m \rightrightarrows_\bullet (F_c, F_a)) \sqsupseteq^{SG_0} \textcircled{\bullet}^{SG} F_a$$

*The above says: (i) $F_c$ is related to $F_a$ through morphism m (def. 37), (ii) SGs of resolved fragments satisfy extra conditions of partial refinement between SGs (def. 23).* □

Definition 39 (Total Fragment Refinement). *A concrete fragment $F_c : Fr$ totally refines an abstract fragment $F_a : Fr$ via morphism $m : GrM$, from concrete to abstract, written $(F_c, m) \sqsupset^F F_a$, whenever the following holds:*

$$(F_c, m) \sqsupset^F F_a \Leftrightarrow (F_c, m) \Rrightarrow^F F_a \wedge \{F_c, F_a\} \subseteq TFr \wedge (\textcircled{\bullet}^{SG} F_c, m \rightrightarrows_\bullet (F_c, F_a)) \sqsupset^{SG_0} \textcircled{\bullet}^{SG} F_a$$

*The above says: (i) fragment $F_c$ is related to $F_a$ through morphism $m$ (def. 37), (ii) both fragments $F_c$ and $F_a$ are total, (iii) SGs of resolved fragments satisfy extra conditions of total refinement (def. 26).* □

Definition 40 (Fragments as Types). *The compliance of a graph with a typing morphism $GwT : GrwT$ (def. 11) to its fragment type $F : Fr$, written, $GwT \ni^F F$, is defined as:*

$$GwT \ni^F F \Leftrightarrow GwT \ni^{SG} (\bullet)^{SG} F$$

*This reduces typing by a fragment to typing by the resolved fragment's SG.* □

Definition 41 (Extra typing for Fragments). *Extra-type compatibility of a source fragment $F_s : Fr$ with respect to a target fragment $F_t : Fr$, written $F_s \dashv\vdash^F F_t$, is defined as:*

$$F_s \dashv\vdash^F F_t \Leftrightarrow fet\, F_s \neq \varnothing_{GM} \wedge ((\bullet)^{SG} F_s, fet\, F_s) \dashv\vdash^{SG} ((\bullet)^{SG} F_t)$$

*The compatibility conditions are as follows: (i) $F_s$'s extra typing morphism must not be empty and (ii) the SGs of the resolved fragments must be extra-type compatible ($\dashv\vdash^{SG}$).*

*Extra typing compliance of a $Gwet : GrwET$ (def. 12), typed by a fragment $F_s : Fr$, to a $Gwt : GrwT$ (def. 11), typed by a fragment $F_t : Fr$, written $(Gwet, F_s) \Vdash^F (Gwt, F_t)$, is defined as:*

$$(Gwet, F_s) \Vdash^F (Gwt, F_t) \Leftrightarrow Gwet^{Gw} \ni^F F_s \wedge GwT \ni^F F_t$$
$$\wedge\ domg\,(Gwet^{ET}) =_p domg\,((fet\, F_s) \circ_G (Gwet^T)) \wedge Gwet^{ET} \in (Gwet^G) \nrightarrow_G (Gwt^G) \wedge F_s \dashv\vdash^F F_t$$

*This states that: (i) Gwet's GwT is typed by $F_s$ — $Gwet^{Gw} \ni^F F_s$ —, (ii) Gwt is typed by $F_t$ — $GwT \ni^F F_t$ —, (iii) the domain of Gwet's extra typing morphism is the domain of the morphism obtained by composing $F_s$'s extra-type morphism with Gwet's counterpart — $domg\,(Gwet^{ET}) =_p domg\,((fet\, F_s) \circ_G (Gwet^T))$ —, (iv) Gwet's extra typing morphism is a partial morphism from Gwet to Gwt— $Gwet^{ET} \in Gwet^G \nrightarrow_G Gwt^G$ —, and (v) $F_s$ is extra-type compatible with $F_t$ — $F_s \dashv\vdash^F F_t$.* □

Definition 42 (Global Fragment Graphs). *A $GFG : GFGr$ is a graph — $GFGr \subseteq Gr$ (def. 7). Set of well-formed GFGs includes all those graphs which are acyclic after the self edges are removed:*

$$GFGr = \{G : Gr \mid \circlearrowright(G \bowtie_{Es} (Es\, G \setminus EsId\, G))\}$$

*Relation $\dashrightarrow$ gives fragments referenced by a particular fragment through the transitive closure:*

$$\_^{\dashrightarrow} : GFGr \rightarrow V \leftrightarrow V \quad GFG^{\dashrightarrow} = (GFG^{\leftrightarrow})^+$$

□

Definition 43 (Models). *A model is a pair $M = (GFG, fd)$ comprising a $GFG : GFGr$ (def. 42) and a total function $fd : Ns\, GFG \rightarrow Fr$ which maps GFG nodes representing fragments to their fragment definitions (set Fr, def. 35). The set of base models is defined as:*

$$Mdl_0 = \{(GFG, fd) \mid GFG \in GFGr \wedge fd \in Ns\, GFG \rightarrow Fr \wedge \boxplus_F\, fd\}$$

*All fragments of the model are required to be mutually disjoint — conjunct $\boxplus_F\ fd$ (see def. 35 for $\boxplus_F$).*

*Projection functions mGFG and mFD extract the components of a model. Function mUFs yields union of all fragments of a model, and from gives fragment provenance of any model node:*

$$mGFG : Mdl_0 \to GFGr \quad mFD : Mdl_0 \to V \nrightarrow Fr \quad mUFs : Mdl_0 \to Fr \quad from : Mdl_0 \to V \nrightarrow V$$
$$mGFG(GFG, fd) = GFG \quad mFD(GFG, fd) = fd$$
$$mUFs\, M = \bigcup_F (\operatorname{ran} mFD_M) \quad from\, M\, v = (\mu\, vf : (Ns \circ mGFG)M \mid v \in fLNs(mFD_M\, vf))$$

*Predicate $\Uparrow M$ says whether the model's fragments comply with the GFG. Each proxy of the model and its referent must be consistent with the $\dashrightarrow$ relation (def. 42) which captures the referencing flow:*

$$\Uparrow M \Leftrightarrow \exists\, UF : Fr_0 \mid UF = mUFs\, M \bullet$$
$$\forall\, p : (NsP \circ fSG)UF \bullet (from\, M\, p, from\, M(\leftrightsquigarrow UF\, p)) \in ((\_ \dashrightarrow) \circ mGFG)M$$

*Set of valid models Mdl includes those whose union of fragments is total — $(mUFs\, M) \in TFr$ — and whose underlying referencing complies with the model's GFG — $\Uparrow M$.*

$$Mdl = \{M \mid M \in Mdl_0 \wedge (mUFs\, M) \in TFr \wedge \Uparrow M\}$$

*From a model, it is possible to obtain the fragment resulting from resolving the union of the model's fragments, called the model resolution fragment, through function $⦿^M : Mdl \to Fr$:*

$$⦿^M = ⦿ \circ mUFs$$

**Laws.**

$$M \in Mdl \vdash ⦿^M\, M \in TFr$$

□

Definition 44 (Model Morphisms). *Given models $M_s, M_t : Mdl$, a fragment morphism $m : M_s \rightarrow_M M_t$, such that $m \in GrM$ (def. 9) defines a mapping from the nodes and edges of the fragments of $M_s$ into their $M_t$ counter-parts. The set of morphisms between two models is defined as:*

$$M_s \rightarrow_M M_t = \{m \mid m \in GrM \wedge \exists\, UF_s, UF_t : Fr_0 \mid UF_s = mUFs\, M_s \wedge UF_t = mUFs\, M_t \bullet m \in UF_s \rightarrow_F UF_t\}$$

*The set of morphisms between the two models is the morphisms between the two fragments with the union of the fragments of each model.*

*The relation between two models $M_s, M_t : Mdl$ via a set of morphisms $ms : \mathbf{P}\, GrM$ is defined as:*

$$(M_s, ms) \Rrightarrow^M M_t \Leftrightarrow \bigcup_{GM} ms \in M_s \rightarrow_M M_t$$

□

Definition 45 (Model Refinement). *A model $M_c : Mdl$ refines another $M_a : Mdl$, through a set of morphisms $ms : \mathbf{P}\, GrM$, where each morphism maps elements of $M_c$ into $M_a$, written $(M_c, ms) \sqsupset^M M_a$, provided the following holds:*

$$(M_c, ms) \sqsupset^M M_a \Leftrightarrow \exists\, UF_c, UF_a : Fr_0 \mid UF_c = mUFs\, M_c \wedge UF_a = mUFs\, M_a \bullet (UF_c, \bigcup_{GM} ms) \sqsupset^F UF_a$$

*Above, union of fragments of $M_c$ must refine union of fragments of $M_a$ via morphism $\bigcup_{GM} ms$.* □

Definition 46 (Models as Types). *A graph with typing GwT : GrwT (def. 11) is a compliant instance of a type model M : Mdl, written GwT* $\ni^M$ *M, provided the following holds:*

$$GwT \ni^M M \Leftrightarrow GwT \ni^F mUFs\, M$$

*GwT must comply to the union of fragments of M, obtained through function mUFs (def. 43).* □

Definition 47 (Extra Typing for Models). *A source model* $M_s$ *: Mdl is extra-type compatible with a target model* $M_t$ *: Mdl, written* $M_s \dashv\vdash^M M_t$*, whenever the unions of the models' fragments are extra-type compatible:*

$$M_s \dashv\vdash^M M_t \Leftrightarrow mUFs\, M_s \dashv\vdash^F mUFs\, M_t$$

*Likewise for extra-type compliance, which reduces compliance between models to compliance between the union of the models' fragments:*

$$(GwET, M_s) \Vdash^M (GwT, M_t) \Leftrightarrow (GwET, mUFs\, M_s) \Vdash^F (GwT, mUFs\, M_t)$$

□