

Is UML Made of DSRP?

A Structural Decomposition of Elements, Relationships, Classes, Behavior, and Views

Derek Cabrera^{a,@}

a: Cabrera Research Lab

@: dac66@cornell.edu

ABSTRACT: Unified Modeling Language (UML) is a standardized modeling language whose vocabulary spans structure, behavior, interaction, deployment, and extensibility. This paper asks whether UML introduces any organizational structure not already expressible by DSRP-484. Following the structural-remainder protocol used in companion analyses of system dynamics and network science, we separate information (I) from organization (O), and we replace the earlier practice of nominating constructs by judgment with a pre-registered extraction rule applied to the OMG UML 2.5.1 normative Abstract Syntax Metamodel. The rule yields 242 metaclasses, of which 100 normalize into their ancestors under a purely structural criterion, leaving a closed candidate set of 142. The inclusion filter excludes none of them, because a normative abstract syntax has already been filtered by the standards body. The entire metamodel, 6,785 elements, is assembled from eighteen element types, of which eight are plainly informational. Ruling the remaining construction kit therefore rules UML: Class is identity and part-whole under two undeclared perspectives; Property performs three jobs on one organization, selected by a flag; Association carries two ordered action and reaction ends; Generalization is Distinction and Relationship simultaneously on one link. Twenty further constructs were selected before ruling on the criterion of whether each could break the claim, including conditionality, the temporal metaclasses, ordering, negation, templates, and the metalevel. A separately counted adversarial block was drawn from the Unified Foundational Ontology, whose authors hold that UML requires more primitives rather than fewer; its four load-bearing challenges are rigidity, existential dependence, four irreducible parthood relations, and relators. No residual organizational primitive X was identified anywhere. A second result concerns what the omissions cost: UML has no way to express negation and builds it eleven times, no way to name a point and hard-codes framing in sixty-nine Boolean flags, no Link construct and builds a surrogate for one, and no loop construct while ninety of its own metaclasses sit inside cycles. The result does not make UML unnecessary. It locates UML as a domain-specific modeling language whose higher-order constructs are generated from a more elemental organizational calculus, and whose recurring workarounds are the measurable cost of the structures it declines to name.

KEYWORDS: DSRP | DSRP-484 | UML | Unified Modeling Language | O-Theory | structural decomposition | organization | information | modeling languages | formalism bias

Cite this paper: Cabrera, D. (2026). Is UML made of DSRP? A structural decomposition of elements, relationships, classes, behavior, and views. *Journal of Systems Thinking*, 6(1), 1–30.

<https://doi.org/10.54120/jost.v26i1.dc6umdd6>

1. The Question Beneath the Diagram

UML is often encountered as a collection of diagram types: class diagrams, sequence diagrams, state machines, activity diagrams, component diagrams, deployment diagrams, and use cases. Beneath those diagrams is a standardized abstract syntax containing Elements, Relationships, Classifiers, Properties, Behaviors, Actions, Events, and other model constructs [1]. The practical success of that vocabulary does not establish that its constructs are organizationally elemental. A modeling language can legitimately treat a construct as primitive for purposes of notation and computation while the construct remains decomposable at a lower level of organization.

DSRP-484 makes the stronger claim that organization is generated through four recursively entangled Patterns - Distinctions, Systems, Relationships, and Perspectives - composed of eight elemental roles and governed by four structural dynamics [2–4]. The question is therefore not whether DSRP can be used to draw a UML diagram. It is whether the organization UML attributes to a modeled target requires anything outside the closure generated by DSRP-484.

The companion runs made this question operational for system dynamics and network science. System dynamics yielded sixty of sixty canonical constructs with no residual; network science yielded thirty-five of thirty-five [5,6]. UML is a particularly useful third test because it is explicitly a meta-modeling language: it does not merely analyze systems but specifies a grammar for representing objects, types, relationships, behaviors, interactions, deployment, and multiple diagrammatic views.

2. Two Levels That Should Not Be Confused

The decomposition depends on a strict distinction between information and organization. Let an organized target be represented generally as $X = I \times O$, where I is the content instantiated and O is the organization that content occupies [2,3]. The present claim concerns O .

The fixed decision rule is the same one used in the companion papers: a feature is organizational if changing or removing it changes what entities or relations exist, or how they are bounded, nested, or ordered. A feature is informational if changing or removing it changes only the content or value assigned to a structure that remains.

This distinction is especially important in UML because the language names many semantic roles. For example, Property --owned by--> Classifier contains a relational organization whether or not the relation carries the specific semantic label owned by. Remove the informational content "owned by" and D_i -- R -- D_i can remain. Likewise, Integer is informational content that can backfill a Distinction, System, Relationship, Perspective, or a composite. Information is not tied to one Pattern; it fills organization.

Multiplicity shows why the split must be performed feature by feature rather than construct by construct. The values 0..1, 1, or 0..* are information, but the allowable set or range they describe is itself a part-whole System. A single UML construct can therefore bundle I and O without introducing a new primitive.

One caution governs every ruling that follows. DSRP-484 is not a set of four bins into which features are sorted. Its Enclosure dynamic holds that no Pattern is ever instantiated alone: wherever a Distinction is drawn there is also System, Relationship and Perspective, necessarily and simultaneously [2–4]. The framework is total in that sense, and it is formally specified rather than impressionistic - four Patterns, eight elements, four special relationships and four dynamics, with the mathematics set out in the unified calculus and in O-Theory [2,3] and with published coding instruments derived from them [7,8]. The question a decomposition can therefore ask is never which Pattern a construct is. It is which Patterns a

formalism names, and which it leaves implicit while relying on them anyway. Each ruling below records two things: what must be present, and what UML writes down. The gap between those two columns is the finding, and it is a fact about UML's vocabulary rather than about organization.

3. Analytical Method: The Structural Remainder Test

The primary nominator is the Object Management Group Unified Modeling Language Version 2.5.1 normative specification, cross-checked against its normative Abstract Syntax Metamodel (UML.xmi) [1]. OMG identifies UML 2.5.1 as the formal specification and publishes both the normative PDF and the machine-readable metamodel. The candidate set is fixed by a pre-registered rule: named nominators, a five-category exclusion log, and a purely structural normalization criterion, all applied to UML.xmi before any construct was ruled. An earlier draft nominated constructs by judgment. That practice was retired because a reader cannot check it, whereas the present set can be re-derived from the file by anyone. Two further nominators were run against the result, one historical [9] and one adversarial [10,11]; they are reported in Sections 5 and 9.

Twenty-eight constructs are ruled individually here: the nine presented in Section 4, from which the entire normative metamodel is assembled, and twenty selected before ruling on a single criterion, whether each could break the claim. Generalization belongs to both groups and is counted once, which is why the total is twenty-eight rather than twenty-nine. The adversarial block is counted separately. The purpose is not to rule 142 metaclasses one at a time. It is to close the candidate set, and then to put the containment claim at risk against the constructs on which UML most visibly depends: Element, Relationship, classification, association, whole-part semantics, behavior, conditionality, time, ordering, negation, templates, the metalevel, and the mechanisms for scope and viewpoint.

For each construct three questions are asked: (1) what informational content is specific to it; (2) what organizational structure is necessary for it to exist; and (3) does that organization require a primitive outside DSRP-484? A residual X is recorded only if a necessary organizational feature cannot be generated through composition or recursion of D, S, R, and P with their eight roles and four dynamics.

The decisive result is not a high percentage of convenient labels. It is whether any construct leaves organizational structure unexplained.

Three features of the procedure keep it from being a labelling exercise, and they are worth stating explicitly because the objection is the natural one to raise. First, nothing is curated. The candidate set is produced by applying a fixed rule to a public file: a metaclass enters exactly when it introduces at least one owned attribute or association end that neither redefines nor subsets an inherited property. Running that rule against UML.xmi returns 142 candidates and 100 normalizations, and it returns them to anyone who runs it.

Second, the ruling on what each candidate contributes is decided by the file rather than by the analyst. Each novel feature carries a declared type, written by OMG. A feature typed Boolean is information because the standard says it is a Boolean; a feature typed by another metaclass is a relation to that metaclass because the standard says so. No step in the classification requires anyone to interpret what a construct means, and the classification leaves nothing over: all 271 novel features across the 142 resolve, with no unresolved residue in which discretion could hide.

Third, and most importantly, the claim is falsifiable by a count rather than by an argument. If UML introduced organization outside the closure, it would appear as a novel contribution of a fourth kind - something a metaclass adds that is neither a relation to another element nor a value on a structure already present. That is a category a script can find, and the number of them is reported rather than asserted. A

labelling exercise cannot fail in that way. This one can, and the sections that follow are an account of where it was put at risk.

4. What UML Is Made Of

UML semantics are prose. The specification describes meaning in natural language, and the language shipped without a formal execution semantics, which is why fUML was built afterwards. A decomposition arguing from UML meanings would therefore be arguing about paragraphs.

Its abstract syntax is a different matter. The normative Abstract Syntax Metamodel is machine-readable, constraint-checked and published by OMG as UML.xmi [1]. The question asked here is therefore not what UML means by a Component, but what a Component is made of in the normative model - a question with an exact answer that the standards body itself supplies.

The whole metamodel, 6,785 elements, is assembled from eighteen element types: Comment (1,584), Property (950), LiteralInteger (787), Constraint (674), OpaqueExpression (674), LiteralUnlimitedNatural (469), Association (418), Generalization (304), Parameter (260), Class (242), Operation (205), EnumerationLiteral (62), LiteralBoolean (60), PackageImport (55), Package (15), Enumeration (13), InstanceValue (12), and Tag (1).

One observation precedes any decomposition, because it is visible in the file without performing one. UML.xmi has exactly three mechanisms for writing organization down and no others. It nests one element inside another, which is how composite ownership is expressed - ownedComment, ownedRule, ownedAttribute, packagedElement. It gives every element an identity: 6,776 of the 6,777 identifiers in the file sit on an element that declares which UML metaclass it instantiates, the single exception being a MOF housekeeping tag. And it points from one element to another through exactly seven named properties - annotatedElement, memberEnd, constrainedElement, subsettedProperty, redefinedProperty, redefinedOperation and precondition - 3,883 pointers in all, of which none dangles. Everything else in the file is a value carried on one of those structures: visibility, multiplicity, the Boolean flags, the constraints.

Two things follow, and they should not be confused. The weaker one is that these mechanisms are not an artefact of XML. A format that has to record organization must record boundedness, identity and reference, because that is what organizing consists of; any format that organizes anything arrives at the same short list, and the seven pointers above are UML properties named in the specification rather than serialization conventions. The stronger and more important one is that this is an opening rather than a result. It would be an error to read the three mechanisms as three Patterns - under Enclosure all four are present in each of them, and nesting is no more purely System than pointing is purely Relationship. What the observation establishes is only that the vocabulary available for writing organization down is very short, and that UML is written in it. What that vocabulary is made of is the question the rest of this paper asks.

Eight of the eighteen are plainly informational: 3,649 of the 6,785 elements are content rather than structure. The remainder is the construction kit, and it is short enough to decompose directly.

Class. A Class classifies a set of objects and specifies the features that characterize the structure and behavior of those objects. Naming is D_i. The lists are S_p^w. Structure and behavior are two P_p, each yielding a part-whole view - and this is not an interpretation imposed on the construct, since the metamodel keeps two separate lists, ownedAttribute and ownedOperation.

Property. The most common structural element in the file, and the clearest single case of the thesis. The specification gives it three jobs: an attribute relating an instance to a value, an end of an Association relating an instance to another object, and a composite part containing another object. All three are the

same organization, S-R-S, differing in role assignment and in the informational content carried on the relation. Which reading applies is selected by a Perspective over two distinctions, owned-by versus member-end, together with one enumeration literal. Nine hundred and fifty elements, one organization, three readings chosen by information.

Association. A link is a tuple of values that refer to typed objects; an Association classifies a set of links. The member ends are R_a and R_r , and memberEnd is ordered, so the tuple reading is enforced by the metamodel rather than inferred. Reifying an Association yields D_i .

Generalization. S_p^w together with D_i^o and R_a^r , the Distinction and the Relationship simultaneous on one link rather than alternative readings of it.

Package. Recursive S_p^w , with containment pointing both upward and downward.

Operation. $S \leftarrow R_a^r \rightarrow S$ between an invoking and a responding system. Parameters carry what crosses; preconditions are information on the action side, postconditions and body conditions on the reaction side.

Parameter. A Relationship carrying a Perspective. The construct declares what may pass - type, multiplicity, direction, effect - and never holds the thing that passes; the content sits on Pins in a different package. It does not merely connect. It frames what may cross.

PackageImport. The fullest construct in the kit, and the only place UML writes all four Patterns down at once: a reified identity, both role-ends inherited from DirectedRelationship, three owned parts, and a stated constraint on who may see what.

Enumeration. S_p^w . The enumeration is the whole and its literals are the parts; the content of any literal is I.

No residual organizational primitive was identified in any of the nine. The tenth structural element type, Constraint, is ruled in Section 6, as is Generalization; both are counted among the twenty there rather than here, so that the twenty-eight constructs ruled individually are not double counted.

5. The Candidate Set

UML.xmi contains 242 metaclasses, distributed across the fourteen packages corresponding to Clauses 7 through 20. Under the normalization criterion of Section 3 - a metaclass is its own candidate only if it introduces an owned attribute or association end not inherited from its generalizations - 100 normalize into their ancestors and 142 remain. The criterion is purely structural, decided from the file before any decomposition, and re-runnable: the candidate list is generated by fetching and parsing the normative XMI rather than transcribed by hand.

The inclusion filter excluded nothing. All 142 pass. This is worth stating plainly because the companion runs did not have this property: the Ford glossary produced seventeen logged exclusions for C_{SD} , and the network extraction logged ten. The difference is not diligence but the nature of the nominator. A prose glossary contains workflow terms, software terms and numerical procedures that a structural test must set aside. A normative abstract syntax contains only abstract-syntax metaclasses. OMG had already performed the filtering.

Nine constructs are retained and ruled trivially informational - the five literal value specifications, InstanceValue, OpaqueExpression, Comment and Image - matching the three so ruled in C_{SD} .

Four constructs came close to exclusion as execution machinery: LinkEndData and its two specializations, together with QualifierValue. They were kept, and the reason is itself a result. The specification's own definition reads: As a link (that is not a link object) cannot be passed as a runtime

value to or from an Action, it is instead identified by its end objects and qualifier values. Excluding these would have discarded UML stating, in its own words, that it has no Link construct and describing the surrogate it built in place of one.

The second nominator adds nothing. Selic's 1998 account of modeling complex real-time systems names the constructs a principal UML architect judged necessary for architectural modeling and adds them to UML 1.1 through the extension mechanism [9]: capsules, ports, connectors, protocols, protocol roles, relay and end ports, plug-in slots, sub-capsules. Every one is already in the candidate set, as StructuredClassifier, Port, Connector, ConnectorEnd, Collaboration, CollaborationUse, ProtocolStateMachine, ProtocolTransition and ProtocolConformance. An independent architect's statement of what a modeling language requires is fully contained in the set the primary rule produced. Because the paper predates UML 2, this is a retrospective containment check rather than a parallel one, and is reported as such.

Closing the set is not the same as ruling it. The normalization criterion of Section 3 admits a metaclass to the candidate set exactly when it introduces at least one owned attribute or association end that neither redefines nor subsets an inherited property. That criterion does more than select: it names, for every one of the 142, precisely what that metaclass contributes that its ancestors did not. The contribution, not the metaclass, is therefore the honest unit of the residual test, and it can be counted exhaustively.

Across the 142 there are 271 such novel contributions. Every one is a relationship end, a primitive value, or an enumeration value. There are 160 of the first, 96 of the second and 15 of the third. There is no fourth kind: not one contribution in the candidate set falls outside those three, and the classification is decided by the declared type of the feature rather than by any reading of what it means. Appendix A lists all 142 with their contributions and the ruling on each; Appendix B lists the 100 that normalize.

This is what converts the containment claim from a sample into a census. Twenty-eight constructs are ruled individually because they carry the argument. The rest are ruled at the level of what they actually add, which is the level at which a residual would have to appear. A residual X would be a novel contribution of a fourth kind, something a metaclass introduces that is neither a relation to another element nor a value on a structure already present. The count of those is zero.

6. The Hard Cases

A decomposition of the construction kit would be worth little if it failed where the language is most demanding. Twenty constructs were therefore selected before any of them was ruled, on a single criterion: could this one break the claim. They are reported here by the pressure each applies.

The root construct, and a pole left implicit. UML Element is the most general model-side constituent, and presents an identity: an Element is something that can be treated as this element rather than another. Its decomposition is D_i. The notable omission is D_o. UML gives the modeler the identity pole and does not make the co-implied other an explicit companion construct. This is not a residual against DSRP but the opposite: Element is a thinner projection of Distinction, retaining D_i and suppressing D_o from explicit grammar.

Relationships, and relationships treated as things. At the language level a Relationship is often presented generically. The generic presentation is not a second elemental kind of Relationship; it is constituted by R_a^r , with some constructs declining to explicate the action and reaction role-ends. When a Relationship is itself treated as a model object, the reified relationship becomes D_i, and that D_i is the relationship-as-object rather than one of the relata. UML supplies an unusually clean example of the rule that Relationships can themselves be distinguished and recursively structured. The metamodel records the

move in its own inheritance: of the twenty-four metaclasses inheriting from Relationship, three plainly relational constructs do not. GeneralOrdering and Message inherit NamedElement and Connector inherits Feature, because a relation that must be named, owned and referenced needs identity. AssociationClass performs the same move openly, inheriting from both.

Conditionality. This is where the decision rule of Section 2 applies most sharply, since a guard determines which relation obtains and the rule counts as organizational anything whose change alters what relations exist. Clause carries a test, a decider, a body and an ordering over sibling clauses; LoopNode adds iteration with an explicit decider; DecisionNode carries an input whose value selects a branch; ConditionalNode asserts whether any branch will fire. In each the test exists, the body exists, and the link between them exists; the Boolean creates and destroys nothing. It selects among structures that are all already present, and selection among present structures is P_p^v . The wrinkle worth stating is that a true test may still lose to a competing clause, so the winner is fixed by ordering together with values - which is a Perspective over existing Clauses, not a further structure.

Time. Behavior is the strongest candidate for a temporal residual, so the constructs UML built expressly for time were tested rather than the state machine alone. Duration is a variable of I on a relation: o-duration-o is o-r-o with I equal to duration, exactly as lag was ruled in the system dynamics run. TimeExpression is a distinguished moment. TimeObservation is a Perspective whose point is the clock and whose view is when this happens, where this may be any I/O structure; DurationObservation is that Perspective taken over a barbell. TimeEvent is a Relationship on a time variable. Applying the subtraction test to each leaves nothing orphaned: remove the length and the relation remains; remove the reading and the viewpoint remains, still pointed at its event.

Order. GeneralOrdering is the thinnest construct in the file, holding only a before and an after. The specification describes it as a binary relation whose purpose is to impose partial order where none was otherwise specified. It decomposes as a System of distinguished occurrences linked by directed Relationships, requiring no temporal primitive.

Negation. UML has no way to say not, and builds negation by hand eleven separate times. The clearest case is the interaction operator neg, which draws a box around a sequence in order to assert that the sequence is invalid: a structure drawn in order to say that it does not obtain. The general ruling is that D_o is negation, and that negation attaches to any Pattern and to either side of $X = I \times O$: not-related, not-a-part, not-viewed. MessageKind is the same operator enumerated rather than expressed, tabulating the four presence-and-absence combinations of a Relationship's two role-ends.

Frameworks. A template is not a Perspective placed on an element. It is an I/O structure whose organization is fully specified and part of whose information is left unfilled, with binding supplying the remainder. Four metaclasses, a signature, a binding and a substitution mechanism exist to hold organization fixed while information is deferred - a mechanism coherent only if the two separate. The same shape recurs in InformationFlow, whose four realizing pointers defer the carrier, and in the default value of a Parameter.

The metalevel. GeneralizationSet collects is-a links and then makes claims about the collection: whether the specific classifiers cover the general one, whether they overlap, and whether a powertype names the partition. The is-a link itself is a Distinction and a Relationship at once - it states that Oak is a Tree and that Oak stands to Tree in the specialization relation - and the equality case is the clean analogue, since a equals b distinguishes a as a b and relates a to b under equality in one structure. The covering flag is treated in Section 8.

Regions and concurrency. An InterruptibleActivityRegion is a barbell whose action is a stop and whose reaction end is a whole of things stopped; a single interrupting edge terminates everything within the

region. An ExpansionRegion is an in-relation, a whole, and an out-relation, with its mode a Perspective whose points are themselves a three-part whole. A JoinNode is a whole whose parts are the incoming Relationships, under a Perspective asking whether the whole is yet complete; when the answer becomes yes the outgoing Relationship obtains. Waiting is therefore not an additional ingredient but completeness seen from somewhere.

Boundaries and architecture. A Component is a bounded whole with internal parts and external relationships. StructuredClassifier makes the recursion explicit: the classifier is recursive S_p^w , Ports are the R_a and R_r role-ends themselves, and what a Port carries at those ends is a single part seen two ways: its required and provided interfaces are one S_p , co-part of both the relata and the relationship, distinguished only by whether a Perspective is applied to it. The required set carries a P_p constraint on that shared part; the provided set is the same part with no P_p applied. Neither is a negation, though the naming invites one to read them as opposites. Connectors are relations between ports, and D_i^o supplies the distinctions among classifier, port and connector. Interface and Realization form a richer composite in which two distinguished elements stand in a Relationship while the Interface itself opens as a whole of required and provided features. What UML calls architecture is recursive part-whole organization populated by distinguished structures and connected through Relationships.

Extension. Profile, Stereotype, Extension and ProfileApplication constitute a subsystem for viewing and labeling existing structures differently by domain. Extension is built as an Association, so the view is attached by a relationship and can be added and later removed without the underlying structure changing. The definition of Stereotype states the operation as plainly as the specification ever does: domain-specific terminology in place of, or in addition to, the ones used for the extended metaclass.

The two that range. InformationFlow describes circulation of information in a general manner and, in the specification's own words, does not specify the nature of the information, the mechanisms by which it is conveyed, sequences of exchange, or any control conditions. Its organization is two distinguished endpoints and a directed Relationship; what is conveyed is information. Constraint does not resolve to one form, and the reason is visible in its three parts: an ordered set of constrained elements, a context that is the Namespace owning it, and a specification that must evaluate true. A set, a place from which the rule has force, and a condition. Which part is in view decides whether the construct reads as Perspective or as Relationship; it is not one fixed form because it is not one thing.

No residual organizational primitive was identified in any of the twenty.

7. Formalism Bias and I/O Compression

The most consistent result of the UML run is not simply that its constructs can be labeled D, S, R, or P. It is that UML repeatedly treats informational distinctions as if they were different kinds of modeling machinery while reusing the same organization underneath. Classifier, Class, Instance, Property, Association, Dependency, Interface, Component, Message, Signal, Deployment, and Constraint differ enormously in domain meaning. At the O level they repeatedly collapse to the same small set of structures.

Three compressions are especially visible. First, Distinction is one-poled in the root Element construct: UML makes D_i explicit while D_o remains largely implicit. Second, Relationship is frequently presented as R_{gen} , suppressing the action-reaction substructure that DSRP treats as constitutive. Third, Perspective appears unevenly. It becomes explicit in scope, profiling, and use-case framing, but much of UML otherwise presents its structures as if observer or frame were absent.

The compression can be measured rather than described. UML declares 950 features across its 242 metaclasses, and their declared types partition exactly: 836 are typed by another metaclass, 98 by a primitive and 16 by an enumeration, with nothing unresolved. Every metaclass-typed feature is an association end and every association end is metaclass-typed, so there is no third case. Eighty-eight per cent of UML's declared vocabulary is therefore relational and twelve per cent is information carried on it. The organization is small and repeated; the variety sits in the content.

The compressions also form a recognisable pattern, and it has a name. The cognitive Pareto law holds that structural thinking is unevenly distributed in a lawful way, and it operates at two levels [8]. At the level of the Patterns, a few are used constantly and the rest rarely. Within each Pattern, one pole of the co-implied pair is made explicit far more often than the other - identity over otherness, part over whole, generic relation over action and reaction, view over point. This second level, the law of structural neglect, makes a specific prediction about what any formalism will omit.

UML's omissions match that prediction item for item. It names identity and leaves otherness to be hand-built. It names parts, and treats having nothing above you as an unremarkable default rather than a boundary choice. It presents relations generically while building one role-end and then filing it where neither participant can reach it. And it frames constantly while almost never naming the standpoint the framing is done from. What Section 8 counts as workarounds are the costs of those four omissions, and of no others - not one entry in the census is a case of UML failing to reach for a Pattern, and every entry is a case of it naming one pole of a pair and leaving the co-implied pole unwritten.

The result resembles the formalism bias observed in the network run. A language naturally sees the structures it explicitly implements and compresses those it does not. In UML, that compression is partly masked by a very large semantic vocabulary. The vocabulary is rich in I even when the organization is recurrent. Under Enclosure the point is sharper than compression: no Pattern is instantiated alone, so the omitted structures are necessarily present in every UML construct and the language simply declines to name them. Three counts make this concrete. UML has no way to say not, and hand-builds negation eleven separate times. It has no way to name a point, and hard-codes the framing in 69 Boolean flags, 13 enumerations, visibility, multiplicity and 674 OCL constraints. And it has no loop construct at all, while 90 of its own 242 metaclasses sit inside cycles and 24 point at themselves.

8. What the Omissions Cost: A Census of Workarounds

The preceding sections report that UML organization decomposes without residual. This section reports something the decomposition alone does not: that UML pays, repeatedly and in public, for the structure it declines to name.

The argument is not that UML is badly built. It is that a formalism does not construct surrogates for structure it does not need. Each entry below is a measurable expenditure, and each is reproducible from the public file.

Recomputing severed paths: 83 derived properties, plus 32 derived unions. ActivityEdge.inGroup, ConnectableElement.end, Class.extension, ActivityGroup.containedEdge, Class.superClass, Association.endType, Port.provided, Component.required. Almost all are reverse lookups, recovering a direction that was built and then put out of reach. The mechanism is exact and readable in the file. Every association has two ends; each end is a Property with its own identifier, distinct from the classifier that holds it. In 328 of the 418 associations one of those two ends is owned by the Association itself rather than by either classifier it connects. In 90 both ends are held by the things they connect. In none are both orphaned: UML always names the action somewhere, and it is the reaction end that is built and then filed

where neither participant can see it. Of 350 pairs of metaclasses related at all, only 104 hold structure at the classifier in both directions.

A surrogate for the missing Link: four metaclasses. LinkEndData, LinkEndCreationData, LinkEndDestructionData and QualifierValue, built because a link cannot be passed as a value, with eleven further metaclasses operating on the thing the metamodel does not contain.

Enumeration literals doing structural work. AggregationKind carries the entire whole-part apparatus in three literals. MessageKind enumerates the four presence-and-absence combinations of the two role-ends of a Relationship, and disowns the fourth as one that should not appear.

Negation, hand-built eleven times. The count is of instances rather than names: isCovering, isDisjoint, isFinalSpecialization and isException once each, isReadOnly on two metaclasses and isAbstract on three, together with the interaction operators neg and break, which are enumeration literals rather than Boolean flags. Nine Boolean instances and two operators make eleven.

Framing, hard-coded sixty-nine times. Sixty-nine distinct Boolean flags, the majority of which state that from some standpoint only some things are permitted, together with thirteen enumerations, visibility on four metaclasses, and multiplicity throughout.

Rules policing what the structure cannot hold: 449 constraints written directly on 156 of the 242 metaclasses. Counting also those attached to individual features, the file contains 674 Constraint elements in all. The most constrained is Property, with eleven. The construct performing three jobs separated by a flag requires eleven written rules to keep those jobs from colliding.

Notation that exists only in prose. The strings {incomplete} and {overlapping} appear nowhere in the normative model. Table 9.1 of the specification supplies them as textual annotations and states that the default values are {incomplete, overlapping}. The default condition of every generalization set is therefore one in which a remainder exists and the specific classifiers overlap, and neither is representable. The specification's own worked example writes the remainder out in words: a single specialization expressed as {incomplete}, which means that a Person may either be an Employee or not.

An extension subsystem: four metaclasses. Profile, Stereotype, Extension and ProfileApplication, added so that the same structures may be viewed and labeled differently by domain. The definition of Stereotype states the operation directly: domain-specific terminology in place of, or in addition to, the ones used for the extended metaclass.

Double inheritance so that a relation may also be a thing. AssociationClass inherits from both Association and Class, and has been a recognized difficulty in the metamodel for that reason.

Relations abandoning Relationship to obtain identity: three. GeneralOrdering and Message inherit NamedElement; Connector inherits Feature. The specification calls GeneralOrdering a binary relation and does not inherit it from Relationship.

Declared effects in place of observed effects: five slots. localPrecondition and localPostcondition on Action, and three condition slots on Operation. The declaration is required because the actor cannot observe: fourteen of forty-two concrete Actions have no output pin.

And one case in which the specification concedes that its own formalism cannot express its own operation. Of ValueSpecification.isComputable: This operation cannot be fully defined in OCL. A conforming implementation is expected to deliver true for this operation for all ValueSpecifications that it can compute.

The census is what converts this run from a labeling exercise into a measurement, and it gives the formalism-bias result its ending. It is not merely that a language sees the structures it implements. It is that the structures it does not implement return as costs, and the costs can be counted.

There is a more exact way to say what a workaround is. A model that is missing structure eventually fails in use: something cannot be expressed, or is expressed wrongly, and the failure comes back. That returning signal is what the DSRP literature calls the love reality loop, the mechanism by which a model is progressively brought into alignment with what it models [7]. For the loop to close, the missing structure has to be recognised as structure. Where it is not - where the modeller has no name for the thing that is absent - the signal still arrives but cannot be acted on structurally, and the available response is to add a feature that covers the symptom.

On that reading the census is a record of a loop that ran and could not close. Eleven separately hand-built negations are eleven occasions on which the absence of a way to say not was felt and patched. So is a four-metaclass surrogate for a Link the metamodel does not contain; so is an extension subsystem added so that the same structures may be viewed and labelled differently; so is a specification conceding that its own formalism cannot express one of its own operations. None of this indicates that UML is badly built. It indicates that the feedback was real, that it was received, and that it was absorbed as vocabulary because the structure it pointed at had no name.

9. An Adversarial Test

A decomposition that consults only sources sympathetic to its conclusion is weak whatever it finds. The candidate set was therefore supplemented with a nominator selected because its authors argue the opposite of the present thesis.

The Unified Foundational Ontology and its modeling language OntoUML hold that UML is ontologically under-specified: that it requires more primitives than it has, and that instantiation, parthood and subtyping are mutually irreducible [10,11]. Its specification defines twenty-four constructs and three principles, every one of which exists because plain UML is held to be unable to express it. None appear in the candidate set; they are ruled here as a separate block and counted separately.

Four are load-bearing.

Rigidity. A Kind applies to its instances necessarily, a Role contingently: a person cannot cease to be a person, but may cease to be a student. This is the strongest residual candidate encountered in the run, because it is modal where the preceding analysis is not. Everything decomposed so far concerns what is and is not the case; rigidity concerns what must be.

The ruling is that Kind and Role are Perspectives on one Distinction: the frame in which identity is held fixed, and the frame in which it is permitted to vary. Rigidity presupposes that an identity criterion is settled, and identity criteria are not found but drawn. A rock is a portion of a mountain; broken, it is two rocks; ground, it is sand. The claim that Socrates is necessarily a person is evaluated only given a criterion for what counts as Socrates, and that criterion is a distinction someone has made.

The modal objection, that rigidity is de re necessity rather than frame-dependence, requires that types be observer-independent universals. That is a defensible position, but it is an assumption carried into the analysis rather than a result of it, and it should be named as one.

Existential dependence. A mode cannot exist without its bearer. The ruling is that the only existential dependence is coimplication: a part implies a whole, and a mode implies a bearer, because the poles co-imply. What does not follow is that a particular part perishes with a particular whole. UML makes the

same move with composite aggregation, and its own text shows where the claim actually lands: all such links are destroyed when the containing StructuredClassifier instance is destroyed. What ends with the whole is the link, not the part.

The four parthood relations. ComponentOf, MemberOf, SubCollectionOf and SubQuantityOf are held to be structurally distinct, chiefly because parthood is not uniformly transitive. The ruling is that these are four grains at which part-whole is drawn - a functional complex, a collection, a collection of collections, a mass - and that apparent failures of transitivity are equivocations across grain. The parts of a person are organs at one grain and cells at another; the chain appears broken only when a question moves between them. Grain is a Perspective, and the four labels are information about which grain is in force.

Relators and mediation. A material relation is held to be founded on a relator: is married to is made true by a marriage, which mediates the two participants, whereas a formal relation such as is taller than holds directly. The ruling is that anything may occupy a relating position. A thing regarded on its own is an identity; the same thing standing between two others is a relationship. The marriage is not a special category of entity but a thing in the relating position, simultaneously Distinction and Relationship. This also explains why AssociationClass must inherit from both Association and Class, and why the difficulty does not arise where the two were never separated. The formal-material distinction accordingly tracks whether the occupant of the relating position has been named, not whether one exists.

No residual organizational primitive was identified in the adversarial block.

10. UML Is Not Displaced

Nothing in this analysis makes UML unnecessary. UML provides a standardized syntax, domain conventions, diagram types, metamodel constraints, tool interoperability, and a mature representational practice. Those are substantial achievements. The claim concerns level of explanation: UML constructs can remain stable and useful higher-order compounds even if their organization is generated from more elemental structures.

The relation is therefore the same one proposed for system dynamics and network science: not replacement, but structural containment. UML asks how to model software and systems using a standardized language. DSRP asks what the organizational structures in that language are themselves made of.

The claim can be put more strongly than not replacement. An interface layer is built on something. The organizational structures recovered here were being used while UML was designed, whether or not anyone named them at the time, and they will be used by whatever UML adds next; a construct introduced in a future version will be assembled from the same elements, because those are the elements available. The argument is not that UML is unnecessary or that it should be replaced by a decomposition. It is that a useful interface layer rests on a grammar it does not itself supply, and that the grammar can be written down.

11. What Would Falsify the Claim

The containment hypothesis would be falsified by an uncontested UML construct whose necessary organizational structure cannot be represented through any composition or recursion of D, S, R, and P with their eight elemental roles and four dynamics. Legitimate failures would include a fifth Pattern, a ninth elemental role, or another irreducible structural dynamic required by UML.

A notation glyph, rendering convention, tool workflow, parsing rule, or computational procedure would not by itself falsify the claim unless it introduced an irreducible form of organization in the model-side ontology. Nor would successful post hoc labeling confirm the claim. The proper burden remains a prospective candidate set with a residual X that can be populated.

No X was identified in the nine kit constructs from which the entire normative metamodel is assembled, in the twenty hard cases, or in the adversarial block; nor in any of the 271 novel structural contributions made by the 142 members of the candidate set. The falsifier is best stated as a procedure rather than a definition: subtract the informational content, and if what remains requires a new element to hold it, that is X. A legitimate X must therefore be a fifth mediation - a special relationship that is not exists/negates, belongs/contains, affects/effects or frames/is-framed. The exhaustive extraction from UML.xmi that an earlier draft proposed as future work has now been performed. The strongest next step is to rule the remaining members of the candidate set individually, and to run the same rule against whatever UML adds next.

12. Convergence: A Third Independent Modeling Tradition

The system dynamics run decomposed sixty canonical constructs; the network science run decomposed thirty-five [5,6]. Across these three runs, 95 construct-level tests drawn from pre-registered candidate sets in system dynamics and network science, together with a closed 142-construct UML candidate set of which twenty-eight were ruled individually, have now been exposed to the same basic question without producing a residual organizational primitive.

The convergence is stronger than the count because constructs that developed in different traditions repeatedly land on the same organization. A feedback loop, network cycle, and behavioral chain are Systems of Relationships. A stock, component, class, package, and aggregation are variations on part-whole structure. Network centrality, UML scope/profile/use-case framing, and SD validity/order constructs expose Perspective. The vocabularies differ because the informational content and modeling purposes differ. The organizational forms recur.

13. Conclusion

UML has a large vocabulary, but its organizational ontology is small. Element is $\$D_i\$$ with the other pole left implicit. Relationships are generic presentations of $\$R_a^r\$$, sometimes exposing their action-reaction roles and sometimes suppressing them. Types, Classes, Components, Packages, Aggregations, Compositions, and Behaviors are recursive Systems. Objects, states, actions, ports, nodes, artifacts, and lifelines are Distinctions occupying roles within those Systems. Associations, dependencies, transitions, connectors, messages, deployments, and information flows are Relationships. Scope, profiles, and use cases explicitly recruit Perspective.

At the same time, UML provides a particularly clean demonstration of the I/O distinction. A substantial portion of its apparent ontological variety is informational terminology instantiated on recurring organization. Change the label, role meaning, type content, message semantics, or constraint predicate and the organizational skeleton may remain. Where the structure itself changes, the change is describable through D, S, R, P, or their composites.

Twenty-eight constructs were ruled individually, drawn from a closed candidate set of 142 extracted from the normative metamodel under a pre-registered rule, together with a separately counted adversarial block. No residual X was identified in any of them. UML therefore appears, at this stage, not as a rival

elemental ontology but as a domain-specific modeling language composed from the same organizational grammar found in the system dynamics and network runs. The remaining risk is where it should be: in the rest of the metamodel, and in whatever UML adds next. One legitimate construct that requires something else is sufficient to break the claim.

References

1. Object Management Group. Unified Modeling Language (UML), Version 2.5.1. Object Management Group; 2017 Dec.
2. Cabrera D. The Unified Calculus of Organization, Structure, Emergence, and Meaning: The Patterns That Connect Mind and Nature. *Journal of Systems Thinking*. 2025;5: 1–41.
3. Cabrera D, Cabrera L. O-Theory: The Science of Organization. *Journal of Systems Thinking*. 2025;5: 1–12.
4. Cabrera D, Cabrera L. The Grammar of Reality: The Organizing Principles of Structure and Emergence. *Journal of Systems Thinking*. 2025;5: 1–18.
5. Cabrera D, Cabrera L. Is Network Theory Made of DSRP? A Structural Decomposition of Nodes, Edges, Centrality, Communities, and Network Growth. *Journal of Systems Thinking*. 2026.
6. Cabrera D, Cabrera L. Is System Dynamics Made of DSRP? A Structural Decomposition of Stocks, Flows, Feedback, Delays, and Dynamic Change. *Journal of Systems Thinking*. 2026.
7. Cabrera D, Cabrera L. DSRP-484: Building an advanced understanding of O-Theory, DSRP Theory, and the DSRP Calculus. *Journal of Systems Thinking*. 2026;6: 1–12.
8. Cabrera D, Cabrera L. The Pareto Structure of Thought: Empirical Discovery of the Six Foundational Mental Moves. *Journal of Systems Thinking*. 2025;5: 1–41.
9. Selic B. Using UML for modeling complex real-time systems. *Languages, Compilers, and Tools for Embedded Systems (LCTES 1998)*. Berlin: Springer; 1998. pp. 250–260.
10. Guizzardi G. *Ontological Foundations for Structural Conceptual Models*. University of Twente. 2005.
11. Guizzardi G, Wagner G, Almeida JPA, Guizzardi RSS. Towards ontological foundations for conceptual modeling: The Unified Foundational Ontology (UFO) story. *Applied Ontology*. 2015;10: 259–271.

Appendix A. The Closed Candidate Set, Ruled

All 142 members of the candidate set extracted from UML.xmi under the pre-registered rule of Section 3, each with the structural contribution it introduces that its ancestors do not, and the ruling on that contribution. Generated by script from the normative file; re-running reproduces the table exactly. Column 4 gives the feature name and its declared type, which is what decides the ruling.

#	Metaclass	Package	Novel contribution (feature : declared type)	Ruling	X
1	AcceptEventAction	Actions	isUnmarshall : Boolean	1 primitive value – 1	none
2	Action	Actions	context : Classifier; isLocallyReentrant : Boolean	1 relationship end – $R_a^r + 1$ primitive value – 1	none
3	ActionExecutionSpecification	Interactions	action : Action	1 relationship end – R_a^r	none
4	Activity	Activities	isReadOnly : Boolean; isSingleExecution : Boolean	2 primitive values – 1	none
5	ActivityEdge	Activities	inGroup : ActivityGroup; interrupts : InterruptibleActivityRegion; source : ActivityNode; target : ActivityNode	4 relationship ends – R_a^r	none
6	ActivityGroup	Activities	containedEdge : ActivityEdge; containedNode : ActivityNode	2 relationship ends – R_a^r	none
7	ActivityNode	Activities	inGroup : ActivityGroup; incoming : ActivityEdge; outgoing : ActivityEdge	3 relationship ends – R_a^r	none
8	ActivityParameterNode	Activities	parameter : Parameter	1 relationship end – R_a^r	none
9	ActivityPartition	Activities	isDimension : Boolean; isExternal : Boolean; represents : Element	1 relationship end – $R_a^r + 2$ primitive values – 1	none
10	AddStructuralFeatureValueAction	Actions	isReplaceAll : Boolean	1 primitive value – 1	none

11	AddVariableValueAction	Actions	isReplaceAll : Boolean	1 primitive value – 1	none
12	Artifact	Deployments	fileName : String	1 primitive value – 1	none
13	Association	StructuredClassifiers	isDerived : Boolean	1 primitive value – 1	none
14	Behavior	CommonBehavior	isReentrant : Boolean; specification : BehavioralFeature	1 relationship end – $R_a^r + 1$ primitive value – 1	none
15	BehaviorExecutionSpecification	Interactions	behavior : Behavior	1 relationship end – R_a^r	none
16	BehavioralFeature	Classification	concurrency : CallConcurrencyKind; isAbstract : Boolean; method : Behavior; raisedException : Type	2 relationship ends – $R_a^r + 1$ primitive value – 1 + 1 enumeration value – 1	none
17	BroadcastSignalAction	Actions	signal : Signal	1 relationship end – R_a^r	none
18	CallAction	Actions	isSynchronous : Boolean	1 primitive value – 1	none
19	CallBehaviorAction	Actions	behavior : Behavior	1 relationship end – R_a^r	none
20	CallEvent	CommonBehavior	operation : Operation	1 relationship end – R_a^r	none
21	CallOperationAction	Actions	operation : Operation	1 relationship end – R_a^r	none
22	Class	StructuredClassifiers	extension : Extension; isActive : Boolean	1 relationship end – $R_a^r + 1$ primitive value – 1	none
23	Classifier	Classification	general : Classifier; isAbstract : Boolean; isFinalSpecialization : Boolean; powerTypeExtent : GeneralizationSet; useCase : UseCase	3 relationship ends – $R_a^r + 2$ primitive values – 1	none
24	ClassifierTemplateParameter	Classification	allowSubstitutable : Boolean; constrainingClassifier : Classifier	1 relationship end – $R_a^r + 1$ primitive value – 1	none

25	Clause	Actions	body : ExecutableNode; bodyOutput : OutputPin; decider : OutputPin; predecessorClause : Clause; successorClause : Clause; test : ExecutableNode	6 relationship ends – R_a^r	none
26	ClearAssociationAction	Actions	association : Association	1 relationship end – R_a^r	none
27	CollaborationUse	StructuredClassifiers	type : Collaboration	1 relationship end – R_a^r	none
28	CombinedFragment	Interactions	interactionOperator : InteractionOperatorKind	1 enumeration value – I	none
29	Comment	CommonStructure	annotatedElement : Element; body : String	1 relationship end – R_a^r + 1 primitive value – I	none
30	Component	StructuredClassifiers	isIndirectlyInstantiated : Boolean; provided : Interface; required : Interface	2 relationship ends – R_a^r + 1 primitive value – I	none
31	ConditionalNode	Actions	isAssured : Boolean; isDeterminate : Boolean	2 primitive values – I	none
32	ConnectableElement	StructuredClassifiers	end : ConnectorEnd	1 relationship end – R_a^r	none
33	ConnectionPointReference	StateMachines	entry : Pseudostate; exit : Pseudostate	2 relationship ends – R_a^r	none
34	Connector	StructuredClassifiers	contract : Behavior; kind : ConnectorKind; type : Association	2 relationship ends – R_a^r + 1 enumeration value – I	none
35	ConnectorEnd	StructuredClassifiers	definingEnd : Property; partWithPort : Property; role : ConnectableElement	3 relationship ends – R_a^r	none
36	ConsiderIgnoreFragment	Interactions	message : NamedElement	1 relationship end – R_a^r	none

37	Constraint	CommonStructure	constrainedElement : Element	1 relationship end – R_a^r	none
38	Continuation	Interactions	setting : Boolean	1 primitive value – l	none
39	CreateObjectAction	Actions	classifier : Classifier	1 relationship end – R_a^r	none
40	DecisionNode	Activities	decisionInput : Behavior; decisionInputFlow : ObjectFlow	2 relationship ends – R_a^r	none
41	DeploymentSpecification	Deployments	deploymentLocation : String; executionLocation : String	2 primitive values – l	none
42	DeploymentTarget	Deployments	deployedElement : PackageableElement	1 relationship end – R_a^r	none
43	DestroyObjectAction	Actions	isDestroyLinks : Boolean; isDestroyOwnedObjects : Boolean	2 primitive values – l	none
44	Duration	Values	observation : Observation	1 relationship end – R_a^r	none
45	DurationConstraint	Values	firstEvent : Boolean	1 primitive value – l	none
46	DurationObservation	Values	event : NamedElement; firstEvent : Boolean	1 relationship end – R_a^r + 1 primitive value – l	none
47	Element	CommonStructure	ownedElement : Element; owner : Element	2 relationship ends – R_a^r + of which 1 composite – S_p^w	none
48	ElementImport	CommonStructure	alias : String; visibility : VisibilityKind	1 primitive value – l + 1 enumeration value – l	none
49	ExceptionHandler	Activities	exceptionInput : ObjectNode; exceptionType : Classifier; handlerBody : ExecutableNode	3 relationship ends – R_a^r	none
50	ExecutionOccurrenceSpecification	Interactions	execution : ExecutionSpecification	1 relationship end – R_a^r	none

51	ExecutionSpecification	Interactions	finish : OccurrenceSpecification; start : OccurrenceSpecification	2 relationship ends – R_a^r	none
52	ExpansionNode	Actions	regionAsInput : ExpansionRegion; regionAsOutput : ExpansionRegion	2 relationship ends – R_a^r	none
53	ExpansionRegion	Actions	inputElement : ExpansionNode; mode : ExpansionKind; outputElement : ExpansionNode	2 relationship ends – $R_a^r + 1$ enumeration value – I	none
54	Expression	Values	symbol : String	1 primitive value – I	none
55	Extend	UseCases	extensionLocation : ExtensionPoint	1 relationship end – R_a^r	none
56	Extension	Packages	isRequired : Boolean; metaclass : Class	1 relationship end – $R_a^r + 1$ primitive value – I	none
57	Feature	Classification	isStatic : Boolean	1 primitive value – I	none
58	GeneralOrdering	Interactions	after : OccurrenceSpecification; before : OccurrenceSpecification	2 relationship ends – R_a^r	none
59	Generalization	Classification	generalizationSet : GeneralizationSet; isSubstitutable : Boolean	1 relationship end – $R_a^r + 1$ primitive value – I	none
60	GeneralizationSet	Classification	generalization : Generalization; isCovering : Boolean; isDisjoint : Boolean; powertype : Classifier	2 relationship ends – $R_a^r + 2$ primitive values – I	none
61	Image	Packages	content : String; format : String; location : String	3 primitive values – I	none
62	InformationFlow	InformationFlows	conveyed : Classifier; realization : Relationship; realizingActivityEdg	5 relationship ends – R_a^r	none

			e : ActivityEdge; realizingConnector : Connector; realizingMessage : Message		
63	InformationItem	InformationFlows	represented : Classifier	1 relationship end – R_a^r	none
64	InstanceSpecification	Classification	classifier : Classifier	1 relationship end – R_a^r	none
65	InstanceValue	Classification	instance : InstanceSpecifi- cation	1 relationship end – R_a^r	none
66	InteractionFragment	Interactions	covered : Lifeline	1 relationship end – R_a^r	none
67	InteractionUse	Interactions	refersTo : Interaction; returnValueRecipien- t : Property	2 relationship ends – R_a^r	none
68	InterruptibleActivityRegi- on	Activities	interruptingEdge : ActivityEdge	1 relationship end – R_a^r	none
69	Interval	Values	max : ValueSpecification; min : ValueSpecification	2 relationship ends – R_a^r	none
70	InvocationAction	Actions	onPort : Port	1 relationship end – R_a^r	none
71	JoinNode	Activities	isCombineDuplicate : Boolean	1 primitive value – l	none
72	Lifeline	Interactions	coveredBy : InteractionFragmen- t; decomposedAs : PartDecomposition; represents : ConnectableElemen- t	3 relationship ends – R_a^r	none
73	LinkEndCreationData	Actions	insertAt : InputPin; isReplaceAll : Boolean	1 relationship end – R_a^r + 1 primitive value – l	none
74	LinkEndData	Actions	end : Property; value : InputPin	2 relationship ends – R_a^r	none
75	LinkEndDestructionData	Actions	destroyAt : InputPin; isDestroyDuplicates : Boolean	1 relationship end – R_a^r + 1 primitive value – l	none

76	LiteralBoolean	Values	value : Boolean	1 primitive value – 	none
77	LiteralInteger	Values	value : Integer	1 primitive value – 	none
78	LiteralReal	Values	value : Real	1 primitive value – 	none
79	LiteralString	Values	value : String	1 primitive value – 	none
80	LiteralUnlimitedNatural	Values	value : UnlimitedNatural	1 primitive value – 	none
81	LoopNode	Actions	bodyOutput : OutputPin; bodyPart : ExecutableNode; decider : OutputPin; isTestedFirst : Boolean; setupPart : ExecutableNode; test : ExecutableNode	5 relationship ends – $R_a^r + 1$ primitive value –	none
82	Message	Interactions	connector : Connector; messageKind : MessageKind; messageSort : MessageSort; signature : NamedElement	2 relationship ends – $R_a^r + 2$ enumeration values –	none
83	MessageEnd	Interactions	message : Message	1 relationship end – R_a^r	none
84	Model	Packages	viewpoint : String	1 primitive value – 	none
85	MultiplicityElement	CommonStructure	isOrdered : Boolean; isUnique : Boolean; lower : Integer; upper : UnlimitedNatural	4 primitive values – 	none
86	NamedElement	CommonStructure	name : String; qualifiedName : String; visibility : VisibilityKind	2 primitive values – $ + 1$ enumeration value – 	none
87	Namespace	CommonStructure	member : NamedElement	1 relationship end – R_a^r	none
88	ObjectFlow	Activities	isMulticast : Boolean; isMultireceive : Boolean; selection :	2 relationship ends – $R_a^r + 2$ primitive values – 	none

			Behavior; transformation : Behavior		
89	ObjectNode	Activities	inState : State; isControlType : Boolean; ordering : ObjectNodeOrderin gKind; selection : Behavior	2 relationship ends – $R_a^r + 1$ primitive value – l + 1 enumeration value – l	none
90	OccurrenceSpecification	Interactions	toAfter : GeneralOrdering; toBefore : GeneralOrdering	2 relationship ends – R_a^r	none
91	OpaqueAction	Actions	body : String; language : String	2 primitive values – l	none
92	OpaqueBehavior	CommonBehavior	body : String; language : String	2 primitive values – l	none
93	OpaqueExpression	Values	behavior : Behavior; body : String; language : String; result : Parameter	2 relationship ends – $R_a^r + 2$ primitive values – l	none
94	Operation	Classification	isOrdered : Boolean; isQuery : Boolean; isUnique : Boolean; lower : Integer; type : Type; upper : UnlimitedNatural	1 relationship end – $R_a^r + 5$ primitive values – l	none
95	Package	Packages	URI : String	1 primitive value – l	none
96	PackageImport	CommonStructure	visibility : VisibilityKind	1 enumeration value – l	none
97	Parameter	Classification	default : String; direction : ParameterDirection Kind; effect : ParameterEffectKin d; isException : Boolean; isStream : Boolean; parameterSet : ParameterSet	1 relationship end – $R_a^r + 3$ primitive values – l + 2 enumeration values – l	none
98	ParameterSet	Classification	parameter : Parameter	1 relationship end – R_a^r	none
99	ParameterableElement	CommonStructure	templateParameter : TemplateParameter	1 relationship end – R_a^r	none

100	Pin	Actions	isControl : Boolean	1 primitive value – l	none
101	Port	StructuredClassifiers	isBehavior : Boolean; isConjugated : Boolean; isService : Boolean; protocol : ProtocolStateMachine; provided : Interface; required : Interface	3 relationship ends – $R_a^r + 3$ primitive values – l	none
102	ProfileApplication	Packages	isStrict : Boolean	1 primitive value – l	none
103	Property	Classification	aggregation : AggregationKind; isComposite : Boolean; isDerived : Boolean; isDerivedUnion : Boolean; isID : Boolean; opposite : Property; subsettingProperty : Property	2 relationship ends – $R_a^r + 4$ primitive values – l + 1 enumeration value – l	none
104	ProtocolTransition	StateMachines	referred : Operation	1 relationship end – R_a^r	none
105	Pseudostate	StateMachines	kind : PseudostateKind	1 enumeration value – l	none
106	QualifierValue	Actions	qualifier : Property; value : InputPin	2 relationship ends – R_a^r	none
107	ReadExtentAction	Actions	classifier : Classifier	1 relationship end – R_a^r	none
108	ReadIsClassifiedObjectAction	Actions	classifier : Classifier; isDirect : Boolean	1 relationship end – $R_a^r + 1$ primitive value – l	none
109	ReadLinkObjectEndAction	Actions	end : Property	1 relationship end – R_a^r	none
110	ReadLinkObjectEndQualifierAction	Actions	qualifier : Property	1 relationship end – R_a^r	none
111	Reception	SimpleClassifiers	signal : Signal	1 relationship end – R_a^r	none
112	ReclassifyObjectAction	Actions	isReplaceAll : Boolean; newClassifier : Classifier;	2 relationship ends – $R_a^r + 1$ primitive value – l	none

			oldClassifier : Classifier		
113	RedefinableElement	Classification	isLeaf : Boolean; redefinedElement : RedefinableElement ; redefinitionContext : Classifier	2 relationship ends – $R_a^r + 1$ primitive value – l	none
114	ReduceAction	Actions	isOrdered : Boolean; reducer : Behavior	1 relationship end – $R_a^r + 1$ primitive value – l	none
115	Relationship	CommonStructure	relatedElement : Element	1 relationship end – R_a^r	none
116	RemoveStructuralFeatureValueAction	Actions	isRemoveDuplicates : Boolean	1 primitive value – l	none
117	RemoveVariableValueAction	Actions	isRemoveDuplicates : Boolean	1 primitive value – l	none
118	ReplyAction	Actions	replyToCall : Trigger	1 relationship end – R_a^r	none
119	SendSignalAction	Actions	signal : Signal	1 relationship end – R_a^r	none
120	SignalEvent	CommonBehavior	signal : Signal	1 relationship end – R_a^r	none
121	Slot	Classification	definingFeature : StructuralFeature	1 relationship end – R_a^r	none
122	State	StateMachines	isComposite : Boolean; isOrthogonal : Boolean; isSimple : Boolean; isSubmachineState : Boolean; submachine : StateMachine	1 relationship end – $R_a^r + 4$ primitive values – l	none
123	StateMachine	StateMachines	submachineState : State	1 relationship end – R_a^r	none
124	Stereotype	Packages	profile : Profile	1 relationship end – R_a^r	none
125	StructuralFeature	Classification	isReadOnly : Boolean	1 primitive value – l	none
126	StructuralFeatureAction	Actions	structuralFeature : StructuralFeature	1 relationship end – R_a^r	none
127	StructuredActivityNode	Actions	mustIsolate : Boolean	1 primitive value – l	none

128	StructuredClassifier	StructuredClassifiers	part : Property	1 relationship end – R_a^r	none
129	TemplateParameter	CommonStructure	default : ParameterableElement; parameteredElement : ParameterableElement	2 relationship ends – R_a^r	none
130	TemplateParameterSubstitution	CommonStructure	actual : ParameterableElement; formal : TemplateParameter	2 relationship ends – R_a^r	none
131	TemplateSignature	CommonStructure	parameter : TemplateParameter	1 relationship end – R_a^r	none
132	TimeConstraint	Values	firstEvent : Boolean	1 primitive value – I	none
133	TimeEvent	CommonBehavior	isRelative : Boolean	1 primitive value – I	none
134	TimeExpression	Values	observation : Observation	1 relationship end – R_a^r	none
135	TimeObservation	Values	event : NamedElement; firstEvent : Boolean	1 relationship end – R_a^r + 1 primitive value – I	none
136	Transition	StateMachines	kind : TransitionKind; source : Vertex; target : Vertex	2 relationship ends – R_a^r + 1 enumeration value – I	none
137	Trigger	CommonBehavior	event : Event; port : Port	2 relationship ends – R_a^r	none
138	TypedElement	CommonStructure	type : Type	1 relationship end – R_a^r	none
139	UnmarshallAction	Actions	unmarshallType : Classifier	1 relationship end – R_a^r	none
140	UseCase	UseCases	subject : Classifier	1 relationship end – R_a^r	none
141	VariableAction	Actions	variable : Variable	1 relationship end – R_a^r	none
142	Vertex	StateMachines	incoming : Transition; outgoing : Transition	2 relationship ends – R_a^r	none

Appendix B. The Metaclasses That Normalize

The 100 metaclasses of UML.xmi that do not enter the candidate set, with the reason in each case. A metaclass normalizes into its ancestors when every attribute or association end it owns either redefines or subsets an inherited property, or when it owns none at all. Neither move introduces organization the ancestors did not already have; both narrow what is already present. This table is given so that the exclusion of these 100 can be checked rather than taken on trust.

#	Metaclass	Package	Owned features, all inherited-constraining	Reason
1	Abstraction	CommonStructure	1	every owned feature redefines or subsets an inherited property
2	AcceptCallAction	Actions	1	every owned feature redefines or subsets an inherited property
3	ActionInputPin	Actions	1	every owned feature redefines or subsets an inherited property
4	ActivityFinalNode	Activities	0	owns no attribute or association end
5	Actor	UseCases	0	owns no attribute or association end
6	AnyReceiveEvent	CommonBehavior	0	owns no attribute or association end
7	AssociationClass	StructuredClassifiers	0	owns no attribute or association end
8	BehavoredClassifier	SimpleClassifiers	3	every owned feature redefines or subsets an inherited property
9	CentralBufferNode	Activities	0	owns no attribute or association end
10	ChangeEvent	CommonBehavior	1	every owned feature redefines or subsets an inherited property
11	ClearStructuralFeatureAction	Actions	1	every owned feature redefines or subsets an inherited property
12	ClearVariableAction	Actions	0	owns no attribute or association end
13	Collaboration	StructuredClassifiers	1	every owned feature redefines or subsets an inherited property
14	CommunicationPath	Deployments	0	owns no attribute or association end
15	ComponentRealization	StructuredClassifiers	2	every owned feature redefines or subsets an inherited property
16	ConnectableElementTemplateParameter	StructuredClassifiers	1	every owned feature redefines or subsets an inherited property

17	ControlFlow	Activities	0	owns no attribute or association end
18	ControlNode	Activities	0	owns no attribute or association end
19	CreateLinkAction	Actions	1	every owned feature redefines or subsets an inherited property
20	CreateLinkObjectAction	Actions	1	every owned feature redefines or subsets an inherited property
21	DataStoreNode	Activities	0	owns no attribute or association end
22	DataType	SimpleClassifiers	2	every owned feature redefines or subsets an inherited property
23	Dependency	CommonStructure	2	every owned feature redefines or subsets an inherited property
24	DeployedArtifact	Deployments	0	owns no attribute or association end
25	Deployment	Deployments	3	every owned feature redefines or subsets an inherited property
26	DestroyLinkAction	Actions	1	every owned feature redefines or subsets an inherited property
27	DestructionOccurrenceSpecification	Interactions	0	owns no attribute or association end
28	Device	Deployments	0	owns no attribute or association end
29	DirectedRelationship	CommonStructure	2	every owned feature redefines or subsets an inherited property
30	DurationInterval	Values	2	every owned feature redefines or subsets an inherited property
31	EncapsulatedClassifier	StructuredClassifiers	1	every owned feature redefines or subsets an inherited property
32	Enumeration	SimpleClassifiers	1	every owned feature redefines or subsets an inherited property
33	EnumerationLiteral	SimpleClassifiers	2	every owned feature redefines or subsets an inherited property
34	Event	CommonBehavior	0	owns no attribute or association end
35	ExecutableNode	Activities	1	every owned feature redefines or subsets an inherited property
36	ExecutionEnvironment	Deployments	0	owns no attribute or association end
37	ExtensionEnd	Packages	2	every owned feature redefines or subsets an inherited property

38	ExtensionPoint	UseCases	1	every owned feature redefines or subsets an inherited property
39	FinalNode	Activities	0	owns no attribute or association end
40	FinalState	StateMachines	0	owns no attribute or association end
41	FlowFinalNode	Activities	0	owns no attribute or association end
42	ForkNode	Activities	0	owns no attribute or association end
43	FunctionBehavior	CommonBehavior	0	owns no attribute or association end
44	Gate	Interactions	0	owns no attribute or association end
45	Include	UseCases	2	every owned feature redefines or subsets an inherited property
46	InitialNode	Activities	0	owns no attribute or association end
47	InputPin	Actions	0	owns no attribute or association end
48	Interaction	Interactions	5	every owned feature redefines or subsets an inherited property
49	InteractionConstraint	Interactions	2	every owned feature redefines or subsets an inherited property
50	InteractionOperand	Interactions	2	every owned feature redefines or subsets an inherited property
51	Interface	SimpleClassifiers	6	every owned feature redefines or subsets an inherited property
52	InterfaceRealization	SimpleClassifiers	2	every owned feature redefines or subsets an inherited property
53	IntervalConstraint	Values	1	every owned feature redefines or subsets an inherited property
54	LinkAction	Actions	2	every owned feature redefines or subsets an inherited property
55	LiteralNull	Values	0	owns no attribute or association end
56	LiteralSpecification	Values	0	owns no attribute or association end
57	Manifestation	Deployments	1	every owned feature redefines or subsets an inherited property
58	MergeNode	Activities	0	owns no attribute or association end
59	MessageEvent	CommonBehavior	0	owns no attribute or association end
60	MessageOccurrenceSpecification	Interactions	0	owns no attribute or association end

61	Node	Deployments	1	every owned feature redefines or subsets an inherited property
62	Observation	Values	0	owns no attribute or association end
63	OperationTemplateParameter	Classification	1	every owned feature redefines or subsets an inherited property
64	OutputPin	Actions	0	owns no attribute or association end
65	PackageMerge	Packages	2	every owned feature redefines or subsets an inherited property
66	PackageableElement	CommonStructure	1	every owned feature redefines or subsets an inherited property
67	PartDecomposition	Interactions	0	owns no attribute or association end
68	PrimitiveType	SimpleClassifiers	0	owns no attribute or association end
69	Profile	Packages	2	every owned feature redefines or subsets an inherited property
70	ProtocolConformance	StateMachines	2	every owned feature redefines or subsets an inherited property
71	ProtocolStateMachine	StateMachines	1	every owned feature redefines or subsets an inherited property
72	RaiseExceptionAction	Actions	1	every owned feature redefines or subsets an inherited property
73	ReadLinkAction	Actions	1	every owned feature redefines or subsets an inherited property
74	ReadSelfAction	Actions	1	every owned feature redefines or subsets an inherited property
75	ReadStructuralFeatureAction	Actions	1	every owned feature redefines or subsets an inherited property
76	ReadVariableAction	Actions	1	every owned feature redefines or subsets an inherited property
77	Realization	CommonStructure	0	owns no attribute or association end
78	RedefinableTemplateSignature	Classification	3	every owned feature redefines or subsets an inherited property
79	Region	StateMachines	6	every owned feature redefines or subsets an inherited property
80	SendObjectAction	Actions	2	every owned feature redefines or subsets an inherited property
81	SequenceNode	Actions	1	every owned feature redefines or subsets an inherited property

82	Signal	SimpleClassifiers	1	every owned feature redefines or subsets an inherited property
83	StartClassifierBehaviorAction	Actions	1	every owned feature redefines or subsets an inherited property
84	StartObjectBehaviorAction	Actions	1	every owned feature redefines or subsets an inherited property
85	StateInvariant	Interactions	2	every owned feature redefines or subsets an inherited property
86	StringExpression	Values	2	every owned feature redefines or subsets an inherited property
87	Substitution	Classification	2	every owned feature redefines or subsets an inherited property
88	TemplateBinding	CommonStructure	3	every owned feature redefines or subsets an inherited property
89	TemplateableElement	CommonStructure	2	every owned feature redefines or subsets an inherited property
90	TestIdentityAction	Actions	3	every owned feature redefines or subsets an inherited property
91	TimeInterval	Values	2	every owned feature redefines or subsets an inherited property
92	Type	CommonStructure	1	every owned feature redefines or subsets an inherited property
93	Usage	CommonStructure	0	owns no attribute or association end
94	ValuePin	Actions	1	every owned feature redefines or subsets an inherited property
95	ValueSpecification	Values	0	owns no attribute or association end
96	ValueSpecificationAction	Actions	2	every owned feature redefines or subsets an inherited property
97	Variable	Activities	2	every owned feature redefines or subsets an inherited property
98	WriteLinkAction	Actions	0	owns no attribute or association end
99	WriteStructuralFeatureAction	Actions	2	every owned feature redefines or subsets an inherited property
100	WriteVariableAction	Actions	1	every owned feature redefines or subsets an inherited property