An Advanced Meta-Meta Model for Visual Language Design and Tooling

Targeting a Property Graph Implementation

Graham McLeod[*,a]

[a] Inspired.org | Univ. Duisberg-Essen

Abstract. *Visual Languages are widely employed in Enterprise Modelling. Our broader research aims to improve visual language design and facilitate rapid adaptation for specific purposes and audiences with a view to improving Return on Modelling Effort (ROME). Most current repositories and tools have hard coded support for notations and meta model concepts of target languages. A small number of tools have facilities to adapt meta models and notations, but these generally require high technical skills. The model described allows definition of arbitrary meta models and supporting notations in a relatively small meta meta model that can be practically implemented economically using property graph concepts. The model supports advanced concepts, including multi-level modelling. These capabilities facilitate tooling which supports rapid visual language definition, iterative improvement and run-time adaptation for purpose or audience.*

Keywords. Meta-Meta Model • Visual Language • Modelling Tools • Return on Modelling Effort

1 Introduction

Visual Languages are widely used in Enterprise Modelling to understand, communicate and design various aspects of (*inter alia*) organisations, capabilities, functions, services, processes, operating models, motivations and information requirements e.g.Multi-Perspective Enterprise Modeling (MEMO) [Frank 2002], Archimate [Various 2019], Business Process Modelling and Notation (BPMN) [Rosing et al. 2015]. They are also widely used in Information Systems to understand, communicate and design system context, process, data, user interaction, system design and other aspects e.g. Unified Modelling Language (UML) [Rayner et al. 2005], Entity Relationship Diagrams (ERD) [Chen 1976] and Data Flow Diagrams (DFD) [Ward 1986].

Prior research has identified that visual languages are often not optimal in terms of their ability to accurately convey information in the best form for the target stakeholder and purpose [Moody 2009]. Ware [Ware 2010] shows how poor most artifacts are at leveraging the human visual system. Bertin [Bertin 1983] highlights the limits of encoding information onto notations. Problems can include:

P1: Mismatch between concepts offered and modelling requirements. This leads users of languages to improvise and use provided concepts and symbols for other meaning and purposes, thereby harming communication and destroying standardisation

P2: Meta Models which do not support multi-level modelling required for advanced / accurate modelling, leading to models which do not properly capture the real world subtleties. This, in turn, leads to inflexible or frustrating systems that expect the world to conform to them [Frank 2014], [Clark and Willans 2014].

P3: Notations which are ill-suited to the orientation and experience of the stakeholders.

* Corresponding author.
E-mail. graham@inspired.org

Business personnel are confused by highly technical notations. Some models may lack the precision to properly capture important details for design or implementation. Notations may be difficult to use in facilitated sessions where they would be most effective from a group dynamic and consensus perspective. [Puhlmann 2019]

P4: Models which do not make important information easy to perceive. We term these "camouflage models" - a great deal of information is presented but insufficiently distinguished to show what is important [BIAN.org 2022]

P5: Notations which do not properly exploit human visual and cognitive abilities. Visual cues, colour, edges and other mechanisms are vital in creating an effective notation, but these are often ignored by language designers who use whatever is on the stencil or in the tool palette. [Bertin 1983; Moody 2009]

To address these shortcomings, we are engaged in research which includes the following Goals:

G1: Support rich meta modelling / ontology definition which allows fully expressing concepts of the domain accurately

G2: Support definition of notations which are appropriate to the domain concept, the stakeholders who will work with them and the purpose of the modelling

G3: Support rapid/iterative evolution of the meta model and visual language to continually improve effectiveness

G4: Support run time tailoring of models, meta model, visual language and tool user interface to support unique requirements ("moldable tools", in the spirit of: [Chiş et al. 2015])

G5: Permit multiple representations for the same semantic models, addressing the needs of different stakeholders and catering for multiple visual languages / methods

G6: As far as possible, achieve an economical implementation of the above capabilities to facilitate implementation of supporting tooling at reasonable effort / cost.

The remainder of this paper discusses requirements to be met to address the goals listed, sources of information and ideas from prior work, limitations of some current implementations and the resultant meta-meta model which excludes user model management and technology adaptation elements (for brevity). It concludes with remarks on remaining challenges and future plans.

2 Broad Requirements

Achieving the goals above requires that we understand who the stakeholders are, what their concerns are, what questions they want to answer and what the relevant concepts are in their domain. The latter should be expressed in a domain semantic model or ontology which includes the definition of concepts, properties which describe instances of these, and relationships which are legal between instances.

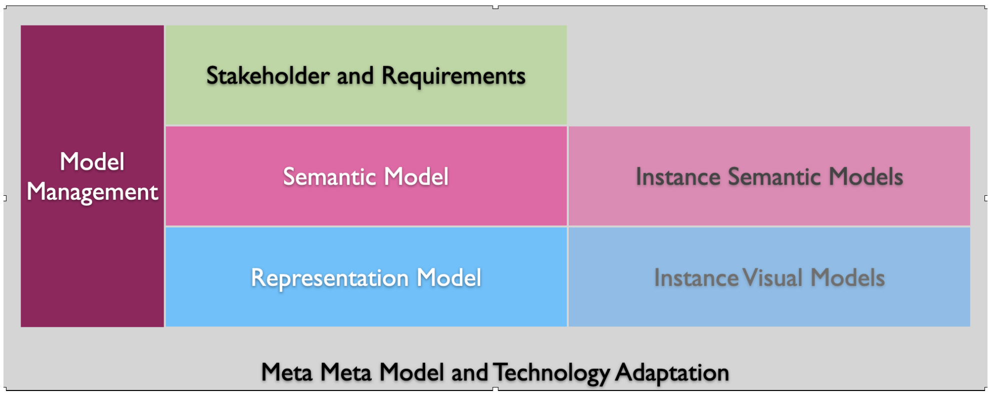
We also need a model that describes the visual language (aka "concrete syntax") which will be used to represent the concepts and relationships in a form that is most useful to particular stakeholders. This should include the model types, symbols and notation conventions that will apply and how these relate to the semantic model. Note that multiple representations should be possible for the same semantic model to meet the needs of different stakeholders or purposes. The meta meta model components are illustrated in Figure 1.

3 Detailed Requirements

More detailed requirements should address the stated shortcomings of current approaches as well as facilitating achievement of the Goals enumerated. These include:

R1: Multi-level modelling must be accommodated to support advanced modelling and to economise on tool size by allowing similar

Figure 1: Meta Model Components



tools to be used on multiple levels as well as support the runtime extension of the domain model/ontology. See [Lara et al. 2014] for a discussion of why and how to use multi-level approach

R2: High level of abstraction to achieve efficiency in model type definitions and agility in changing them when required. This implies a declarative versus procedural solution and prefers configuration over code. [Däcker and Williams 1997, Hartmann and Both 2009] provide evidence for power of abstraction in software and modelling

R3: Support for n-ary relationships and relationship properties. These are necessary to support some types of modelling e.g. [Chen 1976]

R4: Cater for rich data types, provided by the implementation environment, tool classes developed in the implementation language and structures created by users themselves through model definition. We have found this invaluable in our earlier work in the implementation of the EVA Toolset [Inspired.org 2022]

R5: Allow extension of the meta model and notation at run time

R6: Support a rich variety of diagram types and notations as well as facilitate other types of output (e.g. lists, reports, composed documents, matrices, graphs, visualisations)

R7: Support modelling the sequence and grouping required of items

R8: Support definition of validation, constraints, derivation through configurable rules/methods

R9: Provide for documentation of modelling language and evolution/versioning

R10: Support management of collections of things for retrieval in queries, reports and tooling

4 Relevant Sources and Prior Work

Tools and approaches were chosen for examination based upon team experience and their potential match to the stated requirements. In some cases they were used as a "sanity check" on developing concepts and to provide inspiration for how relevant challenges had been addressed by other authors and software engineers. Collectively they represent many decades of applied experience.

4.1 Enterprise Value Architect

The Enterprise Value Architect (EVA) repository and toolset [Inspired.org 2022] has evolved over two decades to address the needs of enterprise modelling, visual modelling and a variety of other requirements in support of strategy, enterprise architecture and information systems work. It has the ability to define a meta model through the web user interface or Graphical Modeller. The meta model is used to customise user interface patterns (at runtime) so that maintenance, navigation and reporting functions are immediately available [McLeod 2001].

Graphical notations include both raster images held as files and vector symbols held as definitions in the tool, in a Logo-like language [Solomon et al. 2020] embedded in XML[1] . Symbols are associated with a model type which links them to the types they represent in the domain, which are defined in the meta model.

Positive things in the architecture which we wish to emulate include:

EVA+1: Meta Model held as data in repository and user modifiable

EVA+2: Notation held as data/code in repository and user modifiable

EVA+3: Management of Model Types, Symbols, Models

EVA+4: Web user interface customised by meta model

EVA+5: Graphical Modeller editor customised by meta model, model type and symbol definitions

EVA+6: Rich data types allow quick definition of sophisticated structures for domain types

EVA+7: Relationships are named semantically (and in both directions), are first class objects and can have properties

EVA+8: Calculated properties allow derived values and inferencing in models

EVA+9: Ordering and Grouping of item display lists is configurable at Meta Level

[1] eXtensible Markup Language

EVA+10: Spatial layout of items within a generated model can be defined at meta level between types

EVA+11: Advanced features in the Graphical Modeler to allow rapid construction of models by dropping in relationships and items for focus object; also to hide selected types to automatically simplify models for display purposes

EVA+12: Many alternate representations are provided for sharing information, including: forms, reports, matrices, tables, graphs, composed documents, visualisations, rich pictures, canvasses and graphical models

EVA+13: Multi-user, SaaS[2] model, easy deployment

EVA+14: Repository, user, security and tool environment utilities

EVA+15: Product architecture works at meta level, which results in small code base which allows extension of capability through modelling, without coding

EVA+16: Pluggable architecture and use of data standards (XML, CSV[3] , JSON[4]) and standard protocols (HTML[5] , XML Messaging, REST[6]) allows easy integration with other tools

Limitations of the EVA architecture

EVA-1: Meta Model does not support generalise/specialise relationships fully

EVA-2: Relationship behaviour is not differentiated, except in tool logic (e.g. handling of hierarchies)

EVA-3: Default values held in an instance record

[2] Software as a Service
[3] Comma Separated Variable
[4] Javascript Object Notation
[5] Hypertext Markup Language
[6] Resource State Transfer

EVA-4: Meta Model does not support Role modelling

EVA-5: Not able to define multiple instances of the same relationship type between items

EVA-6: Constraints are not formally defined in the meta model (e.g. cardinalities)

EVA-7: Meta Model is not partitioned into languages or associated with stakeholder types (except through security model and menus). Domains do allow partitioning meta model but do not provide namespaces to prevent naming conflicts

EVA-8: compound user defined properties are not supported

EVA-9: Symbol definition language has limitations

EVA-10: Configuration of UI presentation is done via Meta Model and at instance level

EVA-11: Performance is limited by the mapping of logical layer to relational persistence

4.2 Eclipse Modelling Platform

The Eclipse environment [Eclipse Foundation 2022] is widely used in computer science and information system communities to implement tools for various visual languages. It is based upon the Eclipse Modeling Framework (EMF) which uses the Ecore meta model. The latter has a fairly close relationship to the Unified Modeling Language (UML) and the Java programming language. It is relatively easy to create tools in Eclipse which adhere to the philosophy of these paradigms. If modelling constructs are required that fall outside these, things become much more difficult and Java programming is generally required.

Positive aspects of the Eclipse Environment

E+1: Stable and well supported

E+2: Supports code editing for various languages and can be configured for DSLs

E+3: Open Source

E+4: Many plug ins and extensions available

E+5: Generated modellers can have good performance

E+6: Large community

Limitations of the Eclipse environment

E-1: Language implementation requires programming skills

E-2: Meta Model is based on EMOF[7] , which has many known limitations [Frank 2011, Clark 2020]

E-3: Tooling is desktop and requires Java runtime (Note: Eclipse Theia is a project to allow using Eclipse Framework to create web based tools, but these still rely on a Java back end server)

E-4: User Interface is crowded and unfriendly for non-technical users

E-5: Does not support creation of other user oriented output types (reports, documents, posters etc.)

E-6: Based upon a code generation approach and static bound language, therefor not runtime extensible

E-7: Models stored in text files rather than repository

4.3 Meta Edit+

MetaEdit+ is an advanced tool environment for building domain specific modelling language (DSML) tools [Kelly and Tolvanen 2021]. It comprises a Workbench for defining a domain model and associated visual language which generates a tool definition for the DSML. The latter drives a Modeller which provides the end user modelling environment.

[7] Essential Meta Object Facility

The tool is highly effective and is widely used in industry to support various kinds of DSML, for example in manufacture of cars and electronic goods. It has a graphical meta modelling language, dubbed GOPPRR[8] , designed specifically for the definition of domain models and graphical modelling languages. It is a DSM language for defining languages!

Positive Aspects of MetaEdit+

M+1: Purpose built and highly effective

M+2: Good meta modelling approach to defining visual language including graphical editor

M+3: Robust and industry proven

M+4: Multi user for modelling tool (not Workbench)

M+5: Model transformation facilities

M+6: API for integration

M+7: Well supported

Negative Aspects of MetaEdit+

M-1: Separate environments for language definition and model use

M-2: Generation paradigm inhibits rapid language evolution

M-3: Proprietary

M-4: Available on limited platforms

M-5: Multi-level modelling not supported

M-6: Models stored in proprietary object database

[8] Graph Object Property Port Role Relationship

4.4 XModeler

XModeler is an advanced and capable meta modelling platform targeted at development of information systems languages and Domain Specific Languages (DSLs). It has its own very compact meta meta model and language for defining meta models. It was developed primarily by Tony Clark [Clark and Willans 2014,Clark 2020].

Positive aspects of XModeler

X+1: Competent and extensible

X+2: Caters for multi-level modelling

X+3: Well defined meta model exploiting object technology

Negative aspects of Xmodeler

X-1: Proprietary Language for definition of models and behaviour

X-2: Limited user community and high learning curve

X-3: More targeted at definition of languages for execution than (esp) business level visual languages

X-4: Not multi-user or web oriented

4.5 Semantic Models

The Resource Description Framework (RDF) from World Wide Web Consortium (W3C) defined as part of the Semantic Web initiative, provides a deceptively simple language for describing semantic knowledge [W3C 2022]. RDF uses the concepts of "triples" which are facts stated in the form [Subject]->[Predicate]->[Object]. Examples would include:

```
John knows Alice
John salary 10000
Alice is a Person
```

A few things are important to observe in the preceding example. The *knows* in the first statement is a semantic relationship between two objects, John and Alice. The *salary* in the second statement points to a literal value of the salary for John. The *is a* in the third statement indicates that Alice belongs to the Person set or class of things. These simple statements therefor include:

1. Relationships between Objects
2. Properties for an Object
3. Values for Properties
4. Typing of an Object

RDF can be serialised in various textual notations for exchange between systems, transmission over networks and persistence in databases and files. RDF is a logical way of describing types and objects and facts about them. It does not, on its own, provide much in the way of type management or validation.

If we want to constrain relationships or values or set membership, then we need to use languages built on RDF such as DAML-OIL [McGuinness et al. 2002]. These provide schema facilities and ability to define constraints. To go further, we could use the Web Ontology Language (OWL) [Motik et al. 2009] which would also allow us to create rules and do inferencing. E.g. We could define a French alias for the concept of salary which would allow us to merge French and English data sets meaningfully.

RDF uses URIs[9] to identify facts, making them addressable on the Internet, thus supporting large distributed knowledge bases. RDF can be stored in semantic stores / triple stores which provide efficient access mechanisms including indexing, caching and memory management to manage large semantic knowledge bases. They are generally good at ingesting, connecting and querying large volumes of facts which are richly related. They are generally not as optimised for data which needs high integrity, ACID transactions and which may be updated frequently. ACID transactions imply Atomicity, Consistency, Integrity and Durability. Popular triple stores supporting RDF data include: Ontotext GraphDB, Allegrograph, Blazegraph and Stardog [Besta et al. 2019]

[9] Uniform Resource Identifiers

4.6 Property Graphs

Property graphs are similar conceptually to semantic graphs and RDF, but differ inthat items with identity (nodes) can have properties stored within the node. This reduces the number of objects in graphs and improves performance. It also allows easier implementation of transaction behaviour and facilitates efficient updates. Property graphs typically make use of internally generated item identities, or user provided ones. Property graph systems are gaining ground rapidly in production applications needing a blend of features normally found in relational systems, plus the ability to deal with rich relationships and large knowledge bases with many complex relationships. Popular systems include Neo4J, AllegroGraph and DGraph [Fernandes and Bernardino 2018].

5 Design Choices

5.1 User and Value First

We adopt an approach where stakeholders, concerns, goals and relevant concepts and model types to address these are taken as the first point of departure. We then apply features and capabilities of meta models and tools to address these in (first) an effective manner and then (second) an efficient manner.

5.2 Multiple Languages and Notations

- A single semantic model may be represented in multiple visual languages
- A single logical model may be represented in multiple visual languages
- We should also anticipate other output forms, including: Forms, Lists, Tables, Matrices, Reports, Composed Documents.

5.3 Rich Properties

These are properties beyond the typical programming language data types. Their handling requires several approaches:

- Composite Properties - e.g. Address. These can be defined as a Concept which can be re-used in the role of a PropertyType
- Logical Data Types: These can be defined in implementation of tooling as classes with special behaviour (e.g. compress, decompress, scale, render etc.). They should all permit serialisation to strings or streams for persistence and transmission
- Physical Properties will be handled by implementation language classes or tool defined classes NOTE: Compressed in meta models shown later to save space and limit complexity

5.4 Logical vs Physical Models

We explicitly model logical vs physical models. The former is a container for semantic information (types, relationships, properties, values) while the latter provides for visualisation (symbols, lines, other visual artefacts). This facilitates separation of concern and allows multiple physical representations of the same semantic information.

5.5 Relationships

1. Unlike the approach of Clark where relationships are held as properties on related objects, we follow a similar approach to Frank where relationships are first class citizens. This is to accommodate n-ary relationships, to support relationship properties and to map more cleanly to graph models. We also want to explicitly support semantic relationships and reusable behaviours for higher level abstractions of kinds of relationships.
2. n-ary support is enabled by having relationships which can reference multiple related items
3. Reusable Semantics - provide consistency in naming and translation to user language and text
4. Reusable Behaviours - provide extensible relationship types beyond the UML association, generalise/specialise, containment. E.g. Roles, Taxonomy For, Dependency, Sequence
5. Support relationship properties e.g. properties for the relationship *enroll* between Student and Course might include *Date* and *payment-Received*

5.6 Dynamic Meta Model

Frank prefers a static Meta Model while Clark advocates a dynamic one. We favour the latter, since we aim to provide moldable behaviours at runtime. Model elements can thus have behaviours (methods/functions). Currently these may be associated with any identity-carrying element in the system.

5.7 Multi-Level Modelling

The design must cater for multiple levels of definition and instances, without constraint. Some authors, e.g. Frank, have adopted a model which relies upon "potency", annotating attributes of types/classes to indicate at what level they should be instantiated. We find this restrictive and awkward to implement. We have adopted a meta circular approach more like that pioneered by Clark. We have adapted this further to leverage the concept of graphs and ideas from RDF and Property Graphs. This will be elaborated in the discussion of the meta meta model following.

5.8 Polymetric Diagramming

This is an approach introduced by Lanza [Lanza 2003] in providing rich visualizations of software systems. The idea is to use a graphical notation and modify aspects of the visual symbols based upon underlying properties of the objects represented. E.g. A box might represent a class. Its height may be calculated based upon the number of methods while the width may represent the number of instance variables. Colour might be determined by the number of other classes which rely upon it, etc. In this way visual cues enhance the value of models and their ability to rapidly identify things of interest to stakeholders. We aim to support polymetric models in the visual representation layer.

5.9 Moldable Tools

This is an approach introduced by Tudor Girba [Chiş et al. 2015] whereby features are provided in tools to create new views, visualisations and workflows at runtime to facilitate interactive and highly effective problem solving. We adopt several of these ideas. The longer term goal is for user interfaces themselves to be models which users can adjust.

5.10 Reuse of Properties

Emulating the approach in EVA we define named properties independently of their inclusion in definition of Concepts (types). This facilitates consistency in naming and semantics.

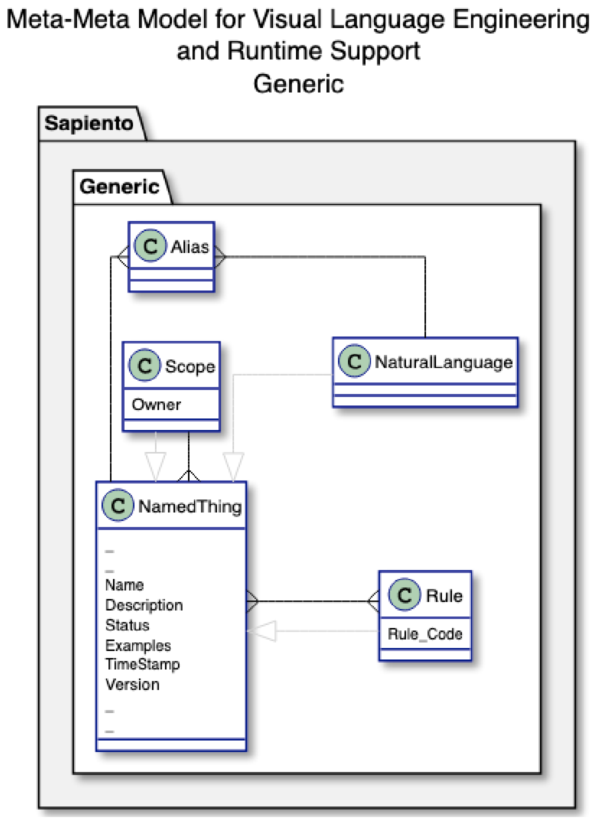
6 Meta Meta Model

6.1 Notation

The meta model is shown using a notation derived from the UML and additional options offered by PlantUML[Open_Source 2022]. Packages are shown as tabbed containers. Classes as multi compartment boxes and relationships as lines. Line styles are as follows:

- Subclass –|> Superclass (inheritance; specialised form to generic form)
- ClassA <>—< ClassB (ClassA contains instances of ClassB)
- ClassA - - -< ClassB (ClassB objects are logical instances of ClassA)
- ClassA []— ClassB (ClassA is a role of ClassB. Roles follow the approach of James Odell [Martin and Odell 1997].

Figure 2: Generic Fragment



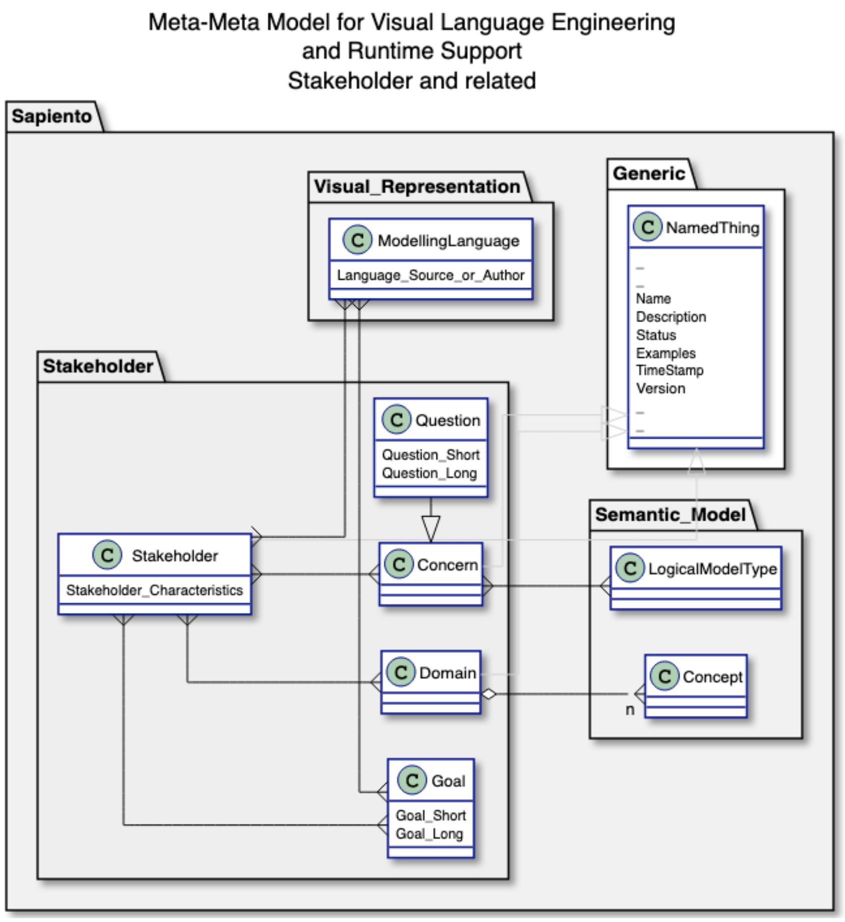
- Crows feet are used to indicate zero or many
- Omission of crows foot or a cardinality at a relationship end implies "1". A specified number (e.g. "n" meaning many, can override this)
- Inheritance lines to NamedThing have been shown in light gray to visually simplify the diagram.
- Some subclasses have been shown within the superclass to simplify the diagram and save space.

The full diagram has been broken into fragments, represented as Packages, to accommodate space constraints in the paper. We will discuss each fragment in turn. Each figure will show the package containing the fragment concepts and other concepts directly referenced in adjacent packages.

6.2 Generic Elements

This fragment contains a few abstractions and generically required items. Refer to Figure 2. NamedThing allows inheritance of identity, description, status, version, time stamp and (where applicable) example instances. Many classes in the meta model are derived from it. Alias and NaturalLanguage allow defining alternate names for most things in the system, including in other human languages. Scope is a mechanism to group things of interest (e.g. Domain of Concepts, Namespace of Objects, Package for export etc). Rule is a place to keep behaviour/methods related to any

Figure 3: Stakeholder Fragment



relevant named object. It allows implementing validation, constraints, derivation of values, layout algorithms etc.

6.3 Stakeholder and Purpose Fragment

This allows us to link types of stakeholders, their concerns and questions to domains, concepts and appropriate modelling languages and model types. Refer to Figure 3.

6.4 Semantic Fragment

This is the core of the meta meta model and holds the definitions of concepts, properties, relationships representing the semantics of domains. Refer to Figure 4. Given that we aim to support multi-level modelling, it must be capable of holding instances as well as definitional elements. The fundamental conceptual model is based upon a property graph with nodes, edges and targets. Nodes can be Concepts or Items. Edges link nodes. They have Targets which can be a Node, a PropertyValue or a PropertyType. There are several important innovations:

I1: Whether something is treated as a Concept or an Item depends upon whether it has instances or not. An Item can become a Concept by having Items as instancces or any edge whose target is a PropertyType

I2: Concepts can have values, which can be inherited by other Concepts. This is similar to the principles in Smalltalk Class Definition [Goldberg and Robson 1983]

I3: Concepts can access a collection of their Items. Unlike object oriented languages where *all instances* returns only those for the subtype, we return all instances of the selected type and any of its subtypes. This is more in line with ontology and set modelling. E.g. Returning *all instances* of Person would return all Persons including subclasses such as Professional and Pensioner

I4: PropertyTypes can be a development language class, an implementation class or a user defined Concept

I5: A dictionary is constructed of NamedProperty instances. This gives a semantic to a data element. E.g. the PropertyType may be Integer, but the named element could be Number-of-Employees. The NamedProperty can be associated with a Concept, where it becomes a LegalProperty. It will normally have the same name there, but may mean something different. E.g. Number-of-Employees may be used on Company and Department.

I6: Actual Items hold PropertyValue objects which contain the values of LegalProperties. To accommodate multi-level modelling, a PropertyValue may be a literal value or a PropertyType.

I7: RelationshipType represents an abstraction for different relationship behaviours, which will include *inter alia*: association, subtype->supertype, hierarchy, containment, role, instances, dependency, taxonomy. The latter allows using elements of one Concept to group instances of another: E.g. Employee could be grouped by Department instances

I8: SemanticRelationship gives a semantic to relationship types. E.g. a role RelationshipType may be used as a *can be used as* SemanticRelationship

I9: LegalRelationship applies a SemanticRelationship between related Concepts. At the instance level, it is instantiated as an actual Relationship between Items

6.5 Visual Representation Fragment

This is the layer in which we hold all information regarding the concrete syntax (graphical presentation). Consider Figure 5. PhysicalModelType defines a representation for a LogicalModelType. It has various subclasses (for space reasons contained within the class symbol). For graphical models (Diagram), it will map Concepts to NotationElements which are Vector or Raster Symbols. It is instantiated by rendering a LogicalModel according to the mapping, possibly with a layout

Figure 4: Semantic Fragment

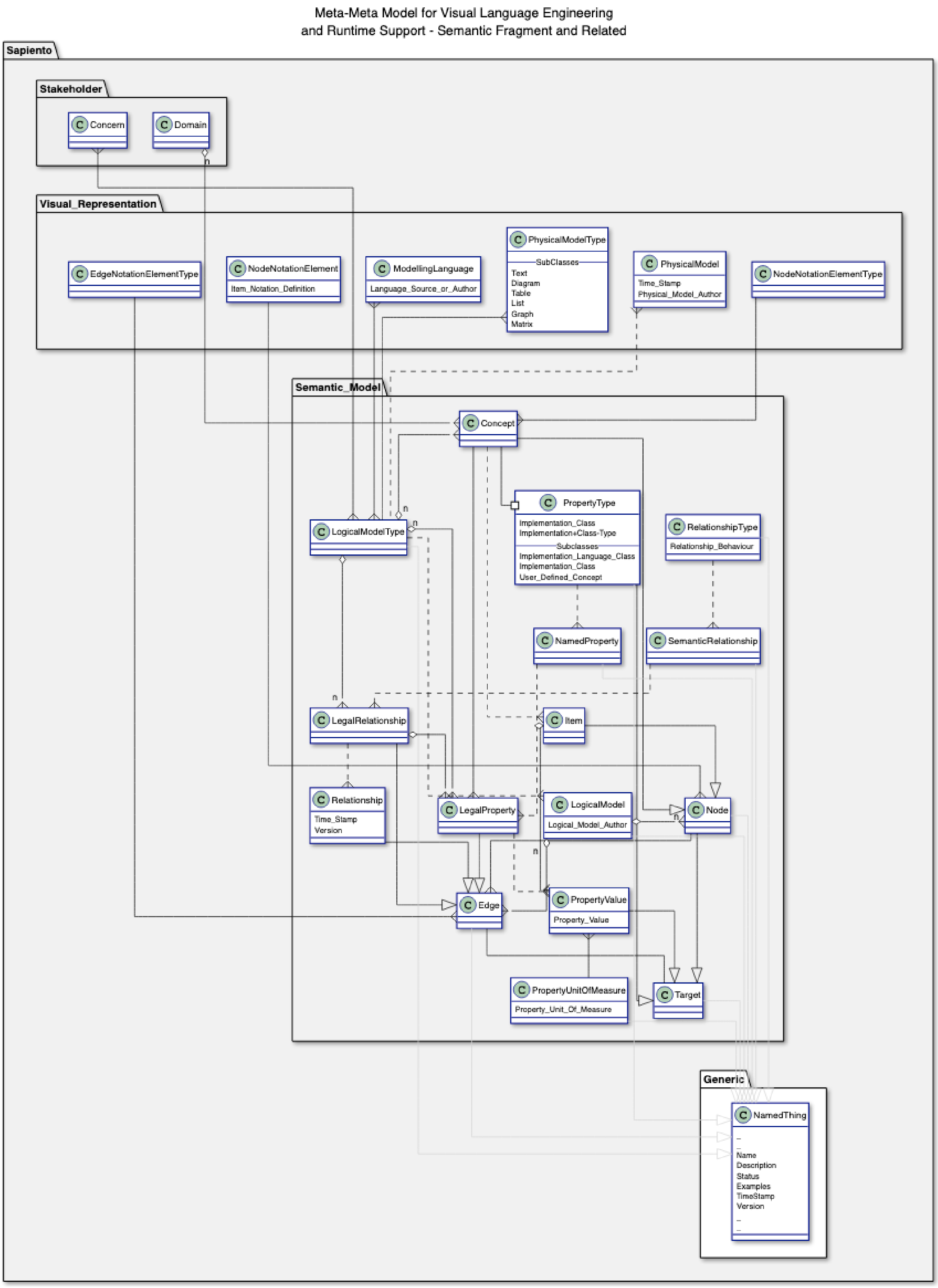
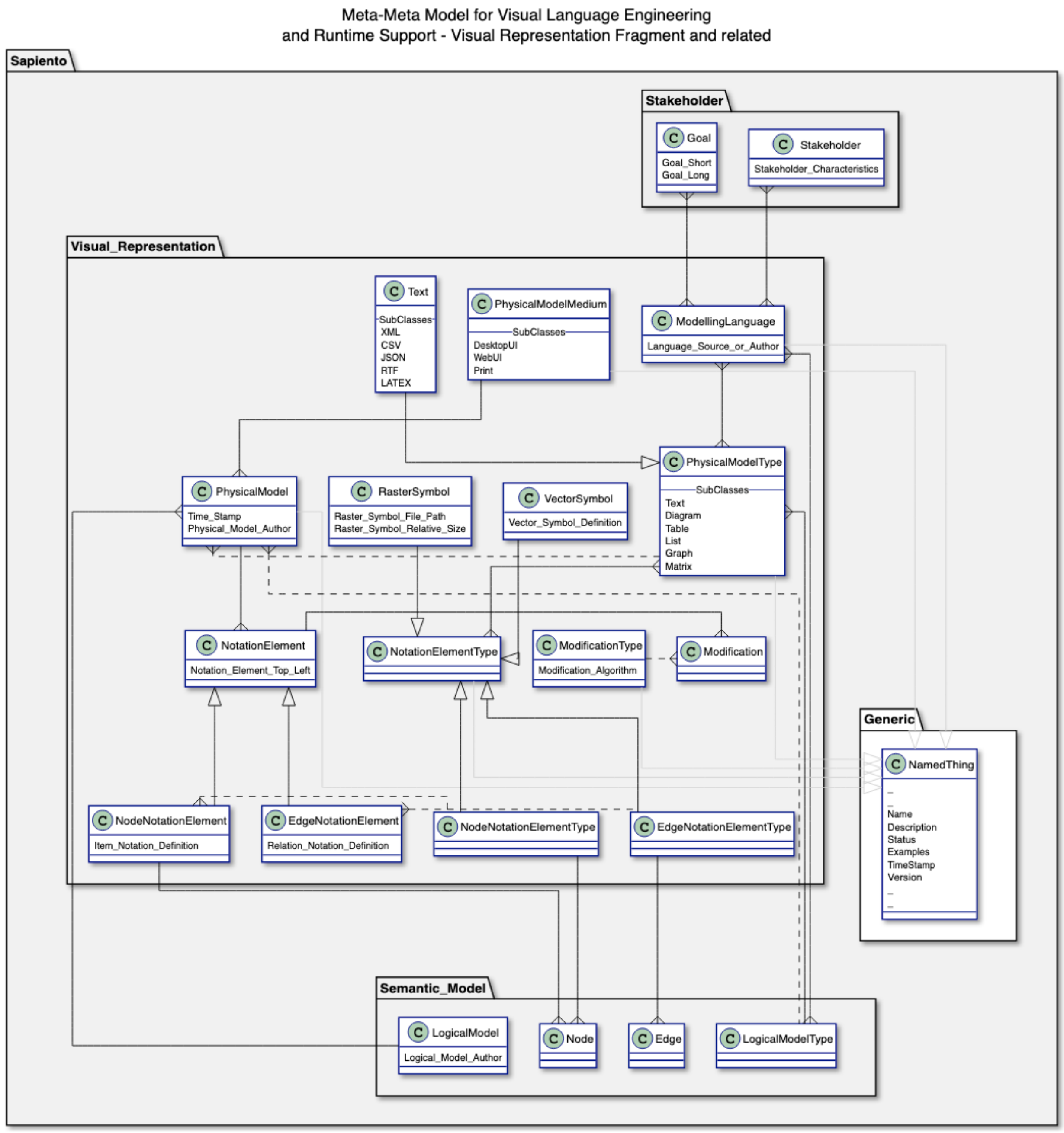


Figure 5: Visual Representation Fragment



specification, which would be a rule attached to the PhysicalModelType.

The PhysicalModel will hold actual symbols (including sizes, positions, layers) and edges (including path and vertices). At the implementation level, these details may be serialised to text (e.g. JSON or XML at the PhysicalModel level). Multiple PhysicalModels of the same PhysicalModelType can be present for a single LogicalModel. This can facilitate modifications at the instance level to highlight important features, modify layout etc.

7 Reflection

Various versions of the meta model have been implemented in prototypes and proof of concept tools during its gestation over the past three years. They show much promise and issues raised during these exercises have been addressed in the model presented. Earlier versions were based upon object technology and did not cater for multi-level modelling.

This is the first iteration to explicitly address this in a serious way, although it was achievable to a limited extent in earlier versions. We are currently engaged in early implementation with a Property Graph system and will refine the model based upon this experience. We are keen to hear of other researchers working on similar problems and collaborate where appropriate.

References

Bertin J. (1983) Semiology of graphics. University of Wisconsin press

Besta M., Peter E., Gerstenberger R., Fischer M., Podstawski M., Barthels C., Alonso G., Hoefler T. (2019) Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries. In: arXiv preprint arXiv:1910.09017

BIAN.org (2022) BIAN Banking Service Landscape 10 https://bian.org/servicelandscape-10-0-0/views/view_51974.html Last Access: 2022-05-02

Chen P. P.-S. (1976) The entity-relationship model—toward a unified view of data. In: ACM transactions on database systems (TODS) 1(1), pp. 9–36

Chiş A., Nierstrasz O., Gırba T. (2015) Towards moldable development tools. In: Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools, pp. 25–26

Clark T. (2020) A Meta-Circular Basis for Model-Based Language Engineering.. In: The Journal of Object Technology 19(3), 3:1

Clark T., Willans J. (2014) Software language engineering with XMF and XModeler. In: Computational Linguistics: Concepts, Methodologies, Tools, and Applications. IGI Global, pp. 866–896

Däcker B. O., Williams M. C. (1997) Breakthrough in software design productivity through the use of declarative programming. In: International journal of production economics 52(1-2), pp. 227–231

Eclipse Foundation. https://www.eclipse.org. Last Access: Accessed: 2022Q3

Fernandes D., Bernardino J. (2018) Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB.. In: Data, pp. 373–380

Frank U. (2002) Multi-perspective enterprise modeling (memo) conceptual framework and modeling languages. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences. IEEE, pp. 1258–1267

Frank U. (2011) The MEMO meta modelling language (MML) and language architecture.. ICB-research report

Frank U. (2014) Multilevel modeling. In: Business & Information Systems Engineering 6(6), pp. 319–337

Goldberg A., Robson D. (1983) Smalltalk-80: the language and its implementation. Addison-Wesley Longman Publishing Co., Inc.

Hartmann U., von Both P. (2009) A declarative approach to cross-domain model analysis. In: Managing It in Construction/Managing Construction for Tomorrow 26, pp. 45–51

Inspired.org (2022) Enterprise Value Architect https://www.inspired.org/eva-home Last Access: 2022-05-02

Kelly S., Tolvanen J.-P. (2021) Collaborative modelling and metamodelling with MetaEdit+. In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). IEEE, pp. 27–34

Lanza M. (2003) Object-Oriented Reverse Engineering Coarse-grained, Fine-grained, and Evolutionary Software Visualization. In:

Lara J. D., Guerra E., Cuadrado J. S. (2014) When and how to use multilevel modelling. In: ACM Transactions on Software Engineering and Methodology (TOSEM) 24(2), pp. 1–46

Martin J., Odell J. J. (1997) Object-oriented methods (UML ed.,) a foundation. Prentice-Hall, Inc.

McGuinness D. L., Fikes R., Hendler J., Stein L. A. (2002) DAML+ OIL: an ontology language for the Semantic Web. In: IEEE Intelligent Systems 17(5), pp. 72–80

McLeod G. (2001) PAMELA: A Proto-pattern for Rapidly Delivered, Runtime Extensible Systems. In: Evaluation of Modeling Methods for Systems Analysis and Design (EMMSAD) 1

Moody D. (2009) The “physics” of notations: toward a scientific basis for constructing visual notations in software engineering. In: IEEE Transactions on software engineering 35(6), pp. 756–779

Motik B., Patel-Schneider P. F., Parsia B., Bock C., Fokoue A., Haase P., Hoekstra R., Horrocks I., Ruttenberg A., Sattler U., et al. (2009) OWL 2 web ontology language: Structural specification and functional-style syntax. In: W3C recommendation 27(65), p. 159

Open_Source (2022) Drawing UML with PlantUML PlantUML Language Reference Guide https://plantuml.com/guide Last Access: 2022-05-02

Puhlmann F. (2019) BPMN 2.0 Wimmelbild Edition http://frapu.de/pdf/BPMN20-Wimmelbild.pdf Last Access: 2022-05-02

Rayner M., Hockey B. A., Chatzichrisafis N., Farrell K. (2005) OMG Unified Modeling Language Specification. In: Version 1.3, © 1999 Object Management Group, Inc

von Rosing M., White S., Cummins F., de Man H. (2015) Business Process Model and Notation-BPMN.

Solomon C., Harvey B., Kahn K., Lieberman H., Miller M. L., Minsky M., Papert A., Silverman B. (2020) History of logo. In: Proceedings of the ACM on Programming Languages 4(HOPL), pp. 1–66

Various (2019) Archimate 3.1 Specification. The Open Group Series. Van Haren Publishing https://books.google.co.za/books?id=kibNywEACAAJ

W3C (2022) Resource Description Framework (RDF) https://www.w3.org/RDF/ Last Access: 2022-05-02

Ward P. T. (1986) The transformation schema: An extension of the data flow diagram to represent control and timing. In: IEEE Transactions on Software Engineering (2), pp. 198–210

Ware C. (2010) Visual thinking for design. Elsevier