Model Sensemaking Strategies: Exploiting Meta-Model Patterns to Understand Large Models

Francisco Martínez-Lasaca
UGROUND
Madrid, Spain
francisco.martinezl01@estudiante.uam.es

Pablo Díez
UGROUND
Madrid, Spain
pdiez@uground.com

Esther Guerra
Universidad Autónoma de Madrid
Madrid, Spain
esther.guerra@uam.es

Juan de Lara
Universidad Autónoma de Madrid
Madrid, Spain
juan.delara@uam.es

***Abstract*—The increasing popularity of model-based and low-code platforms has raised the need to understand large models – especially in industrial settings. However, current approaches mainly rely on graph-based visual metaphors, which do not scale well with large model sizes. To address this issue, we introduce *model sensemaking strategies*: purposeful model visualisations based on alternative visual metaphors. We define them as reusable patterns that yield tailored visualisations when applied to meta-models. This paper presents a catalogue of domain-specific and domain-agnostic sensemaking strategies, and a recommender that suggests suitable strategies for a given meta-model. To showcase the framework's applicability, we have implemented some of these strategies in Dandelion, an industrial, low-code graphical language workbench for the cloud. Using this platform, we have evaluated the effectiveness of the strategies to visualise large industrial models by the UGROUND company.**

***Index Terms*—model sensemaking strategies, large model visualisation, model-driven engineering, low-code platforms**

I. Introduction

Large and complex models have become pervasive in many disciplines and domains, including embedded systems, process modelling, software design, and biology – to name a few. Understanding these models has become, therefore, a ubiquitous task, which calls for the development of specialised techniques. Given the limitations of human cognition [1] and the increasing model complexity [2], many approaches rely on graphical visualisations – typically under umbrella terms such as *graphs*, *diagrams*, *networks*, or *maps* [3]. When these visualisations incorporate domain-specific mechanisms to facilitate the user's understanding process, they can serve as a cost-effective means of carrying out *sensemaking tasks* [4].

The increasing popularity of model-based and low-code platforms has highlighted the need to visualise and understand large artefacts created with these platforms [5]. In these fields (and especially within industrial settings), producing effective visualisations is particularly challenging. Specifically, models can reach millions of elements, exhibit intricate relationships with each other, and contain numerous attributes. Therefore, naïve visualisations are prone to entail overwhelming graphical information, creating cognitive overload. This is especially sensitive in low-code tools, used by *citizen developers* with different backgrounds and limited modelling skills [6].

Graph-like visualisations – typically offered by modelling tools – may not suffice for model sensemaking tasks, since they may fail to reveal underlying information in the explored models. Instead, other visualisations such as charts, plots, maps, or matrices can better support sensemaking tasks. These may rely on summarisation techniques, using derived information by aggregating, filtering, or partitioning the elements within the current model, or sets of them (cf. Fig. 1). These alternative visualisations can be manually created by developers of modelling environments, but their construction is costly. Moreover, flexibility is required to cater to the specifics of particular domain-specific languages (DSLs).

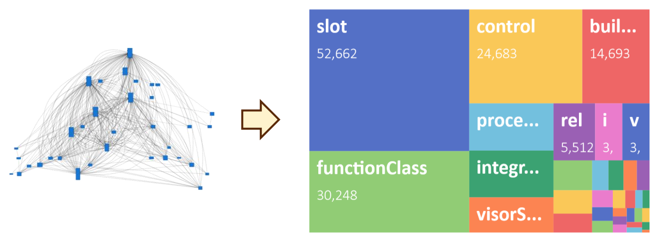


Fig. 1: Sensemaking strategies help unravel model complexity. Left: UGROUND's ROSE industrial meta-model with 35 meta-classes, 300+ edges, and 500+ attributes. Right: the same meta-model through the lens of a Categorical sensemaking strategy: each rectangle represents a meta-class whose area is proportional to its number of instances. This visualisation supports frequent model sensemaking tasks for language engineers: '*Which are the most relevant DSL concepts for the users?*', '*How is the DSL used in practice?*'

To overcome these issues, we propose the new notion of *model sensemaking strategy* (SMS), drawing from the theory of sensemaking [4], [7]. Model sensemaking strategies are aimed at accomplishing specific model understanding tasks, offering visualisation metaphors for it – like charts, plots, maps, graphs, or matrices (cf. Fig. 1). SMSs are *reusable*, since each strategy exposes a small meta-model pattern that, when bound to a target meta-model, results in a tailored visualisation for the target meta-model instances. SMSs are also *flexible*, as they can be used to understand (meta-)models at any level of abstraction, as well as entire modelling ecosystems. Moreover, SMSs can be organised on *dashboards* combining SMSs to perform multiple model understanding tasks simultaneously.

This paper presents the theoretical foundations of SMSs and an extensible catalogue of 10 strategies, which can be depicted using 20 exchangeable visualisations. We have also built a recommender that suggests suitable SMSs for a given meta-model automatically. Our approach is implemented atop Dandelion [8], a scalable, cloud-based graphical language workbench for industrial low-code development.

To demonstrate the approach's applicability, we have evaluated its effectiveness within the UGROUND company.[1] UGROUND uses low-code development within all its projects, in areas like banking, human resources, insurance, services, and transportation. UGROUND models can reach hundreds of thousands of elements and are defined using *ROSE* [9], a proprietary technology that interprets models to yield the final running system [10]. Our challenge was to develop a workbench to facilitate understanding of UGROUND models (to help developers of applications) and the ROSE language itself (to understand how the language is used and help in its future evolution). Our experience shows that Dandelion was able to facilitate the understanding of ROSE, the entire UGROUND modelling ecosystem, and particular models.

Paper structure. Section II analyses related work. Section III overviews the main components of SMSs, and Section IV details how they are applied. Section V presents a catalogue of sensemaking strategies. Section VI discusses the integration of SMSs into Dandelion. Section VII reports on the evaluation of the strategies on UGROUND models. Lastly, Section VIII ends with the conclusions and future work.

II. Related Work

Graphical modelling notations often lack visual scalability. In particular, the amount of information that can be presented is constrained due to human comprehension limits (large models are difficult to grasp [1]) and technology limitations (large models do not fit on the screen and take long to display [11]). Hence, techniques have been proposed to visualise, explore, and comprehend large models.

Next, we revise works on methods for handling large models (Section II-A), techniques to visualise and comprehend large artefacts (Section II-B), and, since our solution relies on reusable patterns, works on pattern-based reuse (Section II-C).

A. Handling large models

Traditionally, handling large models effectively has been one of the main challenges in Model-Driven Engineering (MDE) [2]. For example, part of the problem in the Eclipse Modeling Framework (EMF) ecosystem stems from its dependence on file-based model persistence (often, one model per file), and the lack of proper model modularisation primitives in Ecore [12]. To tackle this problem, some researchers propose model components, which can be composed via interfaces [13]. Additionally, solutions based on model partitioning have been devised to handle *existing* large models. These include model fragmentation according to user-defined strategies [14], [15]; partial model loading [16]; and model decomposition into smaller valid submodels [17]. Moreover, indexers [18] (similar to those in relational databases) and caching techniques for queries [19], [20] have been proposed for faster model element retrieval.

However, while all the previous works alleviate some of the scalability problems of modelling tools, they only target the abstract syntax of models and neglect their concrete syntax and visualisation aspects. Moreover, they do not provide concrete facilities for model comprehension tasks.

B. Visualising and comprehending large graphs and models

Most techniques to visualise, explore, and comprehend large models come from the field of graph visualisation. Large structured datasets are often represented as graphs, and numerous efforts target effective large-graph visualisations [21], [22]. Many of these efforts have been triggered by the field of *Visual Analytics*, which employs analytical reasoning facilitated by interactive visual interfaces [23], [24].

For example, [25] surveys techniques for visualising graph structures, including node colouring, contour drawing, adjacency matrices, and their embedding into graphs. In [26], techniques for visualising dynamic graphs are studied so that interaction techniques do not produce abrupt, disruptive changes in the user experience. FACETS proposes the exploration of large graphs driven by the most interesting neighbourhood of the current node [27].

Related to visual analytics, Pienta et al. [7] propose *graph sensemaking* as "*the iterative process of understanding and making sense out of graph-formatted data, where a user gradually builds up a representation of the information space to achieve the user's goal.*" They also provide a survey of the graph exploration and visualisation literature, and create a taxonomy of graph sensemaking techniques [4].

We have taken inspiration from this trend of works, adapting the idea to the 'modelware' technical space, and making SMSs reusable. Plain graphs have no explicit semantics – nodes and edges have no types or properties –, which enables aggressive simplifications. However, when applying SMSs to models, we need to consider the structure and semantics provided by their meta-models, the possibility of applying SMSs to meta-models themselves, and the analysis of whole modelling ecosystems.

The need to understand and monitor large *code* repositories has triggered the appearance of tools [28] offering dashboards to visualise aspects of the construction process (e.g., commits) and the code itself (e.g., LoC). Similarly, our SMSs can be grouped in dashboards, and are adaptable to different DSLs.

There is little literature on applications for the visualisation and understanding of large models. For instance, visualisation techniques based on semantic zoom [29] have been applied to UML diagrams [30], [31] and OWL ontologies [32] to improve their understanding. Layering has been used with UML diagrams [33]. Similarly, in [34], label management [35] and vertical message compaction are applied to reduce the height of sequence diagrams and make them fit on a screen. While useful, all these works target specific languages. In

[1] https://www.uground.com

contrast, language-independent catalogues of model abstractions have been proposed in [36] to simplify models for better comprehension, and in [11] to facilitate the exploration and processing of large models. Still, these abstractions do not consider the models' concrete syntax. Model views [37] can be defined to extract a relevant portion of the model via a query language. Still, the result would be a model, while SMSs can benefit from different visualisation metaphors.

More recently, a taxonomy of advanced visualisation techniques for conceptual modelling was proposed in [38] based on an analysis of 46 tools (like IntelliJ IDEA, Google Maps or PowerPoint). In most of them, visualisations were built ad-hoc and were not reusable. Finally, a few modelling tools have been designed to permit a flexible definition of concrete syntaxes. This is the case of JJodel [39], a web-based modelling environment supporting the definition of views reactive to variations on the model objects, in the style of Visual Basic. VizDSL is a DSL to define interactive data visualisations, which can be applied to render models [40] (i.e., define their concrete syntax), but does not target or directly support SMSs. Our approach, in contrast, focuses on strategies that are defined once and can be repeatedly reused for different DSLs.

C. Genericity and pattern-based reuse

Our SMSs become reusable by exposing a meta-model pattern, which can then be bound to target meta-models via a binding. This is a type of generic approach that other researchers have also exploited to define generic refactorings [41], generic model abstractions [36], generic model fragmentation strategies [42], or generic model transformations [43], [44]. Several tools, like Kermeta [45] or MetaDepth [46], support the definition of generic operations. All the approaches have in common that the generic operation exposes a small meta-model that has to be bound to make the operation reusable. We provide three contributions in this area: we can bind to the linguistic meta-model to define agnostic SMSs, applicable to any (meta-)model; our binding offers an expression language enabling gathering information from lower meta-levels; and we offer a recommender that automatically precomputes feasible bindings.

Generally, researchers have profited from pattern-based approaches to define model transformations [47], DSLs and services for them [48], and have been employed in specific domains like secure systems development [49] or dependability engineering [50]. However, to our knowledge, ours is the first application of patterns to define SMSs.

III. Overview

Our approach relies on the definition of *model sensemaking strategies* (SMSs) that can be reused and customised for specific DSLs. Fig. 2 overviews their main components.

An SMS yields an interactive *visual representation* for (graph-based) data according to a visual metaphor, e.g., plots, bar charts, or adjacency matrices. In our model-centric approach, data is extracted from the (meta-)model(s) being

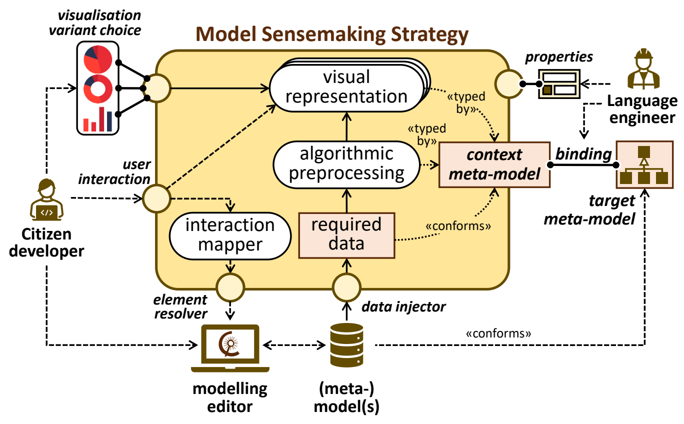


Fig. 2: Components of a model sensemaking strategy (SMS).

explored via a *data injector*. This data may undergo an *algorithmic preprocessing* step, if necessary. Some SMSs support multiple *visualisation variants*, which can be switched at runtime. For example, Fig. 2 shows some supported visualisation alternatives for the Categorical SMS: pie, donut, and bar chart. All the steps depend on the *context meta-model* of the SMS, which the SMS exposes to make it reusable. Finally, SMSs can define and expose *properties* to refine the resulting visualisations (e.g., specifying the axes titles in a bar chart).

Any DSL can reuse an SMS by means of a *binding*. This specifies how the SMS context meta-model elements are mapped to those of the DSL *target meta-model*. For efficiency, we do not rely on an explicit model-to-model transformation from the DSL models to the format of the SMSs. Instead, we use notions of generic programming [51], where the SMS can be seen as a generic component that is instantiated by a binding to the target meta-model. As next section shows, this target meta-model can be a domain meta-model (specifying the abstract syntax of a DSL), or the linguistic meta-model of the modelling environment. In the first case, the SMS is applicable to instances of the meta-model; in the second, to each (meta-)model built within the environment. To support our approach, we include a recommender that computes possible bindings, and provides a language to compute derived expressions.

SMSs provide interaction support at run-time. The *interaction mapper* specifies the (reactive) behaviour of the visualisations upon user interaction. It establishes a bidirectional communication with the modelling editor, bridging the gap between the impacted model elements and their representation in the SMS. For example, when a tree view SMS is equipped with interaction, it becomes a navigation utility that allows the user to navigate the model by clicking on the tree nodes.

While modelling with a DSL, citizen developers (i.e., final users) have available the SMSs previously configured (and bound) by a language engineer. SMSs are presented in a dashboard to allow multiple sensemaking tasks at once.

IV. Applying Sensemaking Strategies

The key component for applying an SMS is its *binding*, which maps every concept of the SMS's *context meta-model*

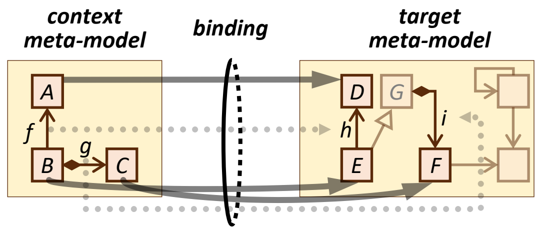


Fig. 3: Binding a context meta-model to a target meta-model.

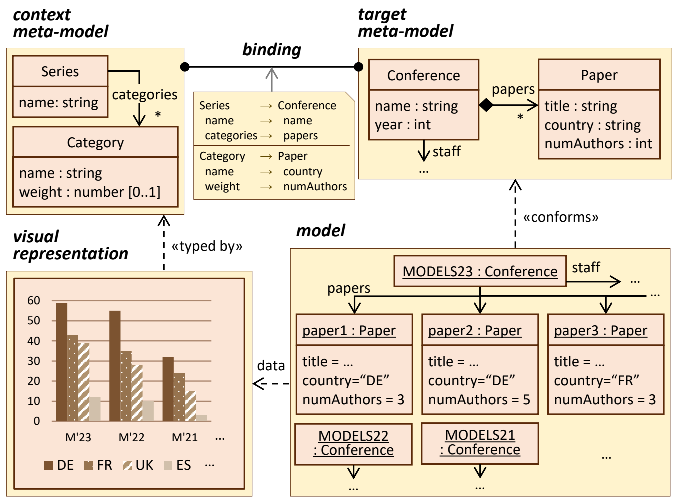


Fig. 4: Application example of the Categorical SMS.

to elements of the *target meta-model* – see Fig. 3.

As an example, suppose the MODELS organisation wants to study the origin countries of the submitted papers per edition of the conference. The Categorical SMS fits this intent. Fig. 4 depicts the application of this SMS to an example meta-model (with conferences and associated papers), and the resulting visualisation for a sample model – more on this SMS in Section V-A. The SMS is applied via a binding, which, once established, allows the visualisation of any conformant model using the strategy. Moreover, if the SMS supports multiple visualisation variants, these can be switched at run-time.

The rest of the section delves into the context meta-model (Section IV-A), the binding (Section IV-B), the target meta-model (Section IV-C), and SMSs properties (Section IV-D).

A. The context meta-model

The context meta-model defines the application pattern of an SMS. It has to be bound to a target meta-model so that the SMS can be used to visualise meta-model instances. Each element of the context meta-model (i.e., classes, attributes, and references) can be deemed conceptual "holes" to be populated by elements of the target meta-model. Context meta-models usually contain few elements, enabling their application to a wide range of target meta-models.

B. The binding

The binding determines how to apply an SMS to (the instances of) a target meta-model. It is a structure-preserving mapping $b : C \to T$ relating each class, attribute, and reference of the context meta-model C to elements of the target meta-model T. We closely follow [36], [52] for the definition of its well-formedness conditions:

Classes. If c is a class in C, then $b(c)$ is also a class in T.

Class subtyping is preserved and reflected: c_1 is a subtype of c_2 in C (written $c_1 \leq c_2$) iff $b(c_1) \leq b(c_2)$.

Attributes. If a is an attribute defined or inherited in class c in C, then $b(a)$ is also an attribute inherited or defined in class $b(c)$. The type of the attribute must be preserved or refined in the binding: $b(a)$.type $\leq a$.type. For instance, an attribute of type double can be bound to an integer.

References. If r is a reference from class c_1 to class c_2 in C, then $b(r)$ is a reference from class c'_1 to c'_2 in T, s.t. $b(c_1) \leq c'_1 \wedge c'_2 \leq b(c_2)$.

Composition is preserved: if r is a composition in C, then so must be $b(r)$.

Cardinalities. If f is an attribute or reference in C, with a multiplicity interval $[l..h]$, then $b(f)$ should have multiplicity $[l'..h']$, with $l \leq l' \wedge h' \leq h$.

Non-injective mappings are supported, e.g., binding two classes c_1 and c_2 in C to one class c' in T is allowed (i.e., $b(c_1) = b(c_2) = c'$). The same holds for attributes and references. Generally, a well-formed binding b ensures that, for any model M conformant to T, slicing M w.r.t. the bound elements $b(C)$ yields a valid model M' of C, as Fig. 5 depicts. Slicing a model M w.r.t. a "smaller" meta-model can be expressed using the categorical notion of *pullback* (P.B.) [53]. We refer to [54] for more details.

$$\begin{tikzcd}
C \arrow[r, "b"] & T \\
M' \arrow[u, dotted, "type'"] \arrow[r, hook] \arrow[ur, phantom, "P.B."] & M \arrow[u, "type"']
\end{tikzcd}$$

Fig. 5: Extracting a model M' of the context meta-model C via a binding b.

Fig. 3 exemplifies these binding criteria. For example, we can map reference g to i because i is inherited by E, which is bound from the owner of g. Formally, $b(g) = i$, which is correct since $b(B) = E \leq G$ (the source of i), and for the target of i, F, we have $F \leq b(C) = F$, as required by the well-formedness criterion for references.

In practice, a structural mapping may be too restrictive. Hence, our approach enables the specification of derived property bindings using an *expression language*. For example, imagine that Paper.numAuthors in the target meta-model of Fig. 4 did not exist, and a Paper.authors reference to a class Author was introduced. The binding of Category.weight could, then, be expressed /this.getReference("authors").target.length. In this case, the keyword this represents the bound Paper object. Section VI-A will give more details about this language.

C. The target meta-model

SMSs can be applied to (meta-)models independently of their level of abstraction. The only requirement is establishing a sound binding. This setting is especially suitable for multi-level modelling as, in this modelling paradigm, (meta-)models can conform in two ways to other models: via ontological or linguistic conformance [55]. This leads to two scenarios,

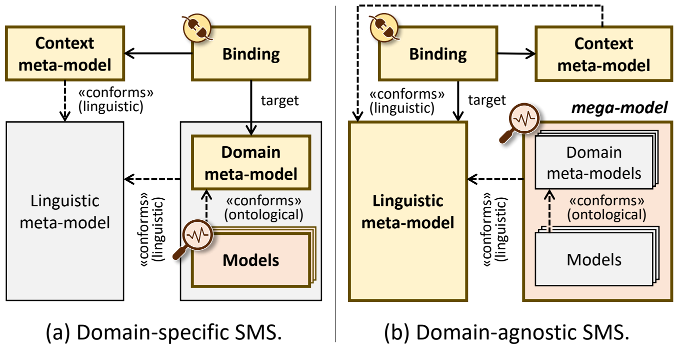


Fig. 6: SMS classification by target meta-model.

domain-specific and *domain-agnostic* SMS applications, depending on the type of conformance to the target meta-model.

Fig. 6 depicts both cases. On the one hand, *domain-specific* SMS applications emerge from ontological conformance. As shown in Fig. 6a, the binding targets a specific domain meta-model, thereby allowing the visualisations of any (ontologically) conformant model. Fig. 4 exemplifies the application of a domain-specific SMS.

On the other hand, *domain-agnostic* SMS applications are bound to the linguistic meta-model of the modelling framework, as depicted in Fig. 6b.[2] As, by construction, every (meta-)model conforms (linguistically) to the linguistic meta-model, this allows the visualisation of any (meta-)model. This approach is helpful in two situations:

a) To understand (complex) meta-models. By targeting the linguistic meta-model, domain-agnostic bindings treat domain meta-models as (linguistic) instances. Therefore, they can be applied to any meta-model, regardless of their domain or complexity. For example, Fig. 10 displays an SMS representing a meta-model as a heat-map, while Fig. 13 shows SMSs counting the number of elements within a meta-model; their division into concrete/abstract classes and enums using a bar chart; the prevalence of attribute names using a cloud map; and the distribution of meta-classes as data- or connection-centric using a scatter plot. Since these SMSs are domain-agnostic, they can be used to understand models as well.
b) To understand whole modelling ecosystems (i.e., mega-models). Models at any level of abstraction are (linguistic) instances of the linguistic meta-model. This means that bindings of agnostic SMSs can gather data about element instances through the expression language. For example, we can use an agnostic SMS to understand how a language is used, by presenting a proportional area chart, where meta-classes are represented as rectangles with size proportional to their number of instances (cf. Fig. 1).

D. Strategy properties

SMSs feature properties to customise the resulting visualisations. Each SMS defines its own set of properties, which can be mandatory or optional, and are typed. For instance, all the SMSs define a title property (a mandatory string), and SMSs displaying Cartesian axes allow the definition of X and Y labels (optional strings). Unlike bindings, properties are independent of the target meta-model and are populated by value.

[2] See Section VI for more details on Dandelion's linguistic meta-model.

V. Catalogue of Sensemaking Strategies

We have developed a catalogue of 10 SMSs, which can be displayed using 20 visualisations. We employ a consistent format organised into sections to describe SMSs – similar to traditional software design patterns [56]. Namely, *intent* (i.e., the goal of the strategy), *presentation metaphor*, *context meta-model*, *visualisation variants*, and *properties*. We accompany our explanations with motivating examples.

Table I summarises the implemented SMSs and the supported visualisation variants. We have grouped similar SMSs by common presentation metaphors (PM), including numerical charts and plots; SMSs grouping elements into categories; metric-based SMSs; SMSs for models with time; and SMSs targeting model structure. The entire catalogue of supported strategies can be found on the tool's website.[3]

For space constraints, we detail only two SMSs: Categorical (Section V-A) and Weighted Hierarchy (Section V-B).

A. Categorical

Aggregation is a powerful technique for summarising large amounts of data: elements that share a feature are grouped together, partitioning datasets into categories. The Categorical SMS exploits this technique in the realm of models to visualise categories and their incidence.

a) Intent: The Categorical SMS aims to understand partitions of objects to a set of (unknown) categories. Some involved sensemaking tasks include: '*What categories emerge from the data?*' and '*What is the incidence of each category?*'. The last task can be refined into discovering the most and least frequent categories.

b) Presentation metaphor: This is a grouping-based SMS. Other examples under the same metaphor include cluster analysis and pattern matching.

c) Context meta-model: It contains two classes, Series and Category, both identified by name (cf. Fig. 4). Each series is associated with a set of categories, refining the scope of the analysis. Categories expose an optional numeric weight, which determines the contribution of the bound object to its category tally. This weight is assumed to be 1 if left unspecified. Objects sharing a Category.name are aggregated into the same category, and the visualisation is replicated for each series.

d) Visualisation variants: Vertical and horizontal bar charts, (semi-circle) donut, pie, and proportional area charts (see Fig. 11).

e) Properties: The SMS introduces optional X_label and Y_label string properties. These only apply to bar charts, the visualisation variants that display axes labels.

f) Motivating example: The example in Fig. 4.

[3] https://miso.es/tools/Dandelion.html

TABLE I: Overview of model sensemaking strategies and their visualisation variants.

PM	SMS	Visualisation variants	Description
D	Numerical	line graph, area chart, scatter plot	For (x, y) coordinates.
D	Numerical + Frequency	bubble chart	For (x, y) coordinates with an associated frequency.
G	Categorical	vertical/horizontal bar chart, (semi-circle) donut chart, pie chart, proportional area chart	To partition data into categories.
M	Metric Distribution	boxplot	To depict the most important percentiles of a metric.
M	Free Metric	highlighted number, icon and number	To visualise an unconstrained value.
M	Bounded Metric	angular gauge, gauge chart	To visualise a value bounded in a min..max range.
M	Literal Metric	word cloud	To represent frequency in a textual field.
T	Time-Based	Gantt chart	For timed tasks with start and finish dates.
S	Connectivity	adjacency matrix/heat map, chord diagram	For cross-referenced objects.
S	Weighted Hierarchy	treemap	For nested objects.

PM = presentation metaphor: D = data, G = grouping, M = metric, T = time, S = structural.

B. Weighted hierarchy

Containment plays a central role in enabling abstraction in software engineering. It describes the relationship between two objects where one object is a part of – or belongs to – the other. It creates hierarchical structures where the parent objects (the containers) contain child objects (the containees). Conceptually, the lifespan of the containees is contingent on that of the container: when the container is deleted, so are the containees. The Weighted Hierarchy SMS exploits the support of composition (a restrictive form of association) in many DSLs to visualise recursive containment relationships, where each element can be attributed a weight.

a) Intent: The Weighted Hierarchy SMS aims to understand (multi-level) hierarchies emerging from containment relationships, and the relevance of their elements. The main sensemaking tasks to answer are '*What is the structure of the examined component?*', '*What is the relevance of the components?*'.

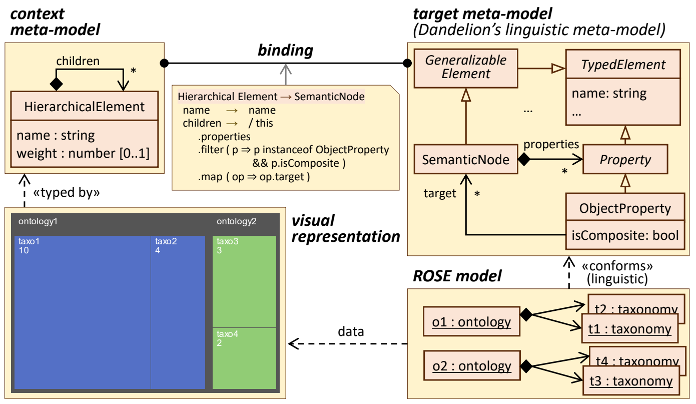
b) Presentation metaphor: This is a structural-based SMS. The other SMS that belongs to this group is Connectivity, which relaxes the composition to an association.

c) Context meta-model: The strategy fuses the concepts of containers and containees into a single class, HierarchicalElement (cf. Fig. 7). These have a name and an optional numeric weight, which is the children collection's size by default. What characterises the SMS is that hierarchical elements are related to themselves via a children composition, supporting recursive containment relationships.

d) Visualisation variants: The SMS can be visualised with Treemaps spanning multiple levels.

e) Properties: The SMS introduces a title property.

f) Motivating example: We can use this SMS to understand the structure of (meta-)models. Fig. 7 depicts an agnostic application of the strategy on Dandelion's linguistic meta-model. The figure shows a small fragment of it, displaying SemanticNode (Dandelion's concept representing both classes and objects) and ObjectProperty, which is the concept representing both references and links. The binding maps



```
Hierarchical Element → SemanticNode
  name      →   name
  children  →   / this
     .properties
     .filter ( p ⇒ p instanceof ObjectProperty
                 && p.isComposite )
     .map ( op ⇒ op.target )
```

Fig. 7: Weighted Hierarchy SMS, bound to Dandelion's linguistic meta-model, and used to understand a ROSE model.

HierarchicalElement to SemanticNode, and children to an expression that obtains the target SemanticNodes of each composition relation. This SMS application can then be used to analyse arbitrary (meta-)models, such as particular UGROUND ROSE models, as illustrated in the figure.

VI. Architecture and Tool Support

We have implemented SMSs atop Dandelion [8],[4] a cloud-based graphical language workbench for industrial low-code development. This section describes the architecture of our solution (Section VI-A), and the tool itself (Section VI-B).

A. Architecture

The architecture of Dandelion is split into two main components: the frontend and the backend, as shown in Fig. 8. The frontend exposes a web-based graphical editor where *citizen developers* create and explore models, and *language engineers* create and configure meta-models. With the incorporation of SMSs support, they can also select SMSs and bind them to the DSL under construction. SMSs are created by an *interaction expert*. All model management is transparent as it takes

[4] https://miso.es/tools/Dandelion.html

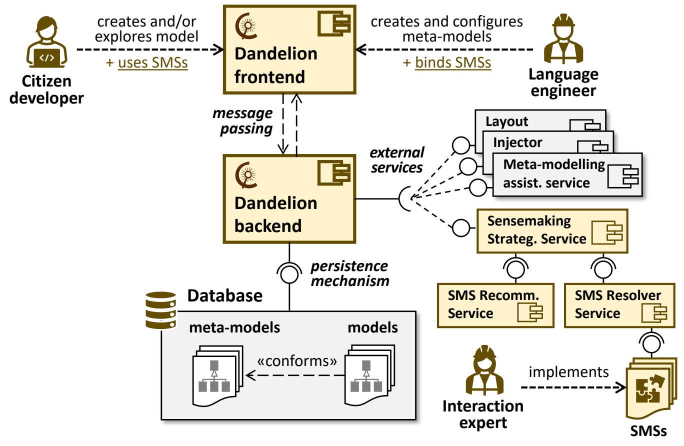


Fig. 8: Extending Dandelion's architecture to support SMSs.

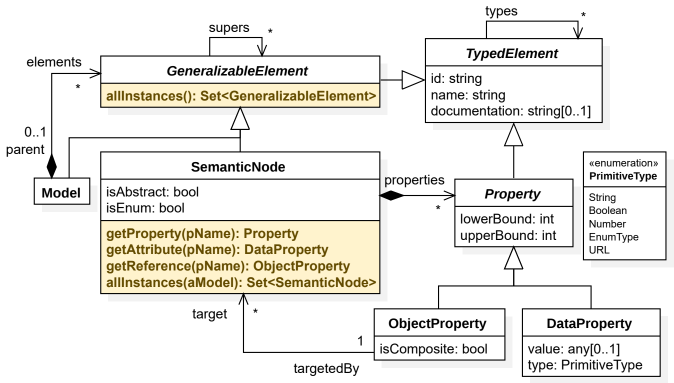


Fig. 9: Dandelion's linguistic meta-model, with operations for derived bindings.

place at the backend, which communicates with the frontend bidirectionally via message passing. The backend implements a flexible persistence mechanism, based on Elasticsearch [57] to integrate with different (meta-)model sources.

SMSs are integrated into Dandelion through the *Sensemaking Strategies Service*. Dandelion already supports a series of external services, such as layouting, to arrange graphical elements on the canvas; injectors, to import/export models from/to other formats, such as EMF [12]; or meta-modelling assistants, like DROID [58], which recommends attributes and references when building a meta-model. The new service is split into a *Recommendation Service*, serving the SMS recommender; and a *Resolver Service*, to execute SMSs.

The **SMS recommender** takes as input an SMS's context meta-model and a target meta-model (Fig. 3) and finds the set of sound bindings between them. This is a variant of the NP-hard subgraph isomorphism problem [59], which is computationally demanding. To avoid a combinatorial explosion of recommendations, we only consider the bindings of classes, and we delegate the binding of attributes and references to the user. Depending on the feasibility of these partial bindings (i.e., whether every attribute of the context meta-model can be bound to, at least, one attribute of the target meta-model), they are tagged *perfect* or *incomplete*. The latter are also useful, as the user can still populate the non-bindable attributes with derived values through the dedicated expression language.

The **SMS resolver** interprets SMS applications and yields interactive diagrams. It requires identifying the type of SMS, a binding to a target meta-model, the values of the SMS properties, and a visualisation variant choice. Dandelion has a catalogue of SMSs, which an interaction expert can extend.

The execution of the resolver comprises three phases: *data extraction*, *algorithmic preprocessing*, and construction of the *visual representation*. First, all the relevant model elements dictated by the binding are obtained. This task is delegated to an internal interface, IDandelionRepository, to exploit Dandelion's integration with different data providers.[5] If there are derived attributes, they are evaluated here. Next, the resolver may perform an *algorithmic preprocessing* step, if needed. For example, metric's boxplots require computing descriptive statistics, while categorical strategies have to aggregate data. The last step is constructing a visual representation (e.g., bar charts or boxplots) with the data obtained in the previous steps. Typically, SMSs expose multiple visualisation variants, which are populated in this step. We rely on two popular libraries for this task: D3.js [60] and Apache ECharts [61]. As this step is decoupled from the rest of the resolver, integration of new visualisation libraries is eased.

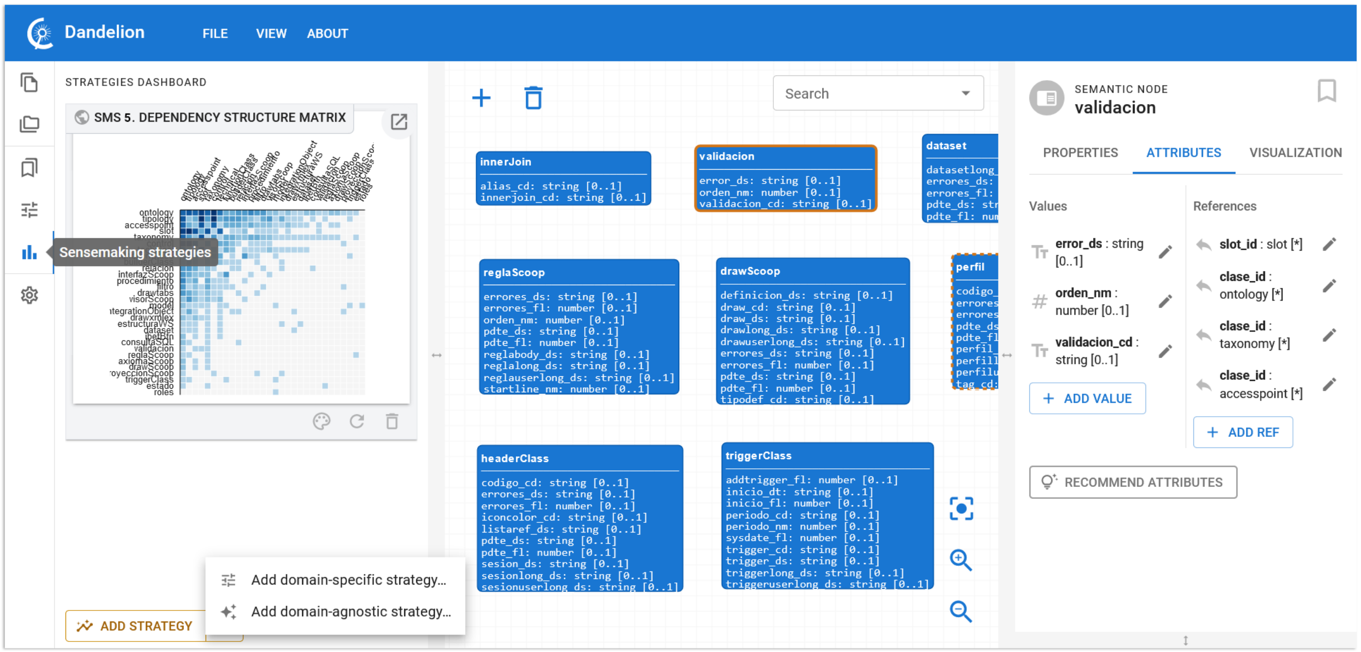
Property bindings can be derived thanks to a dedicated **expression language**. This language is implemented by extending Dandelion's linguistic meta-model with additional operations (Fig. 9). This meta-model supports multi-level modelling [55] and is level-agnostic [62], uniformly representing concepts at any level of abstraction. This way, meta-classes Model, SemanticNode, and ObjectProperty are used to represent meta-models and models; classes and objects; and references and links, respectively. All the attributes (e.g., TypedElement.name) are accessible via getters, and navigability is supported. SemanticNode implements a crucial operation for derived bindings: allInstances(), which returns all the instances of a given SemanticNode or Model, allowing thus penetrating into lower levels to extract information. Additionally, SemanticNode overloads allInstances() to obtain the node's instances within a particular model. Finally, we have implemented this expression language using TypeScript, hence supporting (higher-order) functions such as map or filter; and length for the size of a collection.

B. Tool support

We have integrated SMSs in Dandelion in the *strategies dashboard*: a panel shown next to the modelling canvas (c.f. Fig. 10).[6] Strategies support different operations: maximising in a full-screen view, switching visualisation variants, and forcing a refresh. They can also be rearranged via drag and drop in the dashboard.

[5] We follow a very similar approach to the Epsilon Model Connectivity Layer: https://www.eclipse.org/epsilon/doc/emc.

[6] Edges in the modelling canvas are hidden for clarity.



```
innerJoin
alias_cd: string [0..1]
innerjoin_cd: string [0..1]

validacion
error_ds: string [0..1]
orden_nm: number [0..1]
validacion_cd: string [0..1]

reglaScoop
errores_ds: string [0..1]
errores_fl: number [0..1]
orden_nm: number [0..1]
pdte_ds: string [0..1]
pdte_fl: number [0..1]
reglabody_ds: string [0..1]
reglalong_ds: string [0..1]
reglauserlong_ds: string [0..1]
startline_nm: number [0..1]

drawScoop
definicion_ds: string [0..1]
draw_cd: string [0..1]
draw_ds: string [0..1]
drawlong_ds: string [0..1]
drawuserlong_ds: string [0..1]
errores_ds: string [0..1]
errores_fl: number [0..1]
pdte_ds: string [0..1]
pdte_fl: number [0..1]
tipodef_cd: string [0..1]

headerClass
codigo_cd: string [0..1]
errores_ds: string [0..1]
errores_fl: number [0..1]
iconcolor_cd: string [0..1]
listaref_ds: string [0..1]
pdte_ds: string [0..1]
pdte_fl: number [0..1]
sesion_ds: string [0..1]
sesionlong_ds: string [0..1]
sesionuserlong_ds: string [0..1]

triggerClass
addtrigger_fl: number [0..1]
inicio_dt: string [0..1]
inicio_fl: number [0..1]
periodo_cd: string [0..1]
periodo_nm: number [0..1]
sysdate_fl: number [0..1]
trigger_cd: string [0..1]
trigger_ds: string [0..1]
triggerlong_ds: string [0..1]
triggeruserlong_ds: string [0..1]
```

Values	References
error_ds : string [0..1]	slot_id : slot [*]
orden_nm : number [0..1]	clase_id : ontology [*]
validacion_cd : string [0..1]	clase_id : taxonomy [*]
	clase_id : accesspoint [*]

Fig. 10: ROSE meta-model in Dandelion, showing a strategies dashboard with a dependency structure matrix SMS.

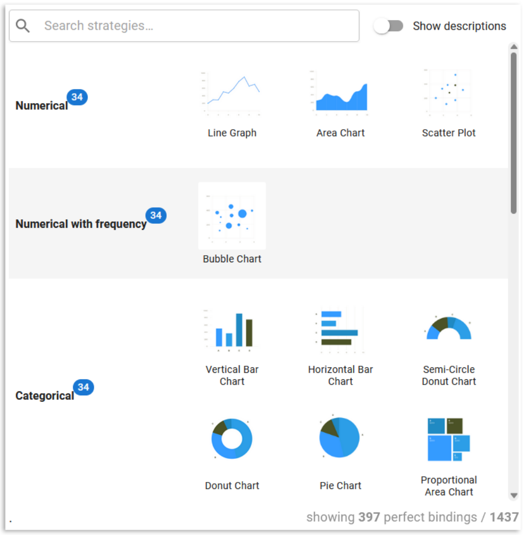
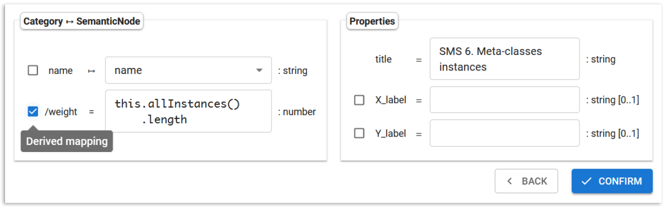


Fig. 11: Recommender panel in Dandelion.



☐	name	↦	name	: string
☑	/weight	=	this.allInstances() .length	: number

	title	=	SMS 6. Meta-classes instances	: string
☐	X_label	=		: string [0..1]
☐	Y_label	=		: string [0..1]

Fig. 12: Finishing the binding of a domain-agnostic SMS in Dandelion, employing a derived mapping.

The creation of an SMS takes place in a multi-step wizard. First, a suitable SMS and a visualisation variant are selected. To guide the decision, the number of suitable bindings is shown next to each SMS (coming from the SMS Recommender) – see Fig. 11. Next, the wizard displays a description and the context meta-model of the SMS together with all the suggested (available) bindings. The last step is completing the selected partial binding (possibly with derived values), and setting values to the appropriate properties of the SMS (c.f. Fig. 12). The strategy is then added to the dashboard.

VII. Evaluation

This section evaluates our approach by demonstrating that SMSs can be leveraged to understand large industrial low-code ecosystems. In particular, we examine UGROUND's modelling ecosystem, consisting of the ROSE meta-model [9] and large model instances (with 175,000+ elements). We aim to answer the following research questions (RQs):

RQ1. Do SMSs help to understand the ROSE meta-model?
RQ2. Can SMSs be used to understand how ROSE is used in practice?
RQ3. Can SMSs be used to understand ROSE models?

Each RQ challenges the utility of SMSs in gaining insight into the modelling ecosystem differently. With RQ1 (Section VII-A), we question their efficacy to understand a complex meta-model, which is a useful task for novel language users and language engineers. With RQ2 (Section VII-B), we consider said meta-model together with all its instances. Hence, we evaluate if SMSs can be useful for language designers to analyse how a DSL is used in practice, and guide its possible evolution. With RQ3 (Section VII-C), we

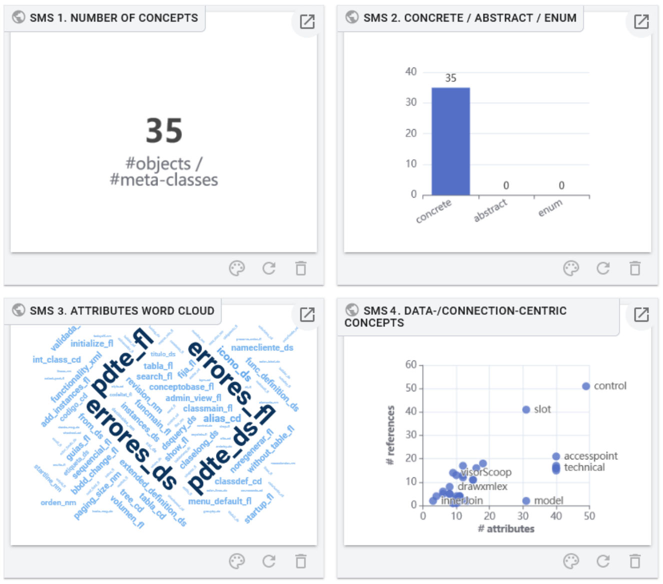


Fig. 13: SMSs #1–4 targeting ROSE meta-model (RQ1).

focus on concrete models to evaluate whether SMSs help creating visualisations tailored for their semantics. This task is helpful for regular language users, who may need help understanding a large model (perhaps built by a third person). Finally, Section VII-D synthesises conclusions, lessons learnt, and threats to validity.

A. RQ1: understanding a complex meta-model

Comprehending meta-models is a frequent activity in the modelling process, which typically entails finding answers to sensemaking tasks (STs) like the following:

ST1. What are the concepts defined in the meta-model?
ST2. Which attributes do concepts define?
ST3. What are the relationships between concepts?

Meta-models are typically built using editors that may not be designed explicitly for sensemaking. For example, graphical editors for meta-models usually rely on a graph-based visualisation metaphor which, while effective for small meta-models, does not scale well for large meta-models (cf. Fig. 1). We argue that SMSs can help alleviating this problem.

To fulfil STs 1–3, we propose the SMSs #1–5 detailed in Table II. We employ one or more SMSs to answer each ST as, according to sensemaking theory, sensemaking tasks benefit from a variety of visualisations for their resolution [7]. Fig. 13 shows SMSs #1–4 applied to UGROUND's ROSE meta-model (left in Fig. 1), and Fig. 10 displays SMS #5.

These SMSs allow answering STs 1–3. For ST1, SMS #1 counts the number of concepts, and SMS #2 categorises them into concrete concepts, abstract concepts, or enums. For ST2, SMS #3 summarises 500+ attributes in a word cloud, highlighting four common attribute names, which suggest the support for error control in many concepts. Additionally, SMS #4, displays the number of attributes and references of each concept in a scatterplot. Complex concepts control

TABLE II: SMSs for UGROUND mega-model evaluation.

	ST	#SMS	Sensemaking question	SMS type
RQ1	ST1	1	How many concepts are there?	Free metric
		2	Which types of concepts are there?	Categorical
	ST2	3	Which attributes are there?	Literal metric
		4	Are concepts data or connection-centric?	Numerical
	ST3	5	Which concepts are coupled?	Connectivity
RQ2	ST4	6	What is the distribution of concepts?	Categorical
	ST5	7	What is the instantiation range of fields?	Categorical
RQ3	ST6	8	Error codes of objects?	Literal metric
	ST7	9	States of Scoop rules (reglaScoop)?	Categorical
	ST8	5	Which concepts are coupled?	Connectivity

and slot stand out, having a high, yet balanced number of attributes and references. Other concepts, like model, are data-centric, with a high number of attributes (>30). This SMS provides a bidirectional interaction: selecting a concept in the plot highlights the corresponding concept in the modelling canvas, and vice versa. The analysis can be complemented, for instance, with SMS #5, which presents a *dependency structure matrix* to grasp the relationships between concepts. As it is sorted by degree of incidence, it is suited to answer ST3. For example, it reveals that concepts like ontology, tipology, or accesspoint (the first ones in the axes) are highly coupled to other concepts.

These SMSs are domain-agnostic (i.e., they target the linguistic meta-model) and can therefore be reused for other meta-models – not necessarily within the UGROUND ecosystem. Some SMSs employ derived bindings, which are specified using the expression language. For example, SMS #2 categorises concepts into 'concrete', 'abstract', or 'enum' by discriminating the isAbstract and isEnum properties from SemanticNode in Dandelion's linguistic meta-model (Fig. 9).

Answering RQ1, SMSs prove effective in understanding complex meta-models such as ROSE. They synergise by being presented in a dashboard, providing a holistic and reactive exploration of the meta-model. Furthermore, the support for agnostic bindings and derived values enables the implementation of reusable SMSs for any meta-model.

B. RQ2: understanding a modelling ecosystem

In this RQ, we evaluate if SMSs are useful to understand how a meta-model is used in practice. This includes analysing the contents and structure of the emerging modelling ecosystem – comprising meta-models and their instances (domain models). We consider two sensemaking tasks to this end:

ST4. What is the usage of each language primitive?
ST5. How are fields used?

SMSs #6–7 aim to answer these questions by their application to UGROUND's modelling ecosystem. SMS #6 (Fig. 1, right) displays a prominent metric of the usage of a meta-model: the number of instances per meta-class, answering ST4. SMS #7 (Fig. 14) focuses on specific meta-model concepts (e.g., technical in the figure) to understand the usage of its optional attributes. For instance, errores_fl with value 100 % means that every instance of technical populates this attribute.

This analysis can be leveraged for language evolution: highly used attributes can be designated mandatory, and scarcely used ones can be deemed redundant. A similar analysis can be used for the instantiation range of references.

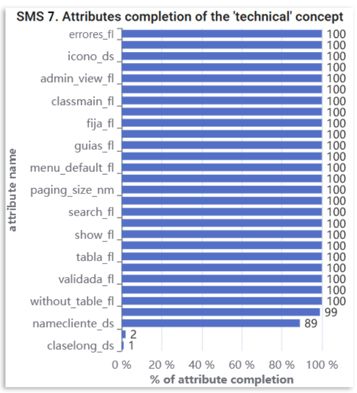


Fig. 14: SMS #7.

Overall, we can answer RQ2 positively. SMSs #6 and #7 are domain-agnostic, collecting data from model instances and not merely from the meta-model. Hence, this allows creating effective, reusable SMSs for any modelling ecosystem.

C. RQ3: understanding specific models

When the meta-model of a domain-specific language is understood, it is possible to devise tailored SMSs for it. SMSs #6-8 have been applied to UGROUND models to resolve:

ST6. Which error codes can be triggered by the objects in this model?

ST7. Which are the states of Scoop rules?

ST8. How are elements connected?

ST6 targets the understanding of the error codes produced by the defined services within the model. We use a literal metric (a word cloud) to visualise them. Stop words are discarded to avoid noise in the visualisation. This way, the user can quickly grasp the most common errors.

Scoop rules are used to describe the processes of the information system, to be enacted by the Scoop rule engine. By understanding the state of these rules (Active, Draft, Obsolete, Pending and others), we can grasp the condition of the services within the model. SMS #9 categorises the states of Scoop rules (reglaScoop in the meta-model) by a estado reference. As multiple states can be grouped under the same term, the SMS aggregates them with a derived binding.

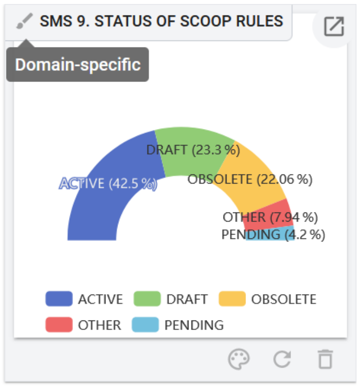


Fig. 15: SMS #9.

Finally, we can reuse SMS #5, a dependency structure matrix, to solve ST8. This is possible because this SMS is domain-agnostic (i.e., like SMSs #1–5). The resulting visualisation is identical to Fig. 10, but the concepts in the matrix are the objects of the currently analysed model.

Hence, we can answer RQ3 positively: we can use a combination of domain domain-specific and agnostic SMSs to understand specific ROSE models.

D. Conclusions, lessons learnt, and threats to validity

Regarding RQ1, we could analyse the ROSE meta-model, realising that most concepts are concrete. This fact, together with a high prevalence of certain attributes among classes may suggest the usefulness of a meta-model refactoring. Moreover, given the insights provided by Dandelion, we intend to make the tool available for new employees in onboarding processes to help them understand the ROSE language.

For RQ2, the analysis was useful in understanding how ROSE is used in practice. This analysis helps understanding scarcely used primitives (like innerJoin) and typical instantiation ranges for fields, providing valuable information for evolving the DSL definition.

Finally, in RQ3 we have facilitated the analysis of models by providing tailored visualisations. UGROUND plans to replace the Scoop engine, and hence needs to migrate its rules. Therefore, SMSs like #9 can help evolve models progressively.

As validity threats, we have used SMSs to analyse UGROUND's ecosystem. Since this is a complex industrial ecosystem, we also expect our SMSs to be used to understand other modelling ecosystems. Our evaluation has been driven by the needs of UGROUND's company employees when working with (meta-)models. We plan to complement this evaluation with user experiments measuring the efficacy of SMSs to resolve concrete sensemaking tasks.

VIII. Conclusions and Future Work

This paper has introduced the notion of *model sensemaking strategy* (SMS), a reusable, generic component for creating flexible, insightful visualisations for large models. SMSs are instantiated by a binding to the – domain or linguistic – meta-model at which they are applied. We have proposed a catalogue of 10 SMSs and 20 exchangeable visualisations, supporting the understanding of meta-models, models, and entire modelling ecosystems. We have implemented SMSs atop Dandelion, an industrial low-code graphical language workbench for the cloud, featuring a flexible binding language and a recommender that suggests suitable SMSs for a given meta-model. We have reported on an industrial evaluation demonstrating the effectiveness of SMSs in understanding UGROUND's large industrial ecosystem.

As future work, we would like to increase our catalogue with new graph-based SMSs, and improve existing ones with more flexibility. We would like to integrate a sensemaking assistant, which, given a description (in natural language) of a sensemaking task, recommends a strategy and a binding. Technically, we are improving the tool, and will work on its extension to support more layouts, model exploration strategies, and injectors from other formats.

Acknowledgements

This project is funded by the EU Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie grant agreement No 813884 and the Spanish MICINN (PID2021-122270OB-I00, TED2021-129381B-C21).

REFERENCES

[1] D. Moody, "The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering," *IEEE Trans. Software Eng.*, vol. 35, no. 6, pp. 756–779, 2009.

[2] D. S. Kolovos, L. M. Rose, N. Matragkas, R. F. Paige, E. Guerra, J. Sánchez Cuadrado, J. De Lara, I. Ráth, D. Varró, M. Tisi, and J. Cabot, "A research roadmap towards achieving scalability in model driven engineering," in *Proceedings of the Workshop on Scalability in Model Driven Engineering*. ACM, 2013.

[3] J. Bertin, *Semiology of Graphics*. University of Wisconsin Press, 1983.

[4] D. M. Russell, M. J. Stefik, P. Pirolli, and S. K. Card, "The cost structure of sensemaking," in *Conference on Human Factors in Computing Systems (INTERACT, CHI)*. ACM, 1993, pp. 269–276.

[5] M. Tisi, J. Mottu, D. S. Kolovos, J. de Lara, E. Guerra, D. D. Ruscio, A. Pierantonio, and M. Wimmer, "Lowcomote: Training the next generation of experts in scalable low-code engineering platforms," in *STAF Co-Located Events Joint Proceedings*, ser. CEUR Workshop Proceedings, vol. 2405. CEUR-WS.org, 2019, pp. 73–78.

[6] D. D. Ruscio, D. S. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, and M. Wimmer, "Low-code development and model-driven engineering: Two sides of the same coin?" *Softw. Syst. Model.*, vol. 21, no. 2, pp. 437–446, 2022.

[7] R. Pienta, J. Abello, M. Kahng, and D. H. Chau, "Scalable graph exploration and visualization: Sensemaking challenges and opportunities," in *International Conference on Big Data and Smart Computing (BIGCOMP)*, 2015, pp. 271–278.

[8] F. Martínez-Lasaca, P. Díez, E. Guerra, and J. de Lara, "Dandelion: a scalable, cloud-based graphical language workbench for industrial low-code development," *Journal of Computer Languages*, vol. 76, p. 101217, 2023, https://miso.es/tools/Dandelion.html.

[9] A. Díez, "Recursive ontology-based systems engineering," 2015. [Online]. Available: https://patents.google.com/patent/US9760345B2/en

[10] A. Díez, N. Nguyen, F. Díez, and E. Chavarriaga, "MDE for enterprise application systems," in *MODELSWARD*. SciTePress, 2013, pp. 253–256.

[11] A. Jiménez-Pastor, A. Garmendia, and J. de Lara, "Scalable model exploration for model-driven engineering," *J. Syst. Softw.*, vol. 132, pp. 204–225, 2017.

[12] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework, 2nd edition*. Pearson Education, 2008.

[13] D. Strüber, S. Jurack, T. Schäfer, S. Schulz, and G. Taentzer, "Managing model and meta-model components with export and import interfaces," in *Proceedings of the 4rd Workshop on Scalable Model Driven Engineering part of the Software Technologies: Applications and Foundations (STAF)*, ser. CEUR Workshop Proceedings, vol. 1652. CEUR-WS.org, 2016, pp. 31–36.

[14] A. Garmendia, E. Guerra, J. de Lara, A. García-Domínguez, and D. S. Kolovos, "Scaling-up domain-specific modelling languages through modularity services," *Inf. Softw. Technol.*, vol. 115, pp. 97–118, 2019.

[15] K. Jahed, M. Bagherzadeh, and J. Dingel, "On the benefits of file-level modularity for EMF models," *Softw. Syst. Model.*, vol. 20, no. 1, pp. 267–286, 2021.

[16] R. Wei, D. S. Kolovos, A. García-Domínguez, K. Barmpis, and R. F. Paige, "Partial loading of XMI models," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS)*. ACM, 2016, pp. 329–339.

[17] Q. Ma, P. Kelsen, and C. Glodt, "A generic model decomposition technique and its application to the eclipse modeling framework," *Softw. Syst. Model.*, vol. 14, no. 2, pp. 921–952, 2015.

[18] K. Barmpis and D. S. Kolovos, "Hawk: towards a scalable model indexing architecture," in *Workshop on Scalability in Model Driven Engineering*. ACM, 2013, p. 6.

[19] G. Daniel, "Efficient persistence and query techniques for very large models," in *ACM Student Research Competition at MoDELS*, ser. CEUR Workshop Proceedings, vol. 1775. CEUR-WS.org, 2016.

[20] G. Daniel, G. Sunyé, and J. Cabot, "Advanced prefetching and caching of models with PrefetchML," *Softw. Syst. Model.*, vol. 18, no. 3, pp. 1773–1794, 2019.

[21] Y. Hu and L. Shi, "Visualizing large graphs," *WIREs Computational Statistics*, vol. 7, no. 2, pp. 115–136, 2015.

[22] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J. Fekete, and D. W. Fellner, "Visual analysis of large graphs: State-of-the-art and future research challenges," *Comput. Graph. Forum*, vol. 30, no. 6, pp. 1719–1749, 2011.

[23] J. J. Thomas and K. A. Cook, "A visual analytics agenda," *IEEE Computer Graphics and Applications*, vol. 26, no. 1, pp. 10–13, 2006.

[24] N. Tovanich, A. Pister, G. Richer, P. Valdivia, C. Prieur, J. Fekete, and P. Isenberg, "VAST 2020 contest challenge: Graphmatchmaker - visual analytics for graph comparison and matching," *IEEE Computer Graphics and Applications*, vol. 42, no. 4, pp. 89–102, 2022.

[25] C. Vehlow, F. Beck, and D. Weiskopf, "Visualizing group structures in graphs: A survey," *Comput. Graph. Forum*, vol. 36, no. 6, pp. 201–225, 2017.

[26] F. Beck, M. Burch, S. Diehl, and D. Weiskopf, "A taxonomy and survey of dynamic graph visualization," *Computer Graphics Forum*, vol. 36, no. 1, pp. 133–159, 2017.

[27] R. S. Pienta, M. Kahng, Z. Lin, J. Vreeken, P. P. Talukdar, J. Abello, G. Parameswaran, and D. H. Chau, *FACETS: Adaptive Local Exploration of Large Graphs*. SIAM, 2017, pp. 597–605.

[28] S. Dueñas, V. Cosentino, J. M. González-Barahona, A. del Castillo San Felix, D. Izquierdo-Cortazar, L. Cañas-Díaz, and A. P. García-Plaza, "Grimoirelab: A toolset for software development analytics," *PeerJ Comput. Sci.*, vol. 7, p. e601, 2021.

[29] F. D. Luca, M. I. Hossain, S. G. Kobourov, and K. Börner, "Multi-level tree based approach for interactive graph visualization with semantic zoom," *CoRR*, vol. abs/1906.05996, 2019.

[30] B. Musial and T. Jacobs, "Application of focus + context to UML," in *Australasian Symposium on Information Visualisation (InVis.au)*, ser. CRPIT, vol. 24. Australian Computer Society, 2003, pp. 75–80.

[31] M. Frisch, R. Dachselt, and T. Brückmann, "Towards seamless semantic zooming techniques for UML diagrams," in *ACM Symposium on Software Visualization*. ACM, 2008, pp. 207–208.

[32] J. F. G. Navarro, R. Therón, and F. J. García-Fernández, "Semantic zoom: A details on demand visualisation technique for modelling OWL ontologies," in *9th International Conference on Practical Applications of Agents and Multiagent Systems (PAAMS)*, ser. Advances in Intelligent and Soft Computing, vol. 89. Springer, 2011, pp. 85–92.

[33] Y. E. Ahmar, S. Gerard, C. Dumoulin, and X. L. Pallec, "Enhancing the communication value of UML models with graphical layers," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS)*. IEEE Computer Society, 2015, pp. 64–69.

[34] C. D. Schulze, G. Hoops, and R. von Hanxleden, "Automatic layout and label management for compact UML sequence diagrams," in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2018*. IEEE Computer Society, 2018, pp. 187–191.

[35] C. D. Schulze, Y. Lasch, and R. von Hanxleden, "Label management: Keeping complex diagrams usable," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE Computer Society, 2016, pp. 3–11.

[36] J. de Lara, E. Guerra, and J. Sánchez Cuadrado, "Reusable abstractions for modeling languages," *Inf. Syst.*, vol. 38, no. 8, pp. 1128–1149, 2013.

[37] H. Bruneliere, E. Burger, J. Cabot, and M. Wimmer, "A feature-based survey of model view approaches," *Softw. Syst. Model.*, vol. 18, no. 3, p. 1931–1952, 2019.

[38] G. D. Carlo, P. Langer, and D. Bork, "Rethinking model representation - A taxonomy of advanced information visualization in conceptual modeling," in *International Conference on Conceptual Modeling (ER)*, ser. Lecture Notes in Computer Science, vol. 13607. Springer, 2022, pp. 35–51.

[39] D. D. Vincenzo, J. D. Rocco, D. D. Ruscio, and A. Pierantonio, "Enhancing syntax expressiveness in domain-specific modelling," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MoDELS)*. IEEE, 2021, pp. 586–594.

[40] R. Morgan, G. Grossmann, M. Schrefl, M. Stumptner, and T. Payne, "VizDSL: A visual DSL for interactive information visualization," in *Advanced Information Systems Engineering*, J. Krogstie and H. A. Reijers, Eds. Cham: Springer International Publishing, 2018, pp. 440–455.

[41] N. Moha, V. Mahé, O. Barais, and J. Jézéquel, "Generic model refactorings," in *Model Driven Engineering Languages and Systems, 12th International Conference, MODELS*, ser. Lecture Notes in Computer Science, vol. 5795. Springer, 2009, pp. 628–643.

[42] A. Garmendia, E. Guerra, D. S. Kolovos, and J. de Lara, "EMF splitter: A structured approach to EMF modularity," in *Proceedings of the*

3rd Workshop on Extreme Modeling co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages & Systems, XM@MoDELS, ser. CEUR Workshop Proceedings, vol. 1239. CEUR-WS.org, 2014, pp. 22–31.

[43] J. Sánchez Cuadrado, E. Guerra, and J. de Lara, "A component model for model transformations," *IEEE Trans. Software Eng.*, vol. 40, no. 11, pp. 1042–1060, 2014.

[44] J. Bruel, B. Combemale, E. Guerra, J. Jézéquel, J. Kienzle, J. de Lara, G. Mussbacher, E. Syriani, and H. Vangheluwe, "Comparing and classifying model transformation reuse approaches across metamodels," *Softw. Syst. Model.*, vol. 19, no. 2, pp. 441–465, 2020.

[45] C. Guy, B. Combemale, S. Derrien, J. Steel, and J. Jézéquel, "On model subtyping," in *Modelling Foundations and Applications - 8th European Conference, ECMFA*, ser. Lecture Notes in Computer Science, vol. 7349. Springer, 2012, pp. 400–415.

[46] J. de Lara and E. Guerra, "From types to type requirements: genericity for model-driven engineering," *Softw. Syst. Model.*, vol. 12, no. 3, pp. 453–474, 2013.

[47] H. Ergin, E. Syriani, and J. Gray, "Design pattern oriented development of model transformations," *Comput. Lang. Syst. Struct.*, vol. 46, pp. 106–139, 2016.

[48] A. Pescador, A. Garmendia, E. Guerra, J. Sánchez Cuadrado, and J. de Lara, "Pattern-based development of domain-specific modelling languages," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS)*. IEEE Computer Society, 2015, pp. 166–175.

[49] B. Hamid, S. Gürgens, and A. Fuchs, "Security patterns modeling and formalization for pattern-based development of secure software systems," *Innov. Syst. Softw. Eng.*, vol. 12, no. 2, pp. 109–140, 2016.

[50] B. Hamid and J. Pérez, "Supporting pattern-based dependability engineering via model-driven development: Approach, tool-support and empirical validation," *J. Syst. Softw.*, vol. 122, pp. 239–273, 2016.

[51] D. P. Gregor, J. Järvi, J. G. Siek, B. Stroustrup, G. D. Reis, and A. Lumsdaine, "Concepts: Linguistic support for generic programming in C++," in *Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. ACM, 2006, pp. 291–310.

[52] J. Sánchez Cuadrado, E. Guerra, and J. de Lara, "Flexible model-to-model transformation templates: An application to ATL," *J. Object Technol.*, vol. 11, no. 2, pp. 4: 1–28, 2012.

[53] S. MacLane, *Categories for the Working Mathematician*. New York: Springer-Verlag, 1971.

[54] J. de Lara and E. Guerra, "Towards the flexible reuse of model transformations: A formal approach based on graph transformation," *J. Log. Algebraic Methods Program.*, vol. 83, no. 5-6, pp. 427–458, 2014.

[55] C. Atkinson and T. Kuhne, "Model-driven development: a metamodeling foundation," *IEEE Software*, vol. 20, no. 5, pp. 36–41, 2003.

[56] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, 1994.

[57] Elasticsearch, https://www.elastic.co/elasticsearch, 2023.

[58] L. Almonte, E. Guerra, I. Cantador, and J. de Lara, "Building recommenders for modelling languages with Droid," in *ASE*. ACM, 2022, pp. 155:1–155:4.

[59] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM (JACM)*, vol. 23, no. 1, pp. 31–42, 1976.

[60] Data Driven Documents, https://d3js.org/, 2023.

[61] Apache ECharts, https://echarts.apache.org/en/index.html, 2023.

[62] C. Atkinson, B. Kennel, and B. Goß, "The level-agnostic modeling language," in *Software Language Engineering - Third International Conference, SLE*, ser. Lecture Notes in Computer Science, vol. 6563. Springer, 2010, pp. 266–275.