PAVEL VLASOV

Beyond Diagrams

MAXIMIZING EFFECTIVENESS AND EFFICIENCY OF THE INFORMATION PIPELINE WITH ADJUSTABLE FORMALISM

Beyond diagrams

Maximizing effectiveness and efficiency of the information pipeline with adjustable formalism

Pavel Vlasov

This book is for sale at http://leanpub.com/beyond-diagrams

This version was published on 2024-01-13

To my family, their patience, support and feedback!

Contents

Introduction

Diagrams is a fundamental form of communication. They've been used by humanity since presitoric times and predate writing.

Wikipedia defines a diagram[1] as "a symbolic **representation** of information using visualization techniques" with **representation**[2] in turn defined as a "**reference** (to an object) conveyed through pictures".

There are many diagrams which can be divided into two major categories:

- Just representation. Such diagrams leave the job on deducing what a particular shape on the diagram means to the beholder. Examples of tools which produce such diagrams include "Draw.io" (diagrams.net) - covered in details below, MS Visio, MS PowerPoint.
- Representations referencing objects belonging to a specific problem domain expressed in a domain language. Examples: UML, BPMN, TIBCO BusinessWorks.

According to the author's experience the IT architecture is dominated by the first type of diagrams with Draw.io and PowerPoint being predominant formats. The reason for this is that enterprises have their own "languages", often not explicitly defined. They don't "speak" in, say, UML or BPMN.

Tools like Eclipse Sirius[3] make it rather easy to create specialized diagramming tools, but they are not widely used (to author's

[1]https://en.wikipedia.org/wiki/Diagram
[2]https://en.wikipedia.org/wiki/Depiction
[3]https://eclipse.dev/sirius/

knowledge) either for a variety of reasons explained in the "Business case" chapter.

This book explains how to take "representation only" Draw.io diagrams and map them to a domian model. The mapping can be many to many - a diagram element may represent zero or more domain (semantic) elements. A semantic element may appear on zero or more diagrams and may appear more than once on a diagram. Also, a diagram may be mapped to more than one domain model.

Once diagrams are mapped and loaded into a model, they can be consumed by human and non-human actors, including AI agents.

A picture is worth a thousand words. An interactive picture… 10 thousand? In the below demos hover over diagram elements for tooltips and click on the to navigate to elements' pages. I hope the demos will convince you to keep reading:

Demo	Domain Model	Description
Nasdanika Documentation[4]	HTML Application[5]	Diagram elements are mapped to actions[6], static HTML is generated from an action model and deployed using GitHub Pages

Demo	Domain Model	Description
Sample Family[7]	Family Model[8]	Mimics Sirius Basic Family tutorial metamodel and model. Also demonstrates representation filtering - a border around Isa is not in the source diagram, it was added during site generation (model loading)
Internet Banking System[9]	Architecture[10]	Mimics C4 sample model[11]. It also demonstrates representation filtering - adding background to Mainframe-BankingSystem-FacadeImpl[12] on Mainframe Banking System Facade[13] diagram.

Demo	Domain Model	Description
Living beings[14]	Graph[15]	Uses an out-of-the-box diagram from Draw.io. Patially maps it. Demonstrates “semantization of geometry” - ordering by angular position of diagram elements.
AWS[16]	Architecture[17]	A simple demo of mapping of an OOTB Draw.io diagram

For those who’s been convinced to read on, the book is very practical and is structured with the “shortest path to value” in mind[18]. Diagram mapping explained in the book is built using Java technologies - Java 17, Maven, Eclipse EMF Ecore, Spring Expression Language (SpEL). If you are not a Java person, you may skim over the technical details and focus on the concepts. If you find them beneficial, you may either bite the bullet and get up to speed with the underlying technologies or find somebody to delegate the technical parts to - more about it in the “Business case” chapter.

We will start with an overview of Draw.io - it has its own meta-model and, in fact, the mapping process is a process of mapping between two models. So this overview will be an overview of the source model.

[18]As a matter of fact, if you’ve liked what you’ve seen in the demos above and want to get your feet wet right away - click on the “Source” link in the demo footer, clone the repo, and start tinkering!

Then we'll take a brief look at Ecore, which is used for domain modeling - the source model above and the target models.

Armed with the that knowledge we'll delve into the minutia of mapping followed by overviews and example of mapping to a few models - family, action, graph, and architecture.

Then, we'll take a look at a few other models, which might not be ready for use right away, but may be a good starting point for your custom models, if/when you get to that point.

The "Semantic search" chapter will show how to make diagrams "consumable" by AI.

The "Business case" chapter explains the "economy" of diagramming - the diagramming/modeling value stream/pipline, variation points, and how to optimize it for a given application/embodiment.

The "Markdown" chapter explains how to write markdown documentation for model elements - add diagrams (representations), etc.

The "Custom model" chapter explains how to create a custom domain model - from scratch or by extending one or more of pre-existing models.

The "Crystal ball" chapter provides a few ideas regarding application of what was presented so far, with a specific focus on "Architecture As Code".

The book closes with mapping reference chapters.

Drawio

"**diagrams.net** (previously **draw.io**) is a cross-platform graph drawing software developed in HTML5 and JavaScript. Its interface can be used to create diagrams such as flowcharts, wireframes, UML diagrams, organizational charts, and network diagrams."[1]

In this book we'll use the term `drawio` to refer to **diagrams.net/draw.io** because it is the default extension for diagram files and a domain name where diagrams.net redirects - https://www.drawio.com/

Here is a quick summary of Drawio capabilities:

- Available on multiple platforms:
  - Web App,
  - Linux
  - macOS
  - Windows
- Available as a plugin in:
  - Confluence
  - VS Code[2]
- Can be self-hosted:
  - WAR file[3]
  - Docker container[4]

[1] https://en.wikipedia.org/wiki/Diagrams.net
[2] https://marketplace.visualstudio.com/items?itemName=hediet.vscode-drawio
[3] https://github.com/jgraph/drawio/releases
[4] https://hub.docker.com/r/jgraph/drawio

- Has a large library of shapes and images - more than 2000 built-in images in version 22.1.17
- Images can be copied/pasted from external sources. For example:
    - 600+ Azure icons[5] (perhaps already a part of the above 2000+)
    - Million+ icons on Icons8[6]
- Features an in-browser diagram viewer which allows to interact with the diagram - click on elements, go full screen, show/hide tags and layers, and switch to the editor
- A diagram document may contain multiple pages with diagram elements linking to pages. It allows to create a page hierarchy.
- Nasdanika Drawio API extends the concept of linking to pages to cross-document linking by using `data:page/name,<URI>#<URL encoded page name>` URL. For example, `data:page/name,my-system.drawio#My+Component` links to `My Component` page in `my-system.drawio` resource. This allows to create a multi-resource network of diagrams. Nasdanika Drawio API also supports loading of documents from arbitrary URI's using URI resolver. For example, `maven://<gav>/<resource path>` to load from Maven resources or `gitlab://<project>/<path>` to load resources from GitLab without cloning a repository, provided there is a handler (`Function<URI,InputStream>`) supporting the aforementioned URI's.
- Tooltips can be used to provide descriptions for diagram elements
- Users can:
    - Add properties to diagram elements

[5] https://learn.microsoft.com/en-us/azure/architecture/icons/
[6] https://icons8.com/icons

  - Create libraries of diagram elements and groups of elements (diagram fragments)

All of the above free and open source!

I'm a huge fan of Drawio since their JGraph times - these guys are really good and know what they are doing - they've been doing since the beginning of the century!

However, it is not enough for me - I need to be able to associate a great deal of details with diagram elements starting with long formatted descriptions - something that exceeds tooltip capabilities. I also need a structured way to navigate the diagram - as a human and programmatically (feed it to AI, for example).

Luckily, it is possible, has been done, and this is what this book is about!

Under the hood the diagram is stored in XML. In the editor you can select "Extras > Edit Diagram" to see the XML. Nasdanika Core Drawio[7] modules provides Java API and EMF Ecore model to read Drawio diagrams, manipulate them, and save as Drawio files or as HTML to view in a web browser.

[7] https://docs.nasdanika.org/core/drawio/index.html

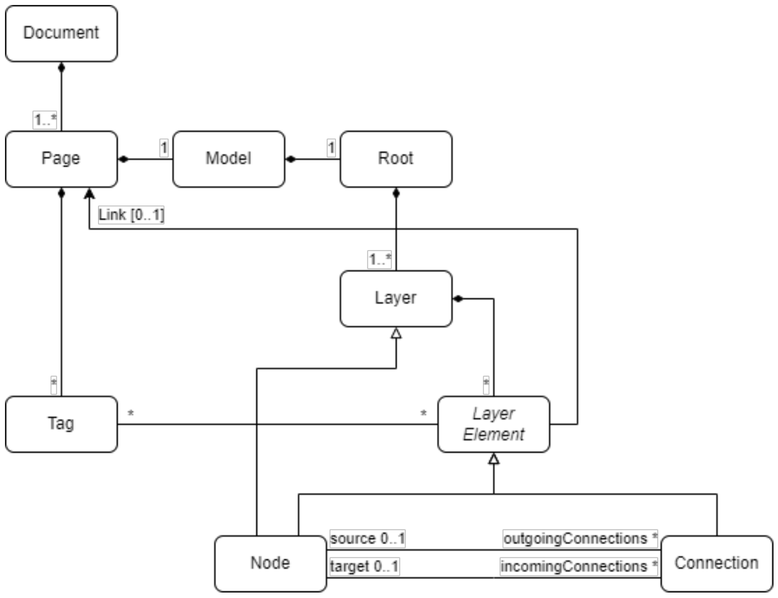

Figure 1. Drawio metamodel

The above diagram (created in Drawio) depicts the structure of a diagram. `Document` is the root object of the diagram. It contains one or more `Pages`. A Page contains a `Model` which in turn contains `Root`. Logically a Page, its Model, and its Root can be thought of as a single entity. `Root` contains one or more `Layers`. A Layer contains layer elements - `Nodes` and `Connections`.

Root, layers, and layer elements have unique ID's and properties. Layer element ID's are editable, the editor prevents duplicates. Pages have unique ID's which are not editable.

Layer elements can be tagged, styled, they can have labels, links, and tooltips.

So, to summarize, we have a "language" with six "parts of speech" - document, page, layer, node, connection, and tag. However, this language has quite a large "vocabulary" of nodes and connections - dozens of styles and millions of images.

Elements of the Drawio language represent/reference objects (by the definition of a diagram). A reference is established using graphics, text (labels and tooltips), links, and properties. In the subsequent chapters we will see how to use properties to establish references from diagram elements to elements of a domain (semantic) model expressed in EMF Ecore.

If you are a software development practitioner, I'd like to give you a reason to continue reading - Generative AI threatening obsoletion of human "coders". Since ChatGPT sprang onto the scene in late 2022, there's been growing realization that the technology gets better and better at writing code. Currently the software industry is dominated by text-based artifacts - Java, Python, YAML, XML, JSON - it is all text. However, humans don't think in text, they think in entities/concepts and relationships. If humans were thinking in text, the phrase "put (something) into words" would not exist - thoughts would already be in words. Historically software development evolved in the direction of heightening the level of abstraction - from machine code to assembly language and so on. Diagrams' level of abstraction is higher than text - humans can explain in a few shapes concepts which'd take pages of text. A good deal of software development efforts is spent on manually "translating" diagrams to code. So, in my opinion, a natural step for human software development practitioners is to "shift left" from text do diagrams and possibly use AI to generate text from diagrams. AI is good at working with text and images. Diagrams are neither text nor images (in the way AI works with images - convolutional neural networks). So, diagramming is something where we, humans, can hold the fort for some time until there is a large body of diagrams to train AI on. AI can help humans to make sense of a large body of diagrams (and graphs in general) with context-specific semantic search and summarization - see "Semantic Search" chapter.

EMF Ecore

Eclipse Modeling Framework (EMF) is an Eclipse-based modeling framework and code generation facility for building tools and other applications based on a structured data model. **Ecore** is the core (meta-)model at the heart of EMF. It allows expressing other models by leveraging its constructs. Ecore is also its own metamodel (i.e.: Ecore is defined in terms of itself).[1]

In this book EMF Ecore is used to create metamodels of problem domains. These metamodels are then used as mapping targets. It is important to mention that metamodels and documentation generated from them have value on their own - they establish a common (ubiquitous) language.[2]

Simply put, Ecore is a way to create (domain specific) languages understandable by both humans and computers.

[1]https://en.wikipedia.org/wiki/Eclipse_Modeling_Framework
[2]https://martinfowler.com/bliki/UbiquitousLanguage.html

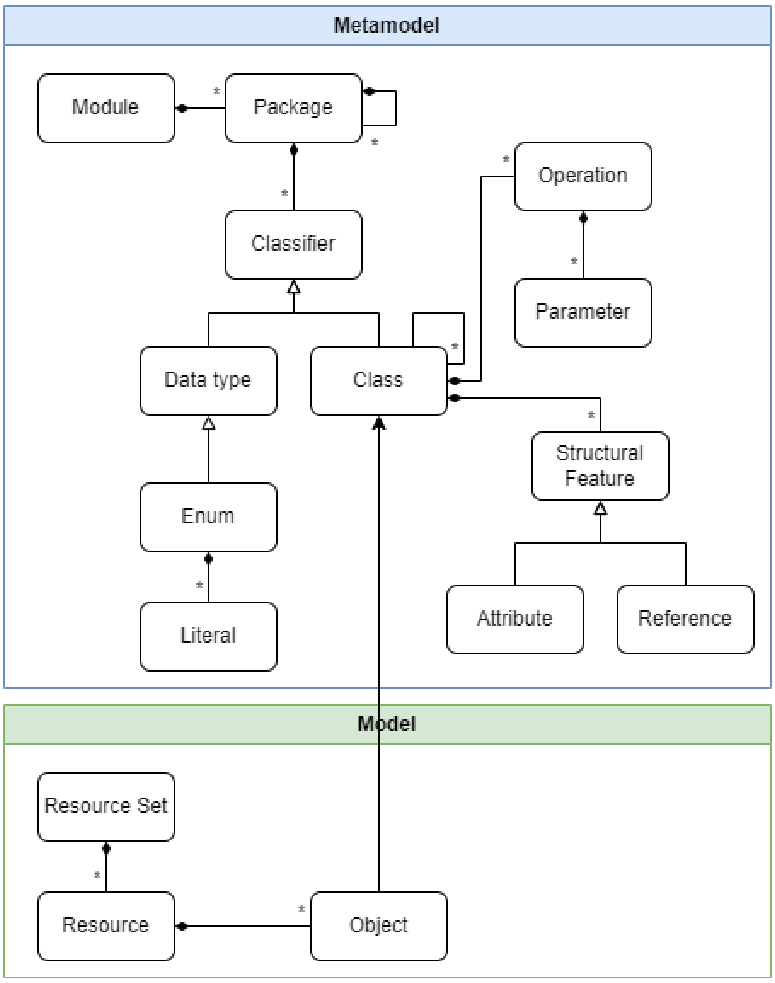

Figure 2. Ecore concepts

The above diagrams shows key Ecore concepts and their relationships. `E` prefix is dropped for clarity. E.g. `EPackage` is shown as `Package`. More detailed documentation can be found here - https://ecore.models.nasdanika.org/ (work in progress)

- Metamodel

  - Module - Java/Maven module or an OSGi bundle. Contains zero or more models with each models containing a root package.
  - Package - a group of classifiers and sub-packages. Packages are identified (keyed) by namespace URI's.
  - Classifier - a class, data type or enumeration.
  - Data type - a bridge to the Java type system.
  - Enumeration - a collection of literals.
  - Class - contains zero or more structural features - attributes and references and zero or more operations. May inherit from zero or more superclasses. Can be abstract and can be an interface.
  - Structural feature - can hold single or multiple values

    * Attribute - holds a "simple" value such as String, Date, number.
    * Reference - relationship between two objects (EObjects). Unidirectional, but can be associated with another reference (opposite) to form a bi-directional relationship.

- Model

  - Resource set - a group of related resources identified by a URI.
  - Resource - a group of objects. A resource is identified by a URI.
  - Object - an instance of a metamodel class.

Below is a concrete example using the Family metamodel[3] and model[4]:

- Metamodel

[3] https://family.models.nasdanika.org/
[4] https://family.models.nasdanika.org/demos/mapping/

  - Module - `org.nasdanika.models.family:model` Maven module.
  - Package - `family` defined in `family.ecore` resource with `ecore://nasdanika.org/models/family` namespace URI.
  - Class - `Man` class inheriting from `Person` class.
  - Structural feature

    * Attribute - `name` attribute of `Man` (inherited from `NamedElement`).
    * Reference - `father` reference.

- Model

  - Resource set - created with a factory which treats `.drawio` diagram files as resources with diagram elements mapped to the family model.
  - Resource - `family.drawio` file.
  - Object - `Paul` and `Elias` are instances of `Man` class. `Elias` references `Paul` as his father.

It is important to note that resources are identified by URI's, not URL's. It allows to load resources from multiple data sources - files, URL's, databases, source repositories such as Git, binary repositories such as Maven. Using Nasdanika classes it is also possible to load objects from multiple sources on access. For example, a person's id and name can be loaded from, say, Excel file or a database. Some other attributes may be loaded by making HTTP requests only if the client code reads those attributes.

EMF Ecore provides facilities to read/write resources from/to XMI files. It also provides tools[5] for creating models.
Eclipse CDO allows to store models in distributed repositories.

Nasdanika provides factories to load models from Drawio diagrams, MS Excel, and CSV files. It also provides a URI handler

[5]https://eclipse.dev/ecoretools/

for loading models from classpath resources and a documentation generator for Ecore models. In addition to the family model mentioned above, you can find examples of Ecore models and generated documentation in Nasdanika Model repositories[6].

I hope that his brief introduction is sufficient for grasping mapping concepts which will be presented in the subsequent chapters. We will take a more detailed look at Ecore in the "Custom model" chapter.

[6]https://github.com/orgs/Nasdanika-Models/repositories

Mapping

base-uri

This property is used to compute base URI for resolving references such as `doc-ref` and `spec-ref`. The base URI is resolved relative to the logical parent element up to the document (diagram) URI. For pages with links from model element the logical parent the first linking element. For connections with sources, it is the connection source. Otherwise, the element's container (`eContainer()`) is the logical parent.

One way to consistently set base URI's is to check "Placeholders" checkbox and then use `%semantic-id%/` value for base URI.

config-prototype

See `prototype` below.

constructor

Spring Expression Language (SpEL)[1] expression evaluating to a semantic element. Takes precedence over `type`. May return `null` - in this case `type` would be used to create a semantic element, if specified.

This property can be used, for example, to look-up semantic elements defined elsewhere. Say, a list of family members can be

[1] https://docs.spring.io/spring-framework/reference/core/expressions.html

defined in an MS Excel workbook, but family relationships in a diagram.

Another example is "progressive enrichment". For example, high-level architecture is defined in some model and a diagram uses constructors for already defined architecture elements and `type` for their sub-elements. This approach can be applied multiple times similar to how Docker images are assembled from layers and base images.

In order to implement lookup constructors, override `configureConstructorEvaluationContext()` in sub-classes of `AbstractDrawioFactory` or `createEvaluationContext()` in Drawio resource factories. Set variables from which the constructor expression would obtain semantic elements.

documentation

Property value is used as documentation for semantic elements which extend `Documented`.

doc-format

Documentation format - `markdown` (default), `text`, or `html`. For `doc-ref` is derived from the reference extension if not set explicitly.

doc-ref

Property value is used as a URI to load documentation for semantic elements which extend `Documented`. The URI is resolved relative to the `base-uri`.

feature-map

A YAML map defining how features of the semantic element and related semantic elements shall be mapped.

The map keys shall be one of the following:

container

Mapping specification for the container element in container/contents permutation. Contains one or more of the following sub-keys with each containing a map of feature names to a mapping specification or a list of feature names.

- `self` - this element is a container
- `other` - the other element is a container

Example

```yaml
container:
  other: elements
  self:
    elements:
      argument-type: ArchitectureDescriptionElement
      path: 1
```

In the above example `other` specifies that this semantic element shall be added to the `elements` feature of its container regardless of the containment path length. `self` specifies that immediate children of this element (`path: 1`) shall be added to this semantic element `elements` collection if they are instances of `ArchitectureDescriptionElement`

contents

Mapping specification for the contents element in container/contents permutation. Contains one or more of the following subkeys with each containing a map of feature names to a mapping specification or a list of feature names.

- `self` - this element is contained by the other
- `other` - the other element is contained by this element

end

For connections - mapping specification for the connection end feature to map the connection target semantic element to a feature of the connection semantic element.

self

A map of feature names to a Spring Expression Language (SpEL)[2] expression or a list of expressions evaluating to the feature value or feature element value.

The expression is evaluated in the context of the source diagram element and has access to the following variables:

- `value` - semantic element
- `registry` - a map of diagram element to semantic elements

[2] https://docs.spring.io/spring-framework/reference/core/expressions.html

source

For connections - mapping specification for the connection source feature to map the connection semantic element to a feature of the connection source semantic element. If the connection semantic element is `null`, then the connection source semantic element is used instead.

target

For connections - mapping specification for the connection target feature to map the connection semantic element to a feature of the connection target semantic element. If the connection semantic element is `null`, then the connection target semantic element is used instead.

start

For connections - mapping specification for the connection start feature to map the connection source semantic element to a feature of the connection semantic element.

feature-map-ref

Property value is used as a URI to load feature map YAML specification. The URI is resolved relative to the `base-uri`.

mapping-selector

Mapping selector is used to select one or more semantic elements for feature mapping. If it is not provided, then the diagram element's semantic element is used for feature mapping.

Mapping selector shall be a YAML document containing either a single string or a list of strings. The strings are treated as Spring Expression Language (SpEL)[3] expression evaluating to a semantic element to use for feature mapping.

Mapping selectors may be used to associate multiple semantic elements with a diagram element for feature mapping purposes.

mapping-selector-ref

Property value is used as a URI to load mapping selector YAML. The URI is resolved relative to the `base-uri`.

operation-mapping

Allows to invoke semantic element operations for mapping purposes. For example, if a `Course` diagram element contains `Student` elements then `Course.enroll(Student)` operation may be used for mapping. The operation may check for course capacity, validate student status, etc.

Operation mapping is a map of operation names to operation specification. Operation specification can be either a map or a list of maps. The first case is treated as a singleton list.

The operation specification map supports the following entries:

arguments

A map of parameter names to SpEL expression evaluating their values in the context of the iterator element (see below). Argument

[3]https://docs.spring.io/spring-framework/reference/core/expressions.html

names are used for operation selection/matching - a candidate operation must have parameters with matching names.
The map does not have to contain entries for all operation parameter. Nulls are used as arguments for parameters which are not present in the map.

iterator

An optional SpEL expression which returns a value to iterate over and invoke the operation for every element. If the result is `java.util.Iterator` then it is used AS-IS. If the result is Iterable, Stream, or array, an iterator is created to iterate over the elements. If the result is null, then an empty iterator is created. Otherwise, a singleton iterator is created wrapping the result.

It allows to invoke the operation zero or more times. E.g. `enroll` for every student. If not defined, the iterator contains the source diagram element.

pass

An optional integer specifying the pass in which this operation shall be invoked. Use for ordering operation invocations.

For example, in the Nature model[4] you may want the Fox[5] to eat[6] the Hare[7], but only after the Hare eats the Grass[8]. In this case you don't specify the pass for `Rabbit.eats()`, so it defaults to zero. An you set the pass for `Fox.eats()` to `1`.

[4]https://nature.models.nasdanika.org/index.html
[5]https://nature.models.nasdanika.org/references/eClassifiers/Fox/index.html
[6]https://nature.models.nasdanika.org/references/eClassifiers/Animal/references/eOperations/eats-1/index.html
[7]https://nature.models.nasdanika.org/references/eClassifiers/Hare/index.html
[8]https://nature.models.nasdanika.org/references/eClassifiers/Grass/index.html

selector

An optional SpEL boolean expression evaluated in the context of the operation to disambiguate overloaded operations.

operation-mapping-ref

Property value is used as a URI to load operation mapping YAML. The URI is resolved relative to the `base-uri`.

page-element

`true` is for true value and any other value or absence of the property is considered false. There should be one page element per page. Having more than one may lead to an unpredictable behavior. For the first top level page the page element becomes a document element. For pages linked from model elements the page element is logically merged with the linking element.

This allows to define a high-level structure on a top level diagram page, link other pages to the diagram elements and refine their definitions.

If the semantic element of a page element extends `NamedElement` then the page name is used as element name if it is not already set by other means.

processor

A SpEL expression evaluating to zero or more `org.nasdanika.drawio.model.util.AbstractDrawioFactory.Processor`s. The expression can evaluate to `null`, a processor instance, iterator, iterable, array, or stream. It is evaluated in the context of the diagram element with `semanticElement`, `pass`, and `registry` variables.

prototype

Spring Expression Language (SpEL)[9] expression evaluating to a diagram element. The semantic element of that diagram element is copied and the copy is used as the semantic element of this diagram element. Also, prototype configuration (properties) is applied to this semantic element. Protypes can be chained.

Example: `getDocument().getModelElementById('6ycP1ahp__-4fXEwP3E-2-5')`

Selectors allow to define common configuration in one element and then reuse it in other elements. For example, a web server prototype may define an icon and then all web server element would inherit that configuration.

Drawio model classes provide convenience methods for finding diagram elements:

- Document[10]:
  - `getModelElementById(String id)`
  - `getModelElementByProperty(String name, String value)`
  - `getModelElementsByProperty(String name, String value)`
  - `getPageById(String id)`
  - `getPageByName(String name)`
- Page[11]:

[9] https://docs.spring.io/spring-framework/reference/core/expressions.html
[10] https://drawio.models.nasdanika.org/references/eClassifiers/Document/operations.html
[11] https://drawio.models.nasdanika.org/references/eClassifiers/Page/operations.html

  - `getModelElementById(String id)`
  - `getModelElementByProperty(String name, String value)`
  - `getModelElementsByProperty(String name, String value)`
  - `getTag(String name)`

- ModelElement[12]:

  - `getDocument()`
  - `getPage()`

- Root[13], Layer[14]:

  - `getModelElementById(String id)`
  - `getModelElementByProperty(String name, String value)`
  - `getModelElementsByProperty(String name, String value)`

A prototype must have a semantic element defined using `constructor`, `type`, `selector` or `ref-id`. If you want to inherit just configuration, but not the semantic element, then use `config-prototype` property instead of `prototype`.

ref-id

Some identifier to resolve a semantic element. You would need to override `getByRefId()` method and define what the reference id means in your case. For example, you may use semantic URI's to lookup elements. Say `ssn:123-45-6789` to lookup a person by SSN.

[12]https://drawio.models.nasdanika.org/references/eClassifiers/ModelElement/operations.html

[13]https://drawio.models.nasdanika.org/references/eClassifiers/Root/operations.html

[14]https://drawio.models.nasdanika.org/references/eClassifiers/Layer/operations.html

Resource factories treat ref-id’s as URI’s resolved relative to the diagram resource URI. Resolved URI’s are then are passed to `ResourceSet.getEObject(URI uri, true)`.

reference

Associates a diagram element with a reference of the semantic element of the first matched logical ancestor for mapping purposes. If the element has mapped descendants, their matching semantic elements are added to the reference.

Reference value can be a string or a map. The string form is equivalent to the map form with just the `name` entry.

The map form supports the following keys:

- `comparator` - see `comparator` in “Feature Mapping”.
- `condition` - a SpEL `boolean` expression evaluated in the context of the logical ancestor semantic element. If not provided, matches the first mapped ancestor. Has access to the following variables:
  - `sourcePath` - a list of logical ancestors starting with the logical parent
  - `registry`
- `expression` - a SpEL `EObject` expression evaluated in the context of the logical ancestor semantic element. If not provided, the logical ancestor itself is used. Has access to the following variables:
  - `sourcePath` - a list of logical ancestors starting with the logical parent
  - `registry`

- `name` - reference name
- `element-condition` - a SpEL `boolean` expression evaluated in the context of the descendant (contents) semantic element. If not provided, elements are matched by type. Has access to the following variables:
  - `sourcePath` - source containment path
  - `registry`
- `element-expression` - a SpEL `EObject` expression evaluated in the context of the descendant (contents) semantic element. If not provided, the descendant itself is used. Has access to the following variables:
  - `sourcePath` - source containment path
  - `registry`

script

Script to execute. The script has access to the following variables:

- `baseURI`
- `diagramElement`
- `logicalParent`
- `pass`
- `registry`
- `semanticElement`

Additional variables can be introduced by overriding `configureScriptEngine()` method. Script class loader can be customized by overriding `getClassLoader()` method.

script-engine

A boolean SpEL expression for selecting a script engine. The expression is evaluated in the context of a candidate `javax.script.ScriptEngineFactory` with the following variables:

- `diagramElement`
- `pass`
- `registry`
- `semanticElement`

script-ref

Script reference resolved relative to the base URI.

selector

Spring Expression Language (SpEL)[15] expression evaluating to a diagram element. The semantic element of that diagram element is used as the semantic element of this diagram element.

Example: `getDocument().getModelElementById('6ycP1ahp__-4fXEwP3E-2-5')`

Selectors allow to use the same semantic elment on multiple diagrams. Drawio model classes provide convenience methods for finding diagram elements:

- Document[16]:

[15]https://docs.spring.io/spring-framework/reference/core/expressions.html
[16]https://drawio.models.nasdanika.org/references/eClassifiers/Document/operations.html

  - `getModelElementById(String id)`
  - `getModelElementByProperty(String name, String value)`
  - `getModelElementsByProperty(String name, String value)`
  - `getPageById(String id)`
  - `getPageByName(String name)`

- Page[17]:

  - `getModelElementById(String id)`
  - `getModelElementByProperty(String name, String value)`
  - `getModelElementsByProperty(String name, String value)`
  - `getTag(String name)`

- ModelElement[18]:

  - `getDocument()`
  - `getPage()`

- Root[19], Layer[20]:

  - `getModelElementById(String id)`
  - `getModelElementByProperty(String name, String value)`
  - `getModelElementsByProperty(String name, String value)`

[17] https://drawio.models.nasdanika.org/references/eClassifiers/Page/operations.html
[18] https://drawio.models.nasdanika.org/references/eClassifiers/ModelElement/operations.html
[19] https://drawio.models.nasdanika.org/references/eClassifiers/Root/operations.html
[20] https://drawio.models.nasdanika.org/references/eClassifiers/Layer/operations.html

semantic-id

If the semantic element extends `StringIdentity`, `semantic-id` property can be used to specify the `id` attribute. If this property is not provided, then Drawio model element ID is used as a semantic ID.

semantic-selector

Spring Expression Language (SpEL)[21] expression evaluating to a semantic element.

Semantic selectors are similar to constructors with the following diffences:

- Semantic selectors are evaluated after constructors
- A constructor may evaluate to `null`, but a semantic selector must eventually evaluate to a non-null value
- A constructor is evaluated once, but a semantic selector maybe evaluated multiple time until it returns a non-null value
- A semantic selector is only evaluated if there isn't a semantic element already

Semantic selectors can be used to evaluate semantic elements using semantic elements of other elements. For example, a semantic selector of a child node may need a semantic element of its parent to resolve its own semantic element.

Overide `configureSemanticSelectorEvaluationContext()` to provide variables to the evaluator.

[21]https://docs.spring.io/spring-framework/reference/core/expressions.html

spec

Loads semantic element from the property value YAML using EObjectLoader.

Example:

```yaml
icon: fas fa-user
```

spec-ref

Loads semantic element from the property value URI using EObjectLoader. The URI is resolved relative to the `base-uri`.

tag-spec

TODO

tag-spec-ref

TODO

top-level-page

Page elements from top level pages are mapped to the document semantic element. By default a page without incoming links from other pages is considered to be a top level page. This property allows to override this behavior. `true` value indicates that the page is a top level page. Any other value is treated as `false`.

type

Type of the semantic element. Types are looked up in the factory packages in the following way:

- If the value contains a hash (`#`) then it is treated as a type URI. For example `ecore://nasdanika.org/models/family#//Man`.
- If the value contains a doc (`.`) then it is treated as a qualified EClass name with EPackage prefix before the dot. For example `family.Man`.
- Otherwise the value is treated as an unqualified EClass name - EPackages are iterated in the order of their registration and the first found EClass with matching name is used to create a semantic element.

Type is used to create a semantic element if there is no `constructor` or the constructor expression returned `null`. A combination of `constructor` and `type` can be used for mapping in different contexts. For example, when loading a stand-alone model `constructor` would return `null` and then `type` would be used. When the same diagram loaded in the context of a larger model, `constructor` may return a semantic element looked up in that larger model.

Namespaces

It is possible to maintain multiple mappings at a single element using namespace prefixes. In subclasses of `AbstractDrawioFactory` override `getPropertyNamespace()` method. By default this method returns an empty string which is used as a prefix for the configuration properties.

Feature mapping

TODO - copy from the mapping, unindent Comparators by functionality - angular, cartesian Semantization of geometry

Feature mapping value can be either a string or a map. If it is a string it is treated as a singleton map to `true` (unconditional mapping).

The below two snippets are equivalent:

```yaml
container:
  other: elements
```

```yaml
container:
  other:
    elements: true
```

The map value supports the following keys:

argument-type

Specifies type of feature elements to be set/added. String as defined in `type`.. Only instances of the type will be set/added. If absent, the feature type is used. Argument type can be used to restrict elements to a specific subtype of the feature type.

comparator

Comparator is used for "many" features to order elements. A comparator instance is created by `createComparator()` method which can be overridden in subclasses to provide support for additional comparators.

The following comparators are provided "out of the box":

clockwise

Compares elements by their angle relative to the node of the semantic element which holds the many reference. In the Living Beings[1] demo "Bird", "Fish", and "Bacteria" are compared by their angle to the "Living Beings" with the angle counted from "12 o'clock" - 90 degrees (default).

Feature mapping with comparators of "Bird", "Fish", and "Bacteria" are defined at the connections from "Living Beings" as:

```
source:
  elements:
    comparator: clockwise
```

To specify the base angle other than 90 degree use the map version of comparator definition where `clockwise` is the key mapping to a number or string value. The number value is used as the angle value in degrees. The string value is treated as a Spring Expression Language (SpEL)[2] expression evaluated in the context of the "parent" node. The expression may evaluate to a number or to a node. In the latter case the result is used to compute the angle between the context node and the result node.

In the Living Beings example "Streptococcus", ..., "Staphyllococcus" are compared relative to the "Bacteria" node with the base angle being the angle between the "Bacteria" node and "Living Beings" node. As such "Streptococcus" is the smallest node and "Staphyllococcus" is the largest. With the default angle of 90 degrees

[1]https://graph.models.nasdanika.org/demo/living-beings/index.html
[2]https://docs.spring.io/spring-framework/reference/core/expressions.html

“Lactobacyllus” would be the smallest and “Streptococcus” would be the largest.

Feature mapping with comparators of “Streptococcus”, …, “Staphyllococcus” is defined at connections from “Bacteria” to the respecive genus nodes as:

```
source:
  elements:
    comparator:
      clockwise: incoming[0].source
```

`incoming[0]` evaluates to the connection from “Living Beings” to “Bacteria” and `source` evaluates to “Living Beings”.

counterclockwise

Reverse of `clockwise`.

down-left

Compares nodes by their vertial order first with higher nodes being smaller and then by horizontal order with nodes on the right being smaller. Nodes are considered vertically equal if they vertically overlap.

down-right

Compares nodes by their vertial order first with higher nodes being smaller and then by horizontal order with nodes on the left being smaller. Nodes are considered vertically equal if they vertically overlap. This comparator can be used for org. charts.

expression

A Spring Expression Language (SpEL)[3] expression evaluated in the context of the feature element with `other` variable referencing the element to compare with. The exmpression has access to `registry` variable containing a map of diagram element to semantic elements.

flow

If one element is reacheable from the other by traversing connections, then the reacheable element is larger than the source element. In case of circular references the element with the smaller number of traversals to the other element is considered smaller. If elements are not connected they are compared by the fall-back comparator. This comparator can be used for workflows and PERT[4] charts.

If this comparator's value is a String, then it is uses as a name of the fallback comparator. In the below example children will be smaller than their parents and siblings will be compared using labels.

```yaml
container:
  self:
    members:
      argument-type: Person
      comparator:
        flow: label
```

If the value is a map, then it may have the following keys:

- `condition` - A boolean Spring Expression Language (SpEL)[5] expression evaluated in the context of a connection being

[3]https://docs.spring.io/spring-framework/reference/core/expressions.html
[4]https://en.wikipedia.org/wiki/Program_evaluation_and_review_technique
[5]https://docs.spring.io/spring-framework/reference/core/expressions.html

traversed. It may be used to traverse only connections which match the condition. For example, only transitions[6] between activities[7] in a process model.

- `fallback` - Fallback comparator.

In the below example…

key

A Spring Expression Language (SpEL)[8] expression evaluated in the context of the feature element. The expression must return a value which would be used for comparison using the natural comparator as explained below.

label

Uses diagram element label converted to plain text as a sorting key. String. In the Family mapping demo[9] family members are sorted by label using the following feature map definition:

```yaml
container:
  self:
    members:
      argument-type: Person
      comparator: label
```

label-descending

Uses diagram element label converted to plain text as a sorting key to compare in reverse alphabetical order. String.

[6] https://flow.models.nasdanika.org/references/eClassifiers/Transition/index.html
[7] https://flow.models.nasdanika.org/references/eClassifiers/Activity/index.html
[8] https://docs.spring.io/spring-framework/reference/core/expressions.html
[9] https://family.models.nasdanika.org/demos/mapping/index.html

left-down

Compares nodes by their horizontal order first with nodes on the right being smaller and then by vertial order with higher nodes being smaller. Nodes are considered horizontally equal if they horizontally overlap.

left-up

Compares nodes by their horizontal order first with nodes on the right being smaller and then by vertial order with lower nodes being smaller. Nodes are considered horizontally equal if they horizontally overlap.

natural

Uses feature element's `compareTo()` method for comparable elements. Otherwise compares using hash code. Null are greater than non-nulls.

property

Uses diagram element property as a sorting key. Singleton map. For example:

```
property: label
```

property-descending

The same as property, but compares in reverse alphabetical order.

reverse-flow

Same as `flow` but with target nodes being smaller than source nodes.

right-down

Compares nodes by their horizontal order first with nodes on the left being smaller and then by vertial order with higher nodes being smaller. Nodes are considered horizontally equal if they horizontally overlap.

right-up

Compares nodes by their horizontal order first with nodes on the left being smaller and then by vertial order with lower nodes being smaller. Nodes are considered horizontally equal if they horizontally overlap.

up-left

Compares nodes by their vertial order first with lower nodes being smaller and then by horizontal order with nodes on the right being smaller. Nodes are considered vertically equal if they vertically overlap.

up-right

Compares nodes by their vertial order first with lower nodes being smaller and then by horizontal order with nodes on the left being smaller. Nodes are considered vertically equal if they vertically overlap.

condition

A Spring Expression Language (SpEL)[10] boolean expression evaluated in the context of the candidate diagram element with the following variables:

- `value` - semantic element of the candidate diagram element
- `path` - containment path
- `registry` - a map of diagram element to semantic elements

path

Either an integer number 0 or a list of boolean SpEL expressions to match the path. If an integer then it is used to match path length as shown in the example below which matches only immediate children

```
container:
  self:
    elements:
      path: 1
```

If a list, then it matches if the list size is equal to the path length and each element evaluates to true in the context of a given path element. Expression have acess to `registry` variable - a map of diagram element to semantic elements.

[10] https://docs.spring.io/spring-framework/reference/core/expressions.html

nop

If `true`, no mapping is performed, but the chain mapper is not invoked. It can be used in scenarios with a default (chained) mapper to prevent the default behavior.

expression

A SpEL expression evaluating to a feature value in the context of the diagram element with with the following variables:

- `value` - semantic element of the diagram element
- `path` - containment path
- `registry` - a map of diagram element to semantic elements

greedy

Greedy is used with containment feature and specifies what to do if a candidate object is already contained by another object:

- `no-children` - grab the object if it is contained by an ansector of this semantic element. This is the default behavior.
- `false` - do not grab
- `true` - always grab

position

A number specifying the position of the element in the feature collection.

type

Type of the feature object to match. String as defined in `type`. Can be used in `other` mappings.

Family model

Intro, detailed description, demos

Generation - Java code, Maven plugin

Action model

Intro, demos

Graph model

Intro, table of flavors, living beings demo, aws

Generation - Java code, Maven plugin

Architecture model

Intro, roadmap, dimensions, demos

Generation - Java code, Maven plugin

Semantic search

This is a planned chapter

Semantic search with Azure OpenAI Java API - Azure OpenAI client library for Java | Microsoft Learn[1], Maven Central[2]. Possibly based on https://github.com/Nasdanika/retrieval-augmented-generation.

Embeddings and indexing

Compute embeddings during loading/generation from Markdown -> HTML -> Jsoup parse -> paragraphs. It will produce a graph of embeddings. The graph can be stored to a variety of stores. For diagrams a vector database might be an overkill - may store to a file , use https://github.com/jelmerk/hnswlib / https://mvnrepository.com/artifact/com.github.jelmerk/hnswlib-core-jdk17 for indexing. Although, even indexing might not be needed for a few thousand embeddings - maybe have some performance comparison.

Take a look/adapt https://github.com/ai-for-java/embeddings4j for an in-memory vector DB / Store. Perhaps have some abstraction `VectorStore` with in-memory implementation and implementations for, say, Milvus (https://milvus.io/, https://milvus.io/docs/install-java.md)

[1]https://learn.microsoft.com/en-us/java/api/overview/azure/ai-openai-readme?view=azure-java-preview
[2]https://mvnrepository.com/artifact/com.azure/azure-ai-openai

Retrieval and generation

Contextual to a graph/diagram node - traverses the graph, collects matches. Final similarity is computed from embdding similarity and graph distance. Graph distance may be computed using different weights for different connection types. E.g. a back-link (incoming connection) may have lower weight than an outgoing connection. Connections to the metamodel may also be traversed. For example (for the family model) Paul -> Man -> Person.

Implementation - Java service and Vue.js client, which is part of the generated UI - navigation action(s) for pages - search/chat. The client passes semantic UUID to the service. The service uses it to find the graph node and do its magic.

Demo

Copy Living Beings demo from Graph to Nasdanika Demos org. Add Wikipedia articles as documentation for all nodes with attribution notes.

Real life applications

System architecture documentation search - release-specific, context-aware.

Provisions for federation. For example, a software component (say, microservice) may reference technology documentation (say, Spring Boot and org-specific guidelines) .

Other models

Different levels of maturity - flow, MCDA, ...

Business case

How diagrams deliver value - diagrammer, mapper, modeler, consumer. Delivery vehicles, federation. Who does what, pace of change, different levels of elaboration. Referenceability.

Ecore DSL - enterprise, mission, language

Java, SpEL, YAML, Markdown - familiar technologies for Spring developers.

“the faintest ink is more powerful than the strongest memory.” Decision binding

Markdown

This chapter explains Markdwon rendering capabilities.

Representations

TODO

Representation filtering

TODO, code snippets, links to demos.

Embedded images

Markdown renderer allows to embed PNG and JPEG using fenced blocks.

PNG resource

~~~
```png-resource
nasdanika-logo.png
```
~~~

Resource location is resolved relative to the base URI.

JPEG resource

~~~
```jpeg-resource
my.jpeg
```
~~~

PNG

~~~
```png
Base 64 encoded png
```
~~~

JPEG

~~~
```jpeg
Base 64 encoded jpeg
```
~~~

Embedded diagrams

Markdown renderer allows to embed PlantUML[1], Draw.io[2], and Mermaid[3] diagrams using fenced blocks. Draw.io diagrams can be edited in a desktop editor or Online editor[4].

Draw.io

[1] https://plantuml.com/
[2] https://www.diagrams.net/
[3] https://mermaid-js.github.io/mermaid/#/
[4] https://app.diagrams.net/

~~~
```drawio-resource
aws.drawio
```
~~~

Resource location is resolved in the same way as for image files as explained above - relative to the base URI.

PlantUML

PlantUML diagrams can be defined inline or loaded from resources.

Loading from a resource

~~~
```uml-resource
sequence.plantuml
```
~~~

In the above snippet `uml` is a dialect supported by PlantUML (see below) and `sequence.plantuml` is a resource containing a diagram definition without `@startuml` and `@enduml`. Resource location is resolved in the same as for image files as explained above.

```
sequence.plantuml
```

Inline

The following language specifications (dialects) are supported:

- `uml` - for the following diagram types:
  - Sequence[5],

[5] https://plantuml.com/sequence-diagram

  - Use Case[6],
  - Class[7],
  - Activity[8],
  - Component[9],
  - State[10],
  - Object[11],
  - Deployment[12],
  - Timing[13],
  - Network[14].

- `wireframe` - for Wireframe diagrams[15]
- `gantt` - for Gantt diagrams[16]
- `mindmap` - for Mind Maps[17]
- `wbs` - for Work Breakdown Structures[18]

UML

Most of the examples below are taken from the PlantUML web site.

Sequence

[6]https://plantuml.com/use-case-diagram
[7]https://plantuml.com/class-diagram
[8]https://plantuml.com/activity-diagram-beta
[9]https://plantuml.com/component-diagram
[10]https://plantuml.com/state-diagram
[11]https://plantuml.com/object-diagram
[12]https://plantuml.com/deployment-diagram
[13]https://plantuml.com/timing-diagram
[14]https://plantuml.com/nwdiag
[15]https://plantuml.com/salt
[16]https://plantuml.com/gantt-diagram
[17]https://plantuml.com/mindmap-diagram
[18]https://plantuml.com/wbs-diagram

~~~
```uml
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response
```
~~~

Component

Component diagram with links to component pages.

~~~
```uml
package Core {
   component Common [[https://docs.nasdanika.org/modules/\
core/modules/common/index.html]]
}

package HTML {
   component HTML as html [[https://docs.nasdanika.org/mo\
dules/html/modules/html/index.html]]
   [html] ..> [Common]
}
```
~~~

Wireframe

Fenced block:

~~~
```wireframe
{
  Just plain text
  [This is my button]
  ()  Unchecked radio
  (X) Checked radio
  []  Unchecked box
  [X] Checked box
  "Enter text here   "
  ^This is a droplist^
}
```
~~~

Gantt

~~~
```gantt
[Prototype design] lasts 15 days and links to [[https://d\
ocs.nasdanika.org/index.html]]
[Test prototype] lasts 10 days
-- All example --
[Task 1 (1 day)] lasts 1 day
[T2 (5 days)] lasts 5 days
[T3 (1 week)] lasts 1 week
[T4 (1 week and 4 days)] lasts 1 week and 4 days
[T5 (2 weeks)] lasts 2 weeks
```
~~~

Mind Map

```
* Debian
** [[https://ubuntu.com/ Ubuntu]]
*** Linux Mint
*** Kubuntu
*** Lubuntu
*** KDE Neon
** LMDE
** SolydXK
** SteamOS
** Raspbian with a very long name
*** <s>Raspmbc</s> => OSMC
*** <s>Raspyfi</s> => Volumio
```

WBS

WBS elements can have links. This type of diagram can also be used to display organization structure.

~~~
```wbs
* [[https://docs.nasdanika.org/index.html Business Proces\
s Modelling WBS]]
** Launch the project
*** Complete Stakeholder Research
*** Initial Implementation Plan
** Design phase
*** Model of AsIs Processes Completed
**** Model of AsIs Processes Completed1
**** Model of AsIs Processes Completed2
*** Measure AsIs performance metrics
*** Identify Quick Wins
** Complete innovate phase
```
~~~

Mermaid

You can define Mermaid[19] diagrams in `mermaid` fenced blocks:

~~~
```mermaid
flowchart LR
  Alice --> Bob & Chuck --> Deb
```
~~~

Loading from a resource

It is also possible to load a diagram definition from a resource resolved relative to the base URI:

~~~
```mermaid-resource
sequence.mermaid
```
~~~

Extensions

- Table of contents[20] - add `[TOC]` to the document as explained in the documentation. This extension will create a table of contents from markdown headers.
- Footnotes[21]
- Strikethrough: `~~strikethrough~~`-> ~~strikethrough~~
- Subscript: `H~2~O` -> H_2O
- Superscript: `2^5^ = 32` -> $2^5 = 32$

[19]https://mermaid-js.github.io/mermaid/#/
[20]https://github.com/vsch/flexmark-java/wiki/Table-of-Contents-Extension
[21]https://github.com/vsch/flexmark-java/wiki/Footnotes-Extension

Custom model

Model

Processors

Maven plug-in

User library

Crystal Ball

How might we?

Innovation, proofs of concepts

Architecture as code

All aspects, federation

Workflow

While there are many workflow engines... people still build custom ones. Reasons - better fit. Textual - hard to read/grasp by business people and even by developers. Options - translate, execute - diagram is a runtime artifact. Composition - Maven dependencies - cross-resource page links, `classpath` scheme

Mapping reference

Mapping

base-uri

This property is used to compute base URI for resolving references such as `doc-ref` and `spec-ref`. The base URI is resolved relative to the logical parent element up to the document (diagram) URI. For pages with links from model element the logical parent the first linking element. For connections with sources, it is the connection source. Otherwise, the element's container (`eContainer()`) is the logical parent.

One way to consistently set base URI's is to check "Placeholders" checkbox and then use `%semantic-id%/` value for base URI.

config-prototype

See `prototype` below.

constructor

Spring Expression Language (SpEL)[1] expression evaluating to a semantic element. Takes precedence over `type`. May return `null` - in this case `type` would be used to create a semantic element, if specified.

This property can be used, for example, to look-up semantic elements defined elsewhere. Say, a list of family members can be

[1] https://docs.spring.io/spring-framework/reference/core/expressions.html

defined in an MS Excel workbook, but family relationships in a diagram.

Another example is “progressive enrichment”. For example, high-level architecture is defined in some model and a diagram uses constructors for already defined architecture elements and `type` for their sub-elements. This approach can be applied multiple times similar to how Docker images are assembled from layers and base images.

In order to implement lookup constructors, override `configureConstructorEvaluationContext()` in sub-classes of `AbstractDrawioFactory` or `createEvaluationContext()` in Drawio resource factories. Set variables from which the constructor expression would obtain semantic elements.

documentation

Property value is used as documentation for semantic elements which extend `Documented`.

doc-format

Documentation format - `markdown` (default), `text`, or `html`. For `doc-ref` is derived from the reference extension if not set explicitly.

doc-ref

Property value is used as a URI to load documentation for semantic elements which extend `Documented`. The URI is resolved relative to the `base-uri`.

feature-map

A YAML map defining how features of the semantic element and related semantic elements shall be mapped.

The map keys shall be one of the following:

container

Mapping specification for the container element in container/contents permutation. Contains one or more of the following subkeys with each containing a map of feature names to a mapping specification or a list of feature names.

- `self` - this element is a container
- `other` - the other element is a container

Example

```yaml
container:
  other: elements
  self:
    elements:
      argument-type: ArchitectureDescriptionElement
      path: 1
```

In the above example `other` specifies that this semantic element shall be added to the `elements` feature of its container regardless of the containment path length. `self` specifies that immediate children of this element (`path: 1`) shall be added to this semantic element `elements` collection if they are instances of `ArchitectureDescriptionElement`

contents

Mapping specification for the contents element in container/contents permutation. Contains one or more of the following sub-keys with each containing a map of feature names to a mapping specification or a list of feature names.

- `self` - this element is contained by the other
- `other` - the other element is contained by this element

end

For connections - mapping specification for the connection end feature to map the connection target semantic element to a feature of the connection semantic element.

self

A map of feature names to a Spring Expression Language (SpEL)[2] expression or a list of expressions evaluating to the feature value or feature element value.

The expression is evaluated in the context of the source diagram element and has access to the following variables:

- `value` - semantic element
- `registry` - a map of diagram element to semantic elements

[2]https://docs.spring.io/spring-framework/reference/core/expressions.html

source

For connections - mapping specification for the connection source feature to map the connection semantic element to a feature of the connection source semantic element. If the connection semantic element is `null`, then the connection source semantic element is used instead.

target

For connections - mapping specification for the connection target feature to map the connection semantic element to a feature of the connection target semantic element. If the connection semantic element is `null`, then the connection target semantic element is used instead.

start

For connections - mapping specification for the connection start feature to map the connection source semantic element to a feature of the connection semantic element.

feature-map-ref

Property value is used as a URI to load feature map YAML specification. The URI is resolved relative to the `base-uri`.

mapping-selector

Mapping selector is used to select one or more semantic elements for feature mapping. If it is not provided, then the diagram element's semantic element is used for feature mapping.

Mapping selector shall be a YAML document containing either a single string or a list of strings. The strings are treated as Spring Expression Language (SpEL)[3] expression evaluating to a semantic element to use for feature mapping.

Mapping selectors may be used to associate multiple semantic elements with a diagram element for feature mapping purposes.

mapping-selector-ref

Property value is used as a URI to load mapping selector YAML. The URI is resolved relative to the `base-uri`.

operation-mapping

Allows to invoke semantic element operations for mapping purposes. For example, if a `Course` diagram element contains `Student` elements then `Course.enroll(Student)` operation may be used for mapping. The operation may check for course capacity, validate student status, etc.

Operation mapping is a map of operation names to operation specification. Operation specification can be either a map or a list of maps. The first case is treated as a singleton list.

The operation specification map supports the following entries:

arguments

A map of parameter names to SpEL expression evaluating their values in the context of the iterator element (see below). Argument

[3]https://docs.spring.io/spring-framework/reference/core/expressions.html

names are used for operation selection/matching - a candidate operation must have parameters with matching names.
The map does not have to contain entries for all operation parameter. Nulls are used as arguments for parameters which are not present in the map.

iterator

An optional SpEL expression which returns a value to iterate over and invoke the operation for every element. If the result is `java.util.Iterator` then it is used AS-IS. If the result is Iterable, Stream, or array, an iterator is created to iterate over the elements. If the result is null, then an empty iterator is created. Otherwise, a singleton iterator is created wrapping the result.

It allows to invoke the operation zero or more times. E.g. `enroll` for every student. If not defined, the iterator contains the source diagram element.

pass

An optional integer specifying the pass in which this operation shall be invoked. Use for ordering operation invocations.

For example, in the Nature model[4] you may want the Fox[5] to eat[6] the Hare[7], but only after the Hare eats the Grass[8]. In this case you don't specify the pass for `Rabbit.eats()`, so it defaults to zero. An you set the pass for `Fox.eats()` to `1`.

[4]https://nature.models.nasdanika.org/index.html
[5]https://nature.models.nasdanika.org/references/eClassifiers/Fox/index.html
[6]https://nature.models.nasdanika.org/references/eClassifiers/Animal/references/eOperations/eats-1/index.html
[7]https://nature.models.nasdanika.org/references/eClassifiers/Hare/index.html
[8]https://nature.models.nasdanika.org/references/eClassifiers/Grass/index.html

selector

An optional SpEL boolean expression evaluated in the context of the operation to disambiguate overloaded operations.

operation-mapping-ref

Property value is used as a URI to load operation mapping YAML. The URI is resolved relative to the `base-uri`.

page-element

`true` is for true value and any other value or absence of the property is considered false. There should be one page element per page. Having more than one may lead to an unpredictable behavior. For the first top level page the page element becomes a document element. For pages linked from model elements the page element is logically merged with the linking element.

This allows to define a high-level structure on a top level diagram page, link other pages to the diagram elements and refine their definitions.

If the semantic element of a page element extends `NamedElement` then the page name is used as element name if it is not already set by other means.

processor

A SpEL expression evaluating to zero or more `org.nasdanika.drawio.model.util.AbstractDrawioFactory.Processor`s. The expression can evaluate to `null`, a processor instance, iterator, iterable, array, or stream. It is evaluated in the context of the diagram element with `semanticElement`, `pass`, and `registry` variables.

prototype

Spring Expression Language (SpEL)[9] expression evaluating to a diagram element. The semantic element of that diagram element is copied and the copy is used as the semantic element of this diagram element. Also, prototype configuration (properties) is applied to this semantic element. Protypes can be chained.

Example: `getDocument().getModelElementById('6ycP1ahp__-4fXEwP3E-2-5')`

Selectors allow to define common configuration in one element and then reuse it in other elements. For example, a web server prototype may define an icon and then all web server element would inherit that configuration.

Drawio model classes provide convenience methods for finding diagram elements:

- Document[10]:
  - `getModelElementById(String id)`
  - `getModelElementByProperty(String name, String value)`
  - `getModelElementsByProperty(String name, String value)`
  - `getPageById(String id)`
  - `getPageByName(String name)`
- Page[11]:

[9] https://docs.spring.io/spring-framework/reference/core/expressions.html
[10] https://drawio.models.nasdanika.org/references/eClassifiers/Document/operations.html
[11] https://drawio.models.nasdanika.org/references/eClassifiers/Page/operations.html

  - `getModelElementById(String id)`
  - `getModelElementByProperty(String name, String value)`
  - `getModelElementsByProperty(String name, String value)`
  - `getTag(String name)`

- ModelElement[12]:

  - `getDocument()`
  - `getPage()`

- Root[13], Layer[14]:

  - `getModelElementById(String id)`
  - `getModelElementByProperty(String name, String value)`
  - `getModelElementsByProperty(String name, String value)`

A prototype must have a semantic element defined using `constructor`, `type`, `selector` or `ref-id`. If you want to inherit just configuration, but not the semantic element, then use `config-prototype` property instead of `prototype`.

ref-id

Some identifier to resolve a semantic element. You would need to override `getByRefId()` method and define what the reference id means in your case. For example, you may use semantic URI's to lookup elements. Say `ssn:123-45-6789` to lookup a person by SSN.

[12]https://drawio.models.nasdanika.org/references/eClassifiers/ModelElement/operations.html
[13]https://drawio.models.nasdanika.org/references/eClassifiers/Root/operations.html
[14]https://drawio.models.nasdanika.org/references/eClassifiers/Layer/operations.html

Resource factories treat ref-id's as URI's resolved relative to the diagram resource URI. Resolved URI's are then are passed to `ResourceSet.getEObject(URI uri, true)`.

reference

Associates a diagram element with a reference of the semantic element of the first matched logical ancestor for mapping purposes. If the element has mapped descendants, their matching semantic elements are added to the reference.

Reference value can be a string or a map. The string form is equivalent to the map form with just the `name` entry.

The map form supports the following keys:

- `comparator` - see `comparator` in "Feature mapping reference".
- `condition` - a SpEL `boolean` expression evaluated in the context of the logical ancestor semantic element. If not provided, matches the first mapped ancestor. Has access to the following variables:
  - `sourcePath` - a list of logical ancestors starting with the logical parent
  - `registry`
- `expression` - a SpEL `EObject` expression evaluated in the context of the logical ancestor semantic element. If not provided, the logical ancestor itself is used. Has access to the following variables:
  - `sourcePath` - a list of logical ancestors starting with the logical parent
  - `registry`

- `name` - reference name
- `element-condition` - a SpEL `boolean` expression evaluated in the context of the descendant (contents) semantic element. If not provided, elements are matched by type. Has access to the following variables:
  - `sourcePath` - source containment path
  - `registry`
- `element-expression` - a SpEL `EObject` expression evaluated in the context of the descendant (contents) semantic element. If not provided, the descendant itself is used. Has access to the following variables:
  - `sourcePath` - source containment path
  - `registry`

script

Script to execute. The script has access to the following variables:

- `baseURI`
- `diagramElement`
- `logicalParent`
- `pass`
- `registry`
- `semanticElement`

Additional variables can be introduced by overriding `configureScriptEngine()` method. Script class loader can be customized by overriding `getClassLoader()` method.

script-engine

A boolean SpEL expression for selecting a script engine. The expression is evaluated in the context of a candidate `javax.script.ScriptEngineFactory` with the following variables:

- `diagramElement`
- `pass`
- `registry`
- `semanticElement`

script-ref

Script reference resolved relative to the base URI.

selector

Spring Expression Language (SpEL)[15] expression evaluating to a diagram element. The semantic element of that diagram element is used as the semantic element of this diagram element.

Example: `getDocument().getModelElementById('6ycP1ahp__-4fXEwP3E-2-5')`

Selectors allow to use the same semantic elment on multiple diagrams. Drawio model classes provide convenience methods for finding diagram elements:

- Document[16]:

[15] https://docs.spring.io/spring-framework/reference/core/expressions.html
[16] https://drawio.models.nasdanika.org/references/eClassifiers/Document/operations.html

  - `getModelElementById(String id)`
  - `getModelElementByProperty(String name, String value)`
  - `getModelElementsByProperty(String name, String value)`
  - `getPageById(String id)`
  - `getPageByName(String name)`

- Page[17]:

  - `getModelElementById(String id)`
  - `getModelElementByProperty(String name, String value)`
  - `getModelElementsByProperty(String name, String value)`
  - `getTag(String name)`

- ModelElement[18]:

  - `getDocument()`
  - `getPage()`

- Root[19], Layer[20]:

  - `getModelElementById(String id)`
  - `getModelElementByProperty(String name, String value)`
  - `getModelElementsByProperty(String name, String value)`

[17] https://drawio.models.nasdanika.org/references/eClassifiers/Page/operations.html
[18] https://drawio.models.nasdanika.org/references/eClassifiers/ModelElement/operations.html
[19] https://drawio.models.nasdanika.org/references/eClassifiers/Root/operations.html
[20] https://drawio.models.nasdanika.org/references/eClassifiers/Layer/operations.html

semantic-id

If the semantic element extends `StringIdentity`, `semantic-id` property can be used to specify the `id` attribute. If this property is not provided, then Drawio model element ID is used as a semantic ID.

semantic-selector

Spring Expression Language (SpEL)[21] expression evaluating to a semantic element.

Semantic selectors are similar to constructors with the following diffences:

- Semantic selectors are evaluated after constructors
- A constructor may evaluate to `null`, but a semantic selector must eventually evaluate to a non-null value
- A constructor is evaluated once, but a semantic selector maybe evaluated multiple time until it returns a non-null value
- A semantic selector is only evaluated if there isn't a semantic element already

Semantic selectors can be used to evaluate semantic elements using semantic elements of other elements. For example, a semantic selector of a child node may need a semantic element of its parent to resolve its own semantic element.

Overide `configureSemanticSelectorEvaluationContext()` to provide variables to the evaluator.

[21]https://docs.spring.io/spring-framework/reference/core/expressions.html

spec

Loads semantic element from the property value YAML using EObjectLoader.

Example:

```yaml
icon: fas fa-user
```

spec-ref

Loads semantic element from the property value URI using EObjectLoader. The URI is resolved relative to the `base-uri`.

tag-spec

TODO

tag-spec-ref

TODO

top-level-page

Page elements from top level pages are mapped to the document semantic element. By default a page without incoming links from other pages is considered to be a top level page. This property allows to override this behavior. `true` value indicates that the page is a top level page. Any other value is treated as `false`.

type

Type of the semantic element. Types are looked up in the factory packages in the following way:

- If the value contains a hash (`#`) then it is treated as a type URI. For example `ecore://nasdanika.org/models/family#//Man`.
- If the value contains a doc (`.`) then it is treated as a qualified EClass name with EPackage prefix before the dot. For example `family.Man`.
- Otherwise the value is treated as an unqualified EClass name - EPackages are iterated in the order of their registration and the first found EClass with matching name is used to create a semantic element.

Type is used to create a semantic element if there is no `constructor` or the constructor expression returned `null`. A combination of `constructor` and `type` can be used for mapping in different contexts. For example, when loading a stand-alone model `constructor` would return `null` and then `type` would be used. When the same diagram loaded in the context of a larger model, `constructor` may return a semantic element looked up in that larger model.

Namespaces

It is possible to maintain multiple mappings at a single element using namespace prefixes. In subclasses of `AbstractDrawioFactory` override `getPropertyNamespace()` method. By default this method returns an empty string which is used as a prefix for the configuration properties.

Feature mapping reference

Feature mapping value can be either a string or a map. If it is a string it is treated as a singleton map to `true` (unconditional mapping).

The below two snippets are equivalent:

```yaml
container:
  other: elements
```

```yaml
container:
  other:
    elements: true
```

The map value supports the following keys:

argument-type

Specifies type of feature elements to be set/added. String as defined in `type`.. Only instances of the type will be set/added. If absent, the feature type is used. Argument type can be used to restrict elements to a specific subtype of the feature type.

comparator

Comparator is used for “many” features to order elements. A comparator instance is created by `createComparator()` method which can be overridden in subclasses to provide support for additional comparators.

The following comparators are provided “out of the box”:

clockwise

Compares elements by their angle relative to the node of the semantic element which holds the many reference. In the Living Beings[1] demo “Bird”, “Fish”, and “Bacteria” are compared by their angle to the “Living Beings” with the angle counted from “12 o’clock” - 90 degrees (default).

Feature mapping with comparators of “Bird”, “Fish”, and “Bacteria” are defined at the connections from “Living Beings” as:

```
source:
  elements:
    comparator: clockwise
```

To specify the base angle other than 90 degree use the map version of comparator definition where `clockwise` is the key mapping to a number or string value. The number value is used as the angle value in degrees. The string value is treated as a Spring Expression Language (SpEL)[2] expression evaluated in the context of the “parent” node. The expression may evaluate to a number or to a node. In the latter case the result is used to compute the angle between the context node and the result node.

In the Living Beings example “Streptococcus”, …, “Staphyllococcus” are compared relative to the “Bacteria” node with the base angle being the angle between the “Bacteria” node and “Living Beings” node. As such “Streptococcus” is the smallest node and “Staphyllococcus” is the largest. With the default angle of 90 degrees

[1]https://graph.models.nasdanika.org/demo/living-beings/index.html
[2]https://docs.spring.io/spring-framework/reference/core/expressions.html

“Lactobacyllus” would be the smallest and “Streptococcus” would be the largest.

Feature mapping with comparators of “Streptococcus”, ..., “Staphyllococcus” is defined at connections from “Bacteria” to the respecive genus nodes as:

```
source:
  elements:
    comparator:
      clockwise: incoming[0].source
```

`incoming[0]` evaluates to the connection from “Living Beings” to “Bacteria” and `source` evaluates to “Living Beings”.

counterclockwise

Reverse of `clockwise`.

down-left

Compares nodes by their vertial order first with higher nodes being smaller and then by horizontal order with nodes on the right being smaller. Nodes are considered vertically equal if they vertically overlap.

down-right

Compares nodes by their vertial order first with higher nodes being smaller and then by horizontal order with nodes on the left being smaller. Nodes are considered vertically equal if they vertically overlap. This comparator can be used for org. charts.

expression

A Spring Expression Language (SpEL)[3] expression evaluated in the context of the feature element with `other` variable referencing the element to compare with. The exmpression has access to `registry` variable containing a map of diagram element to semantic elements.

flow

If one element is reacheable from the other by traversing connections, then the reacheable element is larger than the source element. In case of circular references the element with the smaller number of traversals to the other element is considered smaller. If elements are not connected they are compared by the fall-back comparator. This comparator can be used for workflows and PERT[4] charts.

If this comparator's value is a String, then it is uses as a name of the fallback comparator. In the below example children will be smaller than their parents and siblings will be compared using labels.

```yaml
container:
  self:
    members:
      argument-type: Person
      comparator:
        flow: label
```

If the value is a map, then it may have the following keys:

- `condition` - A boolean Spring Expression Language (SpEL)[5] expression evaluated in the context of a connection being

[3]https://docs.spring.io/spring-framework/reference/core/expressions.html
[4]https://en.wikipedia.org/wiki/Program_evaluation_and_review_technique
[5]https://docs.spring.io/spring-framework/reference/core/expressions.html

traversed. It may be used to traverse only connections which match the condition. For example, only transitions[6] between activities[7] in a process model.
- `fallback` - Fallback comparator.

In the below example…

key

A Spring Expression Language (SpEL)[8] expression evaluated in the context of the feature element. The expression must return a value which would be used for comparison using the natural comparator as explained below.

label

Uses diagram element label converted to plain text as a sorting key. String. In the Family mapping demo[9] family members are sorted by label using the following feature map definition:

```yaml
container:
  self:
    members:
      argument-type: Person
      comparator: label
```

label-descending

Uses diagram element label converted to plain text as a sorting key to compare in reverse alphabetical order. String.

[6]https://flow.models.nasdanika.org/references/eClassifiers/Transition/index.html
[7]https://flow.models.nasdanika.org/references/eClassifiers/Activity/index.html
[8]https://docs.spring.io/spring-framework/reference/core/expressions.html
[9]https://family.models.nasdanika.org/demos/mapping/index.html

left-down

Compares nodes by their horizontal order first with nodes on the right being smaller and then by vertial order with higher nodes being smaller. Nodes are considered horizontally equal if they horizontally overlap.

left-up

Compares nodes by their horizontal order first with nodes on the right being smaller and then by vertial order with lower nodes being smaller. Nodes are considered horizontally equal if they horizontally overlap.

natural

Uses feature element's `compareTo()` method for comparable elements. Otherwise compares using hash code. Null are greater than non-nulls.

property

Uses diagram element property as a sorting key. Singleton map. For example:

```
property: label
```

property-descending

The same as property, but compares in reverse alphabetical order.

reverse-flow

Same as `flow` but with target nodes being smaller than source nodes.

right-down

Compares nodes by their horizontal order first with nodes on the left being smaller and then by vertial order with higher nodes being smaller. Nodes are considered horizontally equal if they horizontally overlap.

right-up

Compares nodes by their horizontal order first with nodes on the left being smaller and then by vertial order with lower nodes being smaller. Nodes are considered horizontally equal if they horizontally overlap.

up-left

Compares nodes by their vertial order first with lower nodes being smaller and then by horizontal order with nodes on the right being smaller. Nodes are considered vertically equal if they vertically overlap.

up-right

Compares nodes by their vertial order first with lower nodes being smaller and then by horizontal order with nodes on the left being smaller. Nodes are considered vertically equal if they vertically overlap.

condition

A Spring Expression Language (SpEL)[10] boolean expression evaluated in the context of the candidate diagram element with the following variables:

- `value` - semantic element of the candidate diagram element
- `path` - containment path
- `registry` - a map of diagram element to semantic elements

path

Either an integer number 0 or a list of boolean SpEL expressions to match the path. If an integer then it is used to match path length as shown in the example below which matches only immediate children

```yaml
container:
  self:
    elements:
      path: 1
```

If a list, then it matches if the list size is equal to the path length and each element evaluates to true in the context of a given path element. Expression have acess to `registry` variable - a map of diagram element to semantic elements.

[10] https://docs.spring.io/spring-framework/reference/core/expressions.html

nop

If `true`, no mapping is performed, but the chain mapper is not invoked. It can be used in scenarios with a default (chained) mapper to prevent the default behavior.

expression

A SpEL expression evaluating to a feature value in the context of the diagram element with with the following variables:

- `value` - semantic element of the diagram element
- `path` - containment path
- `registry` - a map of diagram element to semantic elements

greedy

Greedy is used with containment feature and specifies what to do if a candidate object is already contained by another object:

- `no-children` - grab the object if it is contained by an ansector of this semantic element. This is the default behavior.
- `false` - do not grab
- `true` - always grab

position

A number specifying the position of the element in the feature collection.

type

Type of the feature object to match. String as defined in `type`. Can be used in `other` mappings.