Dynamic Symbol Templates and Ports in MetaEdit+

Steven Kelly
MetaCase
Ylistönmäentie 31
40500 Jyväskylä, Finland
+358 14 641000 ext. 21
stevek@metacase.com

Risto Pohjonen
MetaCase
Ylistönmäentie 31
40500 Jyväskylä, Finland
+358 14 641000 ext. 27
rise@metacase.com

ABSTRACT

A graphical language definition is often divided into its abstract syntax and concrete syntax. The concrete syntax of a DSM language used on paper or in a drawing tool is often rich and varied, whereas many language workbenches only easily support rather simple boxes or icons, with more complex symbols quickly hitting a "customization cliff" and requiring manual programming. In this demonstration, we show the new dynamic symbol functions, templates and ports in MetaEdit+ 5.0. These extend the WYSIWYG symbol definition of MetaEdit+ with iterative and recursive possibilities, useful both for building common complex symbols and opening up new possibilities.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Computer-aided software engineering (CASE).
D.2.6 [**Programming Environments**]: Graphical environments

General Terms

Design, Human Factors, Languages.

Keywords

demonstration, language workbench, concrete syntax.

1. INTRODUCTION

Standard wisdom on graphical languages splits a language definition into its abstract syntax and concrete syntax. The concrete syntax of a DSM language used on paper or in a drawing tool is often rich and varied, whereas many language workbenches only easily support rather simple boxes or icons, with more complex symbols quickly hitting a "customization cliff" and requiring manual programming.

MetaEdit+ has always allowed WYSIWYG specification of complex graphical symbols, with dynamic elements that can be revealed based on the model data, and subobject symbols similar to GMF's Compartments [1]. Subobjects can either be freeform in the diagram, e.g. in UML Use Case Systems, or as lists in the symbol with definable alignment, e.g. a UML Class's Attributes.

MetaEdit+ 5.0 [2] adds a new symbol element, Template, which allows symbols to have elements that can repeat in more complex patterns and contain various content, including recursive symbols. The templates offer declarative definition for the common cases, with the possibility of using the MetaEdit+ Reporting Language for specifying more complex behavior. In this demonstration, we show the new dynamic symbol functions, templates and ports in MetaEdit+ 5.0, both for building common complex symbols and opening up new possibilities.

2. BACKGROUND

The GOPPRR meta-metamodel of MetaEdit+ is designed for graphical modeling languages, with concepts of Graph, Object, Property, Relationship, Role and Port. Roles form the endpoints of relationships, and ports are points or areas on the perimeter of an object that roles can connect to with specific semantics. Before version 5.0, ports were defined statically in the metamodel: an object type might be defined to have an In port and an Out port, or in a more complex case an Electrical port with properties Direction: In, Signal type: Analog, and Voltage: +5V. Constraints can be specified for how points can be connected, e.g. an Out port can only be connected to an In port, or an Electrical port with Signal type: Analog can only be connected to another Electrical port with Signal type: Analog.

These ports thus covered a wide range of languages, in addition to the even wider range of languages already covered by rules on relationships and roles, with no ports needed. However, a certain class of port usage was not supported: where the ports arise dynamically in an object, depending on other information in the model – the list of Attributes in a Class, allowing roles to connect to an Attribute (Figure 1), or the set of Interface objects defined in a subgraph of a Component object.

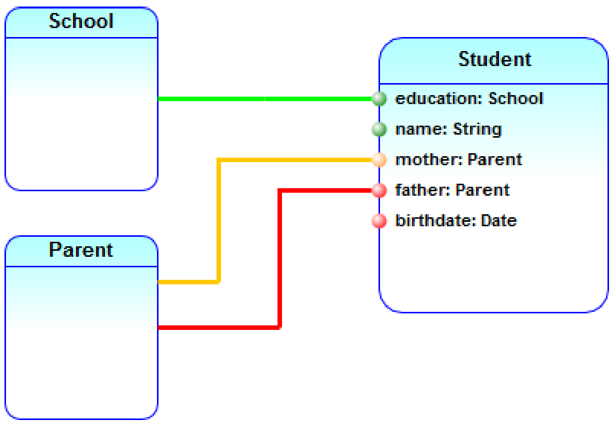


Figure 1: Simple attribute-based dynamic ports

In these cases, the symbol must update with some kind of repeating element, depending on the number of ports it has in the model. Unlike the static ports, such dynamic ports arise only at modeling time, and cannot be specified by the metamodeler. Instead, MetaEdit+ must provide the metamodeler with facilities for specifying how to find the dynamic ports for an object, obtain a small port subsymbol to represent each, and lay out the port symbols as part of the object symbol. We called the new symbol element that implemented these facilities Template. In keeping with the WYWIWYG nature of the MetaEdit+ Symbol Editor, as much as possible of the Template definition is visual and interactive: the iterative layout of the subobjects and the space allocated for each can be edited like any other vector graphic element. Figure 2 shows a Template for ports that would appear on the left, bottom and right sides of the green object symbol, in a small box for each (the dotted blue box shows the area allocated, the actual choice of symbol to display there is made at runtime).

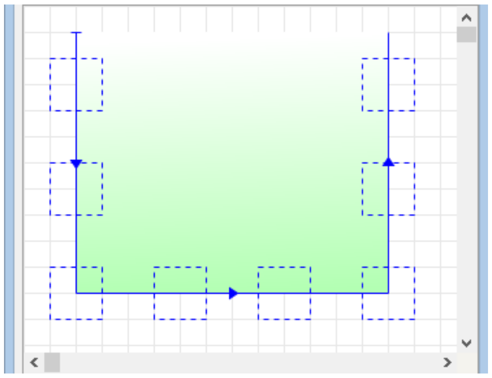


Figure 2. Template layout

During the development of these Template facilities, we noticed that some "ports" may want to appear graphically but not allow connection. For instance, some UML tools display colored ball symbols similar to Figure 1, but within the symbol and not allowing connection. This gave rise to the idea of using the Template facilities to allow the specification of iterating and/or recursive parts of symbols in the more general non-port case. We thus allowed each Template to specify whether it resulted in just subsymbols or also ports.

3. DEMONSTRATION

In the demonstration we will look at some motivating examples from real and representative DSM languages, and show how these can be accomplished in MetaEdit+ 5.0, and how they contributed to the requirements for the Template design and implementation. Some of the examples we will cover can be seen in [3] and [4], others will be new for this demonstration.

As an example, Figure 3 shows the definition for the Class symbol in Figure 1. This symbol uses two overlaid templates to provide a complex visualization: the template on the right takes care of displaying the normal list of attributes with their names etc., whereas the template on the left is used to display the colored balls. Both templates use the same method to find the subobjects, collecting them from the Class's list of Attributes, as shown in Figure 4. The left template also offers its subobjects as ports: roles can connect to the colored balls.

Making these facilities available via a WYSIWYG Symbol Editor and dialogs offering simple selections for common cases enables new users to get started more quickly and proceed more efficiently than if they were provided only as part of a programming framework. Experience to date shows that the Template facilities are able to concisely specify a wide range of needs found in actual languages, and new users have been able to use them in ways we had not even envisaged, learning solely from the supplied documentation.

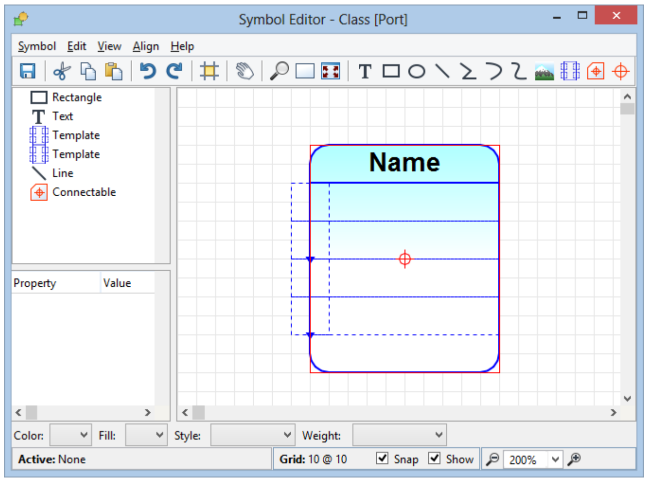
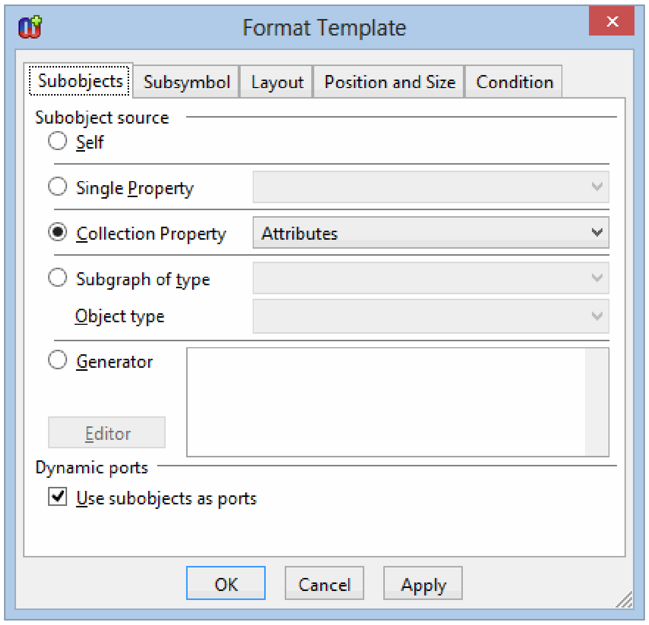


Figure 3. Definition of the symbol for Figure 1



Subobject source
- Self
- Single Property
- [✓ Collection Property] Attributes
- Subgraph of type
- Object type
- Generator
- Editor

Dynamic ports
- [✓ Use subobjects as ports]

Figure 4. Specifying where to find the dynamic ports from

REFERENCES

[1] Eclipse. *GMF Tutorial*. Retrieved 3.10.2013. http://wiki.eclipse.org/GMF_Tutorial_Part_2

[2] MetaCase. *MetaEdit+ Version 5.0 Workbench User's Guide*. Retrieved 18.8.2013. http://metacase.com/support/50/manuals/mwb/Mw.html

[3] Kelly, S., Pohjonen, R. 2011. *MetaEdit+ 5.0 Beta Primer*. Retrieved 18.8.2013. http://www.metacase.com/download/metaedit/MetaEdit+%205.0%20Beta%20Primer.pdf

[4] MetaCase. *Advanced dynamic symbols and ports in MetaEdit+*. Retrieved 18.8.2013. http://www.metacase.com/webcasts/EnhancedDynamicBehaviorWithTemplates.html