Concrete Syntax-Based Find for Graphical DSLs

Elina Kalnina
Institute of Mathematics and Computer Science, University of Latvia
Riga, Latvia
elina.kalnina@lumii.lv

ABSTRACT

There are services available in the most software tools we have got used to like, copy, paste, cut, find, and replace. However, the state of the art is not so good with tools of graphical languages. Even many commercial modelling tools have limited support of the find feature. We propose to add find as a service of graphical DSL tool development frameworks. This way find is available in any DSL built using the DSL tool development framework. The concrete syntax-based find has been implemented as a service of the DSL tool development framework ajoo. Two graph-based languages: UML Activity diagrams and Deterministic Finite Automata (DFA) transition diagrams are used to demonstrate usage of the concrete syntax-based find.

CCS CONCEPTS

• **Computing methodologies** → **Modeling methodologies**.

KEYWORDS

Domain-specific languages, find, Domain-specific modelling languages, DSL tool development frameworks

ACM Reference Format:
Elina Kalnina. 2020. Concrete Syntax-Based Find for Graphical DSLs. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3417990.3422008

1 INTRODUCTION

The find is an essential feature of a modern user interface. Users use the find for various purposes e.g to navigate to a certain element, to find an answer to a specific question or to explore to obtain new information [13]. However, the support of the find is limited in the area of graphical modelling. It is true even for commercial modelling tools. There is an opinion that one of the main reasons why MDE (Model-Driven Engineering) has not reached the expected acceptance in the industry is the poor quality of tool support [20]. We propose an approach on how to improve tool support for find feature. Typically there is only some kind of textual search available in the modelling tools. However, textual search uses only textual parts of the model and it is not aware of the structure of the model and graphical elements. Besides, it is not even possible to define that we should look for the text as a value of the particular property of a specific element. Instead, we propose to use a fragment of a diagram as a specification of a find query. We have implemented the approach in the DSL tool development framework ajoo [18]. As a service of a DSL tool development framework, the find is available for any modelling language built using DSL tool development framework. The DSL tool development frameworks are used to ease the creation of an editor for graphical languages. There are multiple DSL tool development frameworks, for example, Sirius [3], MetaEdit+ [10]. Although we discuss implementation in the ajoo framework [18], the approach is universal and could be implemented in any DSL tool development framework. We discuss implementation in other frameworks in subsection 4.3. We explain the main principles of the concrete syntax-based find in Section 2. We use two different graphical languages in Section 3 to demonstrate how the approach could be applied. We discuss implementation principles in Section 4. Related work is given in Section 5.

2 CONCRETE SYNTAX-BASED FIND

We propose to provide the concrete syntax-based find as a service of a DSL tool development framework. The general ideas have been described in [7]. Since then, the approach has been implemented in the ajoo framework [18] and has evolved in some aspects. In this section, we repeat the main principles. We accent the aspects where the approach has evolved.

The approach is intended for graph-based Domain-Specific Languages, therefore only such languages are considered. The diagrams in such language can be treated as typed attributed graphs [4]. The element types are used to determine the match of the find query.

The element (node and edge) types are defined when creating DSL in a DSL tool development framework. The encoding used in the ajoo framework [18] is shortly described in section 4. In other DSL tool development frameworks different encoding is used, but in some sense, element types persist in all of them. The find query is defined by drawing (or selecting) fragment of a DSL diagram. The graph structure and element types that appear in the find query are used to find other diagrams with the same graph structure and element types.

Besides, it is possible to set constraints on the attribute values of the diagram elements. Currently, only substring or equality constraints are supported. Constraint on an attribute value means that only elements with appropriate element type and satisfying conditions defined by constraint will be considered as matches of the query.

Any DSL diagram can be used to define a find query. Therefore there are no specific find diagrams. The only find specific feature required is a find button. (see Figure 1)

Since concrete syntax-based find is a service of DSL tool development framework, it is available for any DSL built with the DSL tool development framework.

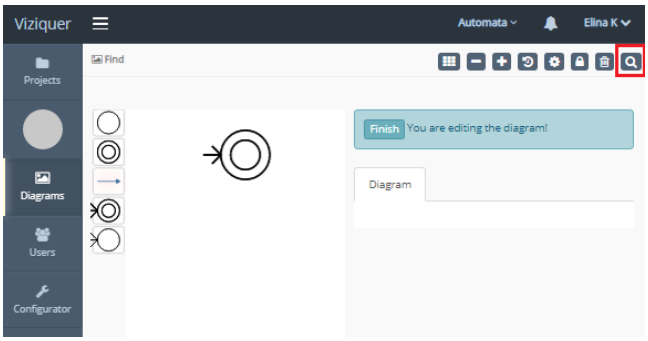

Figure 1: Diagram editor with the "find" button.

3 EXAMPLES

In this section, we demonstrate how the approach could be applied for two different DSLs built with the DSL tool development framework ajoo [18]. The first example is simplified UML activity diagrams [15]. The second example is the Deterministic Finite Automata Transition diagrams [6].

3.1 UML Activity Diagrams

The first example is simplified UML activity diagrams [15] which can be used to describe the flow of processes, performers and other features. The main element of the UML activity diagram is an action. The two most important properties of the action element are the name and performer. The simple use case of find query could be to find all actions performed by a certain actor. For example, a user would like to know what he has to do in the process, or developers of a system would like to know, what the system should do.

Let's look at a simple UML Activity diagram example in Figure 4. The example describes how the transportation tickets are sold by ticket vending machine. The find query which is looking for actions performed by a ticket vending machine is given in Figure 2.

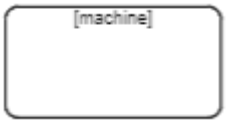

Figure 2: Find query. Find all actions performed by "machine".

The find query is drawn as a new DSL diagram. In the example, it is a UML activity diagram. The query is executed by pressing the "Find" button. The result of the query is a list of diagrams containing elements satisfying conditions defined by the query. Find results are shown in Figure 3. The number of matched diagrams is shown. For each diagram, the number of elements satisfying condition is given. The list of diagrams is clickable. It opens the diagram, where matched elements are highlighted. An example is given in Figure 4.

Figure 3: Find results of find query given in Figure 2.

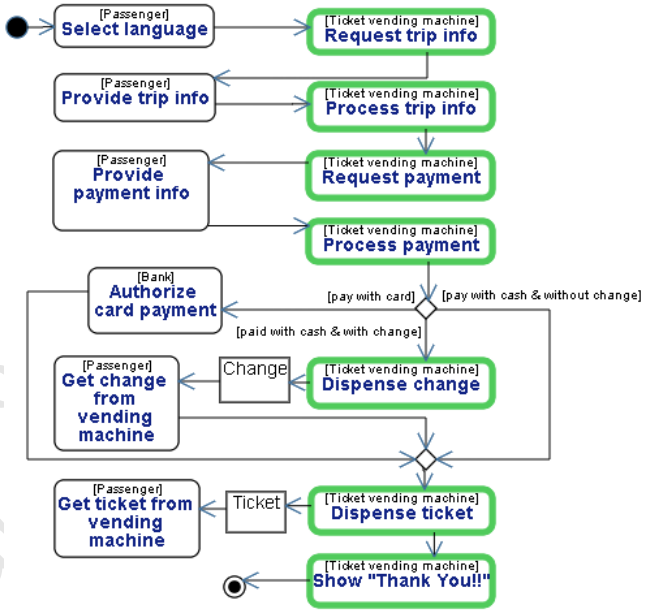

Figure 4: The find results in Figure 3 highlighted in a diagram.

The example is very simple and initially, it may seem that it is possible to do the same using textual search. However, the results of textual find may be different as textual find will reveal matches of name "machine" in any context, e.g. conditions, action name, UML objects etc. However, concrete syntax-based find looks only for "machine" as a performer of an action. Thus the structure of the UML Activity diagram is taken into account. In the example given in Figure 4, the textual search would match two additional nodes: "Get change from vending machine" and "Get ticket from vending machine". It will happen as the actions contain the name "machine" in the action names.

If required it is possible to extend query with other constraints, e.g. restrict name of an action, looking for a specific type of relation, etc. An example of the more complicated query structure is given in the next section.

3.2 Deterministic Finite Automata

The second example is Deterministic Finite Automata (DFA) Transition diagrams [6]. We have created a set of DFA-s. The find query will contain a fragment of the DFA and we use it to find DFA-s from the set where the given fragment has been used.

For example, we may look for automata accepting words $\{\epsilon, 1, 10\}$. The query to find such DFA is given in Figure 5. Since we want a DFA to accept an empty string, the initial state of the DFA is the accepting state. Since we want to accept the word "1", we need transition with the symbol "1" . Since we want to accept the word "10" too, the previous transition should be followed by a transition with the symbol "0" to the accepting state.

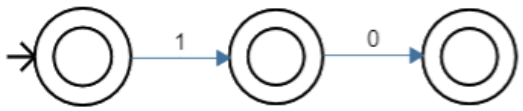

Figure 5: Find query. Find all DFA-s accepting words $\{\epsilon, 1, 10\}$.

Two DFA-s from the set satisfies the condition. The results are shown in Figure 6. As already stated, it is possible to see the matched pattern highlighted in the diagram. Two such examples are given in Figure 7 and Figure 8.

Figure 6: Find results of find query given in Figure 5.

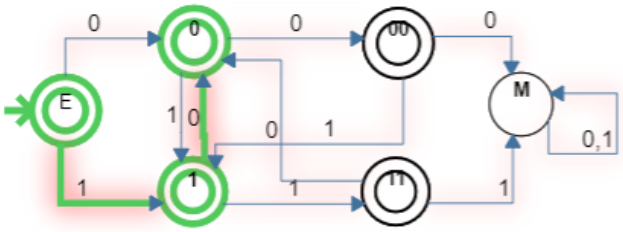

Figure 7: The find results in Figure 6 highlighted in a diagram.

In figure 8 the same node is used as a match for multiple find diagram elements. If multiple elements of the same type are required in the find diagram it matches also recursive structures. However, if the find query, contains a recursive structure, then only recursive structures are matched.

It is possible to use the proposed approach for a more complicated graph structure as well.

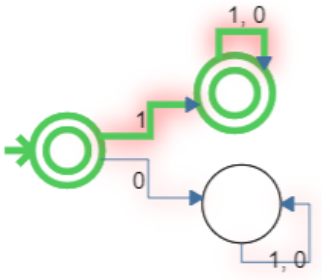

Figure 8: The find results in Figure 6 highlighted in a diagram.

4 IMPLEMENTATION

In this section, we describe the implementation of the proposed approach in a DSL tool development framework ajoo [18]. The approach could be implemented in any graphical DSL tool development framework. However, it should be noted, that the encoding of the DSL is used in the execution of a find query. The encoding used by tools differs, therefore to implement the approach in another tool the find algorithm should be adapted to the encoding used by the tool. The approach should also be adapted to the architecture used by the DSL tool development framework.

Ajoo framework is a web-based tool, it works in a browser. The ajoo is based on the meteor [17] framework. The data are stored in MongoDB [12]. The framework works in a client-server architecture. The find query execution algorithm is implemented in the server-side. The result representation is implemented in the client-side.

4.1 The Find Algorithm

The implementation of the find algorithm uses data encoding of the ajoo framework [18]. The data encoding is shown in Figure 9. The data structure of the ajoo framework is more complicated, but we show only classes, relations and properties which are important in the context of the find implementation. The simplified metamodel of the ajoo platform is shown in Figure 9.

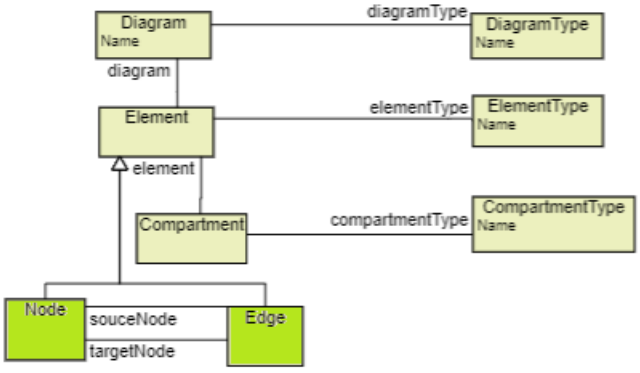

Figure 9: Data encoding used in the ajoo framework.

Ajoo diagrams consist of elements. Since we consider graph diagrams, then there are nodes and edges in the ajoo. There are

element types for elements and diagram types for diagrams. The element type defines the style of the element as well as the properties and tooling - property editors, palette element, etc.. The diagram type defines supported element types in the diagram. There are also compartments and compartment types corresponding to attributes (textual parts) of an element in a diagram. A particular compartment is connected to the particular diagram element. The Yellow boxes correspond to document collections used in the MongoDB.

The find diagram and other DSL diagrams (potential matches of a query) have exactly the same data encoding – they both are DSL diagrams.

The element types are used to find matches of the find query. The element in a DSL diagram is considered a potential match of an element in the find diagram if their types (ElementType in Figure 9) are equal. The edge is considered a potential match if edge types are equal (ElementType in Figure 9) and types of source and target node of an edge are equal as well.

The algorithm simultaneously works with the find diagram or the find query (a fragment of a diagram user want to find) and collection of potential matches. The potential match becomes a find result if it satisfies all conditions defined by the find query.

The find diagram is processed incrementally. Each unconnected subgraph of the find diagram is processed separately. Depth-first style graph traversal algorithm is used to process subgraphs of a find diagram. The initial set of potential matches is created when processing the first edge in a subgraph (if subgraph contains any edges). At each traversal step of the find diagram, the set of potential matches is filtered, leaving in the set only subgraphs where the same type of traversal is possible. If multiple different traversals are available in a potential match then the match is added to the set of potential matches multiple times (once for each traversal).

Beside constraints defined using element types and graph structure, there may be constraints on the attribute values. (Attributes are named Compartments in the ajoo framework.) Only the attributes whose value is set are used as filter conditions in the "find diagram". The appropriate attribute in the potential match is found using the CompartmentType of a Compartment. For attributes of type string, the substring semantics is used. For other attribute types equality is used.

4.2 Result Representation

The client-side code is used to represent results of the find query. It receives a JSON object containing references to the diagrams and element matches inside them. The results are shown in a navigable search result table and its role is similar to classical search result representations. Data on element matches in the diagram are used to show how many elements in the diagram are matched and to highlight find results in the diagram. To highlight matches in the diagram special highlighting mode is used. The style of elements to be highlighted is modified.

4.3 Implementation in another DSL Tool Development Framework

As already stated the approach could be implemented in other DSL tool development frameworks as well. Data encoding of the framework should be analyzed at first in order to implement the approach in other DSL tool development framework. It should be noted that other frameworks provide a wider range of graphical elements, for example, box in a box, boundary box. The extension of the search algorithm is required in order to support these elements, but it wouldn't affect the general idea.

For example, in the Sirius framework [3] diagrams are just views on the domain model. There is a domain model describing the semantics of the language and the presentation model (view) describing graphical representation. The query is defined as a fragment of a diagram respectively - view. It means the find query (or the find diagram) should be translated in terms of the domain model. The instance corresponding to the element in the find query and the match should be instances of the same domain class. The matches are instance graphs similar to the instance graph created by the query. The side effect of the domain and view principle is that it would also allow finding the representations of the model elements which are completely different from the representation of the original query!

As different types of data stores are used in the DSL frameworks, it may be necessary to adapt the search algorithm to effectively use the data store of the framework. For example, in the Microsoft Modelling SDK [11] models are stored as XML files. Here to implement find XML files should be analyzed to find similar XML structure. To conclude the ideas could be applied in other tools as well. The graph traversal principles could be reused as well, however, the data access will be specific to a DSL tool development framework and encoding used by the DSL tool development framework.

5 RELATED WORK

The approach closest to ours was proposed in a VMQL [19] and VMTL [1] projects. There the plugin of the MagicDraw [14] was created allowing the concrete syntax-based search for UML. The plugin extracts MagicDraw diagrams as well as search diagrams, and then it uses a Prolog-based transformation engine to find the exact matches and similarities in the provided diagrams. Although the approach could be applied to any language it requires to create export from a language editor to a Prolog based transformation engine. In contrast, in the approach proposed in the paper, the find is available as out of the box service of DSL tool development framework.

Another difference is that in the VMQL project UML comments are used as annotations to define constraints in the find queries. We think that additional constraints are not necessary and fragments of diagrams are expressive enough to define necessary queries. The purpose of the approach is to improve the usability of the tool. Introduction of the find specific elements creates extra complexity for the user and results in a steeper learning curve.

The approach is also in a sense similar to concrete syntax-based model transformations [2, 5, 16], model transformations by example [8] and model transformations by demonstration [9]. Using model transformation by demonstration or model transformation by example there are multiple examples and the transformation is concluded from examples. In the concrete syntax-based find, there is only one find query. Compeared to concrete syntax-based model transformations there is only pattern matching part of the transformation required in concrete syntax-based find.

6 CONCLUSIONS

The concrete syntax-based find as a service of DSL tool development framework has been proposed. The approach has been implemented in the DSL tool development framework ajoo [18]. The concrete syntax-based find uses a fragment of a diagram as a find query. The find is available in any DSL built using the DSL tool development framework.

The future work is to add find – replace as a service of a DSL tool development framework. Another direction of future work could be to implement the approach in other DSL tool development framework, for example, Sirius [3].

ACKNOWLEDGMENTS

The work has been supported by European Regional Development Fund within the project # 1.1.1.2/16/I/001, application # 1.1.1.2/VIAA /1/16/180 "Concrete Syntax Based Find for Graphical Domain Specific Languages". The author would like to thank Agris Šostaks and Edgars Celms for valuable discussions.

REFERENCES

[1] Vlad Acretoaie, Harald Störrle, and Daniel Strüber. 2016. VMTL: a language for end-user model transformation. *Software & Systems Modeling* 17 (2016), 1139–1167.
[2] Thomas Baar and Jon Whittle. 2007. On the Usage of Concrete Syntax in Model Transformation Rules. In *Perspectives of Systems Informatics*, Irina Virbitskaite and Andrei Voronkov (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 84–97.
[3] Eclipse. 2020. Sirius. https://www.eclipse.org/sirius/
[4] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. 2006. *Implementation of Typed Attributed Graph Transformation by AGG*. Springer Berlin Heidelberg, Berlin, Heidelberg, 305–323. https://doi.org/10.1007/3-540-31188-2_15
[5] Roy Grønmo. 2010. *Using Concrete Syntax in Graph-based Model Transformations*. Ph.D. Dissertation. University of Oslo. Doctoral thesis.
[6] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA.
[7] E. Kalnina and A. Sostaks. 2019. Towards Concrete Syntax Based Find for Graphical Domain Specific Languages. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 236–242. https://doi.org/10.1109/MODELS-C.2019.00038
[8] Gerti Kappel, Philip Langer, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. 2012. Model Transformation By-Example: A Survey of the First Wave. In *Conceptual Modelling and Its Theoretical Foundations: Essays Dedicated to Bernhard Thalheim on the Occasion of His 60th Birthday*, Antje Düsterhöft, Meike Klettke, and Klaus-Dieter Schewe (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 197–215. https://doi.org/10.1007/978-3-642-28279-9_15
[9] Philip Langer, Manuel Wimmer, and Gerti Kappel. 2010. Model-to-Model Transformations By Demonstration. In *Theory and Practice of Model Transformations*, Laurence Tratt and Martin Gogolla (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 153–167.
[10] MetaCase. 2020. MetaEdit+ 5.1. http://www.metacase.com/
[11] Microsoft. 2020. Modeling SDK for Visual Studio - Domain-Specific Languages. https://msdn.microsoft.com/en-us/library/bb126259.aspx
[12] Inc. MongoDB. 2020. MongoDB. https://www.mongodb.org/
[13] Peter Morville and Jeffery Callender. 2010. *Search Patterns: Design for Discovery* (1st ed.). O'Reilly Media, Inc.
[14] Inc. No Magic. 2020. MagicDraw. http://www.nomagic.com/products/magicdraw.html
[15] An OMG Unified Modeling Language Publication. 2017. OMG Unified Modeling Language (OMG UML), Version 2.5.1, formal/2017-12-05. https://www.omg.org/spec/UML/2.5.1/PDF
[16] Markus Schmidt. 2007. Transformations of UML 2 Models Using Concrete Syntax Patterns. In *Rapid Integration of Software Engineering Techniques*, Nicolas Guelfi and Didier Buchs (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 130–143.
[17] Meteor Software. 2020. Meteor. https://www.meteor.com/
[18] Arturs Sprogis. 2016. DSML Tool Building Platform in WEB. In *Databases and Information Systems*, Guntis Arnicans, Vineta Arnicane, Juris Borzovs, and Laila Niedrite (Eds.). Springer International Publishing, Cham, 99–109.
[19] Harald Störrle. 2011. VMQL: A Visual Language for Ad-Hoc Model Querying. *J. Vis. Lang. Comput.* 22, 1 (Feb. 2011), 3–29. https://doi.org/10.1016/j.jvlc.2010.11.004
[20] Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. 2013. Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem?. In *Model-Driven Engineering Languages and Systems*, Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, and Peter Clarke (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–17.