In Proceedings of the 2014 IEEE Symposium on Visual Languages and Human-Centric Computing, Melbourne,Australia, July 27-Aug 1 2014 © IEEE

CONVErT Meets KIELER: Integrating Advanced Layout Algorithms into By-Example Visualisations

Iman Avazpour*
SUCCESS Centre
Swinburne University of Technology
Hawthorn, VIC, Australia
Email: iavazpour@swin.edu.au

Ulf Rüegg†
Department of Computer Science
Christian-Albrechts-Universität zu Kiel
Email: uru@informatik.uni-kiel.de

John Grundy
SUCCESS Centre
Swinburne University of Technology
Hawthorn, VIC, Australia
Email: jgrundy@swin.edu.au

***Abstract*—The CONcrete Visual assistEd Transformation (CONVErT) framework provides facilities to generate reusable notations and compose them to form a wide variety of visualisations. With an increased number of notations in large scale visualisations, it is crucial to use advanced layout algorithms to improve understandability of such complex visualisations. This showpiece paper demonstrates how advanced layout algorithms can be integrated into the notation specifications of CONVErT to generate layouts of complex visualisations.**

I. Introduction

The CONcrete Visual assistEd Transformation (CONVErT) framework[1] provides a user-centric approach to generating visualisations for a wide variety of application domains using drag and drop of visual notations [1]. It uses by-example model transformations in the visualisation process and for mapping of the notations' models to their views, mapping varieties of inputs to notations, and to compose notations to generate visualisations. CONVErT allows layout specifications to be included in notation views. These layout specifications however are hard coded for specific visualisations and altering them requires advanced knowledge of the used visualisations as well as complex low level coding. Therefore, using and updating layouts in CONVErT is not as user-centric as defining visualisations.

We have previously experienced the effectiveness of automatic layout algorithms in improving the comprehension and the aesthetics of complex diagrams [2]. This motivated us to integrate advanced layout mechanisms into the by-example specification of visualisations in CONVErT. The Kiel Integrated Environment for Layout Eclipse Rich Client (KIELER) framework[2] provides a *meta layout infrastructure* that allows using various layout algorithms from libraries such as GraphViz[3] and OGDF[4] and graph formats such as GraphViz's *dot* and KIELER's *JSON graph* [3]. As a result, KIELER is a good candidate for layout specification and integration in CONVErT. This showpiece paper demonstrates how advanced layout algorithms provided by KIELER are integrated into CONVErT to generate complex, graph-like visualisations.

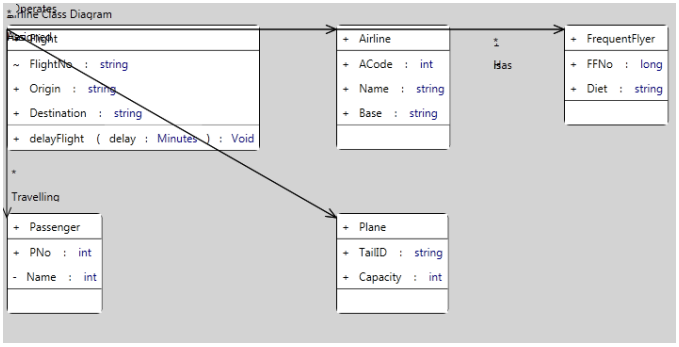
[1]https://sites.google.com/site/swinmosaic/projects/convert
[2]http://www.informatik.uni-kiel.de/rtsys/kieler/
[3]http://www.graphviz.org/
[4]http://www.ogdf.net/

II. Usage demonstration

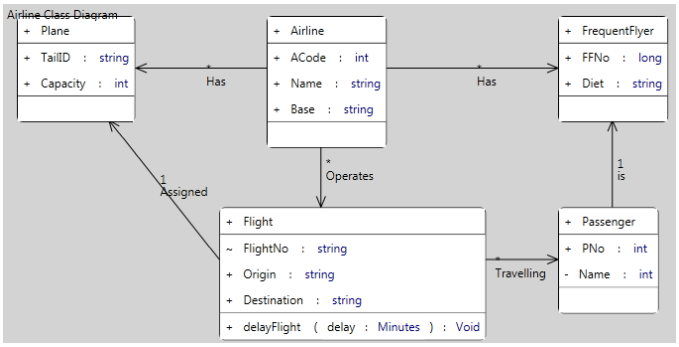
This section provides an example of visualising UML class diagrams in CONVErT using graph-based layout algorithms provided by KIELER. UML class diagrams comprise of classes (graph nodes) and different associations (edges) between these classes. Each class notation is shown as a box with three compartments for name of the class, set of attributes, and set of operations. The attributes and the operations are arranged in vertical lists and separated by a line. Associations, in our adoption of class diagram, are directed lines that connect two classes. Associations provide two labels for showing the cardinality and a name.

CONVErT uses a three step approach to generate visualisations: 1. Create the notations to be used in the visualisation (or reuse existing notations). This is done by importing already existing visualisations as views, providing the notation's model in XML, and annotating the imported visualisations to generate the mapping between the notation's model and the view. 2. The notations are mapped to input data by a drag and drop process of corresponding input elements on the notation's data elements. The drag and drop interactions are translated into rule-based model transformation scripts to transform the input model elements into the notation's model. This step allows to reuse the generated notations for multiple different input files. For example, a bar notation of a bar chart can be reused for visualising sales records, a city's population, and the frequency of cars passing an intersection. 3. The mapped-to-input notations are composed to generate the complete visualisation. Here, for example, a bar notation is included in a chart or map notation.

CONVErT's architecture allows a separation of concerns in generating a notation's view and its model, i.e. notations in CONVErT can be generated from already existing visualisations or can be designed in the provided facilities for drawing visual shapes as views, while mapping the notation's model to its view is done separately by using model transformation scripts. This allows the use of different technologies and approaches for generating notation views. For example, in our class diagram example, the view representing a class is designed first and then the correspondences between the view and the class's model are annotated in the view. These annotations define one-to-one or one-to-many mapping correspondences between a notation's model and its view. The name of a class corresponds to the name provided in the model, and the class includes multiple attributes which result in a



(a) With ad-hoc grid layout.



(b) With advanced graph layout provided by KIELER.

Fig. 1. Samples of class diagram visualisations.

one-to-many relationship. Using these annotations, CONVErT automatically generates the transformation scripts to map class notation models to class views.

To map the notations to input data, examples of the data should be provided to CONVErT. For example, a class diagram's data (in XML or XMI) that should be visualised is presented in CONVErT and the user can drag and drop its elements to the notations. This drag and drop generates transformations to map the input model to visual notations. Once this mapping is done, the mapped notations are composed to generate the full visualisation. The composition step defines what type of notations are included in the host notations. For example, a class diagram includes classes, while a class includes attributes and operations. This composition needs to be specified using one instance of each notation and the resulting transformation script is applied to the example inputs. Once the transformation is applied, depending on the number of classes that exist in the example input, the class view is generated and included in the class diagram notation.

The positioning and layout of visual elements in CONVErT is delegated to notation views. For example, the arrangement of attributes in vertical lists in classes is delegated to the class notation. Similarly, the arrangement of classes in the class diagram itself is done by the class diagram notation. This architecture allows the use of third party layout mechanisms (in this case, KIELER) during the visualisation procedure.

KIELER provides a common interface to select and configure the desired layout algorithm. The KWebS project allows to access the layout infrastructure from different languages (e. g., JavaScript and C#), locally or over the web. We use this service to enable the class diagram notation to position the classes it is comprised of. A routine is provided in CONVErT for generating a JSON graph representation of the class diagram and requesting KIELER to apply a certain layout algorithm. The JSON-encoded class diagram is sent to the layout service. The desired layout algorithm is configured by additional options that are added to the JSON. KIELER enriches the graph with coordinates for every element and returns the result. The returned results are passed to the class diagram notation which reorganises the classes and their association links according to the returned positions. Figure 1(a) depicts an example class diagram visualisation before the integration of advanced KIELER layouts, while Figure 1(b) demonstrates the same class diagram using the automatic layout mechanism. Note the improved arrangement of associations and their labels as well as class notations. A similar procedure can be applied for other visualisation examples.

To check the validity of the visualisation, certain consistency checks are provided. For instance, it is possible to check whether the notations being inserted inside a class diagram notation are in fact classes and associations. CONVErT provides facilities to report inconsistencies as exceptions. Also, notations in CONVErT can be altered to use flexible layout algorithms depending on user interests. A notation's model can be updated to include a feature for specifying the name of the desired layout algorithm, to allow users to specify the layout algorithm during the modelling process. The desired layout algorithm is then sent to the notation's view to generate the notation specific KIELER layout request.

III. Summary and Presentation

This showpiece demonstrates the integration of KIELER's advanced layout mechanism into the by-example visualisations of CONVErT. The provided example of a UML class diagram visualisation shows how class notations in CONVErT can be laid out using these methods. The approach and tool integration presented by this showpiece paper will be further demonstrated using a screen-cast video and an accompanying poster.

Acknowledgment

* This work is partially supported by an ARC Discovery Project and ARC Future Fellowship. Support for the Iman Avazpour from Swinburne University of Technology is gratefully acknowledged.
† Ulf Rüegg is funded by a doctoral scholarship (FITweltweit) of the German Academic Exchange Service.

References

[1] I. Avazpour and J. Grundy, "CONVErT: A framework for complex model visualisation and transformation," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'12)*, Innsbruck, Austria, 2012, pp. 237–238.

[2] P. S. Yap, J. Hosking, and J. Grundy, "Automatic diagram layout support for the marama meta-toolset," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'11)*, Pittsburgh, PA, 2011, pp. 61–64.

[3] M. Spönemann, C. D. Schulze, C. Motika, C. Schneider, and R. von Hanxleden, "KIELER: building on automatic layout for pragmatics-aware modeling," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'13)*, San Jose, CA, USA, 2013, pp. 195–196.