GraphTool - a new System of Graph Generation

Iwona Ryszka, Ewa Grabska
Faculty of Physics, Astronomy and Computer Science
Jagiellonian Universisty
Krakow, Poland
e-mail: iwona.ryszka@uj.edu.pl, ewa.grabska@uj.edu.pl

***Abstract*—The purpose of this paper is to present a new software tool for graph edition and generation. The project focuses on providing a graphical editor for defining different types of graphs and rules describing their transformation. The idea of graph grammar systems has been adopted to extend graph derivation functionality. The system architecture with implementation details is described. To illustrate the features of the tool the examples of a graph building and generation are attached.**

***Keywords*—graph, graph grammar, graph grammar system, tool for graph generation**

I. Introduction

The concepts of a graph and graph transformations are applied to model structures and also to simulate flows or behaviours. The base definition presents a graph as a set of nodes and a binary relation defined on this set. Each element of the relation being a pair of nodes defines an edge that can be directed or undirected. A graph grammar enables the application of local transformations by means of rules called productions to subgraphs of derived graphs. An order of productions application is determined by so called control diagram. A process of graph generation by means of productions is called a derivation. The possibility of attributes assignment extends the graph model by defining an additional semantic layer.

The basic representation of the graph that includes nodes and edges does not require any dedicated graphical software product. However, taking into consideration the requirements for graph grammars and the process of graph derivation, a new application *GraphTool* [1] has been proposed in order to provide an unified graphical environment supporting graph operations. The motivation for implementing a new tool is to provide the editor for custom types such as composite graphs or layered graphs. Additionally, the new approach for defining a grammar system as a grouping mechanism for graph grammars has been addressed within the project. Furthermore, the *GraphTool* application deals with the area of attributes operations.

Section II provides the overview of existing applications supporting graph defining and automatic rewriting. The purpose of Section III is to present the functionality of the *GraphTool* and it is followed by the Section IV that provides example usage of the application for different types of graphs. The Section V describes the implementation details.

II. Related works

There exist a number of tools for generating graphs but they are usually specialized for experts of a particular subject and focus on different areas.

PROgrammed Graph REwrting Systems (PROGRES) [2] offers a visual and operational specification language for defining graphs, transformation rules that is combined with the environment for executing specifications in this language. A transformation rule can be formatted as a simple one when only left-hand and right-hand sides are defined. There is an option to define a combined rule that describe several rules by textual control structures, e.g., loops. Additionally, the framework UPGRADE is available and its purpose is to display the flow of applying graph transformations with some formatting options.

A custom graphical language is also introduced by the *Graph Rewriting and Transformation (GReAT)* and is dedicated for graph transformations in the area of domain-specific modelling languages (DSMLs). The rules specify rewriting operations in the form of a matching pattern (closely related to UML class diagram). The application focuses on model transformations, but there is no verification if the generated graph is correct according to target language.

The area of attributed graphs with the theory on confluence and termination properties is investigated by the *Attributed Graph Grammar System (AGG)* [3]. The application supports attributes of algebraic data types, including Java expressions. The graph derivation process can be both manual and automatic, the intermediate steps are not stored. The GTXL format based on XML is used to export generated graphs.

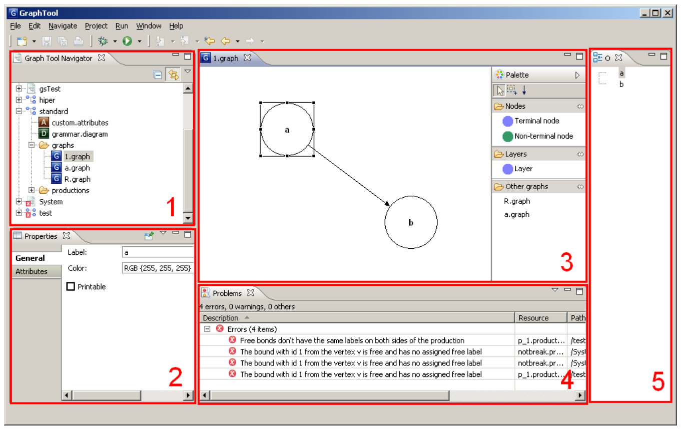
GRaphs for Object-Oriented VErification (GROOVE) [4] project focuses on the verification of object-oriented systems by means of a graph transformation tool set that uses labelled graphs and single push-out (SPO) transformation rules. There is a component for graphical editing of rules and graphs, a generator responsible for creating temporary graphs during the derivation. Additionally, the Model Checker component is available for verification whether the derived system satisfies specific properties.

III. Application overview

GraphTool is a *What You See Is What You Get (WYSIWYG)* editor. The functional areas that are addressed by the tool cover support for creating different types of graphs, building graph grammars by defining productions and a control diagram. Additionally, the process of graph derivation can be simulated and controlled by the user. As an extension the functionality of building grammar systems is offered.

The base application view is presented in Figure 1. Following working areas can be marked within the view:

1) *GraphTool Navigator* is a component responsible for presenting the structure of grouped objects created by the user. By means of the context menu, new elements can be added.



4 errors, 0 warnings, 0 others

Description	Resource	Path
Errors (4 items)		
Free bonds don't have the same labels on both sides of the production	p_1.product...	/test
The bound with id 1 from the vertex v is free and has no assigned free label	notbreak.pr...	/Sys
The bound with id 1 from the vertex v is free and has no assigned free label	notbreak.pr...	/Sys
The bound with id 1 from the vertex v is free and has no assigned free label	p_1.product...	/test

Fig. 1: The main perspective of *GraphTool* application

2) *Properties* is a section providing the functionality of editing properties selected objects in *Graphical editors* area.
3) *Graphical editors* are areas used to build graphs, productions and control diagrams. A dedicated palette with context-specific elements is available.
4) *Problems* represents a supporting section that is used to notify the user about existing issues or warnings automatically detected inside the objects created within graphs.
5) *Outline* is a view showing a hierarchy of objects defined in active *Graphical editiors* area.

A. *Graph building process*

The compulsory action in the process of creating a graph is to specify a *project*. This entity is used by the application to define a context that will be common for all graphs attached to the given project. Such a specification cover:

- the type of the graph: standard graph, hypergraph (included also the option to create hierarchical graphs [10]), composite graphs [5],
- the type of the edge: directed, undirected.

Building a graph is understood as creating nodes, edges and assigning them labels and optionally attributes. In order to simplify the construction process, a feature to add previously defined graphs within the same project as a subgraph is also available. The folder *graphs* is used to store defined objects.

Graph objects defined with *GraphTool* can be exported to XML files using GraphML format [6]. It supports all types of graphs covered by the tool. Using its flexible extension mechanism to add application-specific data the information about attributes and structure of the graph (sizes and locations of nodes) are also optionally available in the exported file.

B. *Attributes*

GraphTool application offers the support for attributed graphs. An attribute is defined as a function assigning the value from its domain to an object. Within each project there is a file called *custom.attributes* that is used for creating declaration of attributes that can be later assigned to any object defined within the project. The declaration of a new attribute is understood as specification of an unique name and its domain (the available types are integer numbers, float numbers, strings, enums and arrays). Additionally, the user can specify the default value.

Having defined attributes, their instances can be assigned with values to nodes, edges and graphs. There is a rule that each element of given type and having the same label contains the same attributes.

The application offers an additional functionality in the area of attributes which is called a *template attribute*. During assignment to an element such an attribute it receives a template value (variable). It can be used within a production to represent values which are not known at the moment of creating the production and during the graph derivation they are replaced with with the real one. The next usage of this type of attributes is defining predicates for productions and similarly they are replaced with the current values from a graph when a production is to be applied to this graph. The template attribute can be defined as an expression containing variables including a conditional expression, for example having att_i defined as integer attribute following template attributes are valid:

- $att_i = var_1 + var_2$ interpreted as a sum of values represented by variables var_1 and var_2
- $att_i = (var_1 > var_2)\ ?\ var_2 : var_3$ In case when the value of variable var_1 is greater than var_2, the att_i gets the value of var_2. Otherwise the value of variable var_3 is assigned to att_i.

C. *Graph grammars*

The concept of the project in the *GraphTool* application is additionally used in the process of defining a *graph grammar* that is specified as a set of productions and a control diagram.

A production entity is represented by two graphs that are called left and right side, respectively. The former describes a subgraph of a derived graph that after application of the production is transformed to the latter. In the *GraphTool* application the user can build a production by defining both sides from the beginning or using graphs available within the project. A folder *productions* is designed to store user's defined productions within *GraphTool* project.

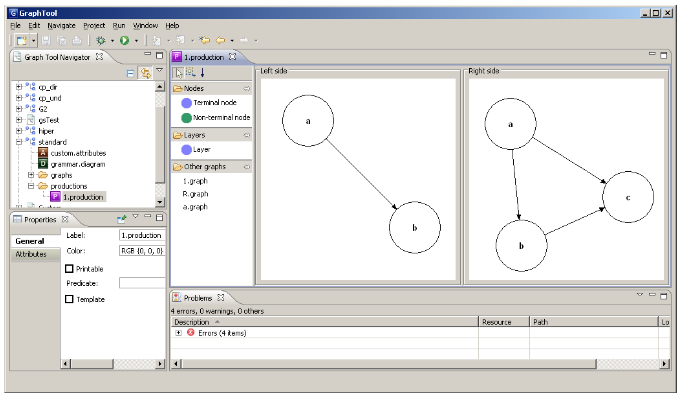


Fig. 2: An example production defined in *GraphTool*

The application offers the feature of an usability predicate for a production. The predicate represents a logical expression and its value is calculated just before application of the production during a graph derivation process. The expression can refer to values of attributes of objects available in the context of the production: a graph currently derived, its nodes, its edges and also a graph from left side of production.

Having defined transformations, the user builds a control diagram that represents productions' flow. The diagram is a directed graph where nodes represent productions. Additionally, there are two marked nodes called start and stop node that point respectively to the node initializing derivation process and the node completing this process. The diagram is valid when at least one path exists from the start to the stop node. It is stored in the file called *grammar.diagram* within the project.

D. *Graph derivation process*

In order to derive a new graph, a grammar with a control diagram and an initial graph are needed. Therefore, in the *GraphTool* application the user creates a *configuration* that involves selecting a project and one of the graph available in its *graphs* folder as an initial graph. The process of derivation can be managed by the user by means of selecting next production from the set of available productions according to the control diagram.

The current state of implementation covers the derivation process only for the composite graphs.

E. *Grammar systems*

As an extension to a graph grammar, the concept of *grammar system* can be introduced. The main purpose of the system is to combine a set of graph grammars and introduce an upper level control diagram that is responsible for switching the context between graph grammars during a graph derivation process. Therefore, the nodes in such a diagram represent graph grammars. Thus, the derivation process is controlled by two diagrams: the grammar system control diagram points to a graph grammar that should be used and derivation follows the control diagram associated with this grammar. When the stop node is achieved, the control is returned to the grammar system control diagram and next grammar can be used. An example derivation within a grammar system is presented in Figure 3.

In the *GraphTool* application the grammar systems can be build from the beginning by defining all graph grammars or by using existing one. The process of graph derivation includes also defining a configuration like in a graph grammar case.

IV. Examples of GraphTool usage

In this section, the examples of advantages of GraphTool for solving computational issues or modeling the structure are presented.

A. *Graph grammar system for* h*-modeling finite element method*

There were several attempts to adopt graphs and graph grammars to model *h*-finite element method [5]. GraphTool application has been used as a support tool to construct a grammar system for modeling three dimensional finite element method (3D FEM) with tetrahedral finite elements [7]. Composite, attributed and undirected graphs have been selected to model a finite element mesh.

The system contains a grammar that is responsible for generating the mesh. Its productions represent the mesh transformation. One of them is shown in Figure 4.

The presented production is an example of usage template attributes as the values of integer attribute *fn* can be

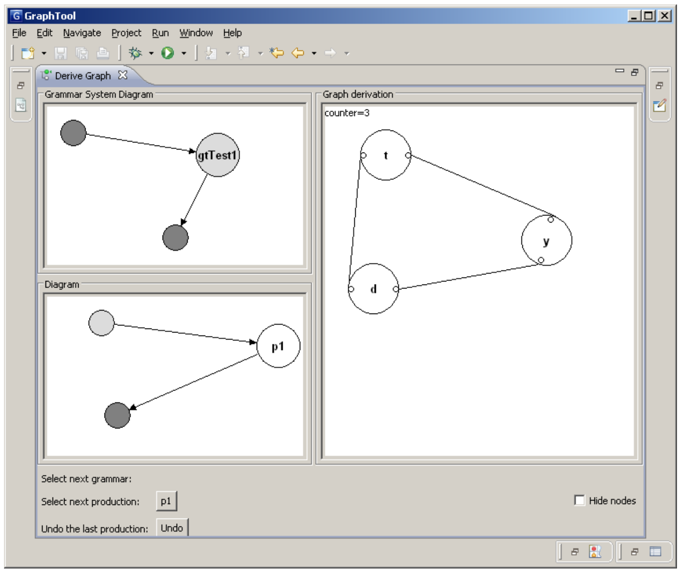


Fig. 3: An example of graph generation process using a grammar system defined in *GraphTool*

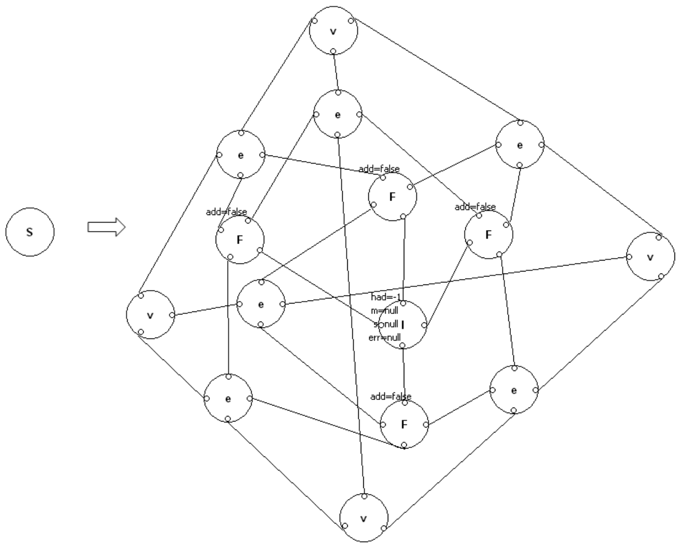


Fig. 4: A mesh generation as a production defined for *h*-modeling finite element method using *GraphTool*

incremented when the production is applied. By means of the template attributes, the number of productions could be reduced.

The next example of graph grammar within the system contains productions that focus mainly on attributes operations. By means of this process some logic operations such as calculating a solution vector can be performed. Thus, array attributes with variable lengths are useful to store information about such a vector. An example of such a production is shown in Figure 5.

The proposed system proves additionally the advantage of introducing the support for graph attributes. Such objects are used as a *global memory* in the given grammar system. The memory is initialized during defining a graph that is used in

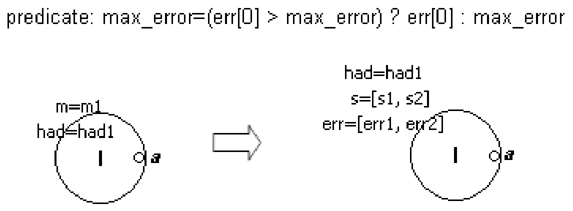

predicate: max_error=(err[0] > max_error) ? err[0] : max_error
m=m1
had=had1
I
a
had=had1
s=[s1, s2]
err=[err1, err2]
I
a

Fig. 5: A production for assigning values to array attributes *s*, *err* for node with label *I* together with a predicate specified using attributes

the derivation process as an initial one. During this process the variables and their current values can be used to calculate values of predicates for productions or values for template attributes. For the grammar system for modeling 3D FEM the memory is defined as two attributes: *error_max* and *accuracy*. For the production shown in Figure 6, the value of its predicate depends on the current value of the attribute *error_max* that is assigned to the generated graph and the value of the attribute of a node in the graph according to the left side of the production.

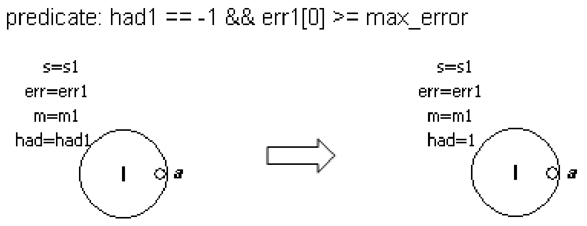


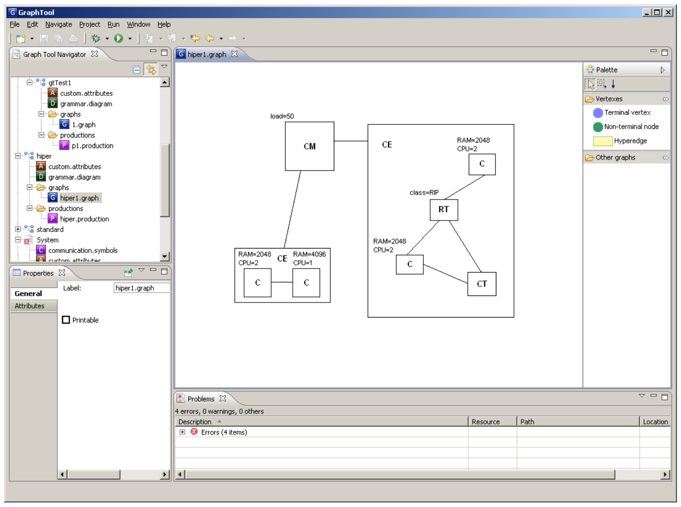
Fig. 6: A production for virtual h-refinement

B. *Hierarchical graphs for generating virtual Grids*

The Grid can be regarded as the implementation of the distributed computing concept. The structure of the environment that contains grouped components can be modeled by means of hierarchical graphs. The graph derivation process can be used to simulate building the grid environment.

The approach proposed by Lu and Dinda [8] suggests separating topology generation from annotations. The application of hierarchical graphs with attributes for the grid generation was proposed in [9] and the concept has been enhanced with the new structure including layers in [10].

GraphTool offers the environment to construct hierarchical graphs. It can be used to create graphs representing the gird structure together with attributes as presented in Figure 7. The application is able to verify the correctness of hierarchical graphs for example in the area of edges that cannot connect a node with its internal nodes. Additionally, the hierarchy can be built for hypergraphs where both nodes and hyperedges can be nested.



4 errors, 0 warnings, 0 others

Errors (4 items)

Fig. 7: A hierarchical graph representing a grid

V. Implementation details

GraphTool is an application based on Java Platform SE 7. It has been created as a plugin for Eclipse Integrated development environment. The required release is at least Eclipse Helios version 3.6.1, available at [11]. The tool can be also shipped as a standalone rich client application. Supported operating systems are both Windows and Unix based and the only additional requirement is an installed Java Runtime Environment in 1.7 version, available at [12].

The plugin uses several open sources libraries. The first is Graphical Editing Framework technology [13] that is regarded as a middleware for providing graphical layer within Eclipse editors and views. For transforming objects between Java and XML JAXB version 2.2.3 is used. Additionally, JUNG library [14] in version 2.0.1 is used to model basic structure of graphs and perform operations on them such as finding shortest path.

The current state of implementation covers the functionality for deriving graphs using both graph grammars and graph grammar systems for the composite graphs. This type of graph uses a constant embedding transformation during a production appliance. Therefore, there is no need to specify any additional conditions for the production. The character of the composite graphs add additional requirements for finding a subgraph during applying a production when the bonds of the edge have to be checked. Additionally, the parsing and calculating values of the expression in attributes JavaScript engine is used.

VI. Conclusion

In this paper, a new software application for editing and generating graphs has been presented. The design of the tool with the focus on task-oriented modules results in the support for creating different type of graphs, including attributed ones and productions for their generation. The concept of template attributes has been also described. By means of grammar systems the process of creating new graphs has been enhanced. The functionality of the GraphTool application has been illustrated on the example concerning defining graph grammar systems for modeling three dimensional finite element method and the hierarchical graph representing the grid environment.

Further research in the application development can cover support for grammar systems where particular grammars represent different types of graphs. Such a feature introduces a possibility of exploration in the area graphs transformations between specific representations. The idea of templates for attributes can be a source for further investigation as it enables to add logic to the graph structure and during the graph derivation process some calculations can be performed.

REFERENCES

[1] I. Ryszka, “Implementation of graphs model generation and their appliance”, PhD Thesis, Jagiellonian University, in press.

[2] A. Schürr, A. J. Winter and A. Zündorf, “Graph Grammar Engineering with PROGRES”, ESEC 1995, pp. 219-234.

[3] G. Taentzer, C. Ermel, and M. Rudolf, “The AGG approach: Language and tool environment” in “Handbook of Graph Grammars and Computing by Graph Transformation”, volume Volume 2: “Applications, Languages and Tools. World Scientific”, 1999, pp. 163–246.

[4] A. Rensink, “The GROOVE Simulator: A Tool for State Space Generation. In: Applications of Graph Transformations with Industrial Relevance (AGTIVE)”, Lecture Notes in Computer Science 3062, Springer 2004, pp. 479-485.

[5] A. Paszyńska, E. Grabska and M. Paszyński, “A Graph Grammar Model of the hp-adaptive Three Dimensional Finite Element Method. Part I”, Fundamenta Informaticae, 114(2), 2012, pp. 149 – 182.

[6] GraphML, http://graphml.graphdrawing.org, September 2013

[7] I. Ryszka, A. Paszyńska, E. Grabska, M. Paszyński and M. Sieniek, “Graph grammar systems for modeling three dimensional finite element method”, in press

[8] D. Lu and P. A. Dinda, “GridG: Generating Realistic Computational Grids”, SIGMETRICS Performance Evaluation Review, Volume 30, num 4, 2003, pp. 33-40.

[9] B. Strug, I. Ryszka, E. Grabska and G. Ślusarczyk, “Virtual ‘Computational Grid by Graph Transformations”, F. Davoli et al. (eds.), Remote Instrumentation for eScience and Re-lated Aspects, Springer Science + Business Media, LLC 2012.

[10] E. Grabska, W. Palacz, B. Strug and G. Ślusarczyk, “A Graph-Based Generation of Virtual Grids, 9th International Conference on Parallel Processing and Applied Mathematics (PPAM 2011)”, Lecture Notes in Computer Science 7203, 2012, pp. 451 – 460.

[11] The Eclipse Foundation open source community, http://www.eclipse.org, September 2013

[12] Java Technology, http://www.java.com, September, 2013

[13] GEF - Graphical Editing Framework for Eclipse, http://www.eclipse.org/gef, September 2013

[14] JUNG - Java Universal Network/Graph Framework, http://jung.sourceforge.net/, September 2013