A SYSTEM FOR VISUALIZATION OF BIG ATTRIBUTED HIERARCHICAL GRAPHS

Victor N. Kasyanov, Timur A. Zolotuhin

Institute of Informatics Systems,
Novosibirsk State University Novosibirsk, Russia

ABSTRACT

Information visualization is a process of transformation of large and complex abstract forms of information into the visual forms, strengthening cognitive abilities of users and allowing them to take the most optimal decisions. A graph is an abstract structure that is widely used to model complex information for its visualization. In the paper, we consider a system aimed at supporting of visualization of big amounts of complex information on the base of attributed hierarchical graphs.

KEYWORDS

Attributed Hierarchical Graphs; Big Graphs; Information Visualization; Visualization Systems

1. INTRODUCTION

Information visualization plays an important role in human life. It is believed that people receive through vision about 90% of all received information. Humanity for thousands of years has overcome the way from simple imaging methods in the form of rock paintings to maps, charts and graphs. At present, visualization of graph models is an inherent part of the processing of complex information about the structure of objects, systems and processes in many applications in science and technology, and at the market there are widely presented software products, using the information visualization on the basis of graph models [1 - 4].

In some application areas the organization of information is too complex to be modeled by a classical graph. To represent a hierarchical kind of diagramming objects, some more powerful graph formalisms have been introduced, e. g. compound digraphs [5] and the clustered graphs [6]. The compound digraphs are an extension of directed graphs and allow both inclusion relations and adjacency relations between vertices. A clustered graph consists of an undirected graph and its recursive partitioning into subgraphs. It is a relatively general graph formalism that can handle many applications with hierarchical information, and is amenable to graph drawing.

The size of the graph model to view is a key issue in graph visualization [2, 4]. Large graphs pose several difficult problems. If the number of graph elements is large it can compromise performance or even reach the limits of the viewing platform. Even if it is possible to layout and display all graph elements, the issue of viewability or usability arises, because it will become impossible to discern between nodes and edges of graph model. It is well known that comprehension and detailed analysis of data in graph structures are easiest when the size of the displayed graph is small. Another weak point is that usual systems for graph visualization do not have a support for many different attributes of graph elements. The standard situation for graph visualization systems is to have one text label for each vertex and, optionally, for each edge.

DOI: 10.5121/ijcnc.2018.10206

The hierarchical graphs are an extension of cluster and compound graphs and can be used in many areas where visualization of complex information is needed [7, 8]. In this paper, the Visual Graph system [9] intended for visualization of big amounts of complex information on the basis of attributed hierarchical graph models is considered.

2. AREA OF APPLICATION

The Visual Graph system has been developed to visualize the internal data structures typically found in compilers and other programming systems. Data structures that occur in these systems are usually represented as attributed graphs of big size. For example, the attributed syntax trees are used as the internal representations of translated programs in almost all compilers or interpreters. Optimizing and restructuring compilers require static analysis of control and data relationships in a translated program and use their presentations in the form of a more general graph model of the program, such as the control-flow graph or the data-flow graph.

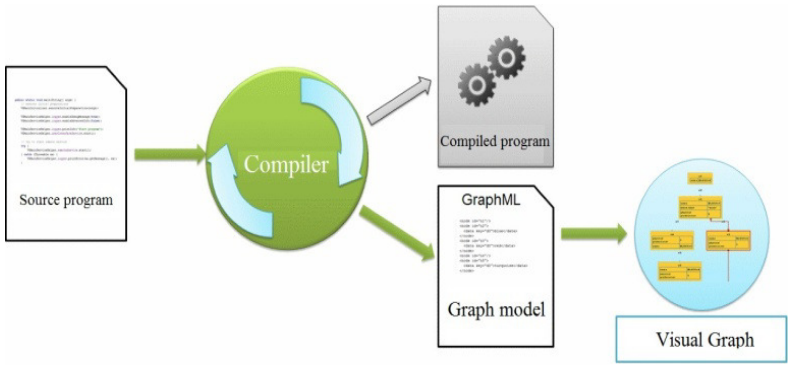

Fig. 1. Application of the Visual Graph system.

It is clear that a graph model used in a programming system may be own, distinct from the others. Moreover, the representation of the same graph model may differ from each other in different systems and even in different versions of the same system. However, all these graph models are specific subclasses of the class of the attributed hierarchical graphs.

It is assumed that the Visual Graph system is used as follows (Fig. 1). First, a compiler (or another programming system) itself or with an auxiliary program transforms a graph model arisen during compiling a source program from its internal representation into a file of one of the formats supported by the Visual Graph system, usually into the GraphML-file. Then the Visual Graph system will be able to read this graph model from the file, to visualize it and to provide a user with different navigation tools for its visual exploration allowing him to take the most optimal decisions about compiler behavior.

3. HIERARCHICAL GRAPHS AND GRAPH MODELS

Let us consider some definitions from [7, 8].

Let G be a graph of some type, e.g. G can be an undirected graph, a digraph or a hypergraph. A graph C is called a *fragment* of G, denoted by $C \subseteq G$, if C includes only elements (vertices and

edges) of *G.* A set of fragments *F* is called *a hierarchy of nested fragments* of the graph *G*, if $G \in F$ and $C_1 \subseteq C_2$, $C_2 \subseteq C_1$ or $C_1 \cap C_2 = \varnothing$ for any $C_1, C_2 \in F$.
A *hierarchical graph H* = (*G, T*) consists of a graph *G* and a rooted tree *T* that represents an immediate inclusion relation between fragments of a hierarchy *F* of nested fragments of *G. G* is called the *underlying graph* of *H. T* is called the *inclusion tree* of *H*.

A hierarchical graph *H* is called a *connected* one, if each fragment of *H* is connected graph, and a *simple* one, if all fragments of *H* are induced subgraphs of *G*.

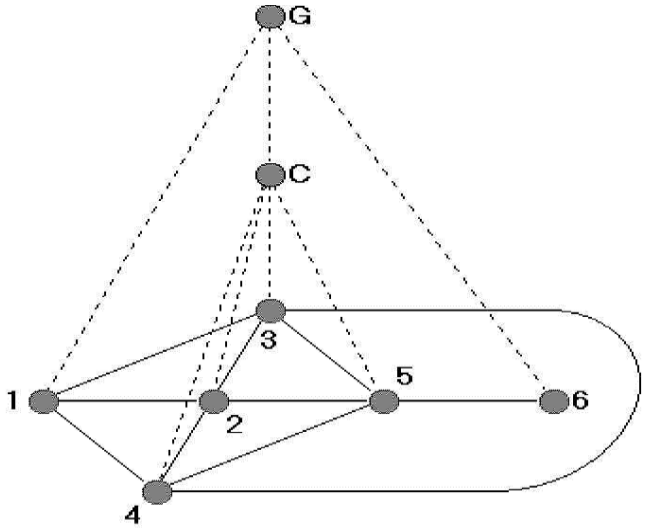

Fig. 2. A simple hierarchical graph *H*=(*G, T*) which has only two nontrivial fragments: *G* and *C*.

It should be noted that any clustered graph *H* can be considered as a simple hierarchical graph *H* = (*G, T*), such that G is an undirected graph and the leaves of *T* are exactly the trivial subgraphs of *G* (See Fig. 2).

Under the graph model, in general, we understand a class of graph objects being attributed (labeled) graphs with a given equivalence relation on it. So in the definition of a graph model we can distinguish between a static (or syntactic) part of the specification which defines a class of graph objects, and a dynamic (or semantic) part which defines a partition of this class into subclasses of graphs being pair-wise equivalent.

Let *V* be a set of objects called simple *labels* (e.g. *V* can include some numbers, strings, terms and graphs). Let *W* be a set of *label types* of graph elements and let a label set $V(w) = V_1 \times V_2 \times \ldots \times V_s$, where $s \geq 1$ and for any i, $1 \leq i \leq s$, $V_i \subseteq V$, be associated with each $w \in W$.

A labeled hierarchical graph is a triple *(H,M,L),* where *H* is a hierarchical graph, *M* is *a type function* which assigns to each element (vertex, edge and fragment) *h* of *H* its type $M(h) \in W$, and *L* is a *label function,* which assigns to each element *h* of *H* its label $L(h) \in V(M(h))$.

When the image of a graph model is made a type of its elements may be associated with a certain geometrical shape of corresponding representations and / or with their certain color range, as well as with the place and manner of representations of attributes relating to the elements of the type.

As for the dynamic part of a hierarchical graph model, it brings to the visualization of graph models different aspects of animation. Two main approaches to the presentation of semantic part of the graph model are used typically: either by explicitly specifying a set of so-called invariants (i.e. properties being common to all equivalent models), which distinguishes the equivalence classes of graph models, or through so-called equivalent transformations of graph models, which retain the specified set of invariants. Both approaches to the presentation of the semantic graph model are based on graph transformations and actively studied and developed in the theory of program schemes (see, for example, [3]).

4. GRAPHML

For a long time among different formats being used for presentation of graphs was not a single one that would be widely accepted as the standard format.As rule tools to work with graph models supported different graph formats, usually consisting of restricted subclasses of graphs and their representations.

Motivated by the goals of tool interoperability, access to benchmark data sets, and data exchange over the Web, the Steering Committee of the Graph Drawing Symposium started a new initiative with an informal workshop held in conjunction with the 8th Symposium on Graph Drawing. As a consequence, an informal task group was formed to propose a modern graph exchange format suitable in particular for data transfer between graph drawing tools and other applications. The main goal of the GraphML language [10, 11] has been formulated by the developers in the following way. The graph exchange format should be able to represent arbitrary graphs with arbitrary additional data, including layout and graphics information. The additional data should be stored in a format appropriate for the specific application, but should not complicate or interfere with the representation of data from other applications. GraphML is designed with this and the following more pragmatic goals in mind [10]:

Simplicity: The format should be easy to parse and interpret for both humans and machines. As a general principle, there should be no ambiguities and thus a single well-defined interpretation for each valid GraphML document.

Generality: There should be no limitation with respect to the graph model, i.e. hypergraphs, hierarchical graphs, etc. should be expressible within the same basic format.

Extensibility: It should be possible to extend the format in a well-defined way to represent additional data required by arbitrary applications or more sophisticated use (e.g., sending a layout algorithm together with the graph).

Robustness: Systems not capable of handling the full range of graph models or added information should be able to easily recognize and extract the subset they can handle.

Designed GraphML language for descriptions of graphs is based on XML. It allows describing oriented, undirected and mixed graphs, hypergraphs and hierarchical graphs, as well as any specific attributes for specific applications. In particular, GraphML language fully supports attributed hierarchical graphs. Thanks to its XML syntax, GraphML can be used in combination with other XML based formats. On the one hand, its own extension mechanism allows attaching <data> labels with complex content (possibly required to comply with other XML content models) to GraphML elements. Examples of such complex data labels are Scalable Vector

Graphics describing the appearance of the vertices and edges in a drawing. On the other hand, GraphML can be integrated into other applications, e.g. in SOAP messages.

5. USER INTERFACE AND TOOLS

The user interface of the Visual Graph system is shown in Fig. 3. It includes desktop, mini-map, navigator and attribute panel.

A. Desktop

Desktop consists of a set of tabs that are opened by a user to visualize the selected fragments of the graph models as their drawings on the plane.

To improve the image automatically obtained, the user can change easily the shape of graph elements (i.e. vertices and edges), the layout and its settings, the representation of attributes, the scale of the visible area and more.

B. Mini-map

Mini-map is a tool that allows a user to observe the whole graph shown in a current tab of the desktop, and also to move and to zoom its visible region, i. e. such its part which is visible in the current tab.

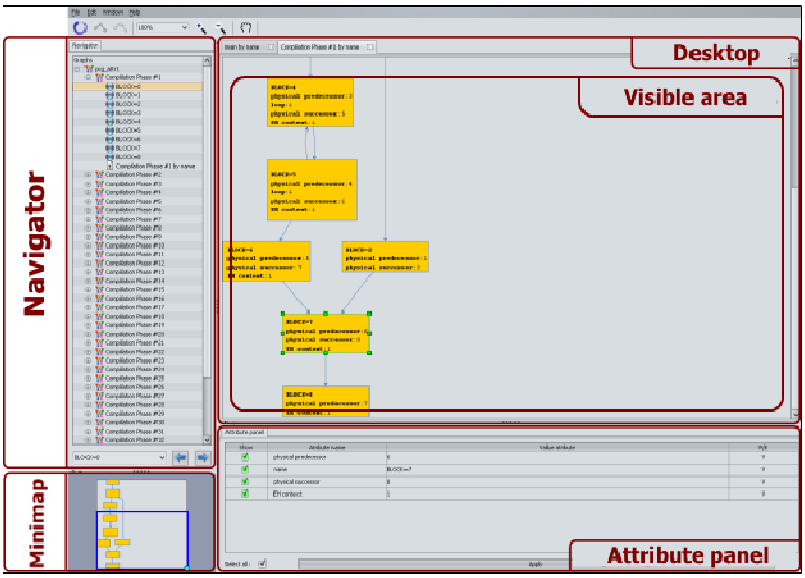

Fig. 3. User interface of the Visual Graph system.

C. Navigator

Navigator is intended for visualization of all graphs with which a user is working as an image where nesting trees of fragments of these graphs are represented via indentations. To quickly search for the elements of the trees, a search line was implemented, which allows the user to use regular expressions. After the search, the user can select the interesting elements and open them in a new tab. The user can also specify a set of attributes that will be visible for these elements.

To work with a big graph navigator can download its necessary elements dynamically every time, when the user uncollapses an element containing inner elements. If the number of inner elements

is too large, the elements can be combined in blocks. Each block contains no more than 100 elements.

D. Attribute Panel

Attribute panel is a tool that allows a user to control the visualization of attributes for the selected vertices and edges in the current tab. For this purpose the user has to select from the graph of the current tab those vertices and edges that are of interest to his/her and then to mark in the attribute panel those attributes that he/she wants to visualize for these elements.

E. Filter

Filter is a tool which supports the search in the current tab all elements (vertices and edges) of the graph model according to some conditions. The conditions are described by the user using the attribute names and values.

User-specified conditions can be combined into expressions by means of logical operators and parentheses. The result of the filter work is the set of all those elements of the graph model that satisfy the specified conditions.

F. Notepad

Notepad is a tool that allows a user to download additional information in the form of text files and link it to the graph model. The user can cross from the graph model elements to the associated additional text information and backward. For example, notepad can be used for connection of a syntax tree with the source code.

G. Structural Analysis Tools

These tools include a variety of algorithms for graph model, which help the user to select and visualize the information he needs in their images. These include tools for finding the shortest path, maximal strongly connected regions, all paths between two given vertices, immediate dominators, and a common subgraph of two graphs.

Note that the problem of finding the maximum common subgraph of two graphs is NP-hard, but appears very often during compiler development. The common subgraph tool uses an approximation algorithm based on a heuristic and finds reasonably good solutions reasonably fast. So, a subgraph obtained by the tool for some two graphs cannot be the maximum common subgraph of the graphs, but a user can always make common subgraph as big as possible by increasing the running time of the tool.

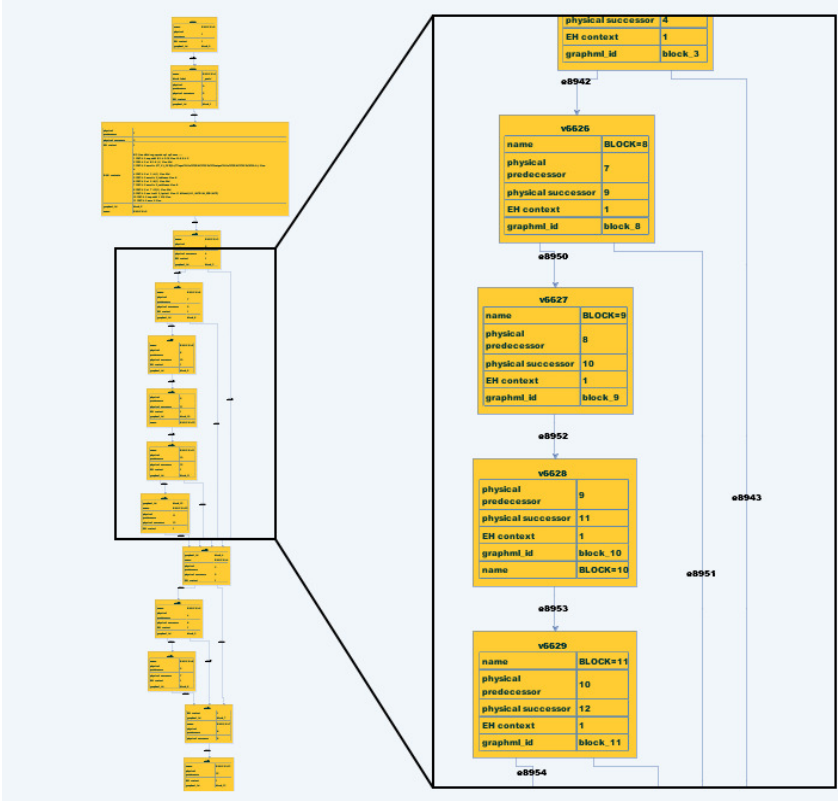

Fig. 4. Example of hierarchical layout for a control flow graph.

H. Layouts

The task of graph layout on the plane does not and cannot have an optimal solution due to the fact that sets of criteria to evaluate the quality of graph layouts are different for different applications [1 - 4]. It is known that for different applications different criteria will be of greater importance than the others in the readability of the graphs. However, for specific classes of applications (and associated classes of displayed graphs) it is possible usually to choose criteria for good layouts as well as effective algorithms for their construction.

The Visual Graph system provides for users several layout tools, the main of which performs hierarchical layouts (Fig. 4). During including the hierarchical layout algorithm turn into the system, it was the most adapted to work with graphs used in compilers. But if a user wants this (and any other) layout support tool can always be changed and /or adjusted for other types of graphs and applications.

As a rule graphs that arise in the compilers have relatively small and simple subgraphs that are visualized in into separate tabs of their multi-aspect layouts (see below). So, the hierarchical layout algorithm works well with these graphs when their multi-aspect layouts are constructed. Depending on the size of the subgraph it can use different heuristics to minimize the height or width of the resulting layout and to reduce the number of intersections between the edges.

6. IMPLEMENTATION FEATURES

A. Storage of Big Graph Models

File size with an input graph model can reach hundreds of megabytes, which prevents the use of the computer's memory to store graphs rendered in the system. It was therefore decided to use the caching data to the hard disk, using a relational database. The embedded SQLite database [12] has been chosen as a relational database. Unlike most popular relational database, the SQLite does not require installing a server, and the client-server architecture is reduced to work with files.

In the Visual Graph system a part of the data is contained in the database and another part is contained in the main memory, and thus the high speed of working with large graph models is achieved.

B. Reduction of Layout Time

Visual Graph was designed to explore large graphs that consist of many hundreds of thousands of elements. However, the layout of large graphs may require considerable time. Thus, there are two main ways to speed up the layout algorithm: multi-aspect layout of graph and control of layout algorithms.

The first way in visualizing a large graph is aimed at avoiding computing the layout of parts of the graph that are currently not of interest. Interactive exploring of a graph is based on step by step construction of so-called multi-aspect layout of the graph being a set of drawings of some subgraphs of the graph. For presentation of multi-aspects layout a set of tabs which includes a separate tab for visualization of each considered subgraph is used. At each step of the construction a layout algorithm is applied to a subgraph being interested to user at this step. To indicate the interested subgraph the user can select its elements in the current tab or in the navigator. The user can also define some condition in the filter or in the search line of navigator. Then the condition will be used for searching of graph elements which will form the interested subgraph. The search can be performed both locally (in some part of graph, e.g. through a subgraph presented in the current tab) or globally (around the entire graph). Multi-aspect drawing of graph models makes every visible part of the graph smaller, thus enabling the layout to be calculated faster and the quality of the layout to be improved.

In order to reduce layout time for big complex graphs, it is possible to control the layout algorithms, e.g. some layout phases can be omitted or the maximum number of iterations of some layout phases can be limited. However, this usually decreases the quality of the layout. The user can improve the automatically obtained layout by hand, e.g. by moving of nodes or changing of their sizes or forms.

C. Expandability of the System

All the features of the Visual Graph system, including possibilities for navigation, visualization and structural analysis, are implemented (and are available to users of the system) using a set of tools, which can be easily extended by both the developers of the Visual Graph system and any third-party developers. To achieve a simple expandability of the Visual Graph system it was

decided to use the Apache Felix product [13], which, in turn, is an implementation of the OSGi specification [14]. This solution is the de facto standard for this type of tasks and allows different developers to easily extend the system by writing new plug-ins.

7. MAIN ANALOGS OF THE SYSTEM

Currently on the market is a fairly wide range of systems for the visualization of graph models. The most famous of them are: aiSee [15], yEd [16], Cytoscape [17] and Higres [18]. Areas of application of these systems overlap with the scope of the Visual Graph system, but generally somewhat wider. So, many of them can be used as editors of graphs, while the Visual Graph system is developed, exceptionally, to visualize existing graph models without the ability to edit them.

The main two tasks which can be solved by the Visual Graph system are to display of big graph models and to navigate through them. Therefore, under these two objectives the characteristics of each of these systems will be mainly considered.

A. The aiSee system

The aiSee is a commercial system that automatically builds the layout of the original graph described in a specially designed language GDL (Graph Description Language) which is an ASCII text representation of the graph. This layout is then displayed, and can be interactively explored, printed, and exported to various formats.

The aiSee system was developed to visualize the internal data structures typically found in compilers. Today it is widely used in many different areas: genealogy, business management, bioinformatics, social networks, linguistics, and so on.

In aiSee there are several layout algorithms, and a lot of settings for them, which in varying degrees affect the results. Quality of layouts which can be obtained for some types of graphs is very high.

For navigation the aiSee system provides:

1. Desktop is an instrument that visualizes the graph model as a static image, which cannot be modified by moving or changing elements, for example.

2. Searcher is an instrument that allows the user to search elements in the graph model using their names. It supports regular expressions, the choice of categories of elements for the searching and saving of previous searches.

B. The yEd system

yEd is a powerful diagram editor that can be used to quickly and effectively generate high-quality drawings of diagrams. The diagrams can be created manually or be imported as an external data. yEd offers powerful support for creating and editing diagrams. yEd can be used to build, modify, and visualize diagrams in an effective and efficient manner. They can be loaded and saved using a variety of different file formats including XML-based GraphML file format.

The yEd system has a large number of layouts and options for them. The user can fine-tune each type of a graph.

The user interface supports comfortable working with diagrams through the following elements:

1. Desktop is similar to the aiSee desktop except that here it is possible to work with several graphs simultaneously (by creating a tab for each graph) and to edit elements of the graph by changing their positions and sizes as well as by specifying their attributes that effect on their visualization.

2. Navigator is an instrument that displays the graph placed in the current tab.

3. Mini-map is an instrument that shows the whole graph placed in the current tab and its visible part.

C. The Cytoscape system

Cytoscape is an open source bioinformatics software platform for visualizing molecular interaction networks and integrating these interactions with gene expression profiles and other state data. Many tools are available for network and molecular profiling analyses, new layouts, additional file format support, scripting, and connection with databases.

Cytoscape has both its own layout algorithms, as well as third-party, among which are the algorithms from yFiles. It should also be noted that the layouts obtained by the yEd system is much better than those layouts which can be obtained by Cytoscape system with default settings.

The main navigation tools offered the Cytoscape system are the followings:

1. Desktop is similar to the yEd desktop. The difference is in that the Cytoscape system cannot work with hierarchical graphs.

2. The mini-map is identical to the mini-map of the yEd system.

3. Attribute panel is a tool to display the current attributes and set new ones. This tool looks like a table using the attribute names as rows and graph elements as columns. This method has grave disadvantages if the number of attributes and graph elements is large.

4. Filter is a tool that allows to search graph elements using some conditions on their attributes.

D. The Higres system

The Higres system is an editor and a visualization tool for attributed not big simple hierarchical graphs and a platform for execution and animation of graph algorithms.

The semantics of a hierarchical graph model is represented in the Higres system by means of labels (attributes) of graph elements and a library of so-called external modules. Each label has its data type, name and several other parameters. Higres provides the run-time animation of algorithms represented by external modules [19].

For automatic graph allocation the Higres system uses a few graph drawing algorithms which are based on the force method and are good for non big simple hierarchical graphs.

As navigation tools the Higres system provides:

1. Desktop is similar to the desktop of the aiSee system, but unlike it makes it possible to close or open one or more fragments as well as to edit the elements of visualized hierarchical graph from by changing of their position, shape and size to by specification of their types and attributes that affect their visualization.

2. Mini-map is similar to the mini-map in the yEd system.

8. CONCLUSION

The Visual Graph system intended for visualization of big amounts of complex information on the basis of attributed hierarchical graph models was considered.

Unlike its analogues the Visual Graph system has the following important properties. It supports the processing of arbitrary attributed hierarchical graphs (including compound and cluster graphs) and uses for specification of the input (visualized) graph model the standard GraphML language. The system makes a multi-aspect layout of an input graph model that consists of separate drawings of only such its fragments which have been interested for user during graph model consideration and constructed on demand. The Visual Graph system provides also rich opportunities to navigate through big graph model, to make its structural analysis and to work with the attributes of its elements, as well as to extend and customize easily the system to specific needs of a concrete user.

At present the Visual Graph system is focused on the visualization of data structures arising in compilers, can simultaneously work with them both in the graph and in text forms, and provides smooth performing of the basic operations on graphs with up to 100 000 elements (vertices and edges).

Its successful test use had been in the Intel Company.

It is also worth noting that using the Visual Graph system is not limited to the visualization of the internal data structures arising in compilers. It can be applied in other related fields which require graph visualization and navigation. For example, the system is used now as a base of the visual debugging tool of a parallel programming system CSS which is under development for supporting cloud supercomputing on the base of the Cloud Sisal language [20, 21]. The CSS system uses the attributed hierarchical graphs for internal representations of Cloud-Sisal-programs and provides means to write and debug Cloud-Sisal-programs on low-cost devices as well as to translate and execute them in clouds. So, the system can open the world of parallel and functional programming to all students and scientists without requiring a large investment in new, top-end computer systems.

This work is supported in part by the Russian Foundation for Basic Research under grant RFBR 18-07-00024.

REFERENCES

[1] G. DiBattista, P. Eades, R. Tamassia, G. Tollis, Graph Drawing: Algorithms for Vizualization of Graphs. Prentice Hall, 1999.

[2] I. Herman, G. Melançon, M.S. Marshall, "Graph visualization and navigation in information visualization: a survey", IEEE Trans. on Visualization and Computer Graphics, vol. 6, pp. 24-43, 2000.

[3] V.N. Kasyanov, V.A. Evstigneev, Graphs in Programming: Processing, Visualization and Application. St. Petersburg: BHV-Petersburg, 2003. (In Russian).

[4] V.N. Kasyanov, E.V. Kasyanova, "Information visualization on the base of graph models", Scientific Visualization, vol. 6, n. 1, pp. 31 – 50, 2014 (in Russian).

[5] Q.W. Feng, R.F. Cohen, P. Eades, "Planarity for clustered graphs, Lecture Notes in Computer Science", vol. 979, pp. 213-226, 1995.

[6] K. Sugiyama, K. Misue, "Visualization of structured digraphs", IEEE Trans. on Systems, Man and Cybernetics, vol. 21, n. 4, pp. 876-892, 1999.

[7] V.N. Kasyanov, "Hierarchical graphs and graph models: problems of visual processing", in Problems of Informatics Systems and Programming, Novosibirsk, 1999, pp. 7-32. (In Russian)

[8] V.N. Kasyanov, "Methods and tools for structural information visualization", WSEAS Transactions on Systems, vol. 12, n. 7, pp. 349-359, 2013.

[9] V.N. Kasyanov, T.A. Zolotuhin, "Visual Graph - a system for visualization of big size complex structural information on the base of graph models", Scientific Visualization, vol. 7, n. 4, pp. 44–59, 2015. (in Russian).

[10] U. Brandes, M. Eiglsperger, J. Lerner and C. Pich, "Graph Markup Language (GraphML)", in Handbook of Graph Drawing and Visualization, CRC Press, pp. 517-541, 2013.

[11] GraphML, http://graphml.graphdrawing.org/specification.html

[12] SQLite homepage, http://www.sqlite.org

[13] Apache Felix homepage, http://felix.apache.org

[14] OSGi Alliance homepage, http://www.osgi.org/Main/HomePage

[15] aiSee homepage, http://www.aisee.com

[16] yEd homepage, http://www.yworks.com

[17] Cytoscape homepage, http://www.cytoscape.org

[18] Higres homepage, http://pco.iis.nsk.su/higres

[19] I.A. Lisitsyn, V.N. Kasyanov, "HIGRES - visualization system for clustered graphs and graph algorithms", Lecture Notes in Computer Science, vol.1731, pp.82-89, 1999.

[20] V.N. Kasyanov, E.V. Kasyanova, "Graph- and cloud-based tools for computer science education", Lecture Notes in Computer Science, vol. 9395, pp. 41-54, 2015.

[21] V.N. Kasyanov, E.V. Kasyanova, "Cloud system of functional and parallel programming for computer science education", in Proceedings of 2015 2nd International Conference on Creative Education (ICCE 2015), London, SMSSI, pp. 270-275, 2015.