GMTE: A Tool for Graph Transformation and Exact/Inexact Graph Matching

Mohamed Amine Hannachi[1,2], Ismael Bouassida Rodriguez[1,2,3], Khalil Drira[1,2], and Saul Eduardo Pomares Hernandez[1,4]

[1] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[2] Univ de Toulouse, LAAS, F-31400 Toulouse, France
[3] ReDCAD, University of Sfax, B.P. 1173, 3038 Sfax, Tunisia
[4] Computer Science Department, Instituto Nacional de Astrofísica, Óptica y Electrónica, C.P. 72840, Tonantzintla, Puebla, Mexico
{hannachi,bouassida,khalil,sepomare}@laas.fr

Abstract. Multi-labelled graphs are a powerful and versatile tool for modelling real applications in diverse domains such as communication networks, social networks, and autonomic systems, among others. Due to dynamic nature of such kind of systems the structure of entities is continuously changing along the time, this because, it is possible that new entities join the system, some of them leave it or simply because the entities relations change. Here is where graph transformation takes an important role in order to model systems with dynamic and/or evolutive configurations. Graph transformation consists of two main tasks: graph matching and graph rewriting. At present, few graph transformation tools support multi-labelled graphs. To our knowledge, there is no tool that support inexact graph matching for the purpose of graph transformation. Also, the main problem of these tools lies on the limited expressiveness of rewriting rules used, that negatively reduces the range of application scenarios to be modelling and/or negatively increase the number of rewriting rules to be used. In this paper, we present the tool GMTE Graph Matching and Transformation Engine. GMTE handles directed and multi-labelled graphs. In addition, to the exact graph matching, GMTE handles the inexact graph matching. The approach of rewriting rules used by GMTE combines Single PushOut rewriting rules with edNCE grammar. This combination enriches and extends the expressiveness of the graph rewriting rules. In addition, for the graph matching, GMTE uses a conditional rule schemata that supports complex comparison functions over labels. To our knowledge, GMTE is the first graph transformation tool that offers such capabilities.

1 Introduction

Graphs are a powerful and natural way of modelling complex systems on an intuitive level. Graph-based modelling is applied in diverse domain such as communication networks, social networks, autonomic systems, data representing, entity relationship and UML diagrams, and visualization of software architectures. Due to the dynamic nature of such systems, graph transformation concept

W.G. Kropatsch et al. (Eds.): GbRPR 2013, LNCS 7877, pp. 71–80, 2013.

takes an important role in order to model dynamic behavior by describing the evolution of graph based structures.

The research area of graph transformation dates back to the seventies, but development of software tools that support this formalism only begun twenty years later. These tools have made graph transformation more and more popular and widely used as a modelling paradigm. Currently, numerous graph transformation tools are under development, covering a wide spectrum of applications such as bioinformatics, process management, object-oriented modeling, architectural design, reengineering, distributed systems, etc. PROGRES [1], AGG [2], GROOVE [3] are well-known tools, which support general purpose graph transformation. For specific purpose we could find VMTS [4], GreAT [5], ATOM3 [6] for model transformation.

In this paper, we expose our general graph matching and transformation engine *GMTE*[1] handling exact and inexact graph matching with expressive graphs and rewriting rules. In section 2, we present the preliminaries concept. In Section 3, we introduce exact graph matching, extensions of model graph definition, vertex matching and consistent valuation building. Graph updating process where rewriting rules consider variable labels, both positive and negative application conditions, connection instructions, modification instruction and conditional rule schemata with label calculation, is presented in this section. Section 4 deals with inexact graph matching based on the graph edit distance and bipartite matching. Comparison to a reference tool is performed in Section 5. The concluding remarks and perspectives are discussed in Section 6.

2 Preliminaries

In this section, we establish the fundamental definitions used in this paper and give the formal problem statement. This paper investigates the subgraph matching and transformation for directed and multi-labeled graphs.

Definition 1. *A multi-labeled graph G is defined as a 6-tuple $G = V, E, L_V, D_{L_V}, L_E, D_{L_E})$, where V is the set of vertices $E \subseteq V \times V$ is the set of edges. D_{L_V} and D_{L_E} are the definition domains of vertex labels and edge labels. $L_V : V \to D_{L_V}$ is the function assigning labels to vertices and $L_E : E \to D_{L_E}$ is the function assigning labels to edges.*

The main idea of graph transformation is the rule-based modification of graphs. The foundation of a rule is a pair of graphs (L,R), called the left-hand side L and the right-hand side R. Applying the rule $p = (L, R)$ means finding a match of L in the source graph and replacing L by R, leading to the target graph of the graph transformation. The problem of finding a match of L is treated by *graph isomorphism.*

Definition 2. *Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there exists a mapping $M \subseteq V_1 \times V_2$ such that for all pairs of vertices $v_i, v_j \in V_1$,*

[1] Graph Matching and Transformation Engine (*GMTE*), available at `http://homepages.laas.fr/khalil/GMTE/`

$(v_i, v_j) \in E_1$ *if and only if* $(M(v_i), M(v_j)) \in E_2$. *M is, in this case, a graph isomorphism between* G_1 *and* G_2. *A subgraph isomorphism is an isomorphism between* G_1 *and a subgraph* G' *of* G_2.

In contrast to the exact graph matching, the inexact (or approximate, error-tolerance) graph matching allows nodes or edges mismatch or both. *Graph edit distance* is one of the most commonly used and well-known approach that defines similarity between graphs. The distance between two graphs is measured by applying a sequence of edit operations (i.e. node and edge insertion, deletion, or substitution) in order to transform one graph into the other. For each edit operation, a cost is assigned. The cost of an edit series is the sum of the individual edit operations. The graph edit distance is the minimum cost necessary for transforming one graph to another.

Definition 3. *Let* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$ *be two graphs. The graph edit distance of* G_1 *and* G_2 *defined by*

$$d(G_1, G_2) = \min_{(e_1 \ldots e_k) \in P(G_1, G_2)} \sum_{i=0}^{k} C(e_i)$$

where $P(G_1, G_2)$ *denotes the set of edit paths transforming* G_1 *into* G_2, *and* C *denotes the edit cost function.*

3 Graph Matching and Transformation Engine

In this section we deal with concepts and theory that our tool is built on them. The *GMTE* encompasses two main processes: the first one is called the pattern matching process and the second one is the graph updating process.

3.1 Exact Matching Process

The matching process between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ consists of finding a mapping M which assigns nodes from G_1 to nodes from G_2, taking into account some predefined constraints. The mapping M is a set of node pairs. Each pair represents the mapping of a node from G_1 with node from G_2 if and only if the mapping is a bijection, preserve adjacency (if two nodes are adjacent in the first graph, their image by the bijection should be adjacent), so M is called isomorphism.

The graph matching algorithm implemented within the *GMTE*, is the one defined in [7]. The algorithm considers a Breadth-first search approach similar to that introduced by the algorithm described by Messmer and Bunke in [8]. The choice to rely on such approach is motivated by the fact that this algorithm is highly effective in cases where matching involves several similar graphs.

For *GMTE* the definition of the left-hand side L is extended to allow the use of variable attributes for nodes and edges labels.

Definition 4. *Let $S_X = X_1, ..., X_i$ be a set of variables. A model graph MG is defined as a 6-tuple $MG = (V, E, L_V, D_{L_V}, L_E, D_{L_E})$, where V, E, L_V and L_E have the same descriptions as given in Definition 1. $D_{L_V} = [\,(\,L_1 \cup S^1_X\,) \cup ... \cup (\,L_n \cup S^n_X\,)]$ is the definition domain of vertex labels, with $S^1_X, ..., S^n_X$ are subsets of S_X. $D_{L_E} = [(E_1 \cup {S'}^1_X) \cup ... \cup (E_n \cup {S'}^n_X)]$ is the definition domain of edge labels, with ${S'}^1_X, ..., {S'}^n_X$ are subsets of S_X.*

Considering the extension previously introduced, we establish some new notions related to label and vertex matching and to valuation merging. Two labels are considered matchable if and only if they are either constant and having same type or if the labels from the model graph is variable it must have the same type as the constant label from the input graph. Based on this notion, two nodes or edges are matchable if and only if they have same number of labels, their labels are, two by two, matchable in respect of their occurrence order, and the results of all parameter matching are consistent. Two valuations Val_1 and Val_2 are consistent if and only if, for every pair $(x_1, value_1)$ belonging to Val_1 and every pair $(x_2, value_2)$ belonging to Val_2, x_1 and x_2 are two different variables (syntactically) or represent the same variable and associate it with the same value $x_1 = x_2$ and $value_1 = value_2$. Typically, a vertex $v_1(x, y, x, 3)$ is not matchable with $v_2(1, 1, 2, 3)$ but is matchable with $v_3 = (1, 2, 1, 3)$ and gives in this case the valuation set $\{(x, 1), (y, 2)\}$ as a result.

3.2 Graph Updating Process

Dealing with graph transformation two major technical problems arise: how to delete L from G and how to connect R with the remaining part of it. To cope with these problems *GMTE* combines two approaches. The first one is the simple pushout *SPO* [9], where a graph transformation rule is (L, K, R). L and R are the left and right hand side graphs of the rule and K is a common subgraph of L and R that will be preserved after the rule application. Also K has a second role which consists of the part that the added nodes will be connected to. The removal of $L \setminus K$ raises the problem of dangling edges (edges without starting or ending node). This approach implies that dangling edges are deleted once $L \setminus K$ is removed.

According to [10] there is another powerful mechanism which is based on *connection instructions* who enrich the formalism of how to connect R. This approach, called *Node Controlled Embedding*, allows connecting nodes from the right-hand side graph to the neighbours of removed nodes from the left-hand side graph.

The *GMTE* rewriting techniques uses the last extension of the *NCE* which is the *edNCE* used for directed and edge labelled graph. For *edNCE* grammars a connection instruction is of the form $(m, \mu, p/q, x, d, d')$ with obvious meaning: a q-labelled edge should be established between x (node from R) and every μ-labelled node of host graph G that is a p-neighbour of m (node from L), with $d, d' \in \{in, out\}$, where d is the old edge direction and d' the new one. The host graph is multi-labelled so μ, p and q are sets of labels. To determine the

applicability of a transformation rule, the approaches presented above are based on the existence of an instance of the sub-graph L. In some cases, it is necessary to express additional conditions to specify conditions relating to the absence of an occurrence in the host graph. Such conditions or restrictions are called negative application conditions (NAC) [11].

3.3 Conditional Rule Schemata

The conditional rule schemata introduced in [12] extend graph transformation with operations on labels. Within the *GMTE* we have two types of operations. Operations on nodes and edges labels of the *Add zone* are called *functions*. When operations are used within nodes and edges labels of the *Delete zone* and *Invariant zone* they are called *conditions*.

Rule graphs used by the *GMTE* are multi-labelled so conditions and functions could be used on any label of node and edge with the respect to the following two conditions:

- Conditions are Boolean expression built in an arithmetic expressions, used as label of nodes and edges within the *Delete zone* and *Invariant zone*. Functions used only on the *Add zone*.
- Var (*Addzone*) $\subseteq$ Var (*Delete zone* + *Invariant zone*).

where $Var(Y)$ is set of all variables in zone Y. Constraint label formalism is added in order to extend the expressiveness of the matching and the transformation. We used the *muParser* [2] which is an extensible high performance math expression parser library written in $C++$. The main objective of this library is to provide a fast and easy way of parsing mathematical expressions.

3.4 Rule and Application Condition

In *GMTE*, the basic representation of a rule is a single graph combining all of the following four zones:

- *Invariant zone*: a subgraph that needs to be present in the input graph in order for the rule to be applicable, this pattern is preserved after the rule application;
- *Absent zone*: a subgraph that must be absent in the input graph to allow the application of the rule;
- *Delete zone*: a subgraph that needs to be present in the input graph in order to be deleted after the rule application;
- *Add zone*: a subgraph that will be added after the rule application.

As we can see, the presence of the *Invariant zone* and *Delete zone* form the positive application condition, and the absence of the *Absent zone* forms the

[2] A fast math parser library Version 2.2.0 (*muParser*), available at `http://muparser.beltoforion.de/`

negative application condition. The rule within the *GMTE* could be extended to meet a much more powerful transformation mechanism assured by the connection instructions as it was defined in the previous Section.

Two main rule application approaches are implemented within the *GMTE*. The first one is to simply apply all rules listed in a rule file to a given input graph. The second one is to recursively apply all rules listed in a rule file to a given input graph, and to all graphs generated by such applications.

4 Inexact Matching

In this section we introduce our approach for inexact graph matching. Graph edit distance is one of the most flexible graph similarity measures. Our approach inline with [13]. In [13] the authors proposed to compute graph edit distance based on bipartite graph matching by means of the Linear Sum Assignment Problem. Their algorithm performs quite efficiently, but it is limited in that it is often applicable for matching two graphs with equal number of nodes. In case of subgraph matching, the authors expand the cost matrix to get a square matrix. Therefore, this expansion increases computation time. As we can see in the sequel, our approach tackles efficiently this problem, a) using a modified version of the $LSAP$ algorithm and b) possibility to directly work on a rectangular cost matrix.

4.1 Node/Edge Edit Distance

To compute the distance between two nodes or edges, we use a modified version of the the Heterogeneous Euclidean Overlap Metric (HEOM) [14] which handles numeric and string labels. But first, we will start by defining a metric to measure the distance between two labels. Given two labels $l_i, l_i^{'}$ (i is the index of the label within the node or the edge attributes) the distance is measured by $labelDistance(l_i, l_i^{'})$ defined as follow:

$$labelDistance(l_i, l_i^{'}) = \begin{cases} 1 & \text{if } l_i \text{ or } l_i^{'} \text{ is missing,} \\ \frac{ed(l_i, l_i^{'})}{\max(|l_i|, |l_i^{'}|)} & \text{if } l_i \text{ or } l_i^{'} \text{ are strings,} \\ \frac{|l_i - l_i^{'}|}{1+|l_i - l_i^{'}|} & \text{if } l_i \text{ or } l_i^{'} \text{ are numerics.} \end{cases}$$

The string edit distance of s and t, denoted by $ed(s,t)$ is the minimal atomic string operations (character insertion, deletion or substitution) needed to transform s into t. when labels are strings, we could see that if they are equal then the distance is 0, and if they are totally different, the distance is 1. In case labels are numeric, the distance is 0 if they have same value and the distance is $\simeq 1$ if $\mid l_i - l_i^{'} \mid \rightarrow \infty$. We define node/edge distance as follow:

$$\delta(n, n') = \sqrt{\sum_{i=0}^{\max(|n|,|n'|)} (labelDistance(l_i, l_i^{'}))^2}$$

$\mid n \mid$ denotes the number of labels within a node or an edge. n and n' could be either two nodes or two edges.

4.2 Cost Matrix

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two multi-labelled graph, where $\mid V_1 \mid = n$ and $\mid V_2 \mid = m$. The cost matrix defined as $C = (c_{ij})_{n \times m}$, where each element $c_{ij} \geq 0$ correspond to the cost of assigning the i^{th} of V_1 to the j^{th} element of V_2. To enhance the matching, information about edges distance could be added to the cost matrix. The technique is somehow similar to [15], except for the cost matrix is rectangular to reduce computation time. For each c_{ij} (assignment cost of node u_i to the node v_j) an adjacency edge cost matrix is generated. The cost resulting from the minimum-cost edge assignment for all edges connected to u_i and v_j is added to c_{ij}.

$$C_{ij} = c_{ij} + \min\{\sum cost(e_{u_i}, e_{v_j})\}$$

where $\min\{\sum cost(e_{u_i}, e_{v_j})\}$ is computed by the algorithm using the adjacency-edge cost matrix of node u_i and v_j. The problem is then to determine the minimum cost of assigning node from G to node from G'.

4.3 Bipartite Graph Matching

Standard graph matching procedures assign nodes and edges of one graph to another using some kind of search tree and trying to minimize the global edit cost. Let n and m be the number of nodes in G and G'. We have $\frac{n!}{m!}$ possibilities of assigning nodes from G to G'. The time complexity of such brute a force algorithm is $O(n^m)$. However, according to [16] the process of assigning nodes can be solved as Linear Sum Assignment Problem $LSAP$. For the previously defined cost matrix the problem is how to match each row to a different column in such a way that the sum of the assignment is minimal. In other words, we want to select n elements of C, so that there is exactly one element in each row and one in each column, and the sum of the corresponding sum is minimal.

4.4 Assignment Algorithm

In [17] the paper considers the classic linear assignment problem with a min-sum objective function, and the most efficient and easily available codes for its solution. Also it gives a survey describing the different approaches in the literature, presenting their implementations, and pointing out similarities and differences. Then it selects eight codes and introduces a wide set of dense instances containing both randomly generated and benchmark problems. According to [17], the modified versions of the Volgenant Jonker algorithm [18] is one of the fastest to solve dense linear assignment problem instances. In [18] authors have made a significant modification that led to speed up the algorithm. This modification is based on a selection procedure that selects a number of small cost elements from the cost matrix.

The *GMTE* adopt the modified versions of the Volgenant Jonker algorithms [18]: for square problem $LAPMOD$ and the non-square problem $LAPMODrow$.

The *LAPMODrow* is a special version of the core oriented *LAP* algorithm, used where the cost matrix is constructed with less rows than columns. For the bipartite matching, *LAPMOD* is used when graphs have equal number of nodes and *LAPMODrow* is used when the nodes number in one graph is less than the other.

5 Comparison with Other Tools

In this section, we compare *GMTE* with other graph transformation tools. The comparison is summarized in Table 1 and covers different criteria. These criteria have been presented in different papers like [3,19]. *GMTE* is categorized as a general purpose tool, however, it can be used through the expressiveness of his rule for model transformations as well.

The third criterion considers advanced rule features, these features increase the expressiveness of rules. *GMTE* and GROOVE support parallel rule application. This rule will be applied to all subgraphs that satisfy the application conditions. Also, *GMTE* like the other tools support standard case application (application on a precise or random matching). *GMTE* supports recursive rules application. GROOVE, PROGRES, Fujaba and GrGen [20] support using regular expressions on edge labels, *GMTE* supports label calculation known as label condition and functions on node and edge labels. Moreover, *GMTE* combines the *SPO* and *edNCE* to support a reach formalism for gluing and embedding technique through the use of connection instructions. Most of the presented tools in this section support either the SPO or the DPO approach, which reduce the expressiveness of the rule.

Table 1. Comparison between *GMTE* and other tools

Tool	Purpose	Typing	Advanced rule features	Editing
AGG	General purpose	Required	-	Graphical
PROGRES	General purpose	Required	Set nodes Star rules Regular expression	Graphical
GReAT	Model transformation	Required	Match condition Recursive pattern	Graphical
GrGen	Multi-purpose	Required	Regular expressions	Textual
VIATRA2	Model transformation	Required	Recursive patterns	Textual
GROOVE	General purpose	Optional	Regular expressions Quantification	Graphical
VMTS	Model transformation	Required	Quantification	Textual
ATOM3	Model transformation	Required	Triple Graph Grammar	Graphical
GMTE	General purpose	Optional	Parallel application, Label calculation, Connection instructions Codification instructions	Textual

The final criterion is whether a tool provides a graphical user interface for editing graphs and rules or is text-based only. *GMTE* can reads the rule graph and the host graph description from input XML files. The standard used is GraphML [21], which is an XML-based file format for graphs. Its main features include supporting directed, undirected, and mixed graphs, hypergraphs, hierarchical graphs, graphical representations, references to external data, application-specific attribute data, and light-weight parsers.

6 Conclusion

In this paper we addressed the problem of tools for graph matching and graph transformation. We presented a tool capable of performing matching and transformation for multi-labelled graphs. Also, we enhanced the rewriting system by extending the expressiveness of rules. To our knowledge, *GMTE* is the first tool that implements the *edNCE* approach and combines it with *SPO* approach, in order to get a rich formalism for both gluing and connecting technique. As well, we improved the formalism through the use of conditional rule schemata which are rule schemata equipped with a Boolean term built on arithmetic expressions. This allows to control rule applications by comparing values of labels. Also GMTE support inexact graph and subgraph matching, by efficiently computing the graph edit distance based on bipartite matching by means of faster and adaptive version the Volgenant-Jonker assignment algorithm.

For other part, we are working on extending our tool with the use of graph transformation system with time. Also, we are looking to implement a faster algorithm for inexact matching in order to reduce computation time. In order to show the efficiency of our approach, we plan to improve autonomic approaches like [22] with graph capabilities for reconfiguration consistency checking purpose.

Acknowledgment. This research is supported by the Itea2 A2Nets (Autonomic Services in M2M Networks) project[3].

References

1. Schürr, A., Winter, A., Zündorf, A.: The PROGRES approach: Language and environment. In: Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G. (eds.) Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools, vol. 3, pp. 487–550. World Scientific (1999)
2. Taentzer, G.: AGG: A tool environment for algebraic graph transformation. In: Münch, M., Nagl, M. (eds.) AGTIVE 1999. LNCS, vol. 1779, pp. 481–488. Springer, Heidelberg (2000)
3. Ghamarian, A.H., de Mol, M., Rensink, A., Zambon, E., Zimakova, M.: Modelling and analysis using groove. STTT 14(1), 15–40 (2012)
4. Lengyel, L., Levendovszky, T., Charaf, H.: Constraint validation support in visual model transformation systems. Acta Cybern. 17(2), 339–357 (2005)

[3] `https://a2nets.erve.vtt.fi/`

5. Balasubramanian, D., Narayanan, A., van Buskirk, C.P., Karsai, G.: The graph rewriting and transformation language: Great. ECEASST 1 (2006)
6. de Lara, J., Vangheluwe, H.: AToM^3: A tool for multi-formalism and meta-modelling. In: Kutsche, R.-D., Weber, H. (eds.) FASE 2002. LNCS, vol. 2306, pp. 174–188. Springer, Heidelberg (2002)
7. Guennoun, K., Drira, K., Diaz, M.: A proved component-oriented approach for managins dynamic software architectures. In: Proc. 7th IASTED International Conference on Software Engineering and Application, Marina Del Rey, CA, USA (2004)
8. Messmer, B.T., Bunke, H.: Efficient subgraph isomorphism detection: A decomposition approach. IEEE Trans. Knowl. Data Eng. 12(2), 307–323 (2000)
9. Ehrig, H., Korff, M., Löwe, M.: Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts. In: Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) Graph Grammars 1990. LNCS, vol. 532, pp. 24–37. Springer, Heidelberg (1991)
10. Rozenberg, G. (ed.): Handbook of graph grammars and computing by graph transformation: volume I. foundations. World Scientific Publishing Co., Inc., River Edge (1997)
11. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundamenta Informaticae 26, 287–313 (1995)
12. Plump, D., Steinert, S.: Towards graph programs for graph algorithms. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 128–143. Springer, Heidelberg (2004)
13. Fankhauser, S., Riesen, K., Bunke, H.: Speeding up graph edit distance computation through fast bipartite matching. In: Jiang, X., Ferrer, M., Torsello, A. (eds.) GbRPR 2011. LNCS, vol. 6658, pp. 102–111. Springer, Heidelberg (2011)
14. Wilson, D.R., Martinez, T.R.: Improved heterogeneous distance functions. J. Artif. Int. Res. 6(1), 1–34 (1997)
15. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vision Comput. 27(7), 950–959 (2009)
16. Riesen, K., Neuhaus, M., Bunke, H.: Bipartite Graph Matching for Computing the Edit Distance of Graphs, pp. 1–12 (2007)
17. Dell'Amico, M., Toth, P.: Algorithms and codes for dense assignment problems: the state of the art. Discrete Appl. Math. 100(1-2), 17–48 (2000)
18. Volgenant, A.: Linear and semi-assignment problems: A core oriented approach. Computers & OR 23(10), 917–932 (1996)
19. Fuss, C., Mosler, C., Ranger, U., Schultchen, E.: The jury is still out: A comparison of agg, fujaba, and progres. ECEASST 6 (2007)
20. Geiß, R., Batz, G.V., Grund, D., Hack, S., Szalkowski, A.: GrGen: A fast SPO-based graph rewriting tool. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 383–397. Springer, Heidelberg (2006)
21. Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., Marshall, M.S.: GraphML progress report. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, p. 501. Springer, Heidelberg (2002)
22. Ben-Halima, R., Jmaiel, M., Drira, K.: A QoS-oriented reconfigurable middleware for self-healing Web services. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2008), Beijing, China. IEEE Computer Society Press (2008)