Transformation-based operationalization of Graph Languages

Erhard Weinell

RWTH Aachen University, Department of Computer Science 3,
Ahornstrasse 55, D-52074 Aachen, Germany,
Weinell@cs.rwth-aachen.de

Abstract

Graph Languages[1] emerged during the seventies from the necessity to process data structures with complex interrelations. Nowadays, various variants of these languages can be found for querying [1][2], in-place transforming [3][4], and translating graph structures [5][6]. Still, new graph languages supporting different paradigms and usage scenarios are proposed regularly. In fact, languages tailored for a dedicated application domain can be restricted to a concise and clear syntax representation, thus reducing effort to learn and apply them. Effectively aiding the development of graph languages, even though considering the already existing ones, therefore remains an important working topic.

Constructing specialized graph languages, considering them as special case of domain-specific modeling languages, is supported by various frameworks and so-called Meta-CASE tools, e.g. [7]. Operational implementations of these languages is usually achieved by customizing template-based code generators. However, graph languages, in contrast to purely static modeling languages, are inherently complex to implement due to the required pattern matching facility, and the possibly required non-deterministic execution engine.

As alternative to the usual code generation approach, I propose a solution to implement graph languages by *transformation*. The approach is based on an extensible core graph language, to which rules modeled in a specialized graph language are transformed. Extensions can be added to the core language to approximate both languages's conceptual levels, and thus to narrow their "semantic gap". In contrast, a code generation module would have to span a significantly larger gap from a high-level specification language to an imperative or object-oriented programming language.

A coarse-grained overview on the presented approach is given in Figure 1. Technically, this platform is built on top of the graph-oriented database DRAGOS [8]. Thanks to DRAGOS' exchangeable backends, language implementations gain access to established storage solutions like relational databases or model repositories.

[1] The term *Graph Language* subsumes languages for querying and transforming graphs, and especially from the overlapping of these areas.

To construct a new graph language, developers usually build an editor based on the language's concrete syntax model, be that a textual or visual one. Based on this, a partly generic export facility transfers instances of this language, e.g. entered by the user at runtime, into the graph database as abstract syntax trees (ASTs). This intermediate storage facility decouples further processing from the actual concrete syntax and from the applied editing technology.

Afterwards, as the first curved arrow suggests, the ASTs are transformed into instances of the provided core graph language, DRAGULA. This transformation, which needs to precisely capture the specialized language's intended semantics, can be modeled in a rule-based way. For this purpose, a simple uni-directional model transformation language is provided. Technically, this language's rule instances are stored in an additional repository in the database, and are transformed to the core language, too.

Finally, the generated core language rules can be evaluated by the corresponding engine, thereby referring to the database's data repository. Both the rule engine and the data repository are subject to user interaction, e.g. to select rules for invocation or to directly inspect the stored graphs. This talk primarily discusses the first curved arrow, whereas the second one has been described in [9], and the core language's extensibility in [10].

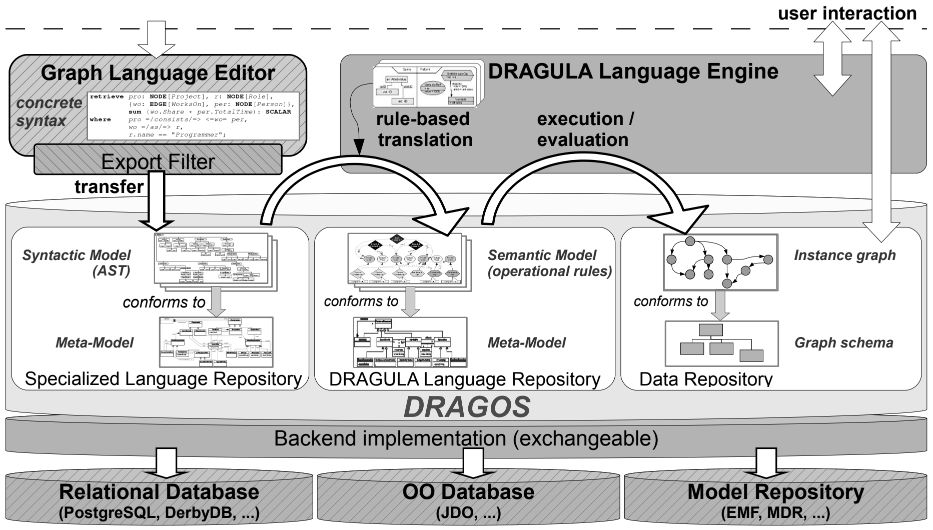
```
retrieve pro: NODE[Project], r: NODE[Role],
         {wo: EDGE[WorksOn], per: NODE[Person]},
         sum (wo.Share * per.TotalTime): SCALAR
where    pro =/consists/=> <=wo= per,
         wo =/as/=> r,
         r.name == "Programmer";
```

Fig. 1. Architectural overview

Summary. The proposed solution eases operational implementations of graph languages, using a rule-based transformation approach. The DRAGULA language is well-suited to implement different kinds of languages, e.g. for queries and transformations. Extensions allow to capture additional functionality.

Related work. Modeling a domain-specific language's dynamic semantics is offered by some meta-case tools, e.g. presented in [11]. In contrast to the present work relying on a core graph language as target domain, the authors apply petri nets for this purpose. Though restricting their approach in expressing graph languages, petri nets naturally provide valuable analytical properties.

Existing core graph languages can also be found in the literature, such as the lately proposed GP [12]. GP offers core functionality for a basic graph model, whilst DRAGULA is tailored towards complex models including hierarchical graphs and hypergraphs. Furthermore, DRAGULA focusses on extensibility to allow concise mapping of specialized graph languages [10]. Like GP already does, DRAGULA will soon support non-deterministic rule application.

References

1. Consens, M., Mendelzon, A.: GraphLog: a visual formalism for real life recursion. In: Proc. of the ACM Symp. on Principles of Database Systems. (1990) 404–416
2. Kullbach, B., Winter, A.: Querying as an enabling technology in software reengineering. In: Proc. of the 3rd Europ. Conf. on Software Maintenance and Reengineering, IEEE Computer Society Press (1999) 42–50
3. Schürr, A., Winter, A.J., Zündorf, A.: The PROGRES approach: Language and environment. [13] 487–550
4. Ermel, C., Rudolf, M., Taentzer, G.: The AGG approach: Language and environment. [13] 551–603
5. Agrawal, A., et al.: The design of a language for model transformations. Software and Systems Modeling **5**(3) (September 2006) 261–288
6. Balogh, A., Varró, D.: Advanced model transformation language constructs in the VIATRA2 framework. In: ACM Symp. on Applied Computing (SAC 2006), ACM Press (2006) 1280–1287
7. Ebert, J., Süttenbach, R., Uhe, I.: Meta-CASE in practice: a case for KOGGE. In: Advanced Information Systems Engineering, Proc. 9th Int. Conf. (CAiSE'97). Volume 1250 of Lect. Notes in Comp. Sci., Springer (1997) 203–216
8. Böhlen, B.: Ein parametrisierbares Graph-Datenbanksystem für Entwicklungswerkzeuge. PhD thesis, RWTH Aachen (2006)
9. Weinell, E.: Adaptable Support for Queries and Transformations for the DRAGOS Graph-Database. In Schürr, A., Nagl, M., Zündorf, A., eds.: Proc. of the 3rd Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE). Volume 5088 of Lect. Notes in Comp. Sci., Springer (2008) 369–411
10. Weinell, E.: Extending graph query languages by reduction. [14]
11. de Lara, J., Vangheluwe, H.: Translating model simulators to analysis models. In Fiadeiro, J.L., Inverardi, P., eds.: FASE. Volume 4961 of Lect. Notes in Comp. Sci., Springer (2008) 77–92
12. Manning, G., Plump, D.: The GP programming system. [14]
13. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools. Volume 2. World Scientific (1999)
14. Ermel, C., Heckel, R., de Lara, J., eds.: Graph Transformation and Visual Modeling Techniques, 7th Intl. Workshop. Volume 10 of Elec. Comm. of the EASST. (2008)