The Graph Rewriting Language and Environment PROGRES

Ulrike Ranger and Erhard Weinell

RWTH Aachen University of Technology
Department of Computer Science 3 (Software Engineering)
Ahornstraße 55, D-52074 Aachen, Germany
Ranger|Weinell@cs.rwth-aachen.de
http://se.rwth-aachen.de/progres

Introduction: PROGRES (PROgrammed Graph REwriting Systems) [1] has been developed since the late 1980s, and thus constitutes one of the eldest implemented graph rewriting languages and environments. It is based on the logic-oriented approach to graph grammars. The PROGRES language allows to model the structure and the behavior of software applications in a visual and declarative way. Thereby, it is not tied to a specific application domain, but may be used for arbitrary software applications (see [2] for a simple example). Besides an extensive language, PROGRES offers an integrated modeling environment, including a syntax-directed editor, an interpreter, and a debugger. Furthermore, the environment supports rapid prototyping by generating executable source code from a specification. The code can be embedded into a visual prototype.

The graph language PROGRES: PROGRES offers language constructs for defining graph schemas and graph transformation rules. Thereby, it uses directed, attributed, node and edge labeled graphs as underlying data model.

Graph Schema: A PROGRES graph schema consists of node types and edge types where the edge types model relations between the node types. Following the object oriented programming paradigm, attributes and graph transformation rules may be defined for every node type. For modeling complex navigations through a host graph, paths may be defined in the schema describing such navigations by using operators like the Kleene star. Furthermore, a schema may contain integrity constraints imposing advanced restrictions on valid host graphs.

Graph Transformation Rules: In PROGRES, graph transformation rules are distinguished into *simple rules* and *combined rules*. A simple rule describes a graph transformation in a visual way, consisting of a left-hand side (LHS) and a right-hand side (RHS). The LHS specifies a graph pattern for which an according match has to be found in the host graph. If such a match could be found, it is transformed according to the RHS. For modeling variable coherences, rules may contain optional, negative or even set-valued nodes and negative edges. Furthermore, embedding clauses may be used for integrating the transformed sub graph into the remaining host graph. In contrast to a simple rule, a combined rule composes several rules by textual control structures, e. g. loops and conditions. Thus, a complex transformation may be modeled by a combined rule. PROGRES also comprises an OCL-like language for formulating constraints in transformation rules or paths.

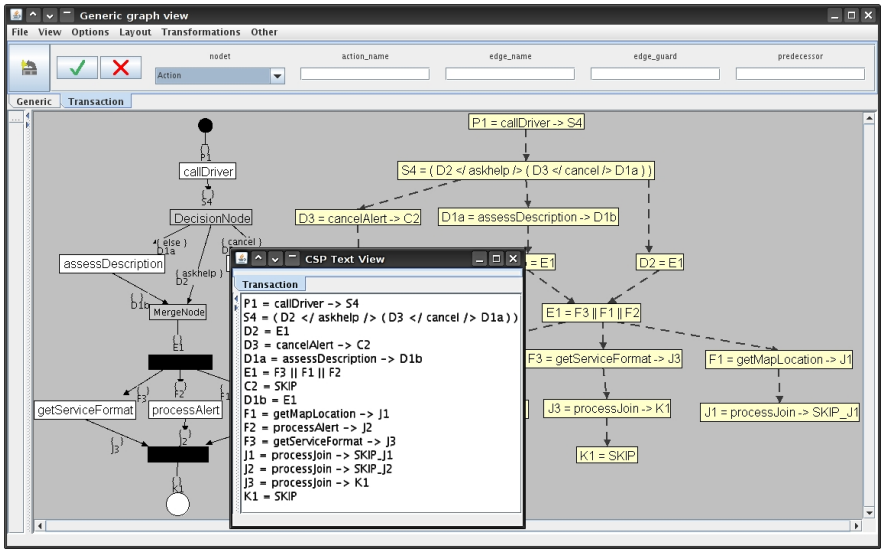

Fig. 1. Screenshot of a generated prototype

Environment: As example for the extensive PROGRES environment, the rapid prototyping framework UPGRADE is sketched here. Using the code generated from a PROGRES specification, UPGRADE is able to display host graphs and to invoke the specified transformation rules on them. As host graphs tend to become too incomprehensible for inspection, UPGRADE offers a display mechanism which is highly user-configurable: *Filters* allow to hide instances of node and edge types not relevant to the user. They also allow to collapse edge-node-edge constructs into a single attributed edge. *Display attributes* modify colors, fonts, and shapes of nodes and edges. All attributes can be refined by conditions, e.g. to mark a node red if one of its attributes exceeds a certain threshold. *Label attributes* assign labels to the displayed graph entities, such as type information or node attributes. All of these attributes can be assigned to each type, or the type's hierarchy.

Figure 1 shows a prototypical editor created for the case study presented in [2]. Besides the graphical representation, a textual view window displays contents derived from the according nodes labels. UPGRADE supports the development of such customized views by a set of extendable base classes.

References

1. Schürr, A., et al.: The PROGRES approach: Language and environment. In Ehrig, H., et al., eds.: Handbook on Graph Grammars and Computing by Graph Transformation. Vol. 2. 1^{st} edn. World Scientific, Singapore (1999) 487–550
2. Varró, D., et al.: Graph Transformation Tools Contest on the Transformation of UML Models to CSP. In: this volume. (2008)