Story Diagrams: A new Graph Rewrite Language based on the Unified Modeling Language and Java

Thorsten Fischer, Jörg Niere, Lars Torunski, Albert Zündorf
AG-Softwaretechnik, Fachbereich 17, Universität Paderborn, Warburger Str. 100, D-33098 Paderborn, Germany; e-mail: [tfischer|nierej|torunski|zuendorf]@uni-paderborn.de

Abstract. Graph grammars and graph rewrite systems improved a lot towards practical usability during the last years. Nevertheless, there are still major problems to overcome in order to attract a broad number of software designers and developers to the usage of graph grammars and graph rewrite systems. Two of the main problems are, (1) that current graph grammar notations are too proprietary and (2) that there exists no seamless integration of graph rewrite systems with common (OO) design and implementation languages like UML and C++ or Java. Story Diagrams are a new graph rewrite language that tries to overcome these deficiencies. Story Diagrams adopt main features from Progres, e.g. explicit graph schemes, programmed graph rewriting with parameterized rules, negative, optional and set-valued rule elements. Story diagrams extend common graph models by offering direct support for ordered, sorted, and qualified associations and aggregations as known from the object-oriented data model. Story Diagrams adopt UML class diagrams for the specification of graph schemes, UML activity diagrams for the (graphical) representation of control structures, and UML collaboration diagrams as notation for graph rewrite rules. Story Diagrams are translated to Java classes and methods allowing a seamless integration of object-oriented and graph rewrite specified system parts.

1 Introduction

At the last graph grammar conference in Williamsburg four years ago, Blostein stated a number of requirements for the industrial use of graph grammars and graph rewrite systems as a design and implementation means, cf. [BFG96]. They should be less difficult to learn. They should be expressive. It must be possible to use them for fractions of a software system (in order to get started). Even applied to larger fractions, they should work seamlessly with standard system parts. Their execution should be fast and environments are needed.

During the last four years theory, implementation and application of graph grammars and graph rewrite systems improved a lot. In theory, the expressive power of most ap-
proaches was increased by attribute conditions, negative application conditions, general constraints, and control structures [Roz97]. Graph grammar and graph rewriting environments emerged and improved, meeting many of the requirements mentioned in [BFG96]. For example the AGG system was extended by a sophisticated graph pattern matching algorithm for the automatic execution of AGG rules, cf. [Rud97]. The Progres environment offers now means for the rapid prototyping of applications from their graph grammar specification [SWZ95a].

Despite these improvements, graph grammars and graph rewrite systems did not yet succeed to attract a broad number of software designers and developers. Two of the main problems are (1) that current graph grammar notations are too proprietary and (2) that there exists no seamless integration of graph rewrite systems and common (OO) design like UML (cf. [UML97]) and implementation languages like C++ or Java.

Story Diagrams are a new graph rewrite language that tries to overcome these deficiencies. Story Diagrams adopt main features from Progres [SWZ95a], e.g. directed, attributed, node and edge labeled graphs, explicit graph schemes, programmed graph rewriting with parameterized rules, negative, optional and set-valued rule elements. However, Story Diagrams extend the Progres graph model by direct support for ordered, sorted, and qualified associations and aggregations. Thus, the data model of Story Diagrams corresponds to the object-oriented data model.

Accordingly, Story Diagrams exploit UML class diagrams for the specification of graph schemes. Story Diagrams adopt UML activity diagrams for the (graphical) representation of control structures. The activities of a Story Diagram contain either program code (like in UML) or graph rewrite rules. The graph rewrite rules use an UML collaboration diagram like notation. Due to our experiences in several industrial projects, this notation looks quite familiar to software engineers.

Chapter 2 introduces the key features of our new graph grammar language. Chapter 3 gives a short introduction of the formal semantics of Story Diagrams. Chapter 4 outlines the translation of Story Diagrams to Java code. Chapter 5 summarizes our results and highlights some future work.

2 Story Diagrams, the language

Story Diagrams rely on an explicit graph scheme that defines static properties of the specified data structures and allows consistency checks of the dynamic specification. We use standard UML class diagrams for this purpose, since they are familiar to our target customers, i.e. software engineers. Figure 1 shows a screen shot of an UML class diagram for a lift simulator used as a running example within this paper. The screenshot is taken from the Fujaba\textsuperscript{1} environment (cf. [FNT98]).

Note, the qualified association between houses and their levels. In standard graph models, multiple links of a certain type attached to a given node are indistinguishable and have no specific order, cf. [Roz97]. The object-oriented data model extends this concept by ordered and sorted and qualified associations. In case of an ordered associations, the

\begin{footnotesize}
1. Fujaba is an acronym for “From UML to Java And Back Again”
\end{footnotesize}