From Graph Transformation to Software Engineering and Back

Luciano Baresi[1] and Mauro Pezzè[2]

[1] Politecnico di Milano,
Dipartimento di Elettronica e Informazione,
Milano, Italy
baresi@elet.polimi.it
[2] Università degli Studi di Milano-Bicocca,
Dipartimento di Informatica Sistemistica e Comunicazione,
Milano, Italy
pezze@disco.unimib.it

Abstract. Software engineers usually represent problems and solutions using graph-based notations at different levels of abstractions. These notations are often semi-formal, but the use of graph transformation techniques can support reasoning about graphs in many ways, and thus can largely enhance them.
Recent work indicates many applications of graph transformation to software engineering and opens new research directions. This paper aims primarily at illustrating how graph transformation can help software engineers, but it also discusses how software engineering can ameliorate the practical application of graph transformation technology and its supporting tools.

1 Introduction

Software engineering aims at developing large software systems that meet quality and cost requirements. The development process that moves from the initial problem to the software solution is based on models that describe and support the development during the different phases. Models are key elements of many software engineering methodologies for capturing structural, functional and non-functional aspects. Popular methodologies prescribe various models – with different degrees of formality, flexibility, and analyzability – to solve the different problems. For example, UML proposes some semi-formal diagrammatic languages: class, object, component, and deployment diagrams for modeling structural aspects, and use case, sequence, activity, collaboration, and statechart diagrams for modeling behavioral aspects [24].

The syntax and semantics of these models are defined informally with different degrees of precision. Although users and tools agree on the main syntactic and semantic aspects, important details are often given different – and frequently incompatible – interpretations.

The scientific community agrees on the need to improve current practice by increasing the degree of formality of these notations. Unfortunately, formal

H.-J. Kreowski et al. (Eds.): Formal Methods (Ehrig Festschrift), LNCS 3393, pp. 24–37, 2005.

methods have not found wide application so far, and cannot be easily used in conjunction with popular modeling notations [6]. This is where graph transformation (GT) provides unique features to strengthen diagrammatic models by adding formality. Similarly to grammars for textual languages, GT can formally describe the concrete and abstract syntaxes of modeling languages, but it can also formalize the semantic aspects, and thus provides a strong basis for reasoning on diagrammatic models at all levels.

Formalizing the concrete and abstract syntaxes eliminates ambiguities and contradictions and supports automatic checks for consistency and correctness. GT allows the designer to describe the semantics both operationally and denotationally. Operational semantics can be given by describing the legal evolutions of models in terms of GT rules. Denotational semantics can be expressed by mapping models onto semantic domains by means of rules that mimic syntactical changes onto the chosen semantic domain [28].

GT is well supported by tools, which are useful for solving many problems and validating new ideas and applications, but often they do not scale well. When the size of the application grows, and requires significant mediation between theory and performance, software engineering principles can provide useful ideas. They can contribute significantly in this direction and help GT experts move towards complex problems and applications.

The main goal of this paper is to frame the opportunities offered by GT to software engineering by illustrating sample cases and proposing additional applications not fully explored yet. The paper also suggests ways for improving GT technology and tools, inspired by well-known software engineering principles, to address practical problems.

The paper is organized as follows. Section 2 discusses the opportunities offered by GT as modeling language by touching the use of GT for modeling and reasoning on particular aspects of software systems. Section 3 illustrates the potentiality of GT for modeling and verifying notations, i.e., for formalizing the concrete and abstract syntaxes as well as the semantics of notations, thus providing a uniform framework for modeling heterogeneous notations, and fostering analysis tools. Section 4 indicates how software engineering can help experiment and introduce graph transformation in current modeling practice. Section 5 proposes some future directions and concludes the paper.

2 Graph Transformation for Models

GT provides a formal and intuitive way for describing systems and their evolution. For example, GT has been proposed to model software architectures. Architectural styles constrain the connectivity among components to guarantee the regularity of architectural models, and thus improve the maintainability and evolvability of systems [10].

GT provides a natural way for formalizing architectural styles by means of rules that support the creation of models that comply with the style by construction. Le Metayér [19] first and then Hirsch et al. [12] investigated different approaches to model software architectural styles by using GT; Baresi et al. [3]