A Formal Framework for the Development of Concurrent Object-Based Systems*

Leila Ribeiro[1], Fernando Luís Dotti[2], and Roswitha Bardohl[3]

[1] Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
leila@inf.ufrgs.br
[2] Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil
fldotti@inf.pucrs.br
[3] International Conference and Research Center for Computer Science Schloss Dagstuhl, Germany
rosi@dagstuhl.de

Abstract. In this paper we present a framework for developing concurrent object-based systems. The framework is based on graph grammars and includes techniques for specification, simulation, animation and verification.

1 Introduction

The development of methods and techniques to aid the construction of correct concurrent software systems has been a challenge for computer scientists for many years. In particular with the boom of intra- and Internet systems as well as the development of highly parallel and vectorial computer architectures, the commercial importance of software applications for distributed and concurrent systems increased significantly and the need for correctness gained a new perspective. This triggered a lot of research activities in the areas of semantical models (e.g. CCS [30], event structures [39], I/O automata [28] – see [40] for an overview), formal specification languages (e.g. LOTOS [37], Estelle [24], TLA [26], Petri nets [32, 33], graph grammars [20, 18]), verification techniques and tools (e.g. SMV [29], SPIN [23]). Many of such activities resulted in quite powerful models that were able to express and reason about concurrent and distributed systems, and were applied successfully to protocol specifications (system states are quite simple because usually they do not have any structure, and important events are state changes triggered by signals or messages). However, software engineers often use (semi-formal) languages like the UML [6] to specify real-life systems with complex states that cannot be captured easily and naturally by existing formal techniques. The formal background of users in industry was rather

* This work was partially sponsored by GRAPHIT (CNPq/DLR), ForMOS (FAPERGS/CNPq), PLATUS (CNPq), IQ-Mobile II (CNPq/CNR) and DACHIA (FAPERGS/IB-BMBF) Research Projects, and partially developed in collaboration with HP Brasil.

H.-J. Kreowski et al. (Eds.): Formal Methods (Ehrig Festschrift), LNCS 3393, pp. 385–401, 2005.

poor, and they were not familiar with the required mathematical notation for using formal specification languages to describe systems and their properties. Due to the size and complexity of distributed and concurrent software applications nowadays, the costs of finding and correcting bugs in the implementation are even higher in this kind of systems. Thus, the software development process would be greatly improved with a comprehensive analysis of functional as well as performance requirements during the specification phase. For such an analysis a formal model of the application would be useful. The development of a formal model, however, is not supported by most of the UML languages used in industry.

With the aim of bridging the gap between the semi-formal languages used in practice and the formal notation required to prove correctness, an international cooperation project between Brazil and Germany, leaded by Prof. Hartmut Ehrig, started in 1994: the GRAPHIT project (DLR/CNPq) [3]. Originally, the project involved two universities (TU-Berlin and UFRGS) and two industrial partners (MSB and Nutec); later further partners joined the project (Universitt Stuttgart and PUCRS). The idea was to develop a formal specification language with a graphical layout, preferably following the object-oriented style used in practice, and relying on few simple and powerful concepts. This would allow users to easily understand and build specifications with this language. Graph grammars seemed to be a perfect basis due to some of its inherent characteristics: graphical description of states (even complex ones can be better understood using a suitable graph representation), changes of states can be easily specified via relationships between graphs (rules), concurrency is naturally described (implicitly, through parallel applications of rules). In the following years a lot of research activities took place in order to define a suitable specification language as well as structuring and analyzing techniques for distributed and concurrent object-based systems in the GRAPHIT and in follow-up projects. In this paper we will review the main results we obtained for specification, simulation, animation and verification of concurrent object-based systems, and present the necessary steps to be followed in order to improve the software development process.

One of the results of the GRAPHIT project was the development of the specification language Object-Based Graph Grammars (OBGG) [12]. This language has been strongly influenced by the composition operators for graph grammars presented in [34] (a Ph.D. thesis advised by Prof. Hartmut Ehrig). The main idea is that a system is constructed by composing objects belonging to different classes. Each class is specified with a graph grammar (with special characteristics giving the language an object-based style natural for non-academic users). In section 2 we present the main concepts of Object-Based Graph Grammars.

To be really useful in practice, there must be techniques (and possibly tools) for the analysis of a specification. With this aim we have built a simulation tool for OBGG (the PLATUS tool[1]). With this tool it is possible not only to execute

[1] The underlying concepts of the simulator as well as a prototype of the tool were developed in the PLATUS project (CNPq).