The Journal of Systems and Software 82 (2009) 292–306

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Specifying behavioral semantics of UML diagrams through graph transformations

Jun Kong [a,*], Kang Zhang [b], Jing Dong [b], Dianxiang Xu [a]

[a] North Dakota State University, Fargo, ND 58105, United States
[b] The University of Texas at Dallas, Dallas, TX, United States

ARTICLE INFO

Article history:
Received 8 October 2007
Received in revised form 16 June 2008
Accepted 16 June 2008
Available online 27 June 2008

Keywords:
Graph transformation
Graph grammars
Visual programming
Visual languages
UML
Behavioral semantics
Object-oriented systems

ABSTRACT

The Unified Modeling Language (UML) has been widely accepted as a standard for modeling software systems from various perspectives. The intuitive notations of UML diagrams greatly improve the communication among developers. However, the lack of a formal semantics makes it difficult to automate analysis and verification. This paper offers a graphical yet formal approach to specifying the behavioral semantics of statechart diagrams using graph transformation techniques. It supports many advanced features of statecharts, such as composite states, firing priority, history, junction, and choice. In our approach, a graph grammar is derived automatically from a state machine to summarize the hierarchy of states. Based on the graph grammar, the execution of a set of non-conflict state transitions is interpreted by a sequence of graph transformations. This facilitates verifying a design model against system requirements. To demonstrate our approach, we present a case study on a toll-gate system.

1. Introduction

Compared with texts, graphs are more intuitive in expressing structural information. Therefore, graphical notations have been extensively used in software development. As a visual modeling language, the Unified Modeling Language (UML) (Rumbaugh et al., 2005) includes various diagrams that specify software artifacts from various points of view. For example, the class diagram models the static structure of a system whereas the statechart diagram describes the behavior of the objects of a class. The intuitive nature of UML notations greatly facilitates distribution and communication of software artifacts among different developers. However, UML lacks a precise semantics, making it difficult to automate verification and analysis. Providing a precise semantics for UML diagrams has gained much attention (Bruel et al., 1998; Evans et al., 1999; Geiger and Zündorf, 2004; Kuske, 2001; Kuske et al., 2002).

Graph transformation offers a computational paradigm of mathematical precision and visual specification (Varró et al., 2002). It provides a means for specifying the semantics of UML diagrams. In general, graph transformation defines computation in a multi-dimensional fashion based on a set of rewriting rules, i.e., *productions*. Each production consists of two parts: a left graph and a right graph. The difference between the two visually indicates the changes caused by a computation. Using graphs to represent the states of a software application, the behavioral semantics can be captured naturally through a sequence of productions, i.e., transitions from one graph (representing the current state) to another (representing the next state). Explaining one set of visual notations (e.g., UML diagrams) by another with a precise meaning (e.g., graph transformation productions) reduces the gap between the specifying language and the specified language. This has been demonstrated by the successful applications of graph transformations to the behavioral semantics of state diagrams (Engels et al., 2000; Kuske, 2001).

Graph-transformation-based approaches (Baresi and Pezzè, 2001; Engels et al., 2000; Gogolla et al., 1998; Kuske, 2001; Varró et al., 2002) are suitable for specifying the semantics of UML statechart diagrams since there is no need to convert from graphical notations to textual/mathematical formalism (Crane et al., 2005). Although some of those approaches support composite states and firing priority, none of them covers the important features of history, junction and choice according to Crane et al. (2005). This paper presents a visual yet formal approach that supports those important features in the UML statechart diagrams. In particular, our approach can automatically translate a UML statechart diagram to a graph grammar with a precise semantics. The automation relieves the burden of learning graph grammar. Furthermore, our approach can lead to the development of a new verification framework. Existing verification frameworks often rely on transformation of system requirements in some formal language, which may cause a mismatch between system requirement and formal specification. In comparison, our approach can directly

* Corresponding author. Tel.: +1 701 231 8179; fax: +1 701 231 8255.
E-mail addresses: jun.kong@ndsu.edu (J. Kong), kzhang@utdallas.edu (K. Zhang), jdong@utdallas.edu (J. Dong), dianxiang.xu@ndsu.edu (D. Xu).

0164-1212/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2008.06.030

verify the defined behavioral semantics against system requirements specified in the form of sequence diagrams.

In our approach, the hierarchy of states in a statechart diagram is automatically formalized as a graph grammar, which extends a graph transformation system by defining an initial graph and classifying terminal and non-terminal objects. Accordingly, the state transition is implemented through a combination of a *parsing process* and a *generating process*. More specifically, starting from an initial graph, the generating process can generate well-formed graphs by iteratively applying productions in the forward direction (Blostein et al., 1994). The parsing process, on the other hand, can recognize the membership of a graph based on a sequence of production applications in the reverse direction (Blostein et al., 1994). It is used to recognize the source state of a state transition while the generating process is to generate the target state. In addition, a set of algebraic structures are abstracted to control the sequence of production applications due to the complex state entries in UML statechart diagrams. As such, our graph-grammar-based approach provides a foundation for directly executing UML models.

Executable UML models (Mellow and Balcer, 2002; Raistrick et al., 2004; Schattkowsky and Müller, 2004, 2005; Starr, 2001), which emphasize the behavioral aspect of a software artifact, can keep specification and implementation consistent. A UML model can be executed by translating it to some platform-dependent code through a code generator. An alternative approach is to directly execute UML models with a precise semantics on a *UML virtual machine* (*UVM*) (Schattkowsky and Müller, 2004, 2005). In our approach, the integrated behavioral semantics of class diagram, statechart diagram and object diagram is defined precisely by two sets of productions. One set of productions are organized in the form of graph grammar that interprets the state transition of objects; and the other set of productions specify the dynamic reconfiguration of object diagrams. The applications of two sets of productions are synchronized according to event dispatching. Our approach is well supported by the methodology of automatic visual language generation (Costagliola et al., 2004; Karsai et al., 2003; Zhang et al., 2001b). The generated language environment is considered to be a UML virtual machine that supports syntax-correct design of visual models and simulates the execution of integrated UML diagrams. Compared with other graph-transformation-based approaches (Ermel et al., 2005; Gogolla et al., 2002; Hölscher et al., 2006; Kuske et al., 2002; Ziemann et al., 2004a,b), which support the execution of integrated UML models, our approach can handle composite states that are typically needed in real-world modeling while it is challenging to effectively recognize and generate composite states due to the state explosion problem.

Various grammar formalisms (Costagliola et al., 1997; Costagliola and Polese, 2000; Rekers and Schürr, 1997; Schürr et al., 1995; Zhang et al., 2001a) have been proposed for different purposes. Most of them use nodes to represent objects and edges to model relations between objects in an abstract syntax. Different from these formalisms, the spatial graph grammar (SGG) (Kong et al., 2006) introduces spatial notions to the abstract syntax. In the SGG, nodes and edges together with spatial relations construct the pre-condition of a production application. The direct representation of spatial information in the abstract syntax can make productions easy to understand since grammar designers often design rules with similar appearances as the represented graphs. Using spatial information to directly model relationships in the abstract syntax is coherent with the concrete representation, which avoids converting spatial information to edges. Therefore, our approach uses the SGG as the underlying formalism to specify the behavioral semantics of statechart and object diagrams.

The contributions of this paper can be summarized as follows:

- Our approach integrates a sound formalism with visual notations to specify the behavioral semantics of statechart diagrams: using one set of visual notations with a precise meaning (e.g., graph transformation) to explain another set of visual notations (e.g., UML statechart diagrams) reduces the efforts of converting graphical notations to textual/mathematical formalisms.
- Among the graph-transformation-based approaches, our approach is the first to address all of the important features of statechart diagrams, including composite state, initial pseudo-state, final state, deepHistory, shallowHistory, join, fork, junction, and choice. Pseudo-states are useful in practical behavior modeling, but they require a sophisticated mechanism to control the state transition when entering a composite state. Our approach defines an efficient control mechanism by mapping each specific state entry onto a corresponding graph transformation rule.
- Our approach automatically converts the hierarchy of states to a graph grammar, and correspondingly applies a parsing process and a generating process to execute state transitions. This efficiently addresses the state explosion problem because a small number of productions can specify a large number of state combinations. Furthermore, it can enforce a consistent state transition.
- This paper also presents a case study, which illustrates the definition of an integrated behavioral semantics and the verification of consistency between a design model and system requirements. The integrated behavioral semantics is defined in the form of a spatial graph grammar, which naturally represents relations through spatial configuration and thus reduces the gap between the abstract and concrete representations of models. Different from other verification methods, our approach can directly apply a use case scenario (represented as a sequence diagram) to the defined behavioral semantics without the need of translating system requirements into some formal language.

In summary, this paper provides a visual yet formal approach to interpreting state transitions in UML statechart diagrams. According to the behavioral semantics of UML statechart diagrams, we define an integrated behavioral semantics, which provides a solid foundation for verifying a design model against system requirements.

The rest of this paper is organized as follows: Following a running example in Section 2, Section 3 introduces the spatial graph grammar – the theoretical foundation of our approach. Section 4 gives an overview of our approach. Section 5 illustrates the semantic specification and verification based on a toll-gate system. Section 6 discusses related work and Section 7 concludes the paper.

2. A motivating example

This section illustrates a toll-gate system (Kong et al., 2005), which is used as a running example to illustrate semantic specification and verification in the following sections. In a road traffic pricing system, drivers of authorized vehicles are charged at toll gates. The tolls are placed at special lanes called *green lanes*. A driver has to install a device (called an *ezpay*) inside his/her vehicle's windshield in order to pass a green lane. The registration of an authorized vehicle having an *ezpay* includes owner's personal data (such as name, date of birth, driver license number, bank account number and vehicle registration number). Each toll gate has a sensor that reads *ezpay*. The information read is stored by the system and used to debit the respective account. When an authorized vehicle passes through a green lane, a green light is turned on. If an un-authorized vehicle passes through it, a camera takes a photo of the vehicle's license plate.