Static Safety Analysis of UML Action Semantics for Critical Systems Development

Zsigmond Pap, Dániel Varró

Dept. of Measurement and Information Systems
Budapest University of Technology and Economics
H-1521 Budapest, Magyat Tudósok krt.2.
[papzs,varro]@mit.bme.hu

Abstract. The Action Semantics for UML provides a standard and platform independent way to describe the behavior of methods and executable actions in object-oriented system design prior to implementation allowing the development of highly automated and optimized code generators for UML CASE tools.
Keywords: Action Semantics, safety criteria, UML, safety critical systems

1 Introduction

Recently, the Unified Modeling Language (UML) [11] has become the de facto standard visual object-oriented modeling language. It is used in systems engineering with a wide range of applications. Its major success is originating in the fact that UML (i) is a *standard* (uniformly understood by different teams of developers) and *visual* language (also meaningful to customers in addition to system engineers and programmers).

Many applications demonstrated the usability and adaptability of UML ranging from small, embedded systems to large scale distributed systems to safety critical applications. In the current paper, the main focus will be put on the design of *safety critical reactive systems* (which is a prospering application field of UML as demonstrated e.g., by [12]) *using UML Action Semantics* specifications.

Unfortunately, writing incorrect specifications in Action Semantics turns out to be disturbingly easy which is unacceptable and dangerous for critical applications. Therefore, we evaluate to what an extent Action Semantics fulfills the famous 47 formal criteria of Nancy G. Leveson [10, 8] which is considered to be the most widely accepted guidelines that should be satisfied to avoid flaws in the specification of safety critical applications.

The rest of the paper is structured as follows. Section 2 provides a brief overview of the safety criteria widely accepted for designing safety critical applications and some existing analysis methods for safety analysis of UML statecharts. Section 3 outlines the main concepts of UML Action Semantics. Section 4 and 5 describe our own contribution providing a static safety analysis technique for Action Semantics. Finally, major conclusions are drawn in Section 6.

2 Basic Safety Criteria

A safe system is free from accidents or unacceptable losses. Safety analysis should identify hazards based on the (formal) model of the system. Accidents related to computers are frequently resulted from flaws in the specification (model). In [10] and [9] 47 formal criteria were defined by Nancy G. Leveson that should be satisfied to avoid inconsistent and ambiguous specifications. These criteria cover general aspects of the specification of a control system, including also peculiar ones like environmental capacity and data age.

The most important desired properties of a specification are *completeness* and *consistency*. Completeness with respect to an embedded control system means that a response is specified for every possible input sequence, including also timing variations (early, late, delayed etc. signals). This means that, there are no "forgotten" relations, transitions, or other elements in the specification. Consistency of the specification implies that there are no conflicting requirements and no (unintentional) non-determinism.

In a previous paper, we formalized the safety criteria of Leveson in terms of the UML Statecharts and presented an approach to check them automatically and [14, 13]. In the current paper, our objective is to apply and formally check these criteria on UML Action Semantics specifications.

4 Safety Problems in Action Semantics

In the current section, we evaluate Action Semantics from a safety point of view and identify contradictions with the safety criteria of Leveson.

Traditionally, the main philosophy of the Action Semantics is the avoidance of "over-specification" [11]. According to the UML standard, the software specification is complete if contains all information necessary to solve the task, but not more. For example, if the task is to initialize two variables, the specification must contain the initialization values, but should not specify the sequence of the two memory writes. This allows the compiler program to make optimizations in the final code. For example using a multi-processor environment, the two initializations can be processed in parallel, which is more optimal than the sequential processing.

If the control flow (or sequence) relations are not defined between two actions, they can run concurrently. Although the UML 2.0 [11] extended the methodology with the elements inherited from activity diagrams, which much better visualize the execution process, but the basic rule remains the same: if two actions are ready to run at the same time, they are still allowed to operate concurrently.

However, from a safety point of view, this means that, if the designer accidentally forgets to define a sequence relation between two actions, they may run in parallel resulting in undesired errors that are frequently context sensitive with unpredictable behavior. As an Action Semantics description is usually large and complex, it is very hard to detect a missing sequence relation prior to compilation. In a safety-critical system a specification with a hidden error is unacceptable, so additional formal criteria and analysis is necessitated.

The second basic static safety criterion is *consistency*. Unfortunately, the designer can define invalid sequence rules, and the UML tools (generally) do not check this. The

result is an inconsistent specification, which will probably run to a deadlock situation. To find these situations, automated analysis methods are necessary.

The third elementary safety criterion is the requirement for the lack of non-determinism to avoid ambiguities. In a safe system, the random or undefined behavior should be avoided, since the operation of the software is unpredictable and/or ambiguous. According to the philosophy of UML, if two operations are in conflict, then the result is undefined and the Action Semantics provides no built-in solutions for the problem of the mutual exclusiveness on shared resources such as object attributes or variables.

5 Structural Model Extensions for Safety Analysis

The first problem, the missing explicit declaration of the parallel operation can be solved defining a new inter-action association type. This association shows that the two activities can be executed in parallel as no causal ordering is defined between them. Unlike the fork construct, the "parallel" association does not specify additional sequence information.

The new type of association forces the designer to think about the possible parallelism in the model or specification. Instead of the original rule that says, "all actions without explicit order information can work concurrently", a new one should be used: "all allowed parallel operation must be explicitly marked using the fork and the join construct, or the new "parallel" association". In this case, if there are two independent actions without a control or data flow ordering or parallelism defined between them, it is considered to be a specification incompleteness error.

Static inconsistency problems are caused if the causal and parallel relations introduced between actions are contradictory. The first inconsistency problem can be detected automatically by searching causal cycles in the Action Semantics model.

Concerning the other case of static inconsistency, if there is an explicit concurrency definition but the two actions are causally ordered, it is an inconsistency error. After mapping the parallelism relations, an analysis tool can examine the variables and memory manipulation actions.

6 Static Safety Analysis Methods

As Action Semantics is part of the UML standard, it is defined with the well-known semi-formal techniques using a metamodel, well-formedness rules, and informal textual descriptions of the semantics. In this respect, concrete Action Semantics expressions can straightforwardly be represented as well-formed *instance graphs* of type graphs.

Regarding the Action Semantics model to be analyzed as a graph allows us the use of all the rich methodologies designed for the graph manipulation. For special our specific safety analysis purposes, we exploit graph matching and graph transformation as implemented in the VIATRA system [7,5].

The graph transformation provides a rule-based manipulation of graphs over metamodels defined as type graphs [1, 6]. During the graph transformation process, the pattern defined by the left-hand side (LHS) of the transformation rule is first tried to be matched into the instance graph representing the system model. In this case of success, the image of the LHS graph pattern is replaced by an image corresponding to the pattern defined by the right-hand side of the transformation rule.

Naturally, an entire transformation process requires several rules. In this case, a single transformation step is repeated as long as a rule can be found which is applicable. If no rules are applicable, the transformation process terminates.

To find safety errors, at first we build a transitive closure of “successor” relations between pairs of action nodes. This can be performed by the following graph transformation rule (Figure 1).

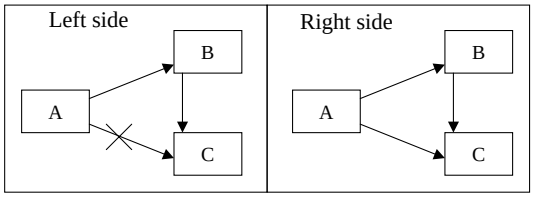

Figure 1: rule to build the transitive closure of the causal dependencies

Applying the rule as long as possible to the input action graph, we obtain an extended graph as the result, where every action node is connected to all other nodes following it in the causal ordering, so it is possible to decide if the two nodes are in a common sequence or not. Two nodes are independent of each other thus they can be operated in parallel, if none of them is the successor of the other according to the action graph (i.e., thus none of them is connected to the other with a “successor” relation).

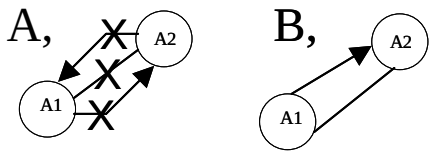

Figure 2: incompleteness (A) and inconsistency (B) checking: graph patterns to match.

To check if all possible concurrent action nodes are connected with the “parallel” association (*completeness analysis*), the analysis tool should be unable to find a counterexample defined by the (negative) pattern that consists of two action nodes that are not connected with neither a “parallel” nor a “successor” edge. Figure 2a graphically depicts the corresponding pattern. If a matching of the pattern is found, then this matching is exactly the place of the incompleteness in the model.

If there are two nodes connected with a “parallel” and a “successor” association at the same time, the model is *ambiguous* or *inconsistent*. This is an error like the cycle in the model. Figure 2b shows the pattern of the error where both types of relations are present between action nodes A1 and A2.

A similar sort of inconsistency is a cycle composed of "successor" edges. To detect sharing violations variable dependencies should be first converted into data flow associations, then into a standard sequence order edge. This technique converts the non-determinism into inconsistency error, which can be found by the checker.

7 Conclusion

In the paper, we dealt with the design of safety critical systems where the behavior of the system is specified by UML 2.0 Action Semantics expressions. The famous safety

criteria of Leveson [10] were first adapted and extended to UML Action Semantics. Then a method using graph transformation was proposed for the automated safety analysis of Action Semantics expressions.

To give a broader environment of the current paper, it is part of the verification and validation framework [7], where UML descriptions are transformed into various mathematical domains (such as Petri nets, SPIN, etc.) to carry out dynamic safety and dependability analyses [4, 3, 2] (which were out of scope for the current paper). For this model transformation purpose, we also used graph transformation as implemented in the VIATRA tool [7, 5]. In this respect, graph transformation provided a general theoretical and implementation framework for various (static and dynamic) sorts of mathematical analysis.

References

1. A. Corradini, U. Montanari, F. Rossi: Graph processes, *Fundamenta Informaticae*, 26(3,4), 1996, 241–266.
2. A. Pataricza: From the general resource model to a general fault modeling paradigm? In: Workshop on Critical Systems Development with UML at UML 2002. Dresden, Germany, 2002.
3. A. Pataricza: Semi-decisions in the validation of dependable systems. In: Suppl. Proc. DSN 2001: The International IEEE Conference on Dependable Systems and Networks. Göteborg, Sweden, 2001, pp. 114–115.
4. D. Latella, I. Majzik, M. Massink: Automatic Verification of a Behavioral Subset of UML Statechart Diagrams Using the SPIN Model-checker. Formal Aspects of Computing, Vol. 11 No. 6 pp 637-664, Springer Verlag, (1999)
5. D. Varró, G. Varró, A. Pataricza: Designing the Automatic Transformation of Visual Languages. Science of Computer Programming, 44(**2**), pp. 205-227, 2002.
6. G. Rozenberg (ed.). Handbook of Graph Grammars and Computing by Graph Transformations: Foundations. World Scientific, 1997.
7. Gy. Csertán, G. Huszerl, I. Majzik, Zs. Pap, A. Pataricza and D. Varró: VIATRA - Visual Automated Transformations for Formal Verification of UML Models. Accepted to the Int. Conference on Automated Software Engineering (ASE 2002), Edinburgh, Scotland (2002)
8. M. P. E. Heimdahl and N. G. Leveson: Completeness and Consistency Checking of Software Requirements. IEEE Trans, on Software Engineering, Vol. 22. No. 6 (1996)
9. N. G. Leveson, J. D. Reese and M. Heimdahl: SpecTRM: A CAD System for Digital Automation. Digital Avionics System Conference, Seattle (1998)
10. N. G. Leveson: Safeware, System Safety and Computers. Addison-Wesley Publishing, USA, 1995 ISBN 0-201-11972-2
11. Object Management Group: Unified Modeling Language Specification v 2.0. (2003).
12. Workshop on Critical Systems Development with UML. October, 2003, San Francisco, USA.
13. Zs. Pap, I. Majzik, A. Pataricza: Checking General Safety Criteria on UML Statecharts. In U. Voges (editor): Computer Safety, Reliability and Security (Proc. SAFECOMP-2001, Budapest, Hungary), pp 46-55, LNCS-2187, Springer Verlag (2001)
14. Zs. Pap: Methods of checking and using safety criteria. 2003. Budapest. Polytechica Periodica, ISSN 0324-6000. 2003/9