VSE - Verification Support Environment

Andreas Nonnengart and **Georg Rock** and **Werner Stephan**
German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
{nonnenga,rock,stephan}@dfki.de

Abstract. The Verification Support Environment (VSE) is a CASE-tool that supports the user in the *formal* development of software. It has a rather broad range of applicability as was already shown in several industrial applications, be it in the safety or in the IT-security domain. We claim that VSE is equally applicable in the domain of plan verification.

1 Introduction

The Verification Support Environment (VSE) supports large parts of the software-engineering process in a systematic way and is general enough to cover several aspects of formal modelling in a common setting. It consists of a basic system for editing and type checking specifications and implementations, a facility to display the development structure, a theorem prover that releases the proof obligations arising from development steps (e. g. refinements), a central database to store all aspects of the development including proofs, and an automatic management of dependencies between development steps. VSE was developed in two phases for the German Information Security Agency (GISA) by consortia from industry and academia. VSE-II (Hutter *et al.* 1999; Rock *et al.* 1999) the follow-up system of VSE-I (Hutter *et al.* 1996) was developed under the lead management of the German Research Centre for Artificial Intelligence (DFKI).

2 Formalisms in VSE

Formal development in VSE is based on two formalisms: *abstract data types* are used to specify data structures and functional computations while a version of *temporal logic* (based on Leslie Lamport's Temporal Logic of Actions (Lamport 2003)) is used to specify the dynamic behaviour of systems with a persistent state. Although there is a fully developed methodology in its own right for abstract data types typically data types are used to provide values for state dependent (flexible) variables in state based systems. Functional computations are then used to model single (uninterruptable) steps of state based systems.

3 Plan Verification

In general, to solve a planning problem means to find a sequence of actions taken from a fixed given set of possible actions - each transforming states (situations) into new states (situations) - such that applying this sequence of actions to a given initial state results in a desired final state.

Slightly more formally this means that a planning problem essentially consists of a pair of states - the initial and the final state (σ_0, σ_n) - together with a set of possible actions $\mathcal{A}$ with $A_i \colon \Sigma \mapsto \Sigma$ for each $A_i \in \mathcal{A}$. The ultimate goal is to find a sequence of actions $\langle A_1, \ldots, A_n \rangle$ that transforms the initial state to the final state, i.e. $A_i(\sigma_i) = \sigma_{i+1}$ for all $0 \leq i < n$.

Plan verification on the other hand addresses the problem of verifying that a given sequence of actions together with a given pair of (initial, final) states really transforms the initial state to the final state. Thus plan verification is not at all concerned with the problem of finding such a sequence; its aim is to prove that this very sequence - regardless of how it was determined - solves the problem.

Obviously, a formal plan generation approach that guarantees the plan correctness would not need such a subsequent plan verification. To verify the correctness of a plan thus comes into play whenever the plan generation did not yet guarantee the plan correctness, be it because the plan happened to be somewhat handcrafted or because an automated plan generation failed for some reason and therefore some gaps remained open that had to be filled by suitable heuristics or human action.

VSE is a tool that supports plan verification, and that manifold. On the one hand VSE's temporal logic (of actions) fits nicely to state transformation systems. On the other hand its internal treatment of abstract data types allows for even highly sophisticated state descriptions and state transformation descriptions (e.g. freely generated and thus inductively defined data structures). But even (dense) real-time considerations can easily be incorporated into VSE's model of actions and state transformations, and that although dense real-time is not directly incorporated into the general idea of TLA (Nonnengart *et al.* 2001).

References

Dieter Hutter, Bruno Langenstein, Claus Sengler, Jörg H. Siekmann, Werner Stephan, and Andreas Wolpers. Deduction in the Verification Support Environment (VSE). In Marie-Claude Gaudel and James Woodcock, editors, *Proceedings Formal Methods Europe 1996: Industrial Benefits and Advances in Formal Methods*. SPRINGER, 1996.

Dieter Hutter, Heiko Mantel, Georg Rock, Werner Stephan, Andreas Wolpers, Michael Balser, Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel. VSE: Controlling the complexity in formal software developments. In D. Hutter, W. Stephan, P. Traverso, and M. Ullmann, editors, *Proceedings Current Trends in Applied Formal Methods, FM-Trends 98*, Boppard, Germany, 1999. Springer-Verlag, LNCS 1641.

Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2003.

A. Nonnengart, G. Rock, and W. Stephan. Expressing Realtime Properties in VSE-II. In *ESA Workshop on On-Board Autonomy*, volume WPP-191, pages 447–454, October 2001.

Georg Rock, Werner Stephan, and Andreas Wolpers. Modular Reasoning about Structured TLA Specifications. In R. Berghammer and Y. Lakhnech, editors, *Tool Support for System Specification, Development and Verification*, Advances in Computing Science, pages 217–229. Springer, WienNewYork, 1999.