The Verification Support Environment VSE

Baur, T. Plasa, P. Kejwal, R. Drexler, W. Stephan, Wolfgang Reif, A. Wolpers, D. Hutter, C. Sengler, E. Canver

Angaben zur Veröffentlichung / Publication details:

Baur, T. Plasa, P. Kejwal, R. Drexler, W. Stephan, Wolfgang Reif, A. Wolpers, D. Hutter, C. Sengler, and E. Canver. 1992. "The Verification Support Environment VSE." IFAC Proceedings Volumes 25 (30): 69–74. https://doi.org/10.1016/s1474-6670(17)49409-2.

THE VERIFICATION SUPPORT ENVIRONMENT VSE

Dr. Baur and T. Plasa, GPP mbH, 8024 Oberhaching, Germany
P. Kejwal, Dornier GmbH, 7990 Friedrichshafen 1, Germany
R. Drexler, W. Stephan, W. Reif, A. Wolpers, University of Karlsruhe, Germany
D. Hutter and C. Sengler, University of Saarbrücken, Germany
E. Canver, University of Ulm, Germany

Abstract The VSE (Verification Support Environment) is a first approach to integrate the methods of formal specification and verification into an existing and industrially approved CASE tool. Safety/security-critical software systems (or parts of) can be developed within the VSE under the guidance of a formal design method, based on the principles of modularization and refinement. VSE support covers the whole development process, starting with fairly abstract *security models* down to the concrete system implementation (which is formally correct w. r. t. its specification, due to the VSE development method). For that purpose, the VSE system integrates an appropriate formal specification language (VSE-SL) and universitary resp. industrially approved components for specification and verification into a single, ready to use software development environment.
The VSE project is a national promoted project, initiated and sponsored by the BSI (Bundesamt für Sicherheit in der Informationstechnik).

Keywords computer applications; formal verification; modeling; reliability; safety; software engineering; specification languages

INTRODUCTION

Nowadays, industry, commerce and public administration are increasingly dependent on the smooth and correct function and continuous availability of technical and information systems based on software components.

This rises the need for adequate methods and tools for the development and evaluation of these systems in order to guarantee the compliance with the safety and security requirements appropriate to the respective risk in different operational environments.

The Verification Support Environment VSE is an attempted answer to this problem. It is an unique approach, to combine the benefits of a CASE environment with the methods and tools for formal specification and verification.

GOALS OF THE VSE

The VSE will be the prototype of an industrially usable software system, consisting of ready to use specification and verification tools, embedded in a common development environment. Its suitability will be examined in realistic and close to practice case studies.

For the realization of the Verifiction Support Environment, the following objectives are pursued:

- Provision of a specification component to identifiy user requirements and categorize them according their criticality
- Support for the definition and analysis of safety / security models
- Different problem oriented specification concepts
- Concepts for dividing a problem into abstraction levels
- Verification in parallel to the development, controlled by a verification management system
- Identification of the links between requirements, safety/security model and program specification
- Automated generation of source code out of the program specification

WHY USING FORMAL METHODS ?

Software Engineering nowadays is conceptually described by a wide range of development methods and supported by a variety of software-tools (CASE tools). Most of the existing CASE tools use pure informal or semi-formal methods in the development process. Their basic concept is to avoid committing errors by a constructive approach (analysis and design methods, tool support, automatic documentation, code generators, configuration management etc.).

The second concept is to detect and remove all those errors, which could not be avoided by performing tests

on the analysis, design and program descriptions.

On one hand it is commonly agreed that this approach will improve the quality of a system to be developed. On the other hand this is only a tendency that cannot be formally measured. Although there are several concepts for a systematic and complete testing, there is no way to assert the overall correctness of software by pure testing alone. All this cannot be the sole basis for the certification of a system.

The solution to this problem is the use of formal specification and formal verification. Formal methods use mathematical descriptions (stated in a formal specification language) to establish models both of the security and safety requirements of the software system. Furthermore, they use an appropriate logic calculus to prove the correctness of the mathematical models as well as the compliance between formal models of the system and the concrete system itself. The advantage of this method is -among others-, that the assertions obtained from formal verification are valid in a mathematical sense.

Since concrete software systems usually are of great complexity, formal system descriptions must be broken down into smaller pieces by modularisation and refinement principles, just as in conventional software engineering.

As a disadvantage of the pure formal methodology is considered the loss of a more "practical" view of things that designers usually get from an informal oriented system description. So the optimal development strategy should use a combination of both formal and informal methods. This integration principle is one of the main features underlying the VSE-philosophy.

FORMAL DEVELOPMENT METHODS AND TOOLS IN THE VSE

The VSE offers an integrated software engineering environment with informal and formal methods. This means, that it delivers both the conceptual frame and the appropriate tools for formal specification and verification in every phase of the life cycle of a software system.

The methods and tools are as follows:

Requirements Analysis

The initial system requirements are described at first informally in an industrial approved way by means of a requirements specification tool. Requirements can then be categorized according to their safety/security relevance and stated in terms of the VSE-specific formal specification language VSE-SL (see below). This defines the *security model* of the system.

Formalization of System Specification

System components that are not safety or security related are developed in a conventional way by means of a design specification language, which supports a wide range of development methods.

For the security relevant system components, the VSE offers a *formal* development method. Formal development involves the design of a *structured* formal specification, the refinement of the individual components of this specification, and proofs of the correctness of these refinements. Structured specifications add horizontal structure to the system design, while the refinement steps add vertical structure.

The use of structured specifications breaks a system down into several pieces that can be developed independently. Furthermore, the method supports design by stepwise refinement: intermediate layers of specifications may be introduced to bridge the gap between the rather abstract specification of a system (e.g. a booking system) and the structures found in the implementation environment (like arrays or database entries) in manageable steps.

The highest level of the vertical structure is given by the security model of the system, the lowest level by the target language (for which Ada has been chosen within the VSE). Together, these principles enforce and encourage a highly modular system design. Apart from its well known advantages, this is a "must" for making verification possible at all.

Since flexibility is especially important when using formal methods, the VSE is not restricted to a single method of modeling the world: systems may be described as abstract data types (see Reif, 1991; Wirsing, 1990) as state based systems (see Rushby, 1991), or as a suitable combination of both. The integration of further techniques is possible.

Formal Verification

While formal specification of a system is usually considered as an advantage in itself, it does not ensure a correct *implementation* of the system. The correspondence between a specification and its implementation can be shown by a formal verification.

While verification of the entire program against the specification of the entire system is prohibitive (due to the complexity of the objects involved), a verification of the individual vertical refinements with respect to their import (the specifications they are based upon) and their export (the specification which they implement) *is* possible.

On the implementation side, the vertical structure of a system's specification is reflected in the programs which link two levels of abstraction. These programs are considered as "abstract programs" in the sense that they do not operate on data structures like arrays, but rather on structures described in intermediate specifications (like "booking systems", "employee lists" etc.).

In the VSE system development method, the correctness of each refinement (consisting of import specifications, a collection of programs and an export specifica-

tion) is formally verified within a logical calculus. The proof obligations are created automatically, the proofs are done either automatically or interactively by the VSE prover components. Since the verification is as modular as the system design, the verification of a refinement step is independent of the implementation of its import specification, and any two refinement steps can be verified independently. As a result, development and verification can proceed in parallel. Overall, the entire process is controlled by a verification management system.

Implementation

The implementation of the final system is done by automatic generation of the source code. The source description for the generated code is built from the *abstract programs* that realize the links between the formal abstraction levels. Due to the VSE development and verification methods, the resulting code is formally correct w. r. t. the system specification.

VSE SYSTEM STRUCTURE

The VSE System consists of several levels or components, which run under a unique graphical user interface, as indicated in Fig. 1.

The VSE components are made up by three existing software tools, which will be integrated into a unique system. Due to the integration, each tool will be

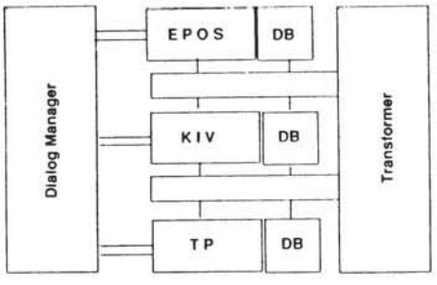

Fig. 1: VSE Sytem Structure

adapted and enhanced to establish the overall VSE-functionality. The basis for formal specification is given by the VSE-SL, a VSE specific formal specification language.

In the following sections, the main features of the different components will be described.

EPOS

The CASE-Tool EPOS is the VSE platform for specification, development management and integrated data management in the VSE System. Among others, it is used for informal and formal requirements analysis, formal specification and abstract programming. It also contains an integrated verification management system, which keeps track of the *verification status* of the development items.

EPOS (Engineering and Project Management Oriented Specification System) is a software tool system designed for computer support of development and project management activities (see Lempp, 1986). Its aim is to release experts, engaged in software/hardware projects, from tedious routine work, enabling them to spend more time on development activities.

Some of the various objectives of the EPOS System are:

- Homogeneous computer support in all phases of a project
- Integrated computer support for development, project management and product management
- Support for various development methods
- Automatic, up to date generation of documentation, according to documentation standards
- Automatic code generation and code feedback (tracing)

The EPOS system is built up by the following items:

- Specification languages for requirement engineering, system specification and project management
- The EPOS project database for storage of all data described by the specification languages
- Several software tool systems for the evaluation of the EPOS project database (error analysis, design method support, documentation system, code generation system, management support)

Development of the EPOS System including the installation, service and maintenance, is carried out by GPP (Gesellschaft für Prozessrechnerprogrammierung mbH), Oberhaching near Munich. Today, EPOS is used world-wide in thousands of installations. It has been used by industry, governmental agencies, research institutes and universities since 1980.

For the integration into the VSE-System, EPOS is - among others- enhanced by a "formal component" to enable formal specification in the VSE specification language (VSE-SL).

Embedded Formal Specification Language VSE-SL

All formal specifications are stated in terms of a general purpose specification language, named VSE-SL. VSE-SL is a strongly typed language based on first-order predicate logic, enriched by features for modeling state-dependent behaviour. Requirements of safety-critical systems may be expressed in an appropriate manner.

Specifications in VSE are organized around units; a unit may contain a first-order theory, an abstract data type or an abstract state machine. Units are the basic blocks in VSE-SL, from which large specifications are built. They are the means of modularizing a system by

the principles of horizontal structuring, vertical refinement, and generic features.

A unit may be parameterized by types, functions and predicates, and constraints may be stated on the parameters; this generates proof obligations for each instantiation of the generic specification unit, but on the other side, enhances the reusability of specifications.

The basis for VSE-SL is first-order typed predicate logic; the primitives of the logic – sorts, functions, predicates, variables, and first order connectives – are explicitly provided in the language. However, because of several extensions, VSE-SL is rather a specification language than just merely a logic. One extension has a pure syntactic character and mainly deals with names, their visibility, and features for overloading operators. Other extensions implicitly introduce axioms into the specification, like pre-defined data types, and again others affect the class of models described by the specification; for example constructors describe only generated models.

VSE-SL provides features for modeling state based systems; these are the primitives *state object* and *operation*. State objects correspond to variables in a procedural programming language, e.g. Ada. Operations cause a state change and thus may affect the values of state objects; they correspond to procedures in conventional programming languages.

A major aspect of VSE-SL is its support for structures, with respect to both, horizontal decomposition and vertical hierarchy. The specification of a system may be structured by composing specification units at the same level of abstraction (horizontal structure). The vertical hierarchy is expressed by the relation between successive levels of abstraction and refinements. A refinement step employs a mapping, which links successive abstraction levels, and an implementation stated in terms of an abstract programming language, which in turn uses specifications written in VSE-SL.

This methodology favors the top-down development of a system, resulting in a hierarchy of abstraction levels, reaching from the requirement specification to the specification of the executable code and primitive data types.

KIV

As mentioned above, the VSE supports the modeling of systems in a variety of ways (abstract data types, state based systems etc.). This raises the need for a matching set of tools to support the corresponding correctness proofs. Within the VSE, the KARLSRUHE INTERACTIVE VERIFIER (KIV) serves the needs of proving correctness of programs wrt. their specification.

Architecture of the KIV system. The KIV system realizes a *shell* for implementing different verification methods on a common logical basis (see Heisel, 1990). This design combines the flexibility required for the various specification methods, a mechanism to ensure the correctness of the implemented verification methods, and the option for future enhancements.

To achieve these goals, the KIV system follows the paradigm of *tactical theorem proving* (see Gordon, 1989), which combines a powerful logical basis with a programming language designed for manipulating proofs. This allows to build a layered system which can be adapted to several needs.

On the lowest level, the built-in logic can be tailored for a specific verification method. This also involves a "reduction" of the tailored version to the built-in logic, and to be successful this task requires expertise with the built-in logic. On the next level, proof tactics are formed from the extended logic. This requires some experience with logic as well. These two lower levels will not be directly accessible within VSE.

The third level combines tactics and heuristics to form proof strategies, and adds a user interface. The user interface hides most technical details from the end user of the VSE system, and consequently using proof strategies requires only marginal knowledge of the particular built-in logic. However, it *does* require the ability to perform some formal reasoning, but the use of the system in courses at the university has shown that computer science students master the proof strategies fairly quick.

Integration into the VSE. Following this way, most of the well known verification methods have been realized within the KIV system. Also, a number of program synthesis methods, where programs are developed starting from the specification, have been integrated. The important feature for the VSE is that the KIV system supports the modularized specification of systems described earlier. There is a special proof strategy for modular verification which performs the proofs necessary to show the correctness of a refinement step.

For the VSE, the management of proofs within the KIV system will be integrated into the VSE's verification management. This includes the possibility of multi-user operation, keeping track of changes to programs (after all, since mistakes are only human, the first attempt to verify a refinement step will usually fail), and tools to analyse, store and document proofs.

Enhancements. Proof strategies are usually *interactive*, in the sense that the "key steps" in the proof are initiated by the user, and the system performs the routine proof steps. Key steps are, for example, the formulation of loop invariants or induction hypotheses, or the selection of the next reduction step out of a choice offered by the system. The actual degree of automation that can be reached currently lies between 80 and 95%. A degree of 95% may sound impressive, but it should be noted that a correctness proof for a refinement step may easily require several ten thousand proof steps.

There are several ways in which this degree of automation can be increased. One of these is to use a better theorem prover to determine properties of the import

specification of a module. The integration of the theorem prover described below is expected to result in further improvements. Another major source of improvement will be the experience gained with the VSE case studies, which will result in enhancements of the existing proof strategies within the KIV system.

While formal verification may be expensive, the remarkable results we have already achieved show that formal verification is possible with today's (hard- and software) technology.

The Theorem Prover

Proving the correctness of programs w. r. t. their specification within KIV results in a bundle of (first order) predicate logical formulas which have to be deduced from the specification. The more these formulas can be proven automatically the more the degree of automation in the KIV system increases. Hence within VSE, this task is handled by a specialized automated theorem prover.

The heart of this theorem prover is the INKA-system (Biundo, 1986) which is an inductive theorem prover based on resolution and paramodulation. Resolution is the best known calculus to handle full first-order predicate logic automatically. The calculus is guided by elaborated strategies based (like the KIV system) on the paradigm of tactical theorem proving.

The theorem proving component has to be capable of several techniques:

Mathematical induction is required for reasoning about objects containing repetition, e.g. lists of numbers. Induction proofs are characterised by the application of so called *induction rules*, of which the simplest and best known example is an induction rule on natural numbers: To prove a formula

$$\forall x : \text{nat}\ P(x)$$

we have to prove both formulas:

$$P(0) \text{ and } \forall x : \text{nat}\ P(x) \rightarrow P(x+1)$$

Within VSE similar and other rules are available for every kind of recursively defined data structure, e.g. list of numbers, integers, trees, etc. In fact, applying induction rules creates infinitely many branching points in search space. Hence, the theorem prover has to be guided by sophisticated heuristics selecting an appropriate induction rule to get rid of the combinatorical explosion. There are other heuristics which contain the knowledge how to prove typically base cases or induction steps (Hutter (1989)).

Within VSE some data structures - integer, sets or arrays - and functions - $+, *, !!$, etc. - operating on these data structures are predefined. This opens up the ability to incorporate highly specialized deduction tools and strategies for dealing with these data structures and operations. A typical example is a built-in semi decision procedure for Presburger Arithmetic (handling integers with $+, -$).

THE VSE PROJECT

It is the objective of the VSE-project to create an integrated software development tool (VSE-tool) to generate security-relevant program systems, which can be proved to be correct.

The VSE is in development by a consortium of industrial companies and institutes (Dornier, GPP, University of Karlsruhe, University of Saarbrücken and University of Ulm) under contract of the BSI (Bundesamt für Sicherheit in der Informationstechnik).

The project organization is shown by Fig. 2.

The individual components (KIV, theorem prover, formal specification language) will be integrated into the EPOS software development environment.

Industrial relevance and usefulness of the VSE-tool will be proved in the course of the project by means of comprehensive case studies based on industrial projects.

These case studies shall cover all aspects of the definition of the VSE-tool requirements, mainly a wide spectrum of the IT security criteria. Additionally, the case studies shall reveal strong and weak points of the VSE-tool and of all components. The features to be determined are, for example, the degree of automation, termination, run time behavior and efficiency. Above all, the case studies shall show, to which extent the user is supported by the VSE-tool.

VSE PROJECT ORGANISATION

- **Commissioner:** Bundesamt fuer Sicherheit in der Informationstechnik (BSI)

Projectpartners	**(Location**	**Responsibility)**
- Fa. Dornier	Friedrichshafen	project management and case studies
- Fa. GPP	Muenchen	integration of components into EPOS
- University	Karlsruhe	interactive verification system (KIV)
- University	Ulm	specification language
- University	Saarbruecken	theorem provers

- **Duration:** 3 1/2 years (February 91 - July 94)
- **Hardware-Platform:** SUN SPARCstation - UNIX

Fig. 2: Project Organisation

For the selection from a great number of industrial case studies which have already been conducted or are currently in preperation, mainly the following criteria were taken as a basis:

- Suitability, extent, and complexity of the case studies
- Relation to reality
- Security requirements
- Assessment of current methods and components
- Development activities required

REFERENCES

Biundo, S., B. Hummel, D. Hutter, C. Walther (1986). The Karlsruhe Induction Theorem Proving System. *Proceedings 8th Conference on Automated Deduction.* Springer LNCS, Vol 230.

Heisel, Reif, Stephan (1990). Tactical Theorem Proving in Program Verification. *Proceedings 10th International Conference in Artificial Intelligence.* Springer LNCS, Vol 449.

Hutter (1989). Complete Induction. In K.H. Bläsius and H.-J. Bürckert (Ed.), *Deduction Systems in Artifical Intelligence*, Ellis Horwood. 1989 Chap 7.

Gordon, Milner, Wadsworth (1979). Edinburgh LCF. *Lecture Notes in Computer Science, 78.* Springer Verlag, Berlin.

Lempp, P. (1986). Development and Project Management Support with the Software Engineering Environment EPOS. In I. Sommerville (Ed.), *Intl. Conf. Software Eng. Environments.* Lancaster, Great Britain.

Reif (1991). *Correctness of Specifications and Generic Modules.* PhD thesis, Universität Karlsruhe. In Deutsch.

Rushby, von Henke, Owre (1991). *An Introduction to Formal Specification and Verification Using EHDM.* SRI International, Menlo Park, California.

Wirsing (1990). Algebraic Specification. In J. van Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, Vol. B, Formal Models and Semantics. Elsevier Science Publishers B.V., pp 675–788.