# High-Precision Retrieval for High-Quality Software

B. Fischer, M. Kievernagel and W. Struckmann

Abteilung für Softwaretechnologie
TU Braunschweig
Gaußstraße 17
D-38092 Braunschweig, Germany
Tel. +49 531 3917579
Fax +49 531 3918111
{fisch,mkiever,struck}@ips.cs.tu-bs.de

**Abstract.** In this paper we present VCR, a high-precision retrieval tool which helps programmers to locate software components which exactly match their needs. It employs implicit VDM specifications as search keys and is thus especially well-suited for a formal software development process based on VDM's specify-refine-implement paradigm. VCR is designed as a filter-inspector-chain which is user-configurable through a graphical user interface. The filters realize deduction-based search methods such as signature matchers, model checkers, and theorem provers. The inspectors maintain links between subsequent filters and thus allow examination and reuse of intermediate retrieval results. First experiments demonstrate the feasibility of our approach.

## 1. Introduction

Formal methods are often believed to be the "one true way" towards the construction of high-quality software. Their application, however, is surrounded by some myths (Hall, 1990; Bowen and Hinchey, 1995) and industrial usage of formal methods does not live up the initial expectations (Bowen and Stavridou, 1993; Craigen et al., 1993; Weber-Wulff, 1993; Hussmann, 1995). Hence, high-quality software remains an exception.

This unpleasant situation may be overcome or moderated if approved (i. e. specified or even verified) software components can be reused. But then adapting the retrieved components to the exact needs of the user remains an error-prone

step and may harm the reliability of the resulting system. Consequently, the reuse approach to high-quality software requires high-precision retrieval methods which only find "plug-compatible" components.

As this approach relies on large component libraries tool support is inevitable. Unfortunately, most classical retrieval methods (Prieto-Diaz, 1987; Maarek et al., 1991) are based on an informal and external keyword classification scheme and are thus not applicable.

In this paper we will present the deduction-based retrieval tool VCR (VDM-based software Component Retrieval). It has been developed in the NORA-HAMMR-project, which in turn is part of the inference-based software development environment NORA (Snelting et al., 1994; Krone and Snelting, 1994). VCR is a high-precision retrieval tool which exploits exact semantic information intrinsic to the components. From a user's point of view it offers three main features:

- a graphical interface,
- a configurable filter-inspector-chain,
- and incremental strengthening of queries.

The graphical interface shields the user completely against the applied deduction methods. No special knowledge but only experience in the applied implementation language and VDM is required to use VCR. The filter-inspector-chain architecture guarantees that results of acceptable precision are available sufficiently fast and thus avoids the bottleneck of insufficient deduction power. Hence, VCR overcomes the two main reasons for the failure of earlier approaches to deduction-based retrieval (Rollins and Wing, 1991; Manhart and Meggendorfer, 1991).

## 2. Deduction-Based Component Retrieval

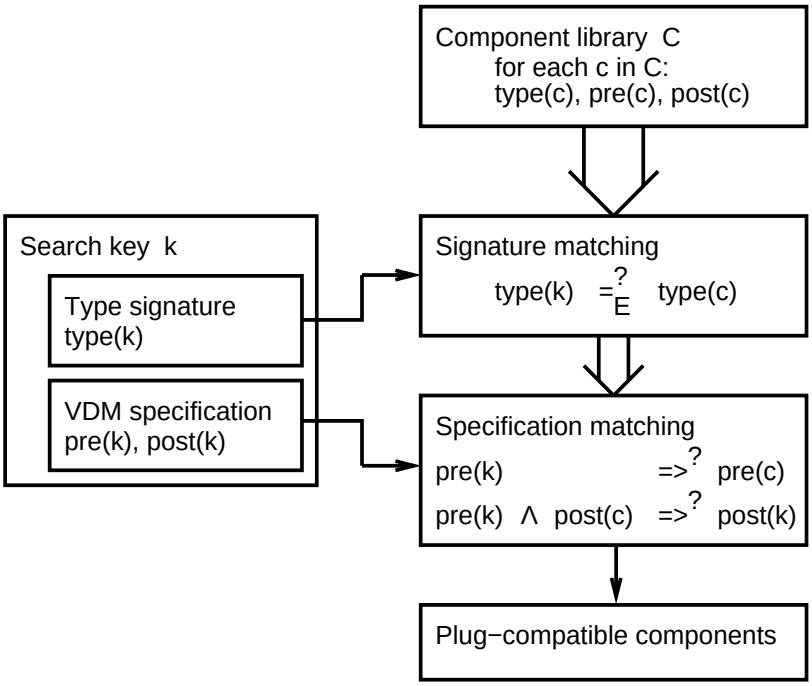
The basic idea of deduction-based software component retrieval is simple:

- a component is characterized by its signature which can be computed using type inference techniques (Damas and Milner, 1982) and its specification which must be given in the form of pre- and postconditions,
- search keys also consist of a $(sig, pre, post)$-triple,
- a component matches if it has a provable "compatible" signature and specification.

A realization of this idea requires the selection of a specification language and precise definitions of compatibility.

In VCR (cf. figure 1) we decided to use a library of Modula-2 procedures (Lins, 1989) which have been specified as VDM operations—this avoids cryptic theorem prover formalisms.

**Figure 1** - Retrieval concept



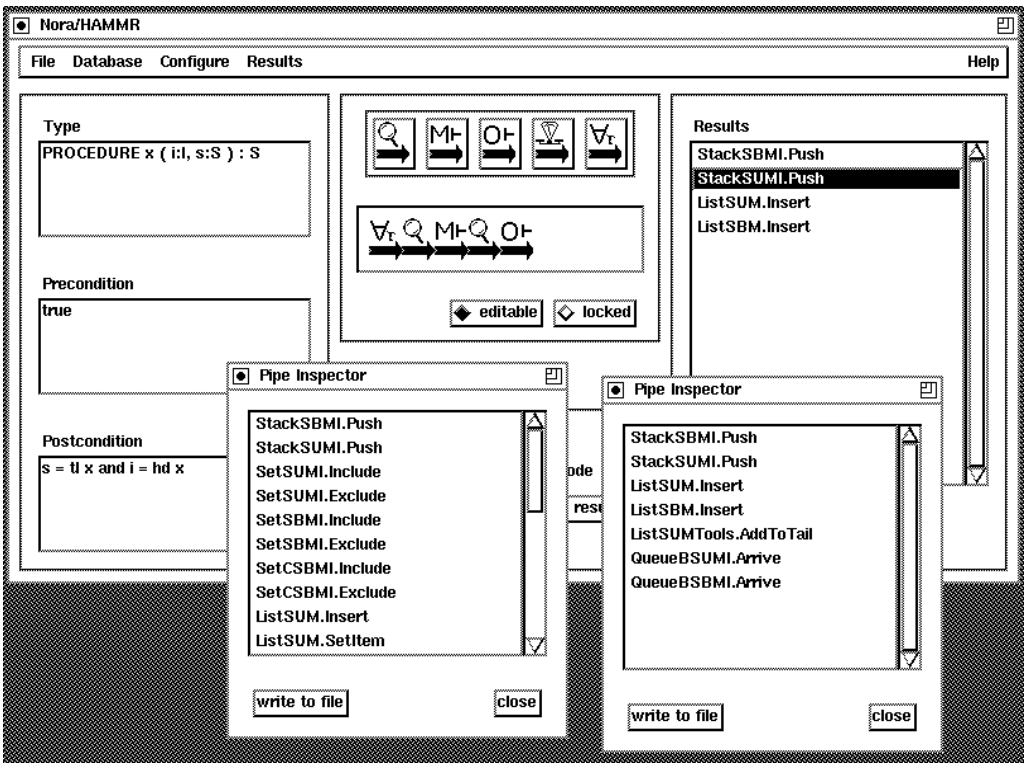

Compatibility of signatures is defined via an equivalence $E$ and checked by $E$-unification (cf. section 3.) Signature terms consist essentially of Modula-2 type definitions extended by type variables to abstract the concrete naming of types.

A component's specification is compatible with that of a search key if it simultaneously contains a weaker precondition and a stronger postcondition. Thus, the proof obligations $\mathsf{pre(k)} \Rightarrow \mathsf{pre(c)}$ and $\mathsf{pre(k)} \wedge \mathsf{post(c)} \Rightarrow \mathsf{post(k)}$ must be valid. The special form of the second condition additionally allows to match components realizing a less partial function than the key.

An implementation of specification matching obviously requires some kind of automatic proof procedure. The undecidability of the proof problem and the complexity and relative weakness of existing theorem provers make it impossible to take a brute-force approach.

Our approach is to split specification matching into simpler steps, each one realized as an independent filter. Hence, VCR is designed as a chain of component filters. This also allows free combination of different retrieval methods—including text-based methods. Moreover, since intermediate results can be inspected at every stage, the overall running time is not critical to the performance of the tool.

**Figure 2** - Graphical user interface




The idea of successive filtering is reflected in the filter chain located in the center of our graphical user interface (see figure 2). Additionally, inspectors grant easy access to intermediate results. This filter-inspector-chain may easily be customized by the user through an icon pad. The configuration displayed contains the filters described in this paper and has also been used in the experiments.

The inspectors maintain their own windows to display the components retrieved by the filters they are attached to. They allow to access the VDM definition and the Modula-2 code of the components. All intermediate sets of components or modules can be selected and used in subsequent retrieval runs with different keys. So a VCR user can start out with stating only weak conditions in the key, which can then be strengthened incrementally as long as the results remain too unprecise. Additionally, selecting the module set of an intermediate result allows to search for typical combinations of procedures (e. g. `push` and `pop`.)

## 3. Signature Matching and Component Transformation

The goal of signature matching is to find components with appropriate calling conventions whatever choices in their implementation have been made. Thus the type equivalence $E$ should abstract the choices a language offers. The

equivalence used in VCR is based on Rittri's work in the context of functional languages (Rittri, 1990) and handles the following features of Modula-2:

- Naming of types.
- Ordering of lists (e.g. parameters and record fields.)
- Return result as VAR-parameter or as function result.

The implementation is based on order-sorted AC1-unification. The resulting substitution is necessary to identify corresponding names in the specifications and consequently to build the proof obligation. It also describes a type transformation from a matched component to the key, which can be used to transform the component automatically into a form exactly fitting the environment of the key.

Let us consider for example the search key:

```
PROCEDURE x( i:I, s:S ) : S
pre   true
post  s = tl x and i = hd x
```

VCR matches the procedure `Push` using the result currying axiom

```
PROCEDURE p( x, VAR y:Y ) = PROCEDURE p( x, y:Y ) : Y
```

and commutativity of parameters. The result of this transformation is shown in figure 3:

**Figure 3** - Component transformation

```
PROCEDURE Push                      PROCEDURE Push'
  ( VAR st:Stack; it:Item );          ( it:I; st:S ) : S;
                                    VAR t :  S;
BEGIN                               BEGIN
⋮                                   ⋮
     WITH st^ DO                         WITH t^ DO
     ⋮                                   ⋮
     END                                 END
⋮                                   ⋮
                                    RETURN t;
END Push;                           END Push';
```

Precautions have to be taken to handle every possible exit from a procedure when a VAR-parameter is transformed into a function result.

## 4. Specification Matching

In this section we will explain how VCR checks a specification against the filtered library components. We decided not to hard-wire a special proof procedure for VDM but to integrate the general purpose theorem prover OTTER (McCune, 1994b) and the associated model finder `anldp` (McCune, 1994a). OTTER is based on the resolution principle and can handle formulas of the non-sorted first order fragment of predicate calculus with equality.

As already mentioned specification matching is currently implemented by two filters, a simple model checker and the actual application of the theorem prover.

The model checker purges obligations which can easily be refuted by testing their validity in a small fragment of the VDM-axiomatization. Obviously, this is a prerequisite for the obligations to be provable in the full theory. The basic idea is to check whether all variable assignments ranging over a finite domain evaluate the obligations to `true`. We have experimented with different models and have obtained good results using a finite model only containing the objects `nul` and `one = suc(nul)` as integers, `nil` and `lnul = cons(nul,nil)` as lists and `inc` denoting illegal terms.

The filter thus extends each obligation by an axiom set for integers and lists which has exactly this finite model. `anldp` then tries to find the designated model using a modified first-order Davis-Putnam procedure (i. e. enumeration of all finite models.) If this fails the component is rejected. Otherwise, the validity in the full theory still has to be tested.

The last filter creates a problem description for OTTER and the full theory. First the problems are split into independent subproblems in order to reduce the search space. A further reduction of the search space is achieved by dynamically linking each problem with an appropriate set of axioms. The complete axiomatization consists of about 120 axioms and lemmata and is based on (Bicarregui et al., 1993) for sequences and on our earlier work in automatic program verification (Kievernagel, 1990; Hohlfeld and Struckmann, 1992) for arithmetic. The problems presented in this paper needed up to 25 axioms from this set.

Finally, OTTER is run on the generated problems and its output is analyzed. The matched components—final or intermediate results—are represented by their location (module) and their name.

## 5. Results and Conclusions

The experiments reported here are based on about half of Lins' library. We have specified all list-like structures from singly-linked lists to priority queues

and deques and the basic set-like data types set and bag. All specifications are based on the structure of the Modula-2 implementation. We represent singly-linked structures and arrays as VDM-sequences. The elementary types and the other structured types are represented in VDM by their nearest counterpart.

The following table displays the filtering effect of the three phases of VCR. The left column gives a short description of the search key. The columns for the type matcher and the model checker give counts for the successfully matched procedures and the modules in which these are contained. The last column gives the results of the respective OTTER runs which are either a successful proof (runtime in seconds[1]) or there was no proof within a short time limit for a valid proof obligation (`np+`) resp. an invalid proof obligation (`np-`).

**Table 1** - Retrieval experiments

| # | description | sig. match | model check | OTTER |
|---|---|---|---|---|
| 1 | Insert at head of seq. | 25/14 | 7/4 | $4 \times 2s / 3 \times$ `np-` |
| 2 | Seq. split at element | 1/1 | — | `np+` $(48s)$ |
| 3 | Seq. split at position | 1/1 | 1/1 | $1s$ |
| 4 | Member?-predicate | 3/3 | 3/3 | $3 \times$ `np+` |
| 5 | Position of element in seq. | 9/9 | 9/9 | $9 \times 1s$ |
| 6 | Remove from front of seq. | 51/20 | 6/3 | $6 \times 2s$ |
| 7 | Remove from back of seq. | 51/20 | 6/3 | $6 \times$ `np-` |

Except for OTTER, there are no runtimes included in this table because some syntax transformations and parts of the process control are not yet fully automated. However, the signature matching filter takes about 2–3 seconds for the whole library depending on the generality of the type key and the number of matching components. The model checker takes about 2–4 seconds per proof obligation varying with their complexity. This time could be drastically reduced by a specialized program because `anldp` needs 1.75 seconds to find the intended model from the axioms of the small theory alone.

Finally, we want to comment on the adequacy of the found procedures. The following table gives the classical retrieval measures of recall ($\mathcal{R}$ = retrieved relevant components / all relevant components) and precision ($\mathcal{P}$ = retrieved relevant components / all retrieved components) for the experiments at each filter (average-2 is computed only from non-empty results.)

[1] All times were measured on a SPARC ELC-10.

**Table 2** - Recall and precision

| problem | sig. match | | model check | | OTTER | | best filter | |
|---|---|---|---|---|---|---|---|---|
| | $\mathcal{R}$ | $\mathcal{P}$ | $\mathcal{R}$ | $\mathcal{P}$ | $\mathcal{R}$ | $\mathcal{P}$ | $\mathcal{R}$ | $\mathcal{P}$ |
| 1 | 0.67 | 0.16 | 0.67 | 0.57 | 0.67 | 1 | 0.67 | 1 |
| 2 | 1 | 1 | 0 | – | 0 | – | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 1 | 1 | 1 | 1 | 0 | – | 1 | 1 |
| 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 6 | 0.75 | 0.12 | 0.75 | 1 | 0.75 | 1 | 0.75 | 1 |
| 7 | 0 | 0 | 0 | 0 | 0 | – | 0 | 0 |
| average | 0.77 | 0.61 | 0.63 | 0.65 | 0.49 | 0.57 | 0.77 | 0.86 |
| average-2 | 0.90 | 0.71 | 0.88 | 0.91 | 0.85 | 1 | 0.90 | 1 |

Even when signature matching produced a result of low precision the remaining filters increased the precision to a very high level. And for two-thirds of the experiments VCR succeeded in proving that the retrieved components are in fact plug-compatible. The computational effort, however, is high. But due to the concept of successive filters, VCR is able to present acceptable intermediate results in short time.

The experiments demonstrate VCR's ability to locate software components via matching of implicit VDM specifications. Once its integration is completed it will be a viable high-precision retrieval tool to support the reuse approach to high-quality software.

**Acknowledgements**

G. Snelting originally proposed the idea of NORA/HAMMR. C. Lindig designed parts of the user interface, J. Rhiemeier implemented the VDM-SL parser, P. Heise and C. v. Grone specified parts of Lins' library, and E. Gode implemented the first version of the signature matching. M. Kievernagel and B. Fischer were supported by DFG, grants Sn11/1-2 and Sn11/2-2.

## 6. References

Bicarregui, J. C., Fitzgerald, J. S., Lindsay, P. A., Moore, R. and Ritchie, B. 1993. *Proof in VDM: A Practitioner's Guide*, FACIT series, Berlin: Springer-Verlag.

Bowen, J. P. and Hinchey, M. G. 1995. Seven more myths of formal methods, *Technical Report TR-357*, Computer Laboratory, University of Oxford.

Bowen, J. P. and Stavridou, V. 1993. The industrial take-up of formal methods in safety-critical and other areas: A perspektive. In *FME 1993*. LNCS 670, Berlin: Springer-Verlag. 183–195.

Craigen, D., Gerhart, S. and Ralston, T. 1993. Formal methods reality check: Industrial usage. In *FME 1993*. LNCS 670, Berlin: Springer-Verlag. 250–267.
Damas, L. and Milner, R. 1982. Principle type-schemes for functional programs. *Proc. 9th Annual ACM Symposium on Principles of Programming Languages*. ACM Press. 207–212.
Hall, J. A. 1990. Seven myths of formal methods. *IEEE Software*. **7(5)**: 11–19.
Hohlfeld, B. and Struckmann, W. 1992. *Einführung in die Programmverifikation*, Reihe Informatik, Mannheim: BI Wissenschaftsverlag.
Hussmann, H. 1995. Indirect use of formal methods in software engineering. In Wirsing, M. (ed.), *Proc. ICSE-17 Workshop on Formal Methods Application in Software Engineering Practice*. 126–133.
Kievernagel, M. 1990. *Auswahl und Installation eines Beweissystems*. Master's thesis. Braunschweig: Technical University of Braunschweig, Germany.
Krone, M. and Snelting, G. 1994. On the inference of configuration structures from source code. *Proc. 16th International Conference on Software Engineering*. IEEE Computer Society Press. 49–57.
Lins, C. 1989. *The Modula-2 Software Component Library*, Springer Compass International, New York: Springer-Verlag.
Maarek, Y. S., Berry, D. M. and Kaiser, G. E. 1991. An information retrieval approach for automatically constructing software libraries. *IEEE Transactions on Software Engineering*. **SE-17(8)**: 800–813.
Manhart, P. and Meggendorfer, S. 1991. A knowledge and deduction based software retrieval tool. *Proc. 4th International Symposium on Artificial Intelligence*. 29–36.
McCune, W. W. 1994a. *A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems*. Technical report. Argonne: Argonne National Laboratory.
McCune, W. W. 1994b. *Otter 3.0 user's guide*. Technical Report ANL-94/6. Argonne: Argonne National Laboratory.
Prieto-Diaz, R. 1987. Classifying software for reusability. *IEEE Software*. **4(1)**: 6–16.
Rittri, M. 1990. Retrieving library identifiers via equational matching of types. In Stickel, M.E. (ed.), *Proc. 10th International Conference on Automated Deduction*. LNCS 449. Berlin: Springer-Verlag.
Rollins, E. J. and Wing, J. M. 1991. Specifications as search keys for software libraries. In Furukawa, K. (ed.), *Proc. of the Eighth International Conference and Symposium on Logic Programming*. Paris: MIT Press. 173–187.
Snelting, G., Fischer, B., Grosch, F.-J., Kievernagel, M. and Zeller, A. 1994. Die inferenzbasierte Softwareentwicklungsumgebung NORA. *Informatik—Forschung und Entwicklung* **9(3)**: 116–131.
Weber-Wulff, D. 1993. Selling formal methods to industry. In *FME 1993*. LNCS 670, Berlin: Springer-Verlag. 671–678.
Woodcock, J. C. P. and Larsen, P. G. 1993. *Proc. FME'93: Industrial-Strength Formal Methods*. LNCS 670, Berlin: Springer-Verlag.