1988

Knowledge Base Support for Rapid Prototyping

Luqi

Naval Postgraduate School

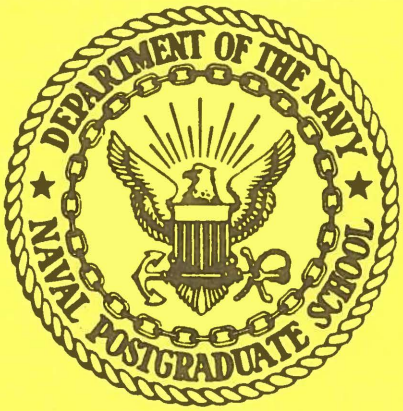
Luqi, "Knowledge Base Support for Rapid Prototyping", Technical Report NPS 52-88-016, Computer Science Department, Naval Postgraduate School, 1988.
https://hdl.handle.net/10945/65222

T 18

NPS52-88-016

NAVAL POSTGRADUATE SCHOOL

Monterey, California



KNOWLEDGE BASE SUPPORT FOR RAPID PROTOTYPING

LUQI

July 1988

Approved for public release; distribution is unlimited.
Prepared for:
Naval Postgraduate School
Monterey, CA 93943
National Science Foundation
Division of Computer and Computation Research
Washington, D. C. 20550

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

H. Shull
Provost

The work reported herein was supported in part by the National Science Foundation and the Naval Postgraduate School.

This report was prepared by:

[redacted]
LUQI
Assistant Professor
of Computer Science

Reviewed by:

[redacted]
ROBERT B. MCGHEE
Chairman
Department of Computer Science

Released by:

[redacted]
KNEALE T. MARSHALL
Dean of Information
and Policy Science

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPS52-88-016	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*) 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Naval Postgraduate School 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11 TITLE (*Include Security Classification*)
KNOWLEDGE BASE SUPPORT FOR RAPID PROTOTYPING (U)

12. PERSONAL AUTHOR(S)
LUQI

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Summary	FROM 87 Oct TO 88 Apr	1988 June	15

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

The knowledge base for rapid prototyping consists primarily of reusable software components. This paper discusses expert systems for retrieving and adapting reusable components eeting a given specification. Such components can be assembled into an executable prototype by a computer-aided prototyping system acting as an assistant to a human prototype designer.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL LuQi	22b. TELEPHONE (*Include Area Code*) (408)646-2735 22c. OFFICE SYMBOL 52Lq

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted.
All other editions are obsolete

☆ U.S. Government Printing Office: 1986—606-243

Knowledge Base Support for Rapid Prototyping

Luqi

Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

ABSTRACT

The knowledge base for rapid prototyping consists primarily of reusable software components. This paper discusses expert systems for retrieving and adapting reusable components meeting a given specification. Such components can be assembled into an executable prototype by a computer-aided prototyping system acting as an assistant to a human prototype designer.

1. Introduction

A software prototype is an executable pilot version of a software system. Prototypes are used to clarify requirements by demonstrating selected aspects of the proposed system behavior to the customer. Since prototypes are used in the initial negotiations leading to a software development project, they must be easy to construct and adapt to requirements changes. The prototyping language PSDL [9] and the associated computer-aided prototyping system [8] have been developed to make this possible. PSDL has been designed for prototyping large systems with real-time constraints, and can be used to express both black-box descriptions of systems and decompositions into networks of simpler operators communicating via data streams. The associated prototyping methodology relies on reusable software components drawn from a software base to speed up prototype construction [7]. In addition to the software base, the computer-aided prototyping system contains a translator for adapting and interconnecting components, schedulers for meeting real-time constraints, and interfaces for entering design decisions [5]. In this paper we discuss the knowledge base of the computer-aided prototyping system and show how expert system technology can be applied to the software base management subsystem, which is responsible for finding reusable software components with specified properties. Knowledge-based approaches to the development of software by means of transformations are described in [1, 3, 4].

We have focused on interactive systems for computer-aided prototyping rather than on completely automated application generators. Computer-aided prototyping differs from application generators by addressing a open-ended class of problems. Application generators are usually restricted to a fixed set of problems in a fixed application domain, and often have constrained problem description languages capable

of describing only the problems in that fixed set. The knowledge base of an application generator contains algorithms spanning the fixed set of problems along with rules for combining those algorithms to create programs corresponding to sentences in the problem description language. Such approaches can be useful in areas where all aspects of the user's problems can be anticipated by the designers of the knowledge base.

Prototyping is often applied in novel or poorly understood domains, in which it is impossible to anticipate all of the customer's problems in advance. An important goal of a prototyping effort is to make unanticipated aspects of the customer's problem surface at the early stages, to enable more effective planning and ensure the detailed design and implementation effort will be spent on the most important directions. The construction of completely automated systems for solving open-ended problems requires progress on currently unsolved research problems, making interactive systems attractive for near-term applications. Consequently, the computer-aided prototyping system has been designed to act as an assistant to a human expert on software design.

It is desirable to automate the process of managing the reusable components in the software base because this frees the human designer from the burden of remembering what software components are currently available. This can be a significant aid to the designer because the number of components in the software base can be very large. Expert system technology is appropriate for the problem because the number of useful software components is so large that it is impractical to store them all explicitly. A more practical approach is to store a representative subset and to provide rules for generating related components. The retrieval of reusable components from a software base is a difficult search problem in a very large space. Exhaustive searching is impractical because the space is much too large. Complete algorithmic solutions to the retrieval problem are unlikely because program equivalence and specification equivalence are both undecidable problems if the programming and specification languages are strong enough to be expressive. These considerations suggest using heuristic search methods, which are a common component of expert systems.

The designer uses the computer-aided prototyping system as follows. The requirements are obtained from the customer as written documents, with extensions and clarifications provided in response to the designer's questions and demonstrations of the prototype. The designer proposes a system interface consistent with the requirements, or makes an adjustment based on customer feedback, and records the

resulting specification in PSDL. The specification is submitted to the software base management system, which attempts to retrieve or adapt available reusable components to meet the specification. If this is not possible, than the designer must decompose the system into a network of simpler components using PSDL, until all of the components can be provided by the software base. This is analogous to the interaction between a mathematician and an automatic theorem proving program, where the mathematician proposes lemmas as intermediate steps if the main theorem is too difficult to be solved by the automatic procedure in a reasonable amount of time.

2. Retrieval Strategies

The purpose of the software base management system is to retrieve reusable software components for meeting a given specification with less effort on the part of the designer than it would take to code the components manually. Time delay is an important factor in the effort required to use the system. It is valuable to be able to produce answers with little perceptible delay, because this avoids disturbing the flow of the designer's thought processes. Since searching a large space can take appreciable amounts of time, a practical design should incorporate several search strategies that are ordered by speed. Those that can be applied with a small perceptible delay should be applied while the designer waits, and if they succeed, an answer should be reported immediately. If the fast methods fail, then the designer should be notified that the retrieval has been spooled, indicating an appreciable delay is in order. At that point the designer has the option of canceling the request and performing a manual decomposition immediately, or letting it continue and shifting to a different part of the problem. The spooled parts of the retrieval can operate in the background, and can be transferred to idle workstations if a network of machines is available. A natural working style with such a system is to refine prototypes in a breadth-first order, incorporating the results of the fast retrievals immediately and allowing spooled retrievals to run until the next refinement cycle comes back to the same part of the prototype.

The fastest and most superficial search strategy is based on exact matches on the component specifications. To facilitate retrievals based on exact matching, the specifications of a component are *normalized*, or transformed into a standard form before being entered into the system [6]. This process reduces variations in the representation of equivalent specifications, increasing the effectiveness of syntactic matching. Specifications are normalized for retrieval requests and for each component entered into the

software base. Components are indexed based on their normalized specifications, so that exact match searches can be performed quickly.

The retrieval strategies based on inexact matches and transformations are much more time consuming, so that it is important to apply these strategies to relatively small portions of the software base. For this reason, the software base is partitioned based on the values of several categorical properties, which are described in more detail in the next section, under the "category" facet. The partition relevant to a given retrieval request is identified based on the values of these properties, and the components in the partition are subjected to a "best first" heuristic search that attempts to create a match by applying various transformations. This can be done by means of an agenda mechanism similar to the one used in the AM system [2].

The knowledge in the knowledge base of the computer-aided prototyping system consists of two different kinds of information: descriptions of reusable software components, and rules for combining or adapting reusable components. This knowledge is described in more detail in the next two sections.

3. Declarative Knowledge

The declarative knowledge in the knowledge base consists primarily of reusable software components. This knowledge is most naturally organized as a frame system [10] where each frame corresponds to a software component. This can be compared to other applications of frame systems such as natural language understanding, where frames represent stereotypical situations, image analysis, where frames represent different viewpoints of a scene, or the exploration of mathematical systems, where frames represent mathematical concepts [2].

The frames in a frame system have a common set of *slots* or *facets*, which have the same interpretation for each frame in the system. A set of facets important for managing a set of reusable software components is described below.

Specification

This facet contains a PSDL specification of the software component. Retrieval is based on the specification, rather than on any attempt to analyze the code in the implementation.

Implementation

This facet contains the code for the software component. We maintain only one implementation for each elementary specification. This is consistent with our principle of specification-based retrieval: if there is a significant difference between two modules, then the specifications of the modules should describe that difference, so that the retrieval mechanisms can be sensitive to it. If there is no significant difference, then it is a waste of space to keep both modules. Differences in performance are sometimes important, and hence must be reflected by the specifications. The implementation of a non-elementary specification is a list of other modules which satisfy the specification (see "generalization below).

Category

The category facet contains several properties used for partitioning the software base into disjoint subspaces. This partitioning is used for improving the performance of the system, by limiting the part of the knowledge base that must be searched. The categorical properties include the programming language used for the implementation, the operating system it runs under, the component type (function, state machine, or data type), and the maximum execution time. The maximum execution time is a number that induces an ordering on each partition, and is used to limit retrievals to only those modules capable of meeting a given real-time constraint. The components in each partition are threaded together in a list that is kept sorted with respect to this ordering, to make it efficient to generate subsets of the components meeting a given bound on the execution time.

Generalizations

The generalizations facet contains a set of non-elementary modules whose specifications are satisfied by the given module. Generalizations are useful for matching requests that are less specific than the specification of a given module. This case arises because there may be many different elementary modules meeting a loosely phrased retrieval request. In such a case all of the elementary modules meet the retrieval request, and their individual specifications differ in respects that were not constrained by the original retrieval request.

Alternatives

The alternatives facet contains links to similar but incompatible modules, together with rules stating

when each link should be considered. These rules contain heuristics for directing the search in near-miss situations, where a module has been found that satisfies some parts of the retrieval specification but not others. These rules can also contain prescriptions for synthesizing composite modules by combining the current module with other modules with specified properties in given ways.

4. Transformations

One of the reasons that it has been difficult to re-use software in practice is that two instances of the "same" software component are rarely exactly alike. Instead, there are many small variations on a theme. The number of small variations can be unboundedly large, and it can be difficult to predict which variation will be needed next. A extreme example of this is the set of array sorting routines in standard Pascal. Since the type and the bounds of the array must be specified in the procedure header, a different sorting procedure is needed for each type and size of array. The standard solution to this problem is generic components, where the specification has one or more formal parameters that must be bound by the matching procedure. The retrieval mechanism must include a transformation which creates the required instantiation of the generic code template. This transformation can be almost trivial for programming languages like Ada that support generic units, and can involve some computation to expand substitutions inline for programming languages like Pascal that do not explicitly support generic units. Some care must be taken to avoid faults due to name collisions in the inline substitution process in such cases.

Another kind of transformation involves small local rearrangements to the interface of the reusable component. These include permuting the input and output parameters, ignoring extra output parameters, and filling in values for extra input parameters. The last process can be done by means of the unification algorithm. In case the body of the specification of a reusable component can be unified with the retrieval specification, the unifying substitution contains the values to be used for the extra input parameters. Such a transformation succeeds only if the values for the extra input parameters are expressible in the implementation language of the reusable component.

Other transformations yield building blocks for a composite component. An important function of an expert system for retrieving re-usable components is to do a limited amount of bottom-up design. This is necessary if we are to insulate the designer from the need to remember all of the reusable components in

the software base. If this knowledge is to be limited to the expert system, then the expert system must be able to steer the decomposition in directions that match available components. One way to achieve this goal is by means of composition rules. These are heuristics attached to particular modules in the software base, which indicate plausible ways to extend the module. This kind of information fits in the "alternatives" facet of a module. Two important categories of composition rules are the guard rules and the filter rules.

Guard rules describe decompositions induced by a case analysis. A guard rule produces a conditional statement by matching the specification of a reusable component against part of the retrieval specification. If the retrieval specification is "S" and the specification of the reusable component has the form "G => S", then the component can be augmented with a PSDL control constraint of the form

TRIGGERED IF G

provided that the guard G can be expressed in a form that is executable in PSDL. This provides a partial implementation, which can be completed by providing another conditional implementation that applies if G is false. A common case in which this pattern may apply is if a reusable component meets the required specification for normal inputs, but raises an exception condition in other cases. If the software base management system can return one or more guarded commands providing a partial implementation, the designer can often complete the job, or may notice that the specification is not satisfiable in the general case, and that some other response is appropriate in the remaining cases. A typical decomposition produced by the guard rule is illustrated in Fig. 1. The "?" represents an unknown operator that must be found or constructed to complete the implementation. The PSDL control constraints below the data flow diagram give the conditions under which each operator is invoked.

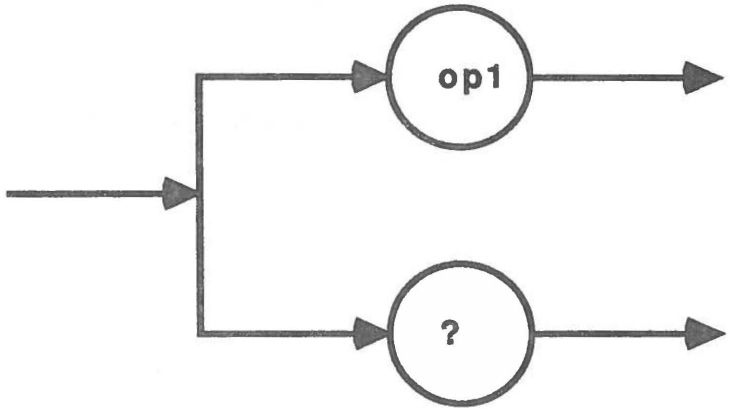
A filter rule factors a specification to allow it to be met by a two-stage data flow decomposition. For example, if a specification has the form

(x IN s <=> P(x)) & sorted(s)

a sorting filter rule suggests seeking a module producing an output sequence s such that

x IN s <=> P(x)

and connecting that output to a sorting operator for sequences of the appropriate data type. A decomposi-



OPERATOR op1 TRIGGERED IF G

OPERATOR ? TRIGGERED IF not G

Fig. 1 A Guard Rule Decomposition

tion produced by the filter rule for the "sort" operator is illustrated in Fig. 2. As before, the "?" represents an unknown operator that must be found or constructed.

5. Conclusions

This paper has outlined the structure of an expert system for retrieving reusable software components from a knowledge base. Such a system can be more effective and produce more accurate automated retrievals than classical information retrieval techniques using keyword searching on closed sets of programs. We believe that rule-based retrieval combined with a limited ability to adapt and combine available components is essential for making extensive software re-use practical in prototyping.

The design and implementation of the system described here is under way. This effort is part of a long-range project, because we have to assemble an effective set of reusable software components in order to refine and test the effectiveness of the proposed system. Important areas for future research are finding more effective sets of transformations and more efficient matching algorithms.

Fig. 2 A Filter Rule Decomposition

1. R. Balzer, "A 15 Year Perspective on Automatic Programming", *IEEE Trans. on Software Eng.*, Nov. 1985.

2. R. Davis and D. Lenat, *Knowledge-Based Systems in Artificial Intelligence*, McGraw Hill, New York, 1982.

3. P. Freeman, "A Conceptual Analysis of the Draco Approach to Constructing Software Systems", *IEEE Trans. on Software Eng. SE-13*, 7 (July 1987), 830-844.

4. C. Green and S. Westfold, "Knowledge-Based Programming Self-Applied", in *Machine Intelligence*, vol. 10 , Wiley, 1982.

5. Luqi, "Rapid Prototyping for Large Software System Design", Ph. D. Thesis, University of Minnesota, 1986.

6. Luqi, *Normalized Specifications for Identifying Reusable Software*, Proc. of the ACM-IEEE 1987 Fall Joint Computer Conference, Dallas, Texas, October 1987.

7. Luqi and V. Berzins, "Rapid Prototyping of Real-Time Systems", *IEEE Software*, July 1988.

8. Luqi and M. Ketabchi, "A Computer Aided Prototyping System", *IEEE Software*, March 1988.

9. Luqi, V. Berzins and R. Yeh, "A Prototyping Language for Real-Time Software", *to appear in IEEE TSE*, 1988.

10. M. Minsky, "A Framework for Representing Knowledge", in *Readings in Knowledge Representation*, R. Brachman and H. Levesque (editor), Morgen Kaufmann, Los Altos, CA, 1985, 245-262.

Initial Distribution List

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Center for Naval Analysis 4401 Ford Avenue Alexandria, VA 22302-0268	1
Office of the Chief of Naval Operations Code OP-941 Washington, D.C. 20350	2
Office of the Chief of Naval Operations Code OP-945 Washington, D.C. 20340	2
Commander Naval Telecommunications Command Naval Telecommunications Command Headquarters 4401 Massachusetts Avenue NW Washington, D.C. 20390-5290	2
Commander Naval Data Automation Command Washington Navy Yard Washington, D.C. 20374-1662	1
Office of Naval Research Office of the Chief of Naval Research Attn. CDR Michael Gehl, Code 1224 Arlington, VA 22217-5000	1
Director, Naval Telecommunications System Integration Center NAVCOMMUNIT Washington Washington, D.C. 20363-5100	1
Space and Naval Warfare Systems Command Attn: Dr. Knudsen, Code PD50 Washington, D.C. 20363-5100	1

Ada Joint Program Office 1
OUSDRE(R&AT)
The Pentagon
Washington, D.C. 230301

Naval Sea Systems Command 1
Attn: CAPT Joel Crandall
National Center #2, Suite 7N06
Washington, D.C. 22202

Office of the Secretary of Defense 1
Attn: CDR Barber
The Star Program
Washington, D.C. 20301

Naval Ocean Systems Center 1
Attn: Linwood Sutton, Code 423
San Diego, CA 92152-5000

National Science Foundation 1
Division of Computer and Computation Research
Washington, D.C. 20550

Director of Research Administration 1
Code 012
Naval Postgraduate School
Monterey, CA 93943

Chairman, Code 52 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100

LuQi 150
Code 52Lq
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100