1987

A Computer Aided Prototyping System

Luqi; Ketabchi, M.

Naval Postgraduate School

Luqi and M. Ketabchi, "A Computer Aided Prototyping System", Technical Report NPS 52-87-011, Computer Science Department, Naval Postgraduate School, 1987.
https://hdl.handle.net/10945/65239

NPS52-87-011

NAVAL POSTGRADUATE SCHOOL

Monterey, California

A COMPUTER AIDED PROTOTYPING SYSTEM

LuQi

Mohammad Ketabchi

April 1987

Prepared for:

Chief of Naval Research
Arlington, VA 22217

This report was prepared by:

[redacted]

LUQI
Associate Professor
of Computer Science

Reviewed by:

[redacted]

VINCENT Y. LUM
Chairman
Department of Computer Science

Released by:

[redacted]

KNEALE T. MARSHALL
Dean of Information and
Policy Science

NOTE:

This project was supported by the NPS Foundation Research Program which was funded by the Chief of Naval Research, Arlington, VA 22217.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-87-011	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (*and Subtitle*) A COMPUTER AIDED PROTOTYPING SYSTEM		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) LuQi; Mohammad Ketabchi, Santa Clara University		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61153N: RR014-01 N0001487WR4E011
11. CONTROLLING OFFICE NAME AND ADDRESS Chief of Naval Research Arlington, VA 22217		12. REPORT DATE April 1987
		13. NUMBER OF PAGES 20
14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*)		15. SECURITY CLASS. (*of this report*) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

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

Rapid prototyping is a promising approach in software technology. A Computer Aided Prototyping System (CAPS) makes rapid construction of software prototypes possible. This paper presents a CAPS based on normalized specifications and reusable software, and describes its integrated prototyping tools.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

A Computer Aided Prototyping System

Luqi
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

Mohammad Ketabchi
Computer Science Department
Santa Clara University
Santa Clara, CA 95053

ABSTRACT

Rapid prototyping is a promising approach in software technology. A Computer Aided Prototyping System (CAPS) makes rapid construction of software prototypes possible. This paper presents a CAPS based on normalized specifications and reusable software, and describes its integrated prototyping tools.

Key Words

Rapid Prototyping, Computer-Aided Design, Reusable Software, Software Base

1. Introduction

The rapidly growing demand for software has shifted towards larger systems and higher quality software, to the point where current software development methods are inadequate. A significant improvement in software technology is needed to improve programming productivity and the reliability of the software products. Computer aided rapid prototyping via specification and reusable components [8] is a promising approach which makes this improvement possible. In this approach the traditional software life cycle used in software design is replaced by a recently proposed alternative life cycle which consists of two phases: rapid prototyping and automatic program generation [16].

Completely automatic generation of programs from very high level specifications is not currently practical but automatic generation of prototypes is feasible. However, current prototyping methods require an impractical amount of time and effort. To reduce the cost of prototyping and to improve the efficiency of the process, a Computer Aided Prototyping System (CAPS) must be developed. Specifying, selecting, retrieving, and composing the reusable components into a prototype which meets a given set of requirements are important research problems which must be addressed before a CAPS becomes available.

This paper outlines a CAPS based on component specification and reusable software. Section 2 discusses the importance and the role of prototyping in software development. Section 3 outlines the process of automated prototyping. Section 4 gives an overview of CAPS architecture and briefly describes its major components. Section 5 presents concluding remarks.

2. Prototyping

A prototype is an executable model or a pilot version of the intended system. A prototype is usually a partial representation of the intended system, used as an aid in analysis and design rather than as production software. The construction activity leading to such a prototype is called prototyping. The goal in constructing a prototype is different than in constructing a production quality software system. Efficient use of designers' time and rapid feedback from the user are more important than robust operation, efficient use of machine resources, or completeness. Prototyping has been found to be an effective technique for clarifying requirements and eliminating the large amount of wasted effort currently spent on developing software to meet incorrect or inappropriate requirements in traditional software life cycles [2,14].

A key issue in the design of large software systems is reaching agreement on the requirements. Lack of agreement on the requirements as specified by the customer and as analyzed by the designer causes inconsistencies between the delivered system and customer expectations, leading to expensive rebuilding [14]. This problem is especially acute for large systems and systems with real-time constraints because the requirements for such systems are complicated to describe and difficult to understand. Because the user can usually recognize whether or not a working software system does what is needed, but usually can't describe the requirements accurately, prototypes are an effective means for achieving stable and accurate requirements early in the development process.

In prototyping the prototype is used in an iterative process of negotiation. The user describes the requirements, and the analyst interprets them and builds a prototype. The analyst then demonstrates the execution of the prototype to the customer. The requirements are adjusted based on feedback from the customer, and the prototype is modified accordingly until both the customer and the analyst agree on the requirements. This process is illustrated in Figure 1.

A prototype can be used to specify a well modularized skeleton design for the intended system and to validate its important attributes such as timing constraints, input and output formats, or interfaces between modules.

Prototyping is a useful tool in feasibility studies. Prototypes of critical subsystems or difficult parts of a complicated system can significantly increase the confidence that the system can be built before large amounts of effort and expense are committed to the project.

Prototyping also helps in estimating costs, since the cost of the intended system is usually proportional to the cost of the prototype. The experiences gained in applying rapid prototyping to special applications, e.g. database design, metaprogramming methods and translator design, have substantiated this cost relationship between the prototype and the completed system [3,7].

An automated support environment is essential for the rapid construction of prototypes.

3. Computer Aided Prototyping

Figure 2 illustrates the major steps in the process of computer aided prototyping using CAPS. The process begins by entering the specifications of the intended software system. To abstract away the syntactical differences the specifications are transformed into a normal form. The normalized specifications are used to search for desired components. If a unique component which meets the specification is found it is retrieved; if more than one component which meet the specification is found the designer must choose one; otherwise, either the specification should be decomposed or the required component should be hand-coded. After a specification is decomposed the entire process is applied to the specification of each component of the decomposition. Information about composing the components into the desired assembly is preserved when the specification is decomposed and is used after components become available.

The process relies on an improved modularization technique [11] and reusable software components. The support environment reduces the efforts of the analyst and designer by automating time consuming tasks of rebuilding large components, and searching for available components.

4. Major Components of CAPS

Software tools are needed to support the process of automated prototyping described in section 3. The major parts of such a system are the prototype description language PSDL, user interfaces for speeding up design entry and preventing syntax errors, an execution support system for demonstrating and measuring prototype behavior and for performing static analyses of the prototype design, a software design management system for retrieving and adapting reusable software components, and a component base which functions as a repository of the reusable components

An initial description of a framework for a rapid prototyping environment based on reusability can be found in [16]. We are developing methods for organizing a software base to aid retrieval of reusable components using normalized specifications, and better automated methods for component organization and retrieval.

Figure 3 illustrates the architecture of CAPS which supports the process illustrated in Figure 2. The following subsections describe the major components of CAPS.

4.1 Prototyping Language and Prototyping Method

The prototyping language PSDL [10] was developed together with a prototyping method [11] for rapidly constructing prototypes for large systems with real-time constraints in a CAPS tool [8].

A good language for expressing design thoughts in terms of a precise model is important for rapid prototyping. It is impossible to do a good design without a language especially designed for this purpose. A powerful, easy to use, and portable prototype description language is also a critical part of an automated rapid prototyping environment. Such a language is needed before the tools in the environment can be built. PSDL (Prototype System Description Language) was designed to serve as an executable prototyping language working at a specification or a design level. The language has special features for real-time system design.

PSDL provides sufficient structures and descriptive ability to describe the internal and external situation for the modules comprising the system. Software systems are modeled in PSDL as networks of operators communicating via data streams, which uses enhanced data flow diagrams for that purpose. The data streams can carry data values of an abstract data type [4] as well as tokens representing exception conditions. Each type or operator is either composite or atomic. Composite operators are implemented by decomposing them into networks of more primitive operators using PSDL. The decomposition of a composite operator is described in PSDL by an enhanced data flow diagram that includes non-procedural control constraints and timing constraints. Atomic operators are realized by retrieving an implementation from a software base [13,15] containing reusable software components.

Good modularity is one of the key factors for increasing productivity, since it significantly reduces the debugging effort for producing a correct executable system, and also influences the understandability, reliability, and maintainability of the developed system, which are especially important in rapid prototyping. A clear and powerful modularization model is introduced in PSDL for building and describing the prototype. This computational model is based on data flow under semantically unified control and timing constraints. It prevents hidden interactions between system components, to encourage designs with good module independence. The language and the associated prototyping method [9] lead to PSDL prototypes with a highly cohesive structure and few coupling problems, since they support the model and combine it with a powerful set of data and control abstractions to make it easy to describe systems at a high level. This structure is suitable for multiple modifications at a specification level during the prototyping iterations of the new life cycle.

The PSDL prototyping method results in a hierarchically structured prototype. The method provides a hierarchical decomposition strategy for filling in more details at any level of the prototype design. It also helps the designer to concentrate on the critical subsystems that must be refined to resolve the problems that motivated the rapid prototyping effort. The prototyping method uses stepwise refinement to selectively refine and decompose critical components. Each higher level component is described in terms of lower level ones and the relations between them. The decomposition of each composite component is a realization of the system at a lower level of detail.

The prototype is designed based on abstract functions, abstract data, and abstract control. This high level view emphasizes the overall configuration at each level without getting bogged down in programming level details. The design is refined by decomposing abstract functions and data types into lower level ones. Functional, data, and control abstractions are used to hide lower level details. Control constraints are combined with the data flow model to achieve the best modularity with sufficient control information. Data flow is used to simplify the interactions between modules, eliminating direct external references and communication by means of side effects.

PSDL and its prototyping method have been designed for use in an environment containing a software base management system, an execution support system, a syntax directed editor with graphics capabilities, and a software design database management system.

4.2 Rewrite Subsystem

Our approach to component specifications is based on term rewriting systems. The approach depends on normalizing specifications to reduce the variations in the representation of software concepts. We seek component specifications that admit effective reductions to a normal form to aid component retrieval. If many different specifications with the same meaning can be reduced to the same normal form, then designer can have freedom of expression while allowing the information retrieval system to have fewer syntactically distinct forms for each semantically distinct module that may appear in the software base. Since they can be unboundedly many syntactic forms for the same semantic description, reduction to a normal form is a more practical approach than attempting to generate all variations and searching the software base for each variation. An example of such a system for an informal approach to component specification is shown below:

TERM	ALIASES
update	change, modify, refresh, replace, substitute
read	fetch, obtain, input, get, retrieve

The rewrite rule defined by such a table simply replaces all occurrences of the aliases by the associated basic terms given in the table. The sentence

"Fetch the order from the transaction file and modify the inventory"

would be rewritten to

"Read the order from the transaction file and update the inventory"

The rewrite subsystem will map equivalent specifications into a normalized specification (see Figure 4) which will be used by Software Design Management System to find and retrieve the required components from the component base. Two kinds of normalization techniques, for formal and informal specifications, are discussed respectively in [12]. In either case, the specifications stored with the components in the component base are normalized. Operations to manipulate normalized specifications are made available by SDMS.

4.3 Software Design Management System

The SDMS is responsible for organizing, retrieving, and instantiating reusable software components from the software base, and managing and alternatives of prototypes. A SDMS must support efficient selection and retrieval of the relevant software components from a component base, because the retrieval must take less effort than constructing the components for computer aided prototyping to be practical.

A SDMS is essentially a database management system which can efficiently manage long transactions, data describing complex objects such as software components, the iterative and tentative nature of the design process which leads to versions, refinements, and alternatives of the design objects, and concurrent design operations in a distributed computing environment. In addition it provides special purpose operations for composition of components, browsing the component base, and manipulating the normalized specifications.

We have compared and contrasted the capabilities of conventional database management systems with the requirements of computer aided design applications in general and computer aided software engineering specifically. Since the conventional DBMSs do not meet the data management requirements of CAD applications we are developing an Object-Oriented DBMS (OODBMS) based on our object-oriented data model ODM [6] for SDMS. ODM meets the requirements of CAD applications with considerable simplicity and economy of concepts. We will tailor ODM and the OODBMS to meet the more specific requirements of CAPS. Note that we are not excluding the use of a commercially available DBMS for the purpose of experimenting with and studying the features of CAPS.

4.4 Software Base

Reusable components must be interpretable and portable [15]; the component base must be easily extensible in order to allow the evolutionary growth of the available components; it must be possible to browse, select, and retrieve components from the component base efficiently. To achieve these goals we are developing a highly structured component base. Three major structural foundations of the component base are generalization per category, approximation of specifications, and composition of components.

Components have certain properties which are used for their categorization. Examples of these properties, called categorical properties [5] are implementation languages, hardware environments, and time and space requirements. Generalization of components according to their categorical properties imposes a lattice structure on the set of components. The lattice structure of the components allows efficient browsing of the component base, supports efficient selection and retrieval of the components, and captures the semantics of component categorization which in turn increases the interpretability of the component base.

The component base contains a large set of normalized specifications corresponding to unique implementations. These specifications are called singleton specifications. A pair containing a singleton specification and its implementation is a singleton component. A normalized specification requested by the designer may not be singleton but an approximation of some singleton specifications. A specification S_i is an approximation of an specification S_j if S_j implies S_i. The SDMS provides operations which allow derivation of best approximation of a set of singleton specifications. The set of singleton specifications and their approximations has a lattice structure. Singleton specifications which are explicitly stored are the basis of this lattice. Other specifications can be derived from singleton specifications. Figure 5 illustrates a component base with four singleton specifications (S_s) and three approximate specifications (S_a)

Some components are compositions of components which themselves may be composed of other components. The components which are not composed of other components are atomic. Only atomic components need to be stored in the component base explicitly. Atomic components are hand-coded while composite components are derived by using a composition template which gives the subcomponents of the desired composite component and their interconnections.

4.5 Execution Support System

A problem oriented top-down strategy is used to focus the prototyping effort on critical problems or selected attributes of the entire system. The major system attributes that must be demonstrated to the user usually appear in a critical subsystem. It is necessary to create a quick sketch of the skeleton of the intended system, because the environment of the critical subsystem must be at least partially simulated to demonstrate the behavior of the prototype. This quick sketch can be built rapidly and understood easily by means of a highly interactive graphics editor for PSDL. The essential advantage of rapidly building the sketch of the prototype is that it provides an initial description of the intended system, which can serve as the basis for analysis and negotiation. The prototype system gradually fulfills the requirements during the iterations of the prototyping effort [8]. Our prototyping method enables each update to the prototype to be made quickly and easily [1].

5. Concluding Remarks

The research described here will solve several key problems in automated prototyping based on reusable software. Conceptual design of an integrated

CAPS, prototyping language and methodology, normal form of the specifications of reusable components, a software design database management system, and a software base which supports efficient retrieval of components by their specification are among these problems.

The results of this research will help to increase productivity in the development of software systems and improve the quality of the resulting products.

The proposed CAPS combines a high level prototyping language PSDL [8], a systematic design method for rapid construction of prototypes [11], an automated prototyping environment with a *software base* [13] containing a large set of reusable software components, and a software design database management system to manage the component base and the prototype design data. Application of the proposed CAPS should sharply reduce the need for requirements changes after implementation has begun, as well as multiple requirements changes during the design of a new feature in an evolving system. Demonstration of the prototypes constructed by using CAPS will provide feedback early enough in the development cycle to enable extensive adaptation of the design without wasting a large investment of effort. This should lead to products that closely match user needs. The approach described here uses specifications as an intimate part of the computer-aided implementation, making documentation a natural by-product of development rather than a costly extra task, and helping to ensure that the documentation corresponds to what the system actually does.

CAPS is an extensible system because it provides facilities for adding and utilizing new components to its software base. The integrated approach taken in CAPS to the maintenance and the management of prototype design data simplifies the adaptation of new tools and techniques. It also provides a knowledge base for expert design and analysis tools.

1. V. Berzins, "On Merging Software Extensions", *Acta Informatica 23* (1986), 607-619.

2. V. Berzins, M. Gray and D. Naumann, "Abstraction-Based Software Development ", *Comm. of the ACM 29*, 5 (May 1986), 402-415.

3. J. Connell and L. Brice, "Rapid Prototyping", in *Datamation*, Aug. 1984, 93-100.

4. J. V. Guttag, E. Horowitz and D. R. Musser, "Abstract Data Types and Software Validation", *Comm. of the ACM 21*, 12 (1978).

5. M. Ketabchi and V. Berzins, "Generalization Per Category: Theory and Application", *Proc. Int. Conf. on Information Systems*, 1986.

6. M. Ketabchi, V. Berzins and S. March, "ODM: An Object Oriented Data Model for Design Databases", in *Proc. ACM Computer Science Conference*, Feb. 1986.

7. L. Levy, "A Metaprogramming Method and Its Economic Justification", *IEEE TSE SE-12*, 2 (Feb. 1986), 272-277.

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

9. Luqi and V. Berzins, "Rapid Construction of PSDL Prototypes", Tech. Rep.-86-17, Computer Science, University of Minnesota, 1986.

10. Luqi, V. Berzins and R. Yeh, "A Prototyping Language for Real-Time Software", *to appear in IEEE TSE*, 1987.

11. Luqi and V. Berzins, "Rapid Prototyping of Real-Time Systems", *Revised for IEEE SOFTWARE*, 1987.

12. Luqi, *Normalized Specifications for Identifying Reusable Software*, submitted to ACM-IEEE 1987 Fall Joint Computer Conference, Dallas, Texas,

October 1987.

13. N. Roussopoulos, "Architectural Design of the SBMS", Quarterly Report for the STARS SB/SBMS Project, Department of Computer Science, Univ. of Maryland, Apr. 1985.

14. R. T. Yeh, *Software Engineering*, IEEE Spectrum, Nov. 1983.

15. R. T. Yeh, N. Roussopoulos and B. Chu, "Management of Reusable Software", *Proc. COMPCON*, Sep. 1984, 311-320.

16. R. T. Yeh, R. Mittermeir, N. Roussopoulos and J. Reed, "A Programming Environment Framework Based on Reusability", *Proc. Int. Conf. on Data Engineering*, Apr. 1984, 277-280.

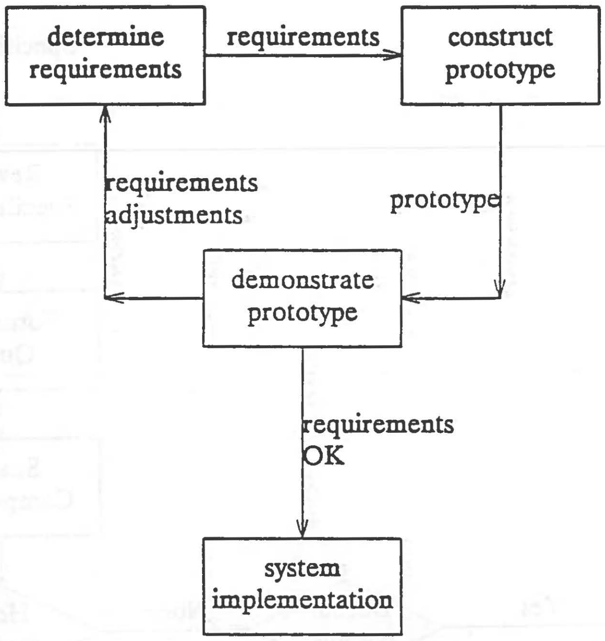

Figure 1 - Process of requirments determination and validation by prototyping

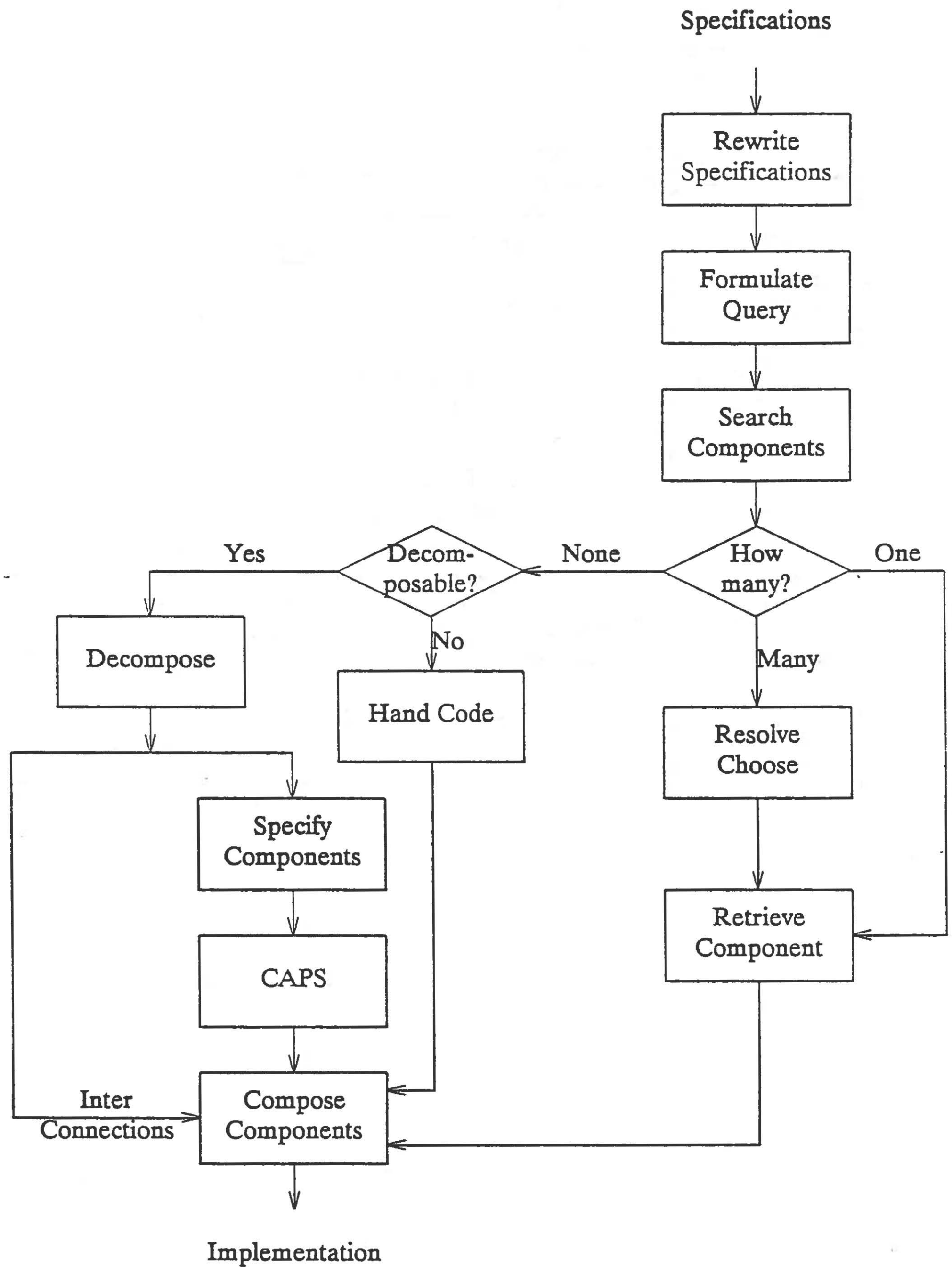

Figure 2 - Prototype development using CAPS

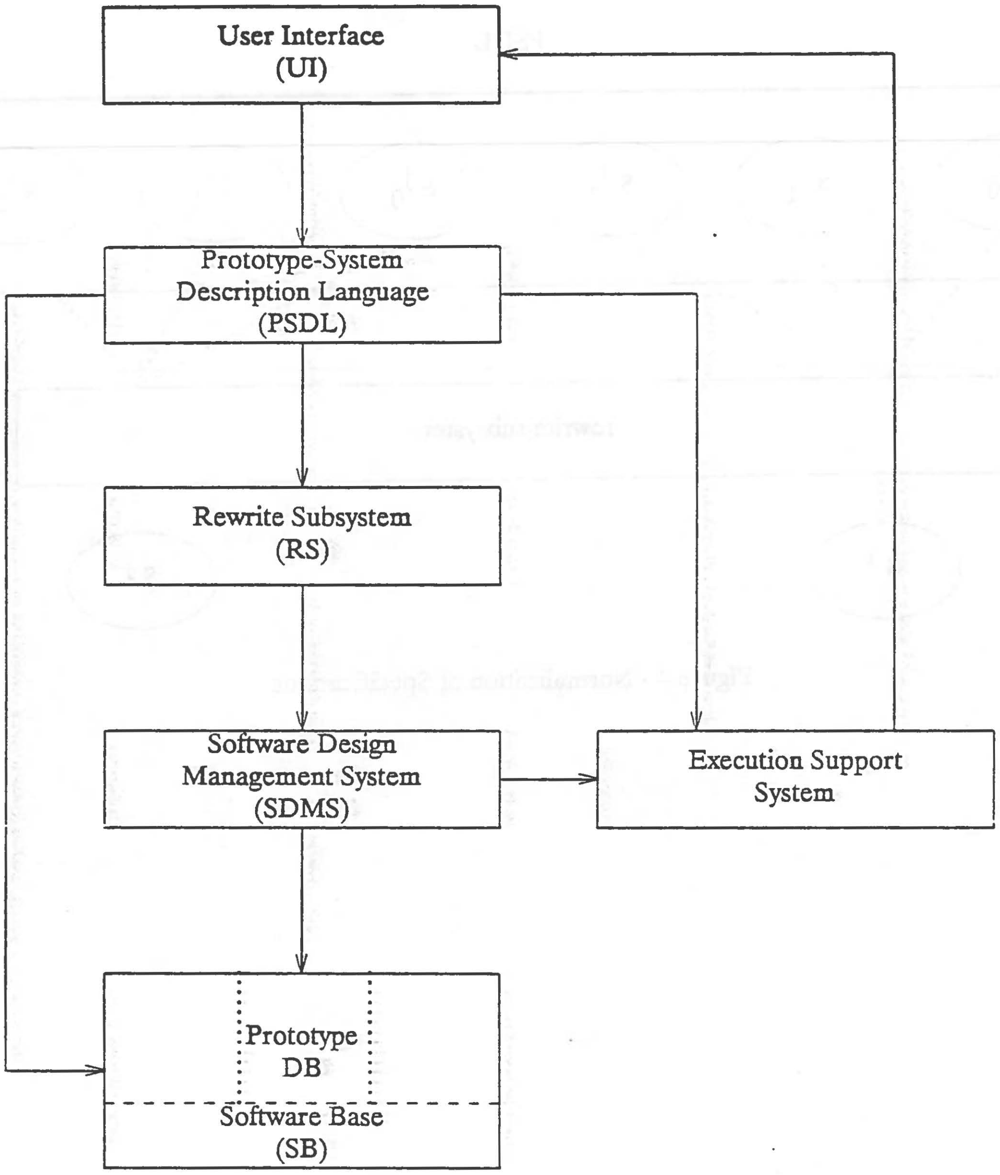

Figure 3 - CAPS Architecture

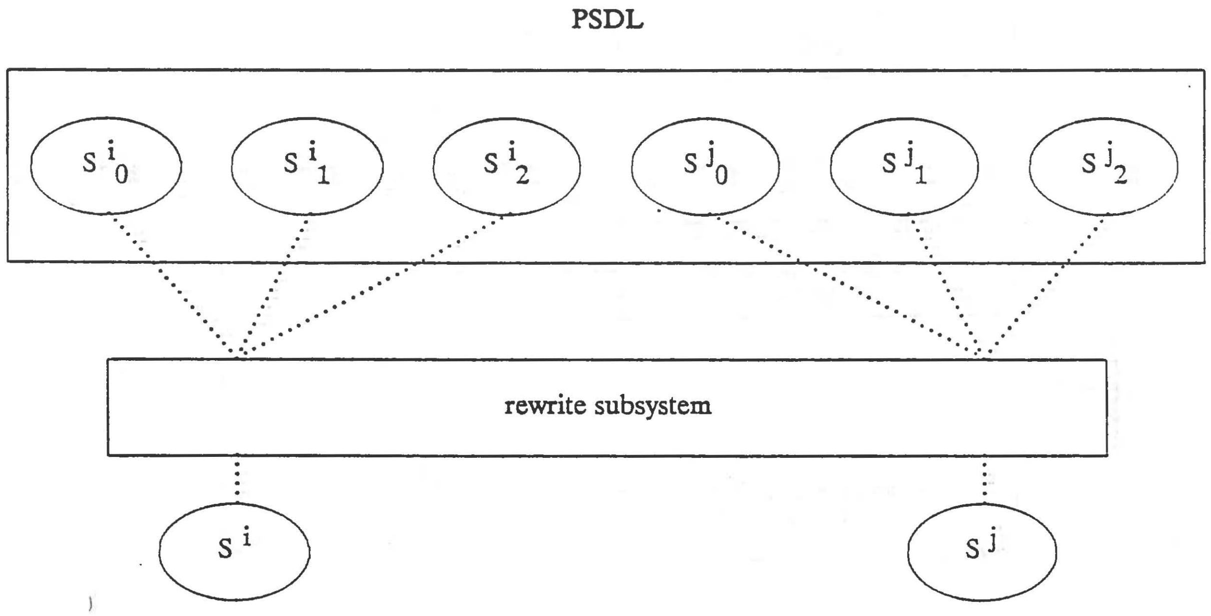

Figure 4 - Normalization of Specifications

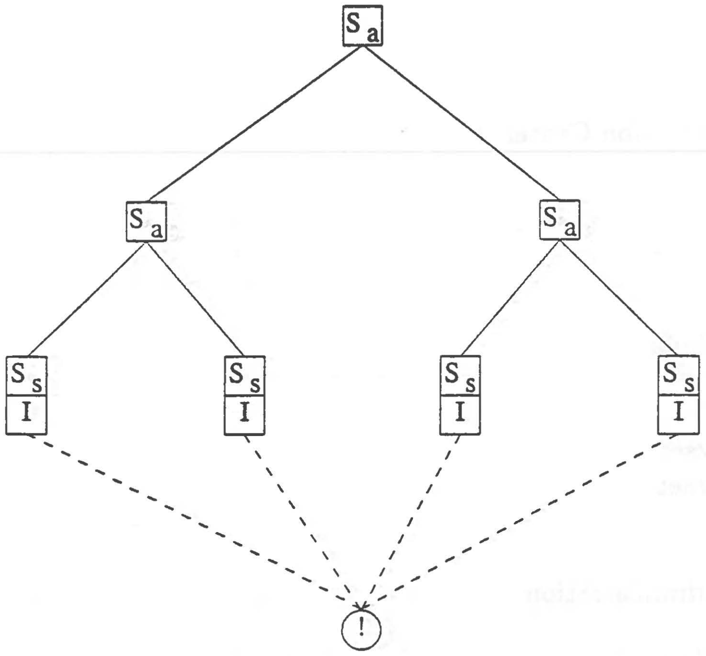

Figure 5 - A component base with 4 singleton components

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 Analyses 2000 N. Beauregard Street Alexandria, VA 22311	1
Director of Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943	1
Chairman, Code 52 Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	1
LuQi Code 52Lq Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	100
Chief of Naval Research 800 N. Quincy St. Arlington, VA 22217-5000	1