Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

460

Jürgen Uhl
Hans Albrecht Schmid

A Systematic Catalogue of Reusable Abstract Data Types

Springer-Verlag
Berlin Heidelberg New York London
Paris Tokyo Hong Kong Barcelona

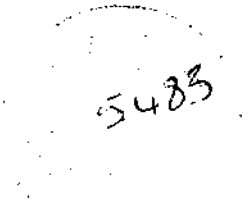
BIBLIOTHEQUE DU CERIST

Editorial Board
D. Barstow W. Brauer P. Brinch Hansen D. Gries D. Luckham
C. Moler A. Pnueli G. Seegmüller J. Stoer N. Wirth

Authors
Jürgen Uhl
Forschungszentrum Informatik, Universität Karlsruhe
Haid-und-Neu-Straße 10–14, W-7500 Karlsruhe, FRG

Hans Albrecht Schmid
Fachbereich Informatik, Fachhochschule Konstanz
Brauneggerstraße, W-7750 Konstanz, FRG

BIBLIOTHEQUE DU CERIST



CR Subject Classification (1987): D.2.2, D.2.m, D.3.3, E.2

ISBN 3-540-53229-3 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-53229-3 Springer-Verlag New York Berlin Heidelberg

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in other ways, and storage in data banks. Duplication of this publication or parts thereof is only permitted under the provisions of the German Copyright Law of September 9, 1965, in its current version, and a copyright fee must always be paid. Violations fall under the prosecution act of the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1990
The copyright for the Ada Specifications (Appendix A) rests with the authors.
Printed in Germany

Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
2145/3140-543210 – Printed on acid-free paper

More and more, people are starting to think of software as an investment. As a consequence, the concern is growing on how to develop software that can be maintained for long periods of time and adapted to new uses easily. This lead to a broad interest in software reusability. The motto "don't redo what is there already" is gaining support.

Reusability can be envisaged at the level of entire systems or at the level of individual pieces or components of a system. Source code can be reused, but also test cases, designs, specifications and even requirements. Software is considered reusable if it can be easily employed for applications it was not intended for originally.

The reuse technology that appears to be most mature is that of generalized components, frequently referred to as building blocks. Libraries of building blocks have been developed and successfully employed in several application areas. At the core of such libraries, particularly if they are intended for applications in system programming, are basic data structures. Ideally they are offered in the form of abstract data types.

The concept of abstract data types is one of the most successful and pervasive structuring concepts introduced into the software development practice. It embodies such software engineering principles as separation of interface and implementation information hiding, localization of functions, and parameterization. Their theoretical properties have been studied quite extensively and they are amenable to algebraic and axiomatic specifications. Designing with abstract data types fosters reuse because it encourages going from a special situation to the more general, or generic, case through classification and abstraction.

Building on a strong technical foundation laid by Kleine [8] several years ago, Uhl and Schmid have designed and implemented a new set of abstract data types that are described in this book. The implementations in this catalogue are done in the programming language Ada.

In contrast to previous similar efforts, Uhl and Schmid introduce two major new ideas. They try to achieve the utmost degree of consistency as far as the external interfaces are concerned. They then device a hierarchical relationship between the various data types which serves as the basis for the implementation, i.e. the more complex members are built out of the more elementary ones – reuse within reuse. The result of this is a very high number of variants that can be produced from a rather small code base.

BIBLIOTHEQUE DU CERIST

Since the advantages of this approach in terms of development cost, code quality, learning effort and maintenance are quite obvious, I like to call this a second generation catalogue.

In addition to the catalogue itself, the authors give valuable practical guidelines on how to design using abstract data types. A comprehensive example at the end of the book illustrates the ideas through a realistic application.

Both authors have spent several years on industrial software projects or on joint studies between industry and academia. They have helped to introduce formal design methods and abstract data types. The entire text reflects this experience.

Albert Endres, Böblingen

BIBLIOTHEQUE DU CERIST

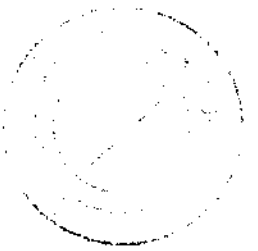
BIBLIOTHEQUE DU CERIST

Acknowledgments

Our thanks are due to the institution and persons who made this work possible, and to our colleagues with whom we had fruitful discussions. Especially, we want to acknowledge

- the support of IBM and, in particular, of Albert Endres, who established the department of research and development in reusability at the IBM Böblingen development laboratory as early as 1983, and
- the base laid by Karl Kleine from FZI Karlsruhe, who made – under the contract of this department – the first steps into the land of catalogues and libraries of abstract data types as reusable components in 1984.

Jürgen Uhl, Karlsruhe
Hans Albrecht Schmid, Konstanz
May 1990



Contents

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

Chapter 1

Introduction

BIBLIOTHEQUE DU CERIST

Reusability is one of the most promising issues in today's arena of software engineering. Some people expect it to put an end to the "software crisis"; others, however, consider it a technique that has been practiced since the beginning of software development and do not expect dramatic impacts. We believe and have experienced ourselves that reuse in the area on which we will focus here, can significantly decrease the cost of software development and maintenance and can improve essential system properties, like modularity and reliability.

Reuse comes in many different flavors. Biggerstaff describes the following framework for reusability technologies in [3]:

Features	Approaches to Reusability				
Component Reused	Building Blocks		Patterns		
Nature of Component	Atomic and Immutable Passive		Diffuse and Malleable Active		
Principle of Reuse	Composition		Generation		
Emphasis	Application Component Libraries	Organization & Composition Principles	Language Based Generators	Application Generators	Trans-formation Systems
Typical Systems	Libraries of Subroutines	Obj. Oriented, Pipe Archs	VHLLs POLs	CRT Fmtrs File Mgmt	Language Transf.

Our focus is on a rather specific and narrow class of reusable components, namely on basic abstract data types, like lists, stacks, trees or sets. On the first glance this falls into the left-hand side categories of Biggerstaff's classification. However, the work also touches some aspects from the right-hand side as we will soon explain.

Our work was triggered from experience which the second author made with IBM where, in systems programming projects, abstract data types formed a considerable part of the components reused [12]. Therefore, he initiated a project that produced a catalogue of abstract data types [8]. This is, to our knowledge, the first published collection – though only internal to IBM – of reusable abstract data types. As such, it had some deficiencies which were the reason and gave the motivation to start this work.

The main objective of our work is to present a practically useful library of efficient components that include the major data structures, which are known and used across different areas, in particular, in systems programming. The sub-objectives and goals that we derived from these objectives will be discussed in chapter 2.

The components are specified and implemented in Ada and can thus be used for realistic applications.

The research on organization and composition principles was not one of our primary objectives. Nevertheless, we had to do some considerable work in this field during the search for an appropriate structure of the library. Our focus was on defining an orthogonal structure that should ease the search, the use and the exchange of components.

Let us summarize here the main features, which distinguish our catalogue of abstract data types from existing work:

- A clear and strict separation between functional aspects and implementation or performance oriented aspects:
  - The functional behavior is defined by around ten abstract data types, which have a standardized interface across all data types.
  - Every abstract data type has in the order of thousand different implementations to be accessed by one uniform interface.

    There are general variations of reusable components with respect to space bounds or potential concurrency, as Booch describes them in [4], which we have defined in a similar way (though we have lesser subclasses and are

BIBLIOTHEQUE DU CERIST

not interested in concurrency in this book). In addition, the objective of efficiency leads to a large amount of abstract data type specific implementation variations.

We call both of them variants, which are considered as points in a library space that is spanned by (mainly orthogonal) basic properties.

According to modern software engineering principles, an abstract data type and its implementation should be determined stepwise. First, the problem "specification" should be stated thus abstracting from the efficiency related properties ("implementation details"). With our catalogue, the abstract data type is to be selected only on the base of this specification.

In terms of this specification one has to reason about the implementation properties, which results in the selection of the variants.

Under this viewpoint the nature of the building blocks as a whole is no longer atomic and immutable (as Biggerstaff puts it in his framework), but becomes similar to a transformational approach. This transformational approach has also been suggested for the use in very high level languages (VHLL), in particular in the area of high level data types. By making the implementation properties of the variants in our catalogue explicit, we hope to contribute to the areas of VHLLs and program transformation paradigms.

- The definition of an (implementation) hierarchy of abstract data types that allows,
  - from the user viewpoint, to select the most general abstract data type suitable for the application. At the same time, it is guaranteed that one can select among all implementation variants that might be available with a less general abstract data type.
  - from the implementor's and maintainer's viewpoint, to implement every data representation and access algorithm only once, but have it available for every suitable abstract data type ("reuse within reuse").

With these features, we are able to meet general requirements to a catalogue of abstract data types, which are derived from the reuse paradigm presented in section 3.3.

How is our attempt related to other areas of reuse? It seems that the reuse of data types is at the lower end of the scale of reusable software, with regard to the size of a single component. Therefore, it may be hard to apply our experience to other, higher level areas of reuse. On the other hand, such higher level components might reuse the lower level ones and we have to consider the impact of reusing the low level components on reusability on the higher level.

BIBLIOTHEQUE DU CERIST

The gain from a library of reusable components is equal to the product from the gain of a single reused component and the frequency of actual uses. For the class of basic data types the gain from reusing a single component is small, compared to the gain of reusing more complete, application oriented solutions. However, these data types are so frequently used that the overall gain is expected to become substantial.

This book is divided into two parts. The first part (chapters 2 - 4) discusses the motivation, suggest a general strategy for reusing abstract data types and gives a language independent introduction to the structure and functionality of our library. The second part is devoted to the realization of these ideas in Ada. It discusses the essential means provided by Ada and the specific design decisions that were based on the language. An extended example shows the use of the catalogue and finally we present the complete specifications of the library components.

BIBLIOTHEQUE DU CERIST