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

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



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

Foreword

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. 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

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

1	Intr	oduction	1						
2	Mot	lotivation and Objectives							
	2.1	Simplicity of the Library Structure	6						
	2.2	Functional Completeness	8						
	2.3	Completeness of Implementation Variants	.0						
	2.4	Maintainability	2						
3	Ноу	v to Reuse Abstract Data Types 1	5						
	3.1	Working With Data Structures - A Counter-Example	6						
		3.1.1 Selecting a Data Structure	7						
		3.1.2 Designing the Operations on the Data Structure	.7						
		3.1.3 Conclusion	8						
	3.2	Reusing Abstract Data Types	.9						
		3.2.1 Selecting "Low Level" Abstract Data Types	20						
		3.2.2 Selecting the "Right" Abstract Data Types	23						
		3.2.3 An Abstract Specification and Solution	25						
		3.2.4 Selection of an Abstract Data Type Variant	27						
	3.3	Reuse Paradigm	29						
4	Stru	acture of the Catalogue 3	7						
	4.1	The Implementation Hierarchy	17						
	4.2	The Abstract Data Types - an Overview	39						
		4.2.1 List	10						
		4.2.2 Stack	ŧ0						
		4.2.3 Queue and Deque	1						
		4.2.4 Tree	1						
		4.2.5 Order	1						
		4.2.6 Set	12						
		4.2.7 Map	12						
		•	_						

		4.2.8	Bag							
	4.3	Varia	nts of Building Blocks							
5	Str	cucture of the Building Blocks 45								
	5.1	Gener	al Properties of the Data Types $\dots \dots \dots$							
		5.1.1	Objects and Values							
		5.1.2	Structure Sharing							
		5.1.3	Compact and Dispersed Representations							
		5.1.4	Recursive Composition 49							
	5.2	Types	50							
		5.2.1	The Type STRUCT							
		5.2.2	The Type INDEX							
		5.2.3	INSERT_POSITIONs and REMOVE_POSITIONs							
		5.2.4	Structure PARTs							
		5.2.5	The Type ITERATION_ORDER							
	5.3	Opera	tions							
		5.3.1	Constructors							
		5.3.2	Operations Based on Indices							
		5.3.3	Access by Position Count							
		5.3.4	Selector Operations							
		5.3.5	Iterators							
		5.3.6	Reduction							
		5.3.7	Find, Skip and Count							
		5.3.8	Existential and Universal Quantifiers							
		5.3.9	Order Dependent Operations							
		5.3.10	Hash Operation							
	5.4	Gener	ic Parameters							
		5.4.1	Equal, Copy, and Transfer							
		5.4.2	Initialization							
		5.4.3	Deallocation 96							
		5.4.4	An Example							
		5.4.5	Key-Info Types							
		5.4.6	Accessed Elements							
		5.4.7	Bounded Collections							
	5.5	Varian	it Parameters							
		5.5.1	Classes of Element Types							
		5.5.2	Collection Management							
		5.5.3	Consistency Checking							
		5.5.4	Algorithms vs. Data							
		5.5.5	Summary							

6	The	Buildi	ing Blocks	111
	6.1	Linked	Collections	. 112
		6.1.1	Synopsis	. 112
		6.1.2	Types and Operations	. 112
		6.1.3	Implementation Overview	. 113
		6.1.4	Specific Generic Parameters	. 114
		6.1.5	Specific Variants	. 114
		6.1.6	General Variants	. 114
		6.1.7	Representation of Linked Collections	. 116
	6.2	Tabula	r Collections	. 117
		6.2.1	Synopsis	. 117
		6.2.2	Types and Operations	. 117
		6.2.3	Implementation Overview	. 118
		6.2.4	Specific Generic Parameters	. 118
		6.2.5	Specific Variants	. 119
		6.2.6	General Variants	. 119
	6.3	Hash t	ables	. 121
		6.3.1	Synopsis	. 121
		6.3.2	Types and Operations	. 121
		6.3.3	Implementation Overview	. 122
		6.3.4	Specific Generic Parameters	. 123
		6.3.5	Specific Variants	. 123
		6.3.6	General Variants	. 126
		6.3.7	Combination of Variants	. 126
		6.3.8	Representation of Hash tables	. 127
	6.4	Lists, l	Linked Lists, Tabular Lists	. 128
		6.4.1	Synopsis	. 128
		6.4.2	Types and Operations	. 128
		6.4.3	Implementation Overview	. 138
		6.4.4	List Specific Variants	. 139
		6.4.5	General Variants of Lists	. 139
		6.4.6	Combination of List Variants	. 140
		6.4.7	Linked List Specific Variants	. 140
		6.4.8	General Variants of Linked Lists	. 142
		6.4.9	Combination of Linked List Variants	. 142
		6.4.10	Representation of Linked List by Linked Collections	. 142
		6.4.11	Tabular List Specific Variants	. 143
		6.4.12	General Variants of Tabular Lists	. 146
		6.4.13	Representation of Tabular Lists by Tabular Collections	. 146
	6.5	Stacks	· · · · · · · · · · · · · · · · · · ·	. 148
		6.5.1	Synopsis	. 148
		6.5.2	Types and Operations	. 148

	6.5.3	Implementation Overview
	6.5.4	Specific Variants
	6.5.5	General Variants
	6.5.6	Combination of Variants
	6.5.7	Representation Specific Variants
	6.5.8	Representation of Stacks by Lists
6.6	Queue	s and Deques
	6.6.1	Synopsis
	6.6.2	Types and Operations
	6.6.3	Implementation Overview
	6.6.4	Specific Variants
	6.6.5	General Variants
	6.6.6	Combination of Variants
	6.6.7	Representation Specific Variants
	6.6.8	Representation of Queues and Deques by Lists
6.7	Trees,	Linked Trees, Tabular Trees
	6.7.1	Synopsis
	6.7.2	Types and Operations 199 159
	6.7.3	Implementation Overview
	6.7.4	Specific Generic Parameters
	6.7.5	Tree Specific Variants
	6.7.6	General Variants of Trees
	6.7.7	Combination of Tree Variants
	6.7.8	Linked Tree Specific Variants
	6.7.9	General Variants of Linked Trees
	6.7.10	Combination of Linked Tree Variants
	6.7.11	Representation of Linked Trees by Linked Collections 112
	6.7.12	Tabular Tree Specific Variants
	6.7.13	General Variants of Tabular Trees
	6.7.14	Representation of Tabular Trees by Tabular Collections 174
6.8	Orders	;
	6.8.1	Synopsis
	6.8.2	Types and Operations 177
	6.8.3	Implementation Overview
	6.8.4	Specific Generic Parameters
	6.8.5	Specific Variants
	6.8.6	General Variants
	6.8.7	Combination of Variants
	6.8.8	Representation Specific Variants
	6.8.9	Representation of Orders by Lists and Trees
6.9	Sets an	nd Maps
	6.9.1	Synopsis

		6.9.2	Types and Operations	5
		6.9.3	Implementation Overview	7
		6.9.4	Specific Generic Parameters	8
		6.9.5	Specific Variants	8
		6.9.6	General Variants	8
		6.9.7	Combination of Variants	9
		6.9.8	Representation Specific Variants	9
		6.9.9	Representation of Sets and Maps 19	3
	6.10	Bags		4
		6.10.1	Synopsis	4
		6.10.2	Types and Operations	4
		6.10.3	Implementation Overview	5
		6.10.4	Specific Generic Parameters	6
		6.10.5	Specific Variants	6
		6.10.6	General Variants	7
		6.10.7	Combination of Variants	7
		6.10.8	Representation Specific Variants	7
		6.10.9	Representation of Bags 19	δ
7	Tecl	hnical	Issues 19	9
	7.1	Selecti	on of Building Block Variants	9
	7.2	Ada D	esign Decisions	0
		7.2.1	Dispersion of Structures Objects	$\overline{2}$
		7.2.2	Composition of Data Types	4
		7.2.3	Storage Management	5
	7.3	Deficie	ncies and Open Problems	6
		-		•
8	Cas	e Stud	y: A File Compression System 20	9
	8.1	The O	verall Task	i9
	8.2	Search	for Reusable Building Blocks (Phase 1)	0
	8.3	Functi	onal Decomposition (Phase 1) $\ldots \ldots \ldots \ldots \ldots 21$	0
	8.4	Modul	ar Decomposition	1
		8.4.1	Words, Codes and Items	2
		8.4.2	The Word-Frequency Collection	4
		8.4.3	The Word-Code Map	5
		8.4.4	The Code-Word Map	6
	8.5	Search	for Reusable Building Blocks (Phase 2)	7
		8.5.1	Words, Codes and Items	9
		8.5.2	The Word-Frequency Collection	9
		8.5.3	The Word-Code Map	3
		8.5.4	Decompression	4

Bibliography

XII

A	Арр	endix: Ada Specifications	229
	A.I	Building Block Utilities	229
	A.2	Generic Parameters	232
	A.3	Linked Collections	234
	A.4	Tabular Collections	236
	A.5	Hash Tables	238
	A.6	Lists	243
	A.7	Stacks	259
	A.8	Queues and Deques	271
	A.9	Trees	282
	A.10	Orders	294
	A.11	Key-Info Orders	304
	A.12	Sets	318
	A.13	Maps	326
	A.14	Bags	338

Chapter 1

Introduction

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		ble
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 JBM + 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 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.

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.