A Library of Generic Algorithms in Ada

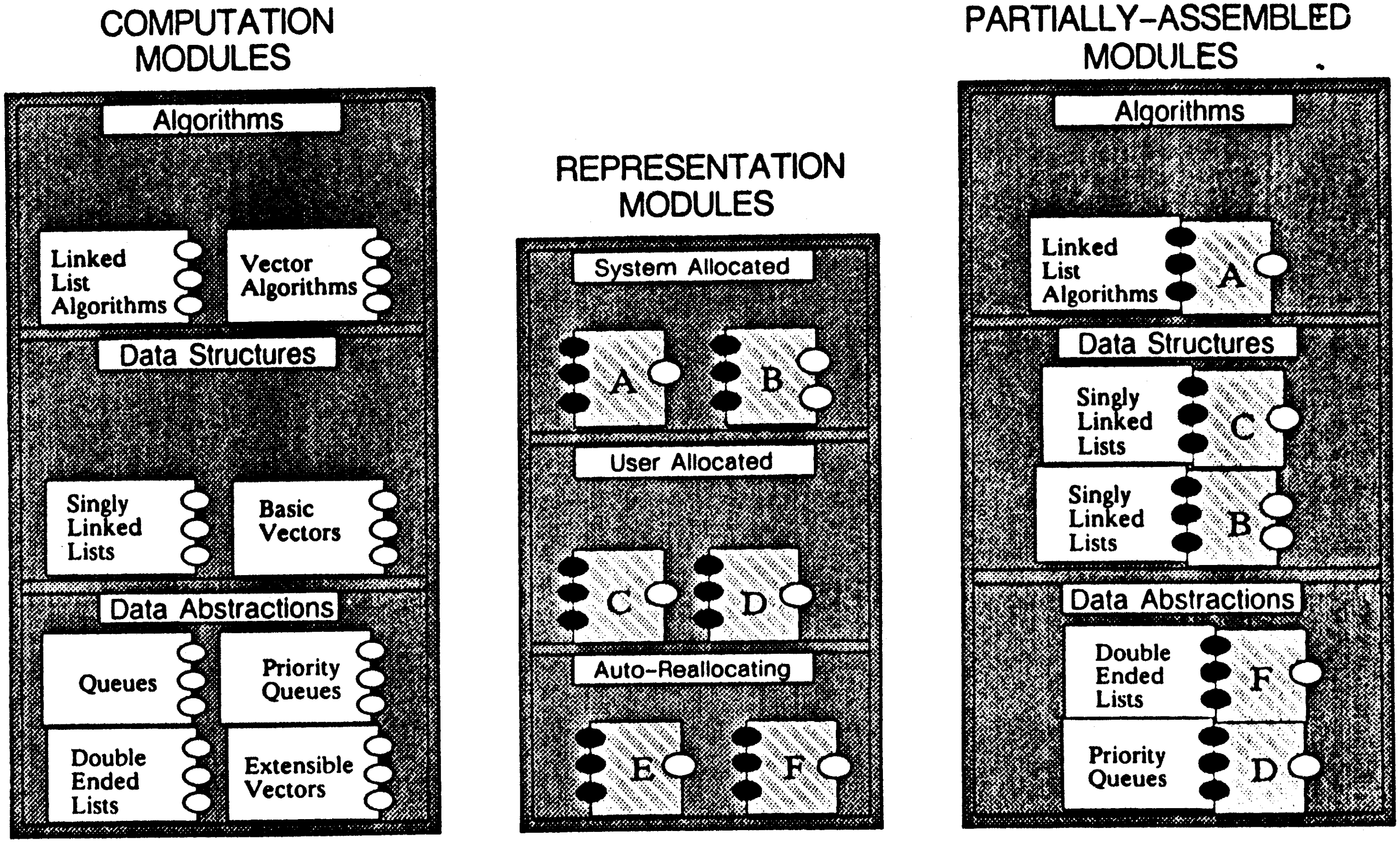
David R. Musser
GE Corporate R&D
Schenectady, NY 12301

Alexander A. Stepanov
AT&T Bell Laboratories
Liberty Corner, NJ 07060

How a Generic Library Differs from a Repository

- Repository—take existing software components, classify them, put them in as is
  - *main effort toward reusability is in proper classification for ease of retrieval*
- Generic Library—commission the creation of software components that are highly reusable
  - *main effort is in design for high quality and high degree of reusability*

OFF-THE-SHELF SOFTWARE COMPONENTS
(Generic Algorithms Approach)



Information Systems Laboratory

Key Ideas of Generic Library

- Use generic algorithms and data types to express general capabilities
  - *A generic algorithm is a template for generating an algorithm by plugging in a set of types and basic operations*
- Generate components for specific applications by instantiation
  - *Small amount of source code yields large number of useful instances*
  - *Library users can easily generate new components*
- Ensure component quality to much higher standard than by usual means
  - *Get it right once at generic level; to show correctness of an instance just show actual parameters meet their requirements*
- Provide highly detailed and cross-referenced documentation
  - *New kinds of classifications for generic components (based on abstraction mechanisms used)*

How Instantiation Works and How It Uses Ada Capabilities

- Define components generically with templates
  - *Parameterized by data type and by basic data operations*
  - *Ada* generic units *are such a template mechanism*
- Obtain specific components (Ada packages and subprograms) by plugging in specific types and operations
  - *Supported in Ada by generic instance declarations*
  - *Ada compiler expands instance declaration into regular package or subprogram*

6.5.12 Delete

Specification

Example from Current Library

```
generic
with function Test(X, Y : Element) return Boolean;
function Delete(Item : Element; S : Sequence)
      return Sequence;
```

Description Returns a sequence consisting of all the elements E of S except those for which Test(Item,E) is true. S is destroyed.

Time order nm

Space 0

where n = length(S) and m = average(time for Test)

Destructive? Yes

Shares? No

See also Delete_If, Delete_If_Not

Examples

```
declare
    function Delete_When_Divides
       is new Lists.Delete(Test => Divides);
begin
   Show_List(Delete_When_Divides(3, Iota(15)));
-- 1  2  4  5  7  8  10  11  13  14
end;
```

Implementation

```
    function Test_Aux is new Make_Test(Item, Test);
    procedure Partition_Aux
      is new Algorithms.Invert_Partition(Test_Aux);
    Temp_1, Temp_2: Sequence := Nil;
  begin
    Partition_Aux(S, Temp_1, Temp_2);
    Free_Sequence(Temp_1);
    return Invert(Temp_2);
  end Delete;
```

Implications of Generic Library Approach

- For software design:
  - *Buiding library components is software design activity*
  - *But compilable, executable designs are result*
- For library maintenance:
  - *Extensive use of standard Ada compiler environment tools*
  - *Need special library maintenance tools for keeping package specs and bodys, documentation, test suites consistent with each other*

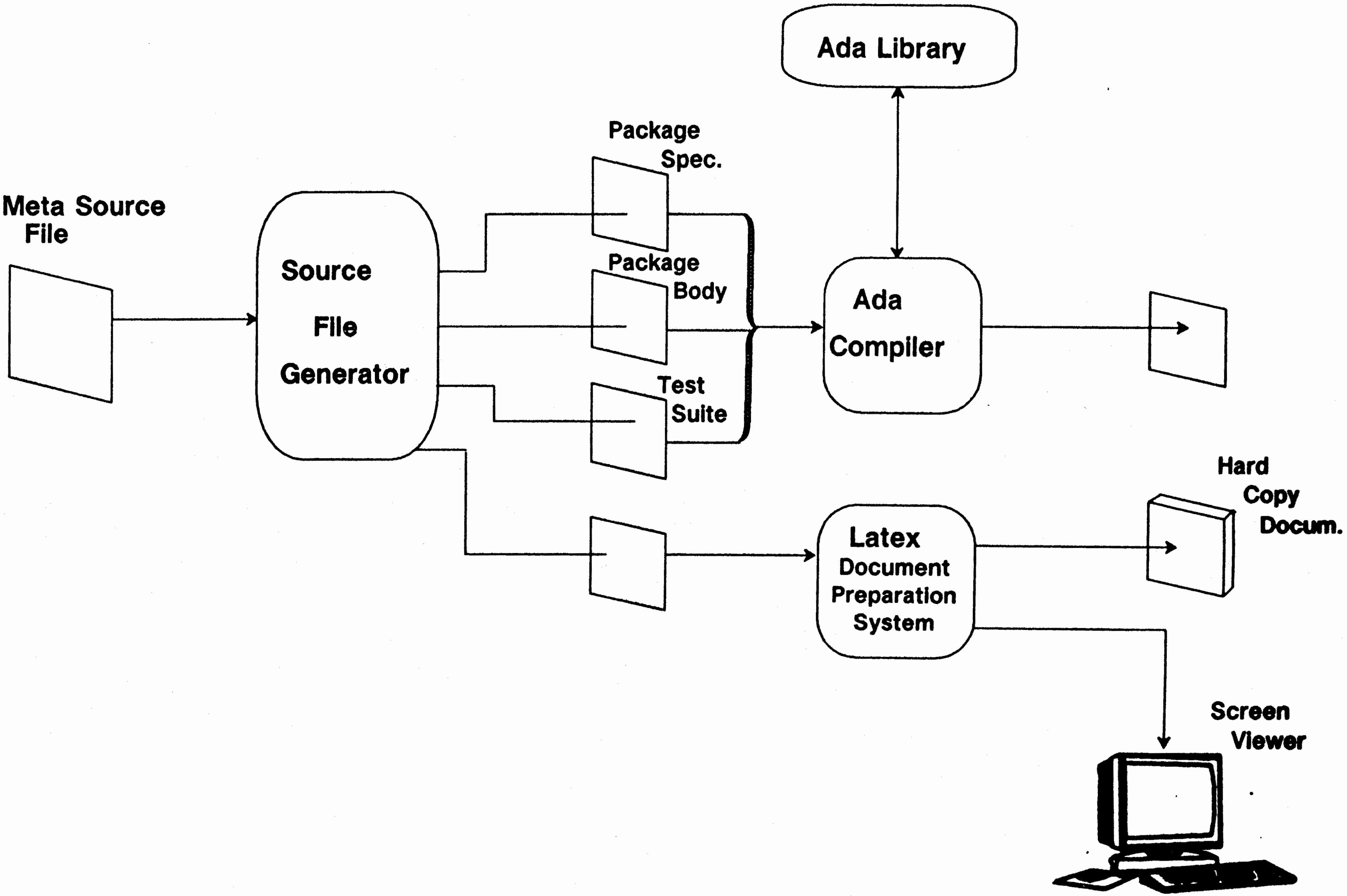
Current Status of Ada Generic Library

- Generic algorithms approach developed and refined
- Volume 1 of Linear Data Structures Packages
  - *Overview of generic library approach*
  - *Overview of linear data structures*
  - *Five packages of linked-list algorithms and data structures (114 subprograms)*
  - *Instructions for use of the packages*
- Volume 2 of Linear Data Structures Packages
  - *Three packages (double-ended lists, stacks, output-restricted deques; 62 subprograms)*
  - *Preliminary examples of generic vector operations*

Current Status of Ada Generic Library (continued)

- Preliminary version of library maintenance system
  - *Aids maintenance of source code, test suites, and documentation*
  - *Originally in Scheme on IBM PC, recently converted into Ada*

Unified Documentation / Code Approach



Data Abstractions Data types with operations defined on them	System_Allocated_Singly_Linked User_Allocated_Singly_Linked {Instantiations of representational abstractions}
Algorithmic Abstractions Families of data abstractions with common algorithms	Sequence_Algorithms Linked_List_Algorithms Vector_Algorithms
Structural Abstractions Intersections of algorithmic abstractions	Singly_Linked_Lists Doubly_Linked_Lists Vectors
Representational Abstractions Mappings from one structural abstraction to another	Double_Ended_Lists Stacks Output_Restricted_Deques

Table 1:

Classification of Abstractions and Example Ada Packages

Diagram of Classification of Abstractions

Related Work

- G. Booch, *Software Components with Ada*, Benjamin/Cummings, Inc., 1987.
- D. Kapur, D.R. Musser, and A.A. Stepanov, "Operators and Algebraic Structures," *Proceedings of Conference on Functional Programming Languages and Computer Architecture*, Portsmouth, New Hampshire, October 1981.
- D.R. Musser and A.A. Stepanov, "On Generic Programming," in preparation.
- Press, et. al. *Numerical Recipes*, Cambridge U. Press, 1987.
- A.A. Stepanov, A. Kershenbaum, and D.R. Musser, "Higher Order Programming," in preparation.

Future Directions

- Extend the library to other data structures and combinatorial algorithms
  - *rectangular data structures, tree and graph processing, string processing, embedded-system control algorithms*
- Explore relation to design stage of software development
  - *train software designers as well as programmers in generic algorithms approach*
- Explore relation to formal software specification and verification
  - *carry out formal proofs for significant library components*