
Teaching Archetypal Design with an Electronic Textbook

Last Revised, November 9, 1993

The eText Group
Department of Computer Science
California Institute of Technology
Mail Stop 256-80, Pasadena, CA 91125
adam@cs.caltech.edu

The primary author and contact for this paper is Adam Rifkin. The ideas presented in this paper were developed by the eText team at Caltech in group discussions.

A modified version of this paper will be presented at the 22nd ACM CSC Conference in Phoenix, March 6-12, 1994.

Keywords: Education, Libraries,
Multimedia, Parallel Programming, Software Engineering

Teaching Archetypal Design with an Electronic Textbook

Abstract

How can parallel programming be made tractable for students in high schools and community colleges, to programmers in four-year colleges, to commercial and government employees, to interested independent users learning on their own, and as CASE tools for professional software designers? The computer science community must address this question if the ability of programmers to harness the power of parallel systems is to maintain pace with technology advances forthcoming in parallel systems. This paper addresses some of the issues of bringing parallel programming to the people, ranging from newly developing programmers with little experience on any computer to seasoned programmers of single-processor machines.

We aim not only to enable people to use more powerful computers, but also to enable people to use computers more powerfully, by nurturing the techniques that enable them to develop efficient, correct code with relative ease. This paper briefly presents the concept of an **Archetype**, a software engineering methodology developed at the Caltech for patterns of problem solving, and for providing media for quick reference and natural software reuse. We then describe **eText**, an interactive multimedia electronic textbook that facilitates the teaching of, navigating through, and referring to Archetypes. Initial experience with Archetypes and the electronic textbook suggests that this approach to teaching parallel programming can aid computer users in the immediate future.

1 Background

Archetypes provide a general purpose design methodology which supports both sequential and parallel source code development, enabling a unification of these two computing models. This paper demonstrates that Archetypes collected in an interac-

tive electronic textbook can be used by a student first to learn parallel programming, presented as a form of algorithm development similar to sequential algorithm development, and later to consult as a reference guide for continued use of parallel programming techniques and reuse of parallel code libraries. In addition, we note that Archetypes provide a means by which not only code, but also design solutions, can be scavenged and reused.

1.1 Motivation: The Problem

In the past decade, the United States has committed many billion dollars to high performance computing. High performance architectures provide the ability to achieve performance simply unattainable with the standard sequential architectures. Computational demands have been steadily rising, also; fortunately, technology continually improves to match it within the track of Moore's Law. As a result, reasonably priced multiprocessor machines, as well as networks of workstations and/or PCs, have recently made high performance parallel and distributed computing affordable to current sequential programmers. The central dilemma arising with the rapid growth of parallel technology is the issue of training people to perform parallel programming in the very near future.

The rise of high performance computing in the 1980s and 1990s has led to an interesting paradox. Problems that were once impractically slow with sequential machines can now be solved on parallel and distributed machines. However, although high performance architectures provide extensive computing power, the source of their strength is also the source of their greatest weakness: the handicap called *parallel programming*. For many reasons, from architecture-specific quirks to the perceived difficulty of distributed debugging, parallel programming has emerged as one of the primary obstacles in reaping the benefits of high performance computing. How can the development of correct, efficient parallel code be made more tractable?

We isolate three desirable characteristics for a parallel programming learning and reference system:

- **Interactive.** A learning environment should make use of instruments such as multimedia as audio, personalized slide shows, and custom animations, and such hypermedia as document links and hypernotes, in effect teaching using the range of tools an online system affords.
- **Systematic.** A productive and methodological approach should enable a good parallel programmer to solve computing problems more efficiently.
- **Extensible.** A constructive system should be developed for adding to, modifying, and extracting from parallel libraries of source code modules that are used in the system. It should also encourage *design scavenging*, as opposed to the mere reuse of source code alone. Design scavenging enables a programmer to borrow from and modify many aspects of the algorithm's design along with source code, such as reliability and performance components.

1.2 Proposed Solution

The main thrust of our solution relies on the development, maintenance, and teaching of parallel and sequential *Archetype libraries*. **Archetypes** [CR93] impart structure to knowledge acquisition by enforcing self-similarity between many levels of abstraction. For parallel programming, Archetypes suggest a consistent solution methodology for meta-algorithms, applications, and implementations. The eText group plans to use Archetypes to provide an evolutionary approach for students to learn parallel programming, based on three tenets.

- **Familiarity.** Archetypes are at a level of abstraction above language, but in the hierarchy described later, we show how they provide libraries and extensions of languages *currently* used by programmers — languages such as C, Pascal, C++ and FORTRAN. Beginners should be able to move gradually from their present programming styles to parallel programming.
- **Ubiquity.** Parallel programs should run on platforms that are the most widely used — namely, workstation networks and PC networks — *in addition to su-*

percomputers. Again, Archetypes emphasize an evolutionary path that requires neither massive expenditures nor massive retraining to begin experimenting with and benefiting from parallel computing.

- **Interactivity.** Archetypes employ tools available on the PC, including multimedia tools and program development utilities, to develop an environment on the PC that encourages people to experiment with parallel computing. This drives knowledge acquisition past passive and even active education towards the realm of interactive learning.

The electronic textbook (abbreviated eText [KR93]) is an interactive multimedia manuscript with a multidimensional navigation system that executes on PCs and is designed to help people learn about parallel programming via Archetypes. Through the use of eText, a programmer can:

- **Navigate** – by browsing through a library of paradigm Archetypes,
- **Learn** – by selecting and understanding an Archetype through the extensive documentation and provided teaching facilities,
- **Experience** – by looking at several applications that use the Archetype, emphasizing how the algorithm for each application was developed from the Archetype in a systematic fashion, and
- **Implement** – by observing sequential and parallel code for each application in C, Pascal, Fortran and C++, again emphasizing the systematic development of the code from the Archetype. The programmer can also
- **Scavenge** – source codes, problem-solving designs, correctness proofs, and performance analysis techniques, by using and modifying components from the electronic textbook's repository.

The three-tiered hierarchy (as discussed in §2.2 and illustrated in figure 3) — many Archetypes, many applications for a given Archetype, and many implementations

for a given application — is designed to help students learn and experiment with parallel programming. Moreover, each level of the hierarchy maintains the same basic structure, affording an intuitive feel for ease of use. Programmers can also browse the text in other ways; for example they can look at the given application and identify different Archetypes used to solve that class of problems (e.g., Sorting). They can also browse using keywords. eText is interactive, uses animation to help explain parallel algorithms, and uses voice narration to allow programmers to focus on animations while listening to an explanation. Since the book is being deployed on Unix-based PCs and workstations, the programs can be executed directly on the same system.

We briefly elaborate on the new ideas that make parallel Archetype libraries and the eText project unique in their approach. Sequential template programming methodologies [VK89], multimedia presentations [BD92, Shn92, vW93], electronic textbooks and hypertext systems [Nel87, BD91, Rad93], and algorithm methodologies [CM89, MS91, vdS93] have been explored previously; the following ideas are original issues being addressed:

- **Archetypes.** These provide a medium for the transfer of parallel processing technology, and the encapsulation of the knowledge held by experienced parallel programmers. In addition, Archetypes allow for design scavenging, which provides a system for reuse at a higher level than mere source code reuse.
- **Programming Style.** Most people program operationally. Some theoretically-oriented people program assertionally. We seek to bridge the gap between these two groups, relating assertions to operations, not only as a method of informally proving program correctness but also as a debugging technique. Archetypes incorporate both correctness verification outlines, and a history of archetypal bugs and errors to aid testing and debugging. Archetypes are a means to this hybrid programming style.
- **Parallel Program Libraries.** These extensive application source code examples. allow for cross referencing and querying.

- **Interactively Teaching Issues Specific to Parallelism.** These issues include, but are not limited to: granularity, parallel structuring, process mapping orthogonal to a given problem structure, relations between machines and programs, and data flow issues.

1.3 Justification

Many computer science curricula are incorporating a variety of classes to teach parallel computing [Mil93]. Our approach to teaching parallel programming — to be used to supplement the Computer Algorithms class at Caltech this fall — is unique from the approaches offered elsewhere in four main aspects:

- **The use of Archetypes.** Central to our instruction is the belief that the systematic approach to problem solving afforded by Archetypes [CR93] encourages more efficient code development. The resulting program designs are more rapidly implemented, and amass properties that can be used to prove correctness of implementation [Cha93b].
- **The presentation of both parallel and sequential algorithm development together.** Many good textbook introductions to parallel programming have been recently published [J92, Les93], but they rarely present algorithms in a manner that unifies the parallel and sequential mind sets. A notable exception is compositional programming [CT92], and this presentation of material has been used to teach a freshman class at Caltech. Perhaps the unified teaching of parallel and sequential issues can ultimately lead to better programmers [Rif93].
- **The use of many programming notations.** In the last ten years a wealth of parallel languages have been developed [Che93]. eText is not bound to any single language, offering the opportunity to learn parallel programming using whichever languages are most comfortable for the student. Furthermore, authoring tools provided by eText allow different language developers to add

source code in their preferred notations to the Archetype libraries easily. To afford a more comfortable transition for programmers to the universe of parallel computation, eText includes popular sequential languages with simple, small extensions and/or libraries. In this manner, users can progress at a reasonable, individualized pace.

- **The use of an electronic textbook.** Special issues must be addressed when dealing with such a medium. Richard Bergman claims [BG93], “The driving force has to be the content, and not the media... Electronic books have two main advantages: ease of access with a search engine and the fact that you can have a collection of books on a subject in one spot for easy cross-reference.” We opted for the electronic book metaphor not only for ease of navigation and access to a potentially exhaustive information content, but also for the interactive simulations and lessons, and for the ability to maintain parallel Archetype libraries *online* for ease of cut-and-paste, modification, compilation and distributed debugging of altered code, all in a run-time environment provided *with* the electronic book.

The primary driving force behind the electronic book is its **content**, specifically Archetypes; in *Applications, Implications* [Ada93], John Adam maintains that multimedia systems are affordable by schools, government agencies, and businesses, but that “content, not technology, will determine success.” We also keep in mind the words of David Guenette [BG93]: “What electronic books need is what all books need: to be edited well.” To this end, Archetypes undergo a rigorous revision and amendment process, allowing them to dynamically and continually improve over time.

The remainder of this paper is organized as follows. Archetypes are described and discussed in §2. The interactive electronic textbook is explained, and its relation to teaching Archetypes is detailed, in §3. We follow with a few examples of Archetypes being implemented using the electronic textbook in §4. Finally, we conclude with a small evaluation of this work thus far in §5.

2 Archetypes

Archetypes represent a systematic, taxonomic and patterned way of grouping similar problems and their respective solutions. Their usefulness resides in the facility for an able programmer to instantiate an Archetype, for a new problem that arises with clues indicating that it follows the same pattern as other applications for that Archetype.

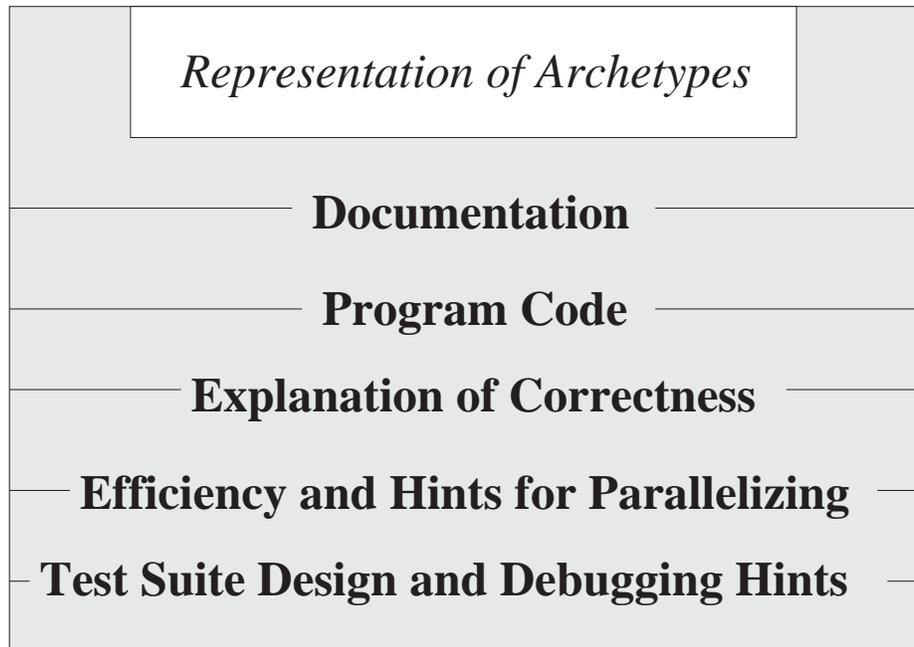


Figure 1: *The components included in the representation of Archetypes.*

An Archetype consists primarily of five components: documentation, program outline, semantic correctness proof outline, efficiency abstraction with performance modeling outline, and test suite design and debugging tips outline. These outlines are further discussed in §2.3, after which we describe why Archetypes are useful and how we make use of them in §2.4 and §2.5, respectively. But first, we describe why Archetypes are needed and our approach to designing and maintaining them.

2.1 How Archetypes Improve Skill

In determining a good way to teach parallel programming, we isolate operational issues necessary to develop proficient users. Archetypes afforded a unique learning environment because of their systematic approach to presenting algorithm design. Briefly, the parallel programmer skill levels and the issues associated with each are:

- **Beginner to Intermediate.** A learning environment should include many source code examples, so students can learn by observing working code and modifying it. Furthermore, examining the code can reveal that parallel and sequential programming conform to similar principles. Finally, when the student tries to write custom code, he or she will be able to construct a solution by analogy to the library code.
- **Intermediate to Advanced.** A unified and methodical approach should be taught concurrently, during these learning sessions, so students could learn to design code, develop test suites, and perform correctness proofs, efficiency analysis, and performance evaluation.
- **Advanced to Expert.** A system should be designed that encourages software reuse. Such a system has been explored for sequential algorithms [VK89], but none to date has been developed for parallel algorithms. Archetypes lend themselves nicely to many software engineering principles including software reuse, leading to a paradigm bridging between sequential and parallel algorithms previously unachieved.

We believe Archetypes represent a manner by which programmers can progress.

Further, by being useful to parallel programmers at every level, Archetypes bridge the gap between different skills of programmers, providing a means by which all programmers can learn and refer to principles at many levels.



Figure 2: *The ability levels of a parallel programmer.*

2.2 The Design of Archetypes

The main feature of our approach to the design of Archetypes is **self-similarity**; at each level of the three-tiered hierarchy, the component components are the same. Sequential and parallel design issues are addressed at each level. The three tiers, as illustrated in figure 3, are:

1. **The Archetypes Level.** At this, the highest level of problem abstraction, we isolate *general patterns* of problem solving for which many algorithms behave. These algorithms may fall under the realm of computer science and/or applied natural sciences. One Archetype, for example, is the Divide-and-Conquer Archetype, described below. Other examples include Matrix Computations, Mesh Computations, Spectral Methods, and n-Body Methods.
2. **The Applications Level.** At this, the middle level of problem solving, we isolate *specific applications* whose algorithms conform to the parent Archetype. For example, the Divide-and-Conquer Archetype would have many applications (that include documentation, pseudo code, correctness proofs, parallelizing techniques, efficiency analysis, debugging tips, and test suites), such as Mergesort, Skyline, String Matching, Nearest Neighbors, and Fast Fourier Transform.
3. **The Programs Level.** At this, the lowest level of problem solving, actual working *source code* implementations which vary by programming language, granularity, and target machine architecture considerations. For example, the Fast Fourier Transform Application might have Programs (that include documentation, correctness proofs, parallelizing techniques, efficiency analysis, debugging tips and test suites) written in sequential C, Pascal, FORTRAN, and

C++, as well as parallel versions of those languages.

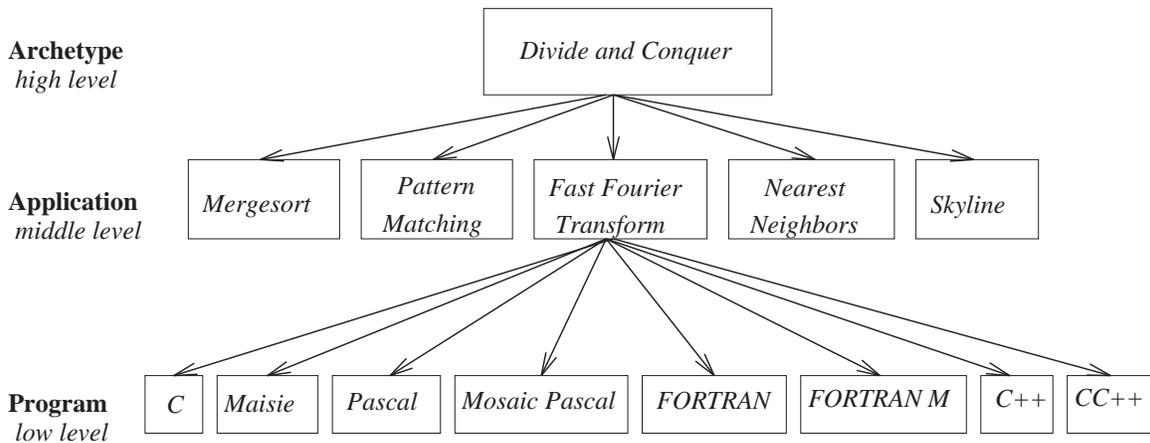


Figure 3: *The three-tiered hierarchy of Archetypes, applications, and programs, instantiated for the Divide-and-Conquer Archetypes example.*

For example, we have developed a *Divide-and-Conquer* Archetype, which represents a methodological approach to solving problems that can be classified under a pattern of “dividing” a problem into similar, smaller subproblems, solving the subproblems (“conquering”), and then merging the solved subproblems to combine a solution for the original problem. This Archetype (as do all) includes documentation, program outline in pseudo-code, correctness obligations, efficiency and performance evaluation components, and test suite design issues, for both sequential and parallel algorithms.

Now, one application of this Archetype is the *Fast Fourier Transform* Application, useful in many scientific and signal processing programs. This application (as do all) *maintains the same structure as its parent Archetype*, including documentation, program outline in pseudo-code, correctness obligations, efficiency and performance evaluation components, and test suite design issues, for both sequential and parallel algorithm development.

Further, this application owns a number of *Fast Fourier Transform* Program Case-books, developed as **Archetype libraries**. These libraries bind together anthologies

of code, easily accessible for browsing and modification. For instance, the *Fast Fourier Transform* Application owns source code examples in sequential notations such as C, Pascal, FORTRAN, and C++, and in parallel notations such as a parallel C called Maisie, a parallel Pascal called Mosaic Pascal, a parallel Fortran called FORTRAN M, and a parallel C++ called Compositional C++. It also owns source code examples in sequential notations with channel libraries added to obtain parallelism. Looking at any one of these instances, for example the *Fast Fourier Transform Pascal Program*, we notice that this program (as do all) *maintains the same structure as its parent application (and Archetype, for that matter)*, including documentation, program in actual Pascal code, correctness proof presentation, efficiency and performance evaluation analysis, and small but thorough test suites, for the sequential algorithm developed. Students can peruse the source code, compare parallel and sequential algorithms for the same programming notation, and discover the issues of parallel programming by probing the code provided and modifying it in a special environment. The navigational engine, text browsers, and runtime environment form the crux of eText, as discussed in [KR93]. The whole concept of Archetypes is fully explained in [CR93].

2.3 The Components Archetypes Include

Here we succinctly and informally describe the characteristic components contained within an Archetype (and application, and program); for a full description of these components, please refer to [CR93]. eText provides a rich navigational interface to Archetypes, as exemplified by figure 4. Each component consists of a number of elements, outlined below.

- **Documentation.** Documentation includes all accompanying explanatory text, to discuss nuances of the given Archetype and the subtleties of the given approach. Typically, documentation will include, but is not limited to, an analogical introduction to the methodology (see figure 4), clues describing how to recognize the problem pattern, an annotated description of the Archetype, and

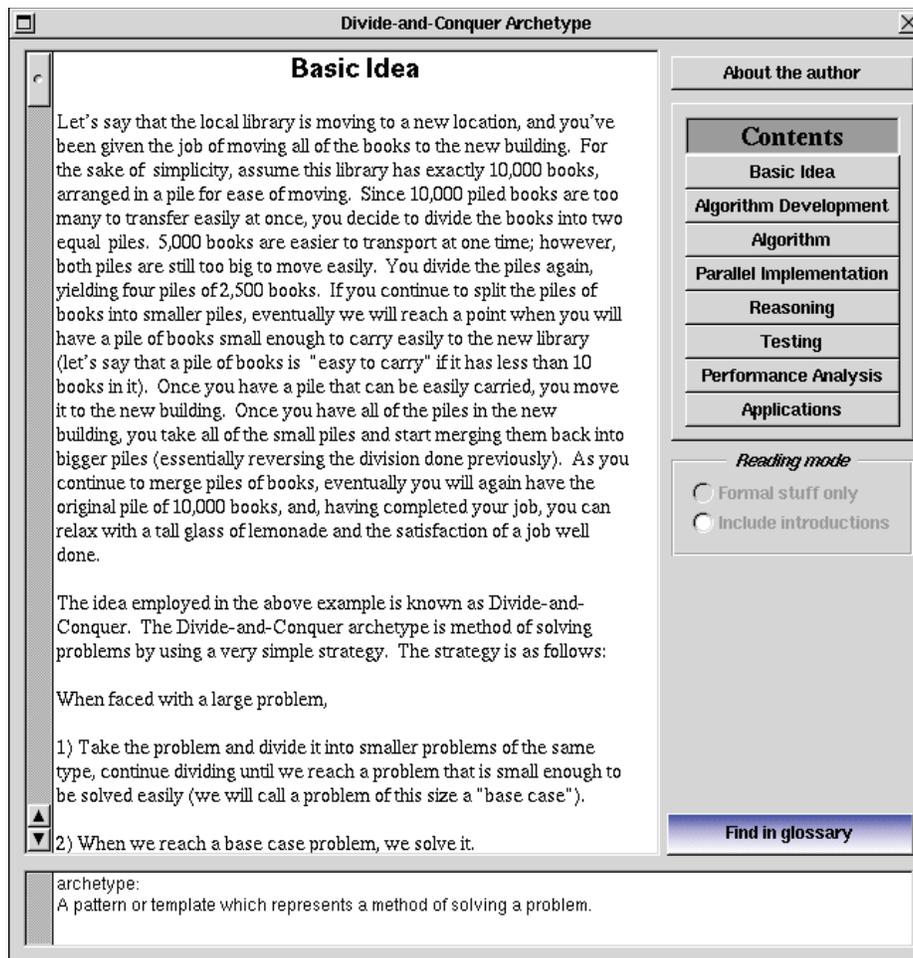


Figure 4: *The eText interface for the Divide and Conquer Archetype.*

a “further reference” reading list, which will eventually contain hyperlinks to other online documents.

- **Algorithm.** In this component, lessons describe the creative steps necessary for sequential and parallel algorithm development for problems its parent Archetype. When specified for an Archetype or application, this section will often include pseudo-code; instantiated for a program, this section provides actual code in the given notation. Also, a methodology for parallelizing a given sequential algorithm is given, including a stepwise refinement process which

enables performance tuning of parallel implementations.

- **Correctness Abstraction.** For this component, the following verification issues are confronted and discussed: the general algorithm outlines from the Algorithm component are critiqued, assertional debugging techniques to consider for this algorithm are described, and a systematic proof outline is presented. Again, for some applications, and for all programs, these are instantiated for the specific problem being solved in the given notation. The systematic correctness outline for an Archetype or application may include proof obligations for user-supplied components, safety and progress considerations for distributed algorithms, invariants, and termination verification strategies.
- **Efficiency Abstraction.** This component covers the essentials of sequential and parallel performance evaluation. This could include: a sequential efficiency analysis overview; parallel and distributed issues such as granularity, mapping, and communication; a comparison of task-parallel and data-parallel approaches; a look at control flow and data flow considerations; and a comparison of the sequential and parallel algorithms prescribed. For some applications, and for all programs, these are instantiated for the specific problem solved in the given notation. Performance analysis is always included with implementations, sometimes included with applications, and rarely included with the Archetypes themselves.
- **Debugging Tips/ Test Suite Design.** Strategies for developing a thorough yet sufficiently small test suite are discussed in this component. For some applications, and for all programs, these are instantiated for the specific problem solved in the given notation. In addition, techniques for debugging canonical errors are presented; in this manner we capture programmer experience.

As implemented using the interactive learning environment enabled by eText, Archetypes also include a number of teaching abstractions. These include interactive figures, slide shows, menus, process diagrams, animations, and laboratory exercises. Further,

eText provides a facility for navigation and reference to the Archetype libraries of instantiated code, for direct use or modification.

2.4 Why Archetypes Are Useful

We consider the benefits derived from the use of Archetypes, as a justification of their utility.

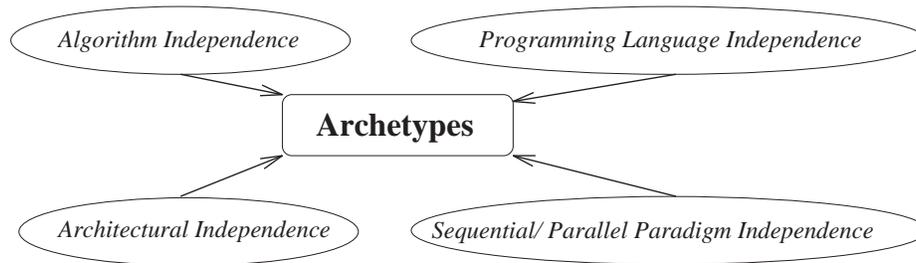


Figure 5: *The independences that Archetypes foster.*

- **Software Engineering.** Archetypes espouse a principled approach to problem solving, supporting the software engineering ideals of abstraction, specification, instantiation, design, and reuse. Because the same structure exists at each level (Archetypes, applications, and programs), there is reuse *at every level*.
- **Independence.** Archetypes can set programmers free. As illustrated in figure 5, Archetypes represent an abstraction subsuming any particular algorithm, an abstraction above any particular machine architecture, an abstraction above any particular programming language, and an abstraction above any particular paradigm. This represents a unique level of independence of problem solving.
- **Paradigm Unification.** Archetypes depict an unconventional breaking of the artificial wall created to separate sequential and parallel programming styles. Through their continued use, programmers discover that algorithm development

follows similar conventions under either paradigm. Archetypes also integrate object-oriented methodologies by presenting them in an intuitive manner.

- **Learning Tool.** Archetypes provide a coherent, cohesive approach to problem solving. This approach can be thought of as a helpful learning aid, with guides on problem solving and careful considerations of subtle issues.

In short, Archetypes not only teach people how to program; they teach people how to program efficiently, correctly, and easily. They provide instant access to a wealth of reference information, and espouse reuse and other software engineering principles.

2.5 How Archetypes Are Used

Presently a number of prototype Archetypes have been developed at Caltech as a demonstration of their feasibility and utility. These examples (as discussed in §4) use the eText interface, described in the next section.

3 The Electronic Textbook

We briefly describe the interactive learning and referencing environment called eText. For an exhaustive investigation of the issues involved in the design and use of eText, please refer to [KR93].

3.1 Design and Implementation of eText

The electronic book is designed to facilitate the authoring and usage of Archetypes, incorporating the following features:

- **Interactive Learning Environment.** eText serves as a teacher, guiding the student through the documentation and various components of a given Archetype. The use of multimedia, hypermedia, and interactive media (see

§3.2) provides a variety of learning aids. Perhaps the most useful tool eText provides is the customizable interactive figure, as illustrated by the Mergesort example in figure 6. Authors can design special simulations and animations to supplement the traditional components of Archetypes. eText also provides a facility that allows the author to develop different “guided tours” through the information space, so that a reader can have a different teaching companion (with different accompanying levels of instructing text) based on his or her level of expertise or goals.

- **Navigational Engine.** Once a student has learned how to use a particular Archetype, he or she will need a suitable referencing environment that allows for speed of navigation and location of information to help improve his or her programming skills. eText provides hypermedia features such as multidimensional indexing, a table of contents, links from parts of one document to other parts of the same document or parts of other documents, and automatic cross referencing for ease of travel through the Archetypes’ hierarchy.
- **Annotations.** All items in the hierarchy (Archetypes, applications, and programs) are considered documents with various attributes that can be modified to better suit the author or the reader. Further, documents are self-aware, in the sense that they know their relationships to other documents, and the eText engine. Printing capabilities allow users to make hardcopies of relevant material for portability.
- **Runtime Environment.** To encourage students to play with provided code libraries, eText provides a code browser with cut-and-paste capabilities. Students can use an editing environment to slice and house relevant portions of code, which can then be modified, compiled, debugged, executed and evaluated in an integrated environment. This provides students with immediate hands-on experience, breaking down the often-intimidating barriers that accompany one’s initial reaction to parallel programming as to where to even *begin*.

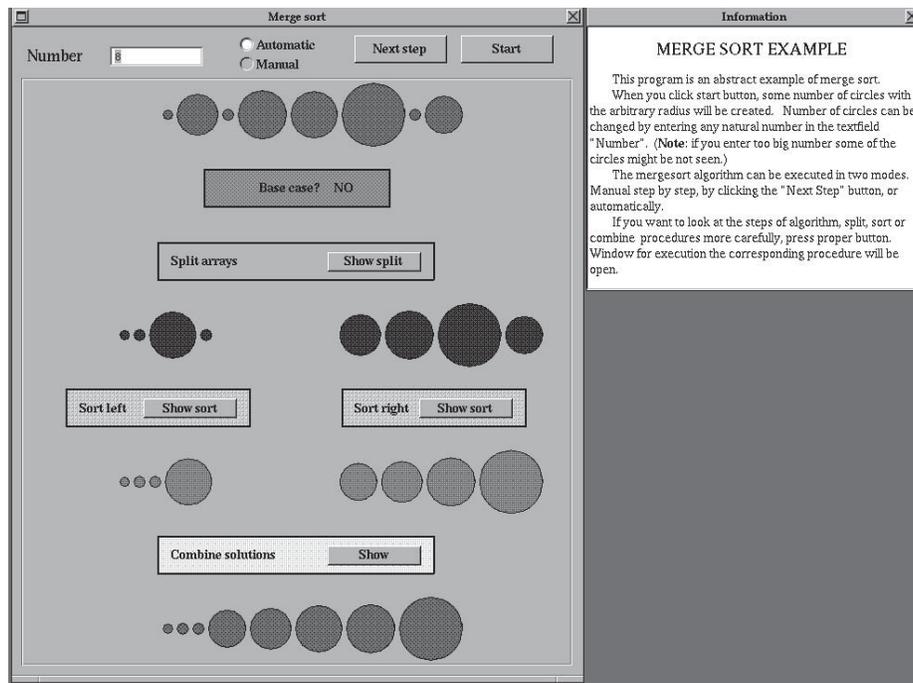


Figure 6: An interactive figure incorporated into eText, that demonstrates how Merge-sort works by using colored circles to animate the Divide-and-Conquer algorithm.

3.2 The Use of Interactive Media in eText

The electronic textbook utilizes a variety of media to intensify its teaching and reference capabilities. A medium can either be an *annotation* to a document (such as text, audio, or pictures), or a *navigation* through a document (such as footnotes, hyperlinks, or indices). As shown in figure 7, eText has its foundations built upon the pyramid of currently available state-of-the-art features. These are briefly discussed below.

- **Conventional Media.** “Traditional” media encompasses devices commonly associated with printed books and papers. eText provides facilities for standard text, diagrams, charts, and static pictures. Furthermore, it affords canonical conventions for navigation, including: tables of contents, chapter and section divisions, indices, footnotes, glossaries, and bibliographies.

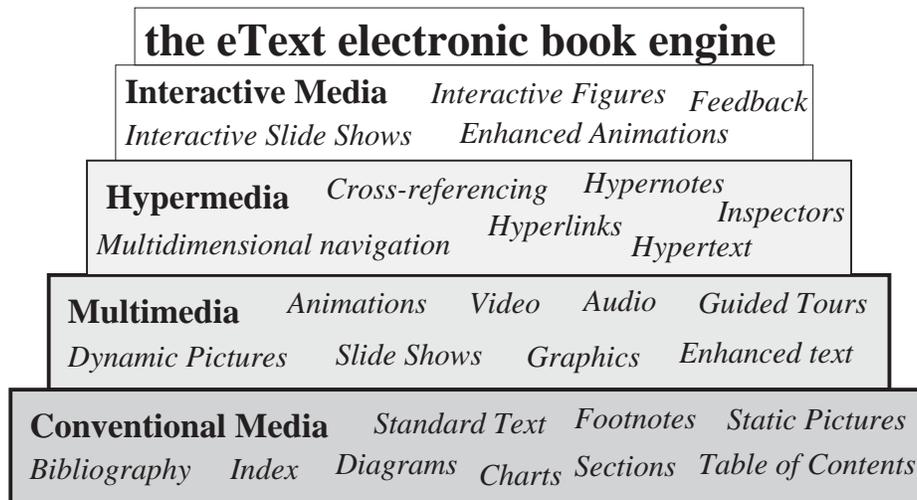


Figure 7: *The pyramid of media upon which eText is constructed.*

- **Multimedia.** Because eText is an online system, it can provide several kinds of live, time-based annotations, such as audio (voice, sound effects, and music), branching slide shows, computer animations, and video. In addition, eText provides for enhanced text format by allowing a variety of fonts, colors, and formatting options. The multimedia features are designed to be as easy to use as to the aforementioned annotations of conventional media, but do not interfere with the presentation flow because the use of such functionality is under user control.
- **Hypermedia.** Hyperfeatures allow a user to click on an annotation and jump to elsewhere in the document, or elsewhere in another document. They provide eText with rich navigation facilities for cross-referencing, inspecting, and hyperlinking (as illustrated in figure 8). A user can click on a special hypernote or hypergraphic left by an author, or utilize the hypertext buttons to look up words in glossaries or elsewhere in other documents. Hypermedia make provisions for true multidimensional navigation, meaning that eText can provide a thick book of information disguised as a thin book of information. Any ideas that need further clarification can be expanded by clicking the proper button;

annotations can be collapsed or traveled similarly. The multimedia features are designed to be as easy to use as to the navigations of conventional media listed previously, and do not interfere with the presentation flow because the use of such functionality is under user control.

- **Interactive Media.** Through the use of interactive figures, interactive slide shows, and enhanced animations, eText provides a feedback facility that allows the electronic book to modify its lessons based on the perceived learning level and desires of the user.

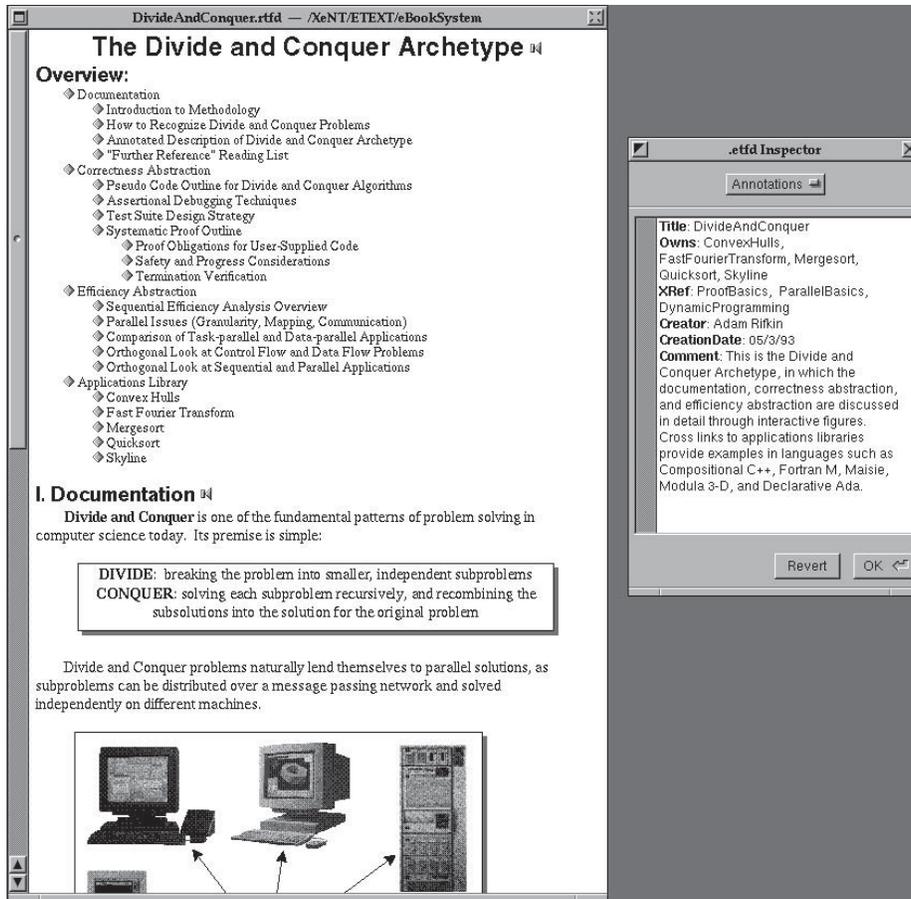


Figure 8: A group of hyperlinks within the eText navigational engine table of contents for the Divide and Conquer Archetype.

3.3 Using eText to Teach Archetypes

eText wraps the extensive features described in §3.1 and §3.2 in a intuitive graphical user interface. It provides modes for authoring documents, modifying created documents, and reading existing documents. For authoring (and printing), to borrow from the parlance of desktop publishing, eText is WYSIWYG. The reader has the full range of navigation and inspection features described earlier, as depicted in figure 8. For example, if the user chose the documentation for the Divide-and-Conquer Archetype, he or she would arrive at the window illustrated in figure 4, after which a guided tour would take him or her through the Archetype's various components. We have developed a number of Archetypes; these instantiations are summarized in the next section.

4 Case Study: Examples

We concisely explore the currently developing Archetypes that demonstrate not only the feasibility of their usage, but also the utility they offer.

4.1 Introduction to Programming

Although not a formal Archetype, the Introduction to Programming chapter provides an appropriate beginning for readers initially learning programming principles. It discusses basic programming ideas, assertional thinking about programs, Hoare triples, and preconditions and postconditions, as presented in [vdS93]. It continues with some mathematical fundamentals, and then proceeds with programming constructs, including assignment statements, sequenced statements, conditional statements, and looping statements. The intricacies of modular design, recursion, data structures, data abstraction, encapsulation, and object-oriented programming styles are described in a programming language-independent manner. Also, specific programming notations are discussed, providing examples of usage. Lessons in parallel programming styles

are presented concurrently with the lessons for sequential programming styles, making use of the concepts of spawned processes, multiple threads of control, and synchronization and communication through channels. The Introduction to Programming chapter is an example of how to use eText for learning and reference, and provides the foundation for the Archetypes.

4.2 Computer Science Archetypes

Presently, the Dynamic Programming (as illustrated in figure 9) and Divide-and-Conquer Archetypes [AK93] are being completed for use with Caltech's Computer Algorithms class this fall. Both provide illustrations of the niceties supplied by Archetypes. Future plans may include Archetypes for the following methodologies: Greedy Algorithms, Hill Climbing Algorithms, Iterative Methods, Numerical Methods, Branch-and-Bound Algorithms, Graph Algorithms, Discrete-Event Simulations, and Monte Carlo Simulations.

Later goals include Archetypes for a number of distributed algorithms, including Global Snapshots and reactive systems such as the Dining Philosophers' Problem, as discussed in [CM89].

4.3 Other Archetype Examples

Presently the eText group at Caltech is working on Archetypes for scientific programming, under the realms of Matrix Computations, Grid Computations, Spectral Methods, and n-Body Problems. It is conceivable, and we hope that, other groups will develop Archetypes for applied computational systems as diverse as financial algorithms and genetic algorithms. New Archetypes, applications and programs are continually being added to the eText electronic textbook. Archetypes are limited only by our imagination, time, and budget constraints. We expect the number of Archetypes, applications, and programs in eText to balloon as the number of groups contributing to the project blossoms in the near future.

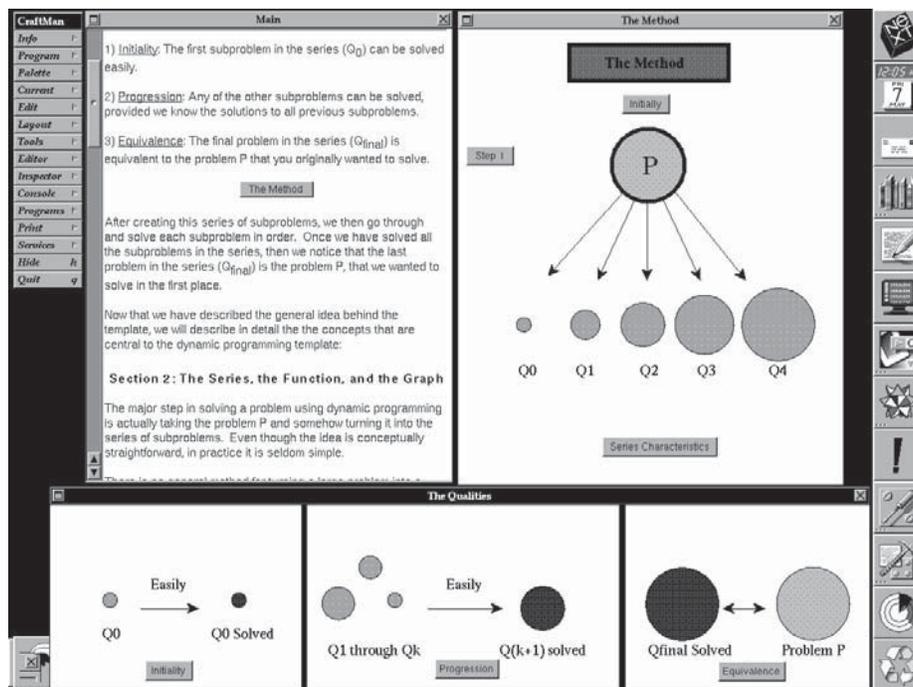


Figure 9: A lesson designed in eText for the Dynamic Programming Archetype.

5 Summary

The efforts discussed in this paper, and the ensuing testing that accompanies them, indicate that Archetypes will influence the way students can learn to parallel program in the future. It is our belief that a large number of application programs (greater than 80%) can be accommodated by a compact set of Archetypes (less than 20), making them useful and feasible for teaching parallel programming and for the technology transfer of parallel programming. On the learning front, Archetypes allow students to learn parallel programming by associating it with the analogous skills involved with sequential programming. And, since they abstract design solutions, Archetypes provide reuse *at the design level* as well as at the source code level; as a result, they represent a knowledge-acquisition system that encapsulates the experience of seasoned parallel programmers for use by developing parallel programmers.

In addition, Archetypes integrate parallel and sequential models of thinking, providing a systematic methodology for patterns of problem solving. Archetypes provide components for documentation, algorithm and code, correctness verification, efficiency and performance analysis, and test suite design. Through the interactive environment founded on eText's electronic textbook, Archetypes (and the applications and programs that accompany them) can be used for learning and reference. The Archetypal libraries of source code provide a wealth of implemented applications for browsing and using. Evidence from early experiences with Archetypes and the electronic textbook indicate they will instill important software engineering practices in the people who use them. As a final note, we observe that Archetypes and eText are, and will continue to be, organic; they are expected to continue to grow and flourish.

Acknowledgments

Software Archetypes were first conceived by K. Mani Chandy, who has been instrumental in refining them, coordinating and inspiring the eText group, and editing this manuscript. Also, special thanks go to Svetlana Kryukova, Paul Ainsworth, and Siddhartha Agarwal, who developed the Divide-and-Conquer and Dynamic Programming Archetypes; to Rohit Khare, architect of the eText electronic book publishing system; to Rajit Manohar, who worked on the Grid Computation Archetype; Adam Rifkin and John Thornley, who were instrumental in adding and removing (respectively) a number of quirky colloquialisms and in offering helpful comments to improve this document; to We are also grateful to the other members of the eText project team at Caltech: Alan Blaine, Greg Davis, Diana Finley, Diane Goodfellow, Paul Kim, Tal Lancaster, and Ted Turócy. The following people have been helpful consultants throughout the course of the eText project: Ulla Binau, JoAnn Boyd, Peter Carlin, James Cook, Robert Harley, Carl Kesselman, Rustan Leino, Berna Massingill, Paul Sivilotti, and Gail Stowers. We thank Cindy Ferrini and Nancy Zachariasen for their patience while this document was being written over a number of months.

This paper reflects a larger overall effort [Cha93a], led by K. Mani Chandy at Caltech,

to develop methods and tools to aid in the software engineering of parallel programs, for a variety of natural science, mathematics, and computer science applications. The methods deal with the systematic development of parallel programs from specifications — and in many cases, the specification is a sequential program which is required to be “parallelized.” The tools support reasoning about parallel programs, then debugging them on workstations, and finally porting the source code from workstations to parallel machines. The methodology has been used for such applications as fluid dynamics computations. eText, Archetypes, and PEN play central roles in the overall effort. The research on libraries of Archetypes was sponsored by ARPA under contract N00014-91-J-4014, and this support dovetails with CRPC support for education and parallel scientific applications, under cooperative agreement CCR-9120008. The government has certain rights in this material.

Demonstrations are available on request through the eText Group at the Department of Computer Science at Caltech. Presently two electronic textbook user interfaces have been implemented: one is running on the applications layer of NEXTSTEP, and the other executes on XMosaic, a layer over the Internet World Wide Web. Future plans include ports to PCs and/or Macs. A modified version of this paper will be presented at the Computer Science Conference of the ACM, to be held in Phoenix, Arizona, March 6-12, 1994.

References

- [Ada93] John A. Adam. Applications, implications. *IEEE Spectrum*, pages 24–31, March 1993.
- [AK93] Paul Ainsworth and Svetlana Kryukova. A multimedia interactive environment using program archetypes: Divide and conquer. *submitted to SIG CSE 94*, 1993.
- [BD91] Emily Berk and Joseph Devlin, editors. *Hypertext/ Hypermedia Handbook*. McGraw Hill, 1991.

- [BD92] Meera M. Blattner and Roger B. Dannenberg, editors. *Multimedia Interface Design*. ACM Press, 1992.
- [BG93] Richard Bergman and David Guenette. What is the state of electronic books? *New Media*, page 12, May 1993.
- [Cha93a] K. Mani Chandy. Archetypes and the systematic development of parallel programs. Technical Report forthcoming, Center for Research on Parallel Computing, California Institute of Technology, 1993.
- [Cha93b] K. Mani Chandy. Properties of concurrent programs. Technical Report Caltech-CS-TR-93-24, Computer Science Department, California Institute of Technology, 1993.
- [Che93] Doreen Y. Cheng. A survey of parallel programming languages and tools. Technical Report RND-93-005, NASA Ames Research Center, Mail Stop 258-6, Moffett Field, CA 94035 -1000, March 1993.
- [CM89] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison Wesley, May 1989.
- [CR93] K. Mani Chandy and Adam Rifkin. An archetype-based approach to parallel program libraries. *technical report forthcoming*, 1993.
- [CT92] K. Mani Chandy and Stephen Taylor. *An Introduction to Parallel Programming*. Jones and Bartlett, 1992.
- [J92] Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison Wesley, 1992.
- [KR93] Rohit Khare and Adam Rifkin. etext: A document-centric hypermedia publishing environment. *submitted to USENIX '94*, 1993.
- [Les93] Bruce P. Lester. *The Art of Parallel Programming*. Prentice Hall, 1993.

- [Mil93] Russ Miller. The status of parallel processing education: 1993. Technical Report available through anonymous ftp, Computer Science Department, State University of New York at Buffalo, August 1993.
- [MS91] Bernard M.E. Moret and Henry D. Shapiro. *Algorithms from P to NP - Volume I: Design and Efficiency*. The Benjamin/ Cummings Publishing Company, 1991.
- [Nel87] Ted Nelson. *Computer Lib/ Dream Machines*. Microsoft Press, 1987.
- [Rad93] Roy Rada. *Hypertext: From Text to Expertext*. McGraw Hill, 1993.
- [Rif93] Adam Rifkin. Teaching parallel programming and software engineering concepts to high school students. Technical Report CRPC-93-4, Center for Research on Parallel Computation, California Institute of Technology, 1993.
- [Shn92] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison Wesley, 2nd edition, 1992.
- [vdS93] Jan L.A. van de Snepscheut. *What Computing Is All About*. Springer-Verlag, 1993.
- [VK89] Dennis M. Volpano and Richard B. Kieburtz. The templates approach to software reuse. In *Software Reusability, Volume I: Concepts and Models*, pages 247–255. ACM Press, 1989.
- [vW93] Mark von Wodtke. *Mind Over Media: Creative Thinking Skills for Electronic Media*. McGraw Hill, 1993.