Concurrent Program Archetypes

K. Mani Chandy
Computer Science, 256-80
California Institute of Technology
Pasadena, CA 91125. mani@cs.caltech.edu

Abstract

A program archetype is (a) a program design strategy appropriate for a restricted class of problems, and (b) a collection of program designs with (c) implementations of examplar problems in one or more programming languages and optimized for a collection of target machines. The program design strategy includes archetype-specific *information about methods of deriving a program from a specification, methods of parallelizing sequential programs, the program structure, methods of reasoning about correctness and performance, empirical data on performance measurements and tuning for different kinds of machines, and suggestions for test suites. This paper reports on a project at Caltech which is exploring the question: Can a library of parallel program archetypes be used to reduce the effort required to produce correct efficient programs?*

1 Introduction

1.1 Programmer Productivity

The "software crisis," now a household phrase, is particularly severe in concurrent computing for several reasons such as those listed below.

Rate of Change of Parallel Computing Parallel computers have become widely used in industry and government over a short span of time, and so there is pressure to develop a large number of parallel applications over a short period of time. This telescoping of the time available for parallel program development makes productivity of parallel programming even more important than productivity of sequential programming.

Correctness Developing correct parallel programs is even more difficult than developing correct sequential programs because programmers may have to deal with nondeterminacy and multiple threads of execution. Nondeterminacy and concurrent threads are difficult to deal with both for formal reasoning and for debugging. The issue of productivity of parallel programs is exacerbated by the difficulty of reasoning about parallel programs.

Performance Optimizing a program to obtain adequate performance is simpler for a sequential program than for a parallel program. The diversity of parallel architectures and the ranges of performance parameters (such as message latency) make optimization of parallel programs difficulty and machine-specific. Since parallel programs are developed because they execute faster than sequential programs, efficiency is critical. The need for efficiency of program execution is another reason that achieving programmer productivity is more difficult for parallel programming than for sequential programming.

Absence of Standards There is no standard in hardware. Shared-memory multiprocessors, networks of workstations, and data-parallel machines compete for the attention of applications developers. Though standards for software (such as HPF) are developing, they have been slow in coming. The development of CASE tools proceeds more rapidly when there are a small number of industry standards. The inadequacy and wide diversity of tools hinders productivity.

1.2 Reuse and Program Libraries

One approach to improving programmer productivity is reuse of programs. Though program libraries help a great deal, they are inadequate for dealing with the difficulties of parallel programming for several reasons including the following.

Language-Specific A program library is written in one language using one runtime system. Given the range of architectures, and the large number of sequential applications that need to be ported to parallel machines, a library in one language and using one runtime system is inadequate. There is insufficient time to develop libraries in all the parallel programming languages and runtime systems that are available.

Risk of Commitment Many applications developers are (quite properly) conservative, and do not want to commit to a significant amount of effort to develop a parallel application if their effort is wasted because some machines, languages or runtime systems are no longer supported. The risk of wasted effort is much lower in application development for sequential machines than for parallel machines, especially since *de facto* industry standards are emerging such as C, C++, Fortran and object interfaces such as OLE, OpenDoc, and CORBA. Commitment to a single parallel library, language and runtime system is a risk.

Compositionality and Mapping Objects to Architectures In sequential programming, there is only one form of composition: sequential composition. Furthermore, all the data of a sequential program is mapped on a single address space. So, all we need to be concerned about in designing a sequential library is whether its routines can be called in some sequence, and whether routines deal with the data structures used in the program.

In parallel programming, we have to be concerned with sequential composition and parallel composition: We may want procedures in the library to cooperate in a concurrent execution. This can be difficult because each parallel procedure may expect a specific distribution of its data across the memory of the machine. The difficulty of composing procedures in a library makes libraries more difficult to use in parallel programming than sequential programming.

2 Parallel Program Archetypes

An archetype is (a) a method of problem-solving, (b) a program design strategy associated with the method, (c) a collection of examplar problems to which the method can be applied, and a collection of reference implementations in different languages and runtime systems and on different architectures of each examplar problem. The program design strategy associated with an archetype has several parts including:

1. The software architecture of a class of programs, where the software architecture includes the control and data-flow structure of the program.
2. Methods of developing parallel and sequential applications from specifications. In many cases, the de facto specification is an existing sequential program that has to be parallelized; so, associated with an archetype are methods for parallelizing sequential code.
3. Frameworks for reasoning about correctness and performance of programs that are instances of the archetype.
4. Suggestions for test suites and debugging based on the software architecture of the archetype and from experience of others who have developed similar applications.
5. Suggestions for performance tuning on different target architectures. These suggestions are based on performance models as well as experience and measurement of performance of applications executing on different machines.
6. Detailed documentation of the archetype, and suggestions for tailoring the archetype documentation to get documentation specific for an application.
7. A description of other archetypes used by this archetype.

The Hypothesis This work is predicated on the following hypothesis: a library of parallel program archetypes can be developed, and it can reduce the effort required to develop correct efficient parallel programs using languages, runtime systems and machines that the user chooses.

Experiments conducted over several years will tell whether the hypothesis has merit.

3 Education and Hypermedia Documentation on the Net

Documentation and Educational Material Documentation of a sequential program library can take the form of a user's manual that specifies what each procedure (or class) does. By contrast, documentation for a parallel archetype library is more complex and has a significant educational component. For instance, explaining the performance consequences of different data distributions is primarily a pedagogical activity. We should think of documentation of parallel archetype libraries in a much broader sense than we think of documentation of sequential program libraries. Indeed, *documentation of an archetype library should be viewed as a collection of course material.*

Course Material and Multimedia Courses are interactive: instructors and students questioning and responding to each other. If we view documentation of archetype libraries as course material, then this documentation must be interactive as well. The obvious solution is to use hypermedia authoring tools to develop archetype documentation. Animation and voice are particularly helpful in explaining performance issues using simulations.

Course Material on the Net To enable wide use of archetype libraries, its documentation must be made available on the Internet. A first step towards hypermedia course materials on the net is the development of HTML documents available through World Wide Web. In the mean time, we can evaluate hypermedia documentation of archetype libraries using tools on Macintosh, NEXTSTEP, Windows and other platforms.

Summary The utility of an archetype library depends on the quality of its documentation. Therefore, we should use the best technologies available to provide documentation in the most appropriate media, and we should develop tools to support browsing and scavenging.

4 The Archetype Library Project

4.1 Overview

How to get it The archetype library project at Caltech has four main parts: The development of archetypes for (a) scientific applications, (b) Combinatorics and optimization, (c) reactive systems, and the development of technology to support documentation of archetype libraries. All the archetypes are part of an electronic document called *etext* which can be obtained on World Wide Web under URL `http://www.etext.caltech.edu`. The NEXTSTEP version, which includes more interactive multimedia material can be obtained by sending a message to `kryukova@cco.caltech.edu`. (The NEXSTEP version requires the NEXTSTEP operating system, and it also requires purchase of the *Craftman* tool which is the electronic document authoring system that we used.) The project is ongoing, and a tremendous amount of work remains to be done.

One way to describe the use of archetypes is to step through a few examples. Next, we discuss two small examples, one from combinatorics and the other from scientific computation.

Archetype Support for Software Design Before discussing the examples, I want to emphasise that archetype libraries do not mechanically produce code from specifications — archetype libraries support program development methods, and as in all design methods, intelligence and creativity are required.

Finding out whether a problem fits an archetype exactly, or approximately, is not automatic, though the navigation tools of etext will help. If a problem does not appear to fit any of the archetypes in the library, then either the library cannot be used for this problem, or the problem has to be partitioned into subproblems, some of which may match an archetype. The archetype library does not provide support for the process of stepwise refinement unless the specification fits an archetype. In many cases, *a significant amount of design work may have to be carried out before an archetype can be used.*

4.2 A Combinatorics Example

The Problem You are faced with the following (well known) problem from computational geometry. Given a set of points in a rectangle on the plane. Each point is specified by its x and y coordinates. The set contains at least two points. Develop a program to determine the distance between the nearest points.

You would like to develop a program in C++ and you would like to explore the use of C++ with a message-passing library such as MPI, and you would also like to try out a C++-extension parallel language such as CC++ (which was developed at Caltech by Carl Kesselman and myself). You are particularly interested in implementations on (a) networks of small numbers of workstations connected by common ethernet, and (b) the Intel Paragon.

Does the Archetype fit the Problem? The first step is to determine if the divide and conquer archetype fits the given problem. Depending on your familiarity with the divide and conquer method of problem solving, you will (a) decide that for yourself without using the documentation, or (b) read the abbreviated description, or (c) study the detailed description. If you are using WWW (World Wide Web), you will get into etext, and then click on the divide and conquer archetype full version (written by Svetlana Kryukova) or the short version. The full version has a detailed description of the divide and conquer strategy and its parallel implementation. I have hopes that high-school students will develop some parallel programs on a variety of machines, and the detailed documentation is at a level that can be followed by a high-school student.

The structure of a divide and conquer program is as follows. Associated with each problem is its size. Assume that size is a natural number. Problems that are small enough that they can be solved by some (usually trivial) method known to the programmer are called *base-case problems*. If the problem is a base-case problem, then it is solved using a procedure, and let's call this procedure the *base-case solution*. If the given problem is larger than all base-case problems, the given problem is partitioned into a list of problems, each of size strictly smaller than the given problem. The solutions to each of the smaller problems are merged to obtain a solution to the given problem.

```
procedure divide_and_conquer(in problem P, out soln S)
begin
   if base_case(P) then
     {* base case *}
     find_base_case_solution(in P; out S)
   else begin
     {* larger than base case *}
     split(in P; out P1, P2, ..., Pn, Other);
     for i from 1 to n do
          divide_and_conquer(in Pi, out Si);
     merge(in S1, S2, ..., Sn, Other, out S);
   end
end;
```

The archetype specifies what the programmer should provide: (a) the base-case test, (b) the base-case solution, (c) the procedure for splitting non-base-case problems into smaller subproblems, and (d) the procedure for merging subproblem solutions. More importantly, the archetype also describes parallel implementations of divide and conquer programs.

Our first step is to determine if we can obtain the procedures required by the archetype for the given nearest-neighbors problem. Having determined that we can do so, we next turn to parallel implementations.

Parallel Structures Click the parallel implementation section of the text. This section describes different approaches to developing parallel algorithms based on the divide-and-conquer archetype.

The section points out that one approach is to first split non-basecase problems and then do all the subproblems in parallel, and then merge the results. The subproblems are solved concurrently with each other, but the subproblems are not solved concurrently with the split and merge. This approach is simple because the concurrently executing programs do not share data with each other.

The section points out an alternate approach which is to execute *concurrently* split, the procedures that compute the solutions to the subproblems, and the merge. In this case, split shares data with the subproblem solutions; in particular, split outputs the subproblems that become the input to the subproblem-solution procedures. We must adopt some protocol to share data between concurrently executing processes. Since the data shared between concurrent processes in this structure is written by at most one process, and we know *a priori* which processes read the data, the section suggests using either message-passing or single-assignment data structures. *The value added by archetypes is such detailed design information about parallel process and communication structure which is* **specific to the archetype**.

Performance Issues The section on performance describes methods for determining process and message granularity, and for mapping processes to processors. If a non-basecase problem is small enough, it is more efficient to solve the problem sequentially rather than in parallel, because the overhead of communication and process spawning exceeds the time saved by parallel execution. For networks of workstations, with ratios of communication speeds to computation speeds that are low, it is more efficient to solve relatively large problems sequentially; by contrast, for parallel machines with fast switches, it is more efficient to solve problems in parallel until the problem size gets to be much smaller. A simple performance model suggests appropriate design choices for different target architectures.

Example Applications Our next step is to look at example applications. Click on the `mergesort` example to see how the divide and conquer archetype is used for mergesort. Look at the programs in different languages that have been implemented for mergesort. Then look at other applications.

Inspecting a collection of applications, all of which use the same archetype, is helpful in understanding how to tailor an archetype to a specific problem.

Language Issues We can postpone deciding on a language and runtime library as much as we wish, because the issues we considered earlier, such as program structure, reasoning about correctness and performance, are largely independent of language and runtime library. Having decided to use C++, we explore implementations using C++ for some of the problems, starting with `mergesort`.

We first explore a C++ language extension called CC++. Clicking on CC++ within `mergesort` brings you to a CC++ program. The documentation gives you the C++ skeleton structure, and it tells you exactly what parts you have to fill, and then it provides a detailed example of the parts to be filled in for `mergesort`. We see that your obligations to tailor this archetype to a specific problem are *identical* whether the target language is CC++, or C++ and PVM. Furthermore, your obligation is to develop sequential code; the archetype shields you from having to deal with parallelism.

Debugging The next step is to debug your program. One would expect that programs developed so carefully would not have bugs. But if they did, the section on test suites and debugging suggests common bugs for this archetype. For instance, if the program does not terminate, a common bug is that the split program is incorrect because one of the subproblems is not smaller than the original problem. Again, the value of the archetype is archetype-specific debugging information.

A particularly useful aspect of archetypes are assertions provided with the program structure. When you fill in the program skeleton, you also fill in an assertional skeleton. Thus as you tailor an archetype to obtain a specific program, you also tailor its assertional structure to obtain assertions specific to the program. Assertions (for example in C) are necessary for reasoning and are very helpful in debugging.

Execution Your next step is to execute the program on some target machines. The performance tuning information, and suggestions for collecting performance data will help you tune your program to get satisfactory efficiency. The archetype library includes performance data of divide and conquer examples executing on your target machines (such as networks of workstations and the Intel Paragon), and this data and the methods of collecting the data described in the archetype will be helpful to you in instrumenting your application and in understanding its performance.

Documentation The documentation you need to write is limited to the procedures that you add. The basic documentation for divide and conquer is provided in great detail, and so all you need to do is to describe the sequential procedures that you added. Our idea here is not to provide a documentation template, but rather to split the documentation into two parts: documentation of the archetype (and this already exists in the template library) and linked to this is your problem-specific documentation.

Summary In summary, archetypes are designed to support you every step of the way from concept to implementation on parallel machines. The advantage of archetypes is that it capitalizes experience and insight in developing parallel programs for a class of related problems.

4.3 A Scientific Application

Data Parallel Archetypes Divide and conquer is an example of a task-parallel archetype; tasks are created and deleted dynamically. Many scientific applications are data parallel, and there are important problems which have both data parallel and task parallel components. Next, I discuss a simple data-parallel archetype: mesh computations.

The Problem You are asked to parallelize a Fortran77 program that has been developed over a period of years. The program is a computational fluid dynamics application. You want to execute your program on parallel machines to get faster turnaround. You would like to explore executing your program on networks of workstations and the IBM SPx.

A difference between this problem and the previous one is that the specification of your problem is an existing sequential program. You want a parallel implementation of your program that produces exactly the same results as the sequential program (assuming that the floating point units are identical).

Steps towards a Parallel Implementation Your sequential program has a mesh structure. The simulation consists of computing the values of variables at each point of an n-dimensional mesh. Each point of the mesh is computed in terms of values at neighboring points.

You click on the mesh archetype (written by Berna Massingill). The mesh archetype description has an overview of mesh computations, and a detailed description of parallel implementations. The mesh archetype describes the essential data-mappings and associated procedures to obtain parallel implementations. The parallel program structure is as follows. The mesh is partitioned into regions. Neighboring regions share a boundary. The values for each region and its associated boundary are computed, and then neighbors exchange the values of boundaries that they share.

A parallel mesh computation has sequential components and specifically parallel components. The sequential components compute new values for each region given old values for the region (including its boundaries) The operations that deal with parallelism are: IO to read input data into proceses and to output data from regions, and procedures to exchange boundaries. The parallel archetype is structured so that you can focus your attention on the problem-specific sequential components while using archetype-specific parallel procedures from the library.

Next, you could explore a specific parallel application in the library such as Dan Meiron's CFD application that simulates flow in a toroid. The point is not the similarity (or lack of it) between Dan's application and yours; the point is to study how an actual application is developed in a straightforward systematic manner from the archetype. Applications in the library are written to emphasise the similarity between the application and the template. In particular, applications are structured so that procedures that deal with parallelism are obtained from the archetype, and problem-specific procedures are sequential.

Dan's CFD application is written in Fortran M, a parallel extension of Fortran developed by Ian Foster at Argonne and myself. Key features of Fortran M are that it supports completely dynamic task and communication structures, while the compiler can verify that the program is deterministic (i.e., all runs of the program with the same inputs produce the same output.) If you want to develop a Fortran M application and execute it on an IBM SP, you can incorporate sequential Fortran 77 code derived from your sequential application into the communication and IO routines provided by the archetype.

Performance measurements given in the archetype, especially graphs of execution time versus the number of processors, will be particularly helpful to you as you tune your application. A simple performance model is provided to help you port your application to machines for which the archetype does not provide empirical evidence.

You may also choose to develop your application in Fortran 77 and PVM. In this case too, the structure of the archetype hides language-specific and runtime-specific details. So, most of the time you take in developing your parallel application will not be wasted whether you use Fortran M, or Fortran and PVM, or if you switch from one to the other.

Parallelizing Sequential Programs A component of the mesh archetype is not in the divide and conquer archetype: this component (written by Berna Massingill) describes a detailed systematic sequence of small steps to convert a sequential mesh computation into a parallel mesh computation. The value added by the archetype is that the sequence of steps associated with the mesh archetype is designed specifically to support mesh applications; different sequences of steps may be adopted for other kinds of applications.

A sequence of steps is carried out in the sequential domain; each of these steps produces a sequential program (say in Fortran 77) executing in a sequential environment familiar to the programmer. The intent is to do as much of the parallelizing work in an environment familiar to the programmer. Familiar sequential environments seem particularly important for debugging.

The final transformation converts a (transformed) sequential program into a parallel program. This conversion is mechanical, and does not introduce errors. Thus programmers can develop parallel mesh applications in a semi-mechanical way.

4.4 Archetype Hierarchies

One archetype can use another, and our archetype library will have a partial-ordering of archetypes, based on the uses relation. One archetype using another is more complex than one procedure calling another. When one archetype X uses another archetype Y, X must be able to use all aspects of Y including Y's software architecture, methods of reasoning about correctness and performance, methods of parallelization, and even test suites and documentation.

The reason for the existence of archetypes is reuse. Therefore, though archetype composition is complex, it is essential that archetypes in the library use other archetypes effectively. Next, we give a few examples of dependence between archetypes.

Multigrid Berna Massingill has begun work on a multigrid archetype with help from Dan Meiron. The multigrid archetype uses the mesh archetype. In essence the multigrid archetype consists of the mesh archetype for each grid, and has new computation and communication structures for converting between finer and coarser grids. Though the multigrid archetype is not complete yet, it does appear that the multigrid archetype can truly use all aspects of the mesh archetype.

Neighbor-Exchange The mesh archetype uses a simpler SPMD mesh computation-communication archetype in which processes are structured in an n-dimensional mesh, and processes communicate in an SPMD fashion to all neighbors in the mesh. Let us call this simpler archetype the mesh neighbor-exchange archetype. The neighbor-exchange archetype deals with IO and communication with neighbors; it does not specify what data is associated with each process. By contrast, the mesh archetype requires that the data structure be a grid, and is explictly designed to deal with ghost boundaries.

The neighbor-exchange archetype is a special case of the collective communication on a process mesh archetype, which in turn is a special case of the collective communication archetype.

In summary, a great deal of thought has to go into structuring a library of archetypes. The complexity of this task is more than that of structuring a library of programs because one archetype uses much more than the code structure of another archetype.

5 Reactive Distributed Systems

I will discuss, very briefly, the group's ideas on archetypes for reactive systems. Our primary focus is message-passing systems, and our target implementation platforms are primarily C (or C++) and Unix, or Ada. Archetypes for scientific computation and combinatorics are based on the computation, communication and proof structure of classes of programs. By contrast, the archetypes we are working on for reactive systems are closer to classes that deal with specific problems such as mutual exclusion, multiprocess synchronization, distributed checkpointing, distributed clocks, collective communication operations such as atomic broadcasts, and so on.

We plan to explore archetypes for commercial applications on the National Information Infrastructure. The hypothesis here is that commerce on the NII can be categorised into a relatively small number of groups, each of which have similar applications. If this hypothesis is correct, then we can have some hope that archetype-based design will help the large numbers of businesses, particularly small businesses, that operate part of their businesses on the net.

A specific concern for reactive systems is reuse of methods of reasoning about programs, and incorporating formal reasoning with software engineering methods such as testing. This concern is raised by the subtleties of reasoning about the correctness of reactive systems.

6 Education

The archetypes project deals as much with education as with library design. A good archetype library is an educational tool for students as well as a reference for practising programmers. A project at Caltech, called the *etext* project deals with using hypermedia and networking technology to support education based on archetypes.

One of the problems we had in this project was dealing with a multiplicity of target audiences. We have hopes that etext and archetypes can be used by high school students to develop parallel applications on networks of workstations and scientific supercomputers. We also want college students in the sciences and engineering to use it as a textbook. Finally, we want programmers who are beginning to use parallel machines, to use the document as a tool to help them obtain parallel implementations. Our hope is that the electronic medium will allow each type of user to employ the document in a manner suitable for that user. This turns out to be something of a problem depending on the technology used.

We have learned that the success of the archetypes project depends on whether the documentation can be used for education and for reference in a very simple way.

7 Archetypes: A Critical Evaluation of an Ongoing Project

The archetypes project is risky. We have no conclusive evidence that an archetype library can be constructed effectively, or that it will serve its purpose of simplifying parallel programming and improving productivity. The project can fail for several reasons, a few of which are listed below.

1. Most applications may be so diverse that there may be very little similarity between different ap-

plications. The archetype project is predicated on the assumption that solution methods can be categorized into groups of similar methods.

We also expect to be able to "scavenge" code, ideas and documentation from an archetype if an archetype is similar to an application, but does not exactly fit the application. Scavenging takes material from an archetype and then modifies it to fit the problem. At this point, we do not have evidence that scavenging from archetypes is effective.

2. An archetype may be so general, that the amount of effort required to tailor the application may be as much as the time required to develop the application from scratch.

3. The amount of effort required to develop an archetype library, even for a few reference implementations (say C, Fortran with MPI and threads package, and a couple of programming languages) and reference architectures (say networks of workstations, single address space multiprocessors, and distributed address space supercomputers) may be so large that we can only develop a small number of archetypes.

4. We have limited experience with structuring archetypes so that archetypes use one another. Though we are hopeful that we can develop methods and theories of archetype composition, we have not done so yet. Success of the effort will depend on the ease with which archetypes can be composed.

5. The structure and ease of use of the electronic documentation is critical to the success of archetype-based design. We may not do this crucial part of the project well, even though we have a lot of very useful code; and if we don't do the documentation effectively, we will not succeed.

8 Previous Work

The idea of a parallel program archetype library draws upon a great deal of earlier work in software engineering, particularly dealing with code reuse. Related projects, with different emphases are underway at the University of Tennessee with Jack Dongarra and others on templates. Tony Skjellum and his group at Mississippi State are also developing template libraries with similar goals.

Mary Shaw at CMU has been working on software architectures with much the same goals. Doug Smith at Kestrel Institute has developed a collection of design tools for automated software engineering; the tools are called KIDS (Kestrel Interactive Development System). The goals of the projects are similar, though our emphasis is primarily on concurrency, whereas KIDS seems to have focused attention on sequential programming. There are several collective communication libraries, on top of NX for instance, and though some of the goals for communication libraries are similar to ours, the emphases are very different.

An Invitation

If you would like to explore the possibility of perhaps participating in the archetypes project either by giving suggestions or by developing some archetypes yourself, we would be delighted to have your cooperation. Please send a message to Dan Meiron, `dim@ama.caltech.edu`, or to me.

Acknowledgements

The Archetype Library Project at Caltech has four parts.

Scientific Applications One project is to build parallel archetype libraries for scientific computation in general, and more particularly for simulation of physical systems. This project is part of the NSF Center for Research in Parallel Computing. The project is directed by Prof. Dan Meiron of the Applied Mathematics Department at Caltech and myself. The project has contributions from three postdocs: Michael Holst, Ron Henderson and Yu-chung Chang. A PhD student, Berna Massingill, is completing her dissertation on parallel archetypes for scientific computation. In addition, several superb undergraduates from Caltech and from the CRPC Women's Summer Program provided valuable advice and did a great deal of work.

Combinatoric Applications A few graduate students and several undergraduates are working on a parallel archetype library for combinatorics and optimization. Svetlana Kryukova has begun a thesis on archetype design, and she is developing archetypes for divide and conquer, tree search including alpha-beta search and branch and bound.

Formal Methods and Reactive Systems The Air Force Office of Scientific Research funds a project to develop a theory of archetype-based design. This project develops formal methods for both correctness and performance analysis to suppport archetype-based design. The project also investigates the use of archetypes in the design of distributed reactive systems and scientific problems relevant to the Air Force. Adam Rifkin, a doctoral student, is developing performance models for the archetypes. Berna Massingill has used archetypes for developing electromagnetic applications.

The Electronic Textbook The documentation for the archetype library is part of the electronic textbook (etext) project. The electronic text has chapters from Svetlana Kryukova,and Paul Ainsworth, and papers from several students. Adam Rifkin, Svetlana Kryukova and Rohit Khare played central roles in designing the structure of the electronic text.