Sather Provides Nonproprietary Access to Object-Oriented Programming

Stephen M. Omohundro

Sather supports strong typing, garbage collection, object-oriented dispatch, multiple inheritance, and parameterized types

Modern scientific computing is placing ever more stringent requirements on the authors of scientific code, and on the languages and programming environments that they use. At one time, most scientists were happy with systems that could efficiently execute relatively simple repetitive operations on fixed-sized arrays. The tools appropriate for that task are woefully inadequate for the challenges posed by the computational sophistication of recent algorithms for scientific computing.

Such challenges take many forms. For example, modern finite-element code must adaptively change the structure of the underlying grid, dynamically adding and deleting nodes during a computation. Complex geometry algorithms are required for efficient manipulation and visualization of geometric structures. Modern nonlinear optimization techniques utilize many interacting software components. More and more models use complex graph-based structures such as neural networks for some portion of the representation. These tasks are far more like general-purpose computing than traditional scientific computing. The appropriate software tools are correspondingly more like the tools developed for general-purpose tasks.

Object-oriented programming has been one of the most notable recent developments in modern programming languages. Unfortunately, it is often associated with reduced computational efficiency, and so has not seemed appropriate for scientific computing. This article describes a new language, "Sather," which was designed to retain the powerful benefits of object-oriented programming without sacrificing efficiency.[1,2,3]

Sather was derived from the language Eiffel,[4,5] and is designed to be very efficient and simple, while supporting object-oriented dispatch, strong typing, multiple inheritance, parameterized types, garbage collection, and a clean syntax. Each of these aspects will be described in turn in this article. Sather was initially developed to meet the needs of research projects at the International Computer Science Institute (ICSI), which required a simple, efficient, nonproprietary, object-oriented language. ICSI is involved in several areas that require the construction of complex software for computationally intensive tasks. Examples include a general-purpose connectionist simulator, a high-level vision system based on complex geometric data structures, the support software for a high-speed parallel computing engine for speech recognition, and CAD tools for integrated-circuit design.

We investigated several existing languages, including C++,[6] Objective C,[7] Eiffel,[4,5] Self,[8] Smalltalk,[9] and CLOS.[10] Only C++ was efficient enough for our needs, but its overall complexity and lack of garbage collection, object type tags (needed for persistency and object distribution), and parameterized types, led us to begin our work with Eiffel. Our experience with Eiffel allowed us to identify the features that were essential for our purposes. We required a nonproprietary compiler, however, to serve as a base for

Stephen M. Omohundro is a research scientist at The International Computer Science Institute, 1947 Center St., Suite 600, Berkeley, CA 94704; e-mail: om@icsi.berkeley.edu.

developing a parallel object-oriented language to run on new experimental parallel hardware.

Sather was developed to incorporate the features of Eiffel that were essential to us as well as others aimed at enhancing efficiency and simplicity. The initial Sather compiler was written in Sather over the summer of 1990. The compiler generates portable C code and links easily with existing C code. In June 1991, ICSI made the language publicly available by anonymous FTP over the Internet (from "ftp.icsi.berkeley.edu" in the United States, "ftp.gmd.de" in Europe, "lynx.csis.dit.csiro.au" in Australia, and "sra.co.jp" in Japan). The release includes documentation, a compiler, a symbolic debugger, a GNU emacs programming environment, and several hundred library classes. Within a few weeks of the release, several hundred research groups from around the world had obtained copies. Since that time new class development has become an international cooperative effort.

Why Object-Oriented Computing?

The primary desired benefit from object-oriented languages is the ability to effectively reuse code. In Sather, one writes a program as a collection of modules called "classes." Each class should encapsulate a well-defined abstraction. If these abstractions are chosen carefully, they can be used in a variety of situations. For example, the Sather library has a vector class that encapsulates the common operations on vectors (e.g., addition, dot product, tensor product, etc.). An obvious benefit is that less code needs to be written for an application if it can use classes that have already been written. An even more important benefit is that the resulting code is often better written, more reliable, and easier to debug. This is because programmers are willing to put more care and thought into writing and debugging code that will be used in many projects. In a good object-oriented environment, programming should feel like gluing together preexisting building blocks to perform new functions. In such a setting one can be confident that most bugs will lie in the 10% or so of newly-written code, and not in the 90% of the code made up by well-tested library classes. This can simplify the debugging process tremendously, and lead to far greater reliability.

It is important to understand the benefits that object-oriented programming gives over traditional subroutine libraries, which also aim to support reuse. A subroutine library makes it easy for newly-written code to make calls on old code, but does not make it easy to get old code to call new code. For example, one might write a visualization package that displays data on a certain kind of display by making calls on display interface routines. At some later time, one wishes to have the package display its output on a new kind of display. Without an object-oriented approach there is no easy way to get the previously written visualization routines to make calls to the new display interface.

Another example might be a system that makes use of a matrix to hold information. One might later decide to use a sparse representation for the matrix, and want to avoid having to rewrite all the code that operates on the matrix. In each of these examples one might want the code sometimes to operate on one type of display or matrix, and sometimes on the other. In fact, the actual kind of display or matrix may not be known until the calls are made.

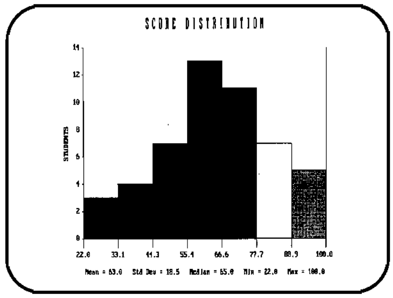
We use the term "object-oriented dispatch" to refer to this late choice of which code to actually execute. In Sather, each class defines the structure of corresponding "objects." The class corresponding to an object defines its "type." Objects are chunks of memory that are dynamically allocated while the program is running, and which have a set of routines associated with them. These routines form the "interface" to the object, and are defined in its corresponding class. For example, the class `DISPLAY_1` might have an interface defined by the routines `draw_line` and `draw_circle`. Any code that performs actions on an object of this type must do so via these routines. We might later define a class `DISPLAY_2` that also defines the routines `draw_line` and `draw_circle`, but for a different kind of display. A variable `disp` can be declared to hold objects of either of these types.

When a call like `disp.draw_line` is made, the actual type of the

Physics Academic Software

PEER-REVIEWED EDUCATIONAL SOFTWARE

Published by the American Institute of Physics in cooperation with American Physical Society American Association of Physics Teachers



GRADEBOOK

Russell A. Stevens

GRADEBOOK provides teachers with a fast, convenient way to record student scores and compute grades automatically. Some of the options are plus-or-minus grading, weighting scores in up to ten categories, handling incomplete grades, granting extra credit, dropping low scores, and issuing interim reports.

The program can combine classes when calculating distributions and display the result in histograms. Scores can be organized on summary sheets or as individual records for students.

Includes 58-page User's Manual.

For the IBM PC

$49.95

$149.95 with a ten-copy site license
$39.95 for members of AIP Member Societies

TO ORDER

Order using Visa, MC, AmEx, check, or institutional PO. Specify 3.5" or 5.25" format. S/H $3.50 for first item ($7.50 foreign) and $.75 for each additional item.

TASL
Box 8202, NCSU • Raleigh, NC 27695-8202

Call Toll-Free (800) 955-TASL
(919) 515-7447/FAX (919) 515-2682

Ask for details of AIP's SATISFACTION GUARANTEE

Circle number 11 on Reader Service Card

object held in disp is checked, and the appropriate choice of draw_line code is chosen to execute. This provides wonderful flexibility, because one can cause new code to be called by old code by defining a new class with the same interface as an existing class. Unfortunately, it reduces computational efficiency because the code must check an object's type and use that to select the instructions to execute. Sather uses several techniques to minimize the cost of this checking.

Why Strong Typing?

In languages like Smalltalk, any variable can hold any object. In Sather, every variable is declared to be of a specific type, and this declaration restricts the objects that can be held by the variable. There are both conceptual and computational benefits to this. It helps the compiler to generate efficient code, because it puts restrictions on what can occur during execution. It helps the programmer, because it provides a conceptual structure for organizing and understanding the code. The declared type of a routine argument delimits what values may be legally passed. All Sather classes are organized into a hierarchy, and this hierarchy is used to determine which objects a variable can hold. A class is called a "descendant" of another class if it lies below it in the class hierarchy.

The Sather type system is even stronger than those of other strongly typed object-oriented languages. In most languages, if a variable is declared to be of type T, it can legally hold any object for which the type is a descendant of T in the type hierarchy. This is often very useful in that it allows new subtypes to be defined and acted upon by existing code. It also introduces computational inefficiency and conceptual ambiguity. There are many situations in which the type of object that a variable can hold is precisely known by the programmer.

Sather allows one to distinguish between variables that can only hold objects of a particular type and variables that can hold any descendant of a declared type. The declaration v:VECTOR means that the variable v can only hold objects of type VECTOR. The declaration v:$VECTOR means that the variable v can hold objects of type VECTOR or any descendant of VECTOR in the hierarchy. Object-oriented dispatching is only performed on the second kind of declaration. When the type is precisely specified, the compiler generates direct calls, which are exactly as efficient as corresponding C code would be.

The Sather compiler is itself a large program written in Sather, and uses a lot of dispatching. Each of the nodes in the Sather code trees uses dispatching on its children (e.g., each type of statement and each type of expression is represented by a separate object type). The performance consequences of dispatching were studied by comparing a version in which all references are dispatched to the standard version.[11] The use of explicit typing causes an 11.3% reduction in execution time and approximately one-tenth the number of dispatches.

The Sather type system is stronger than that of most languages, because it allows more distinctions to be specified. It therefore has more of the benefits and pitfalls of strong typing. Strong typing offers not only computational efficiency, but also conceptual advantages. The compiler is able to perform stronger type checking, and so catches errors that would not be caught with weaker typing. It also allows Sather classes to be structured in ways that more naturally reflect the represented concepts.

For example, a simple two-dimensional geometry system might have a class POLYGON with descendants TRIANGLE and SQUARE. It might be important for the POLYGON class to define a routine add_vertex, which increases the number of sides by one. Such a routine is not appropriate for the descendant classes TRIANGLE and SQUARE, because they have a fixed number of sides. In Sather, these classes can undefine this inherited routine. If a variable declared to be a POLYGON could hold a TRIANGLE at runtime, the compiler could not check the possible application of the illegal routine add_vertex to a TRIANGLE object. Because Sather allows one to declare variables that can only hold POLYGON objects, add_vertex may be applied to them with complete safety.

This distinction is also relevant to the basic types representing integers, characters, real numbers, etc. In many object-oriented languages, objects of these types have tags or tag bits that specify their type. At runtime, extra tag-checking code must run in addition to any operations performed on the objects. This is nice from the point of view of conceptual purity, but is simply not acceptable for the performance goals of Sather. In Sather, objects declared to be of these basic types are guaranteed to actually hold them. This allows the compiler to generate code with no tag checking. In addition to saving the cost of the check, it allows the wide variety of optimizations that modern compilers provide. Much scientific computing code intensively computes with such types, and in Sather the result is as efficient as in C.

Why Multiple Inheritance?

Many object-oriented languages support only "single inheritance," which means that each class can have only one direct parent in type hierarchy. Natural types in the world, however, are more complex. The hierarchy of sets of object is typically a "directed acyclic graph" (DAG), in which classes can have more than one direct ancestor. For example, in a simulation task one might have a class for CHARGED_OBJECTS, which defines a variable for the charge, and one for MOVING_OBJECTS, which defines a variable for the velocity. Because ELECTRON objects are both charged and moving, they naturally descend from both classes. We would like to be able to apply to electrons all code that works on charged objects, as well as all code that applies to moving objects. If one is restricted to single inheritance, one would either have to require that all charged objects move or that all moving objects have a charge. In Sather, each class can inherit from an arbitrary number of other classes.

Why Parameterized Types?

One central feature of the Sather design is the use of parameterized classes. These are classes with one or more type parameters for which values are specified when the class is used. For example, the array class is declared as ARRAY{T}. When used, however, the type variable T is specified to be the type contained in the array. Thus, ARRAY{INT} declares an array of integers and ARRAY{STR} declares an array of strings. Parameterization supports a very common and important form of reuse. In conjunction with Sather's strong typing it causes no increase in performance overhead.

To achieve this high performance, it was decided to generate separate code for each instantiation of a parameterized class. This allows the code to compile in the targets of calls. The increase in the size of the code does not appear to be substantial. Typically, there are a few small classes such as `LIST{T}` that cause the generation of many instantiations, but most classes are not so replicated. The compiler determines the minimal set of self-consistent instantiations required. It also recognizes instantiations which are only used for class inheritance, and does not generate corresponding code.

Why Garbage Collection?

The languages derived from C are typically not "garbage collected." This means that the programmer is responsible for explicitly creating and destroying objects. Unfortunately, these memory-management issues often cut across the natural abstraction boundaries. Usually, the objects in a class do not have enough information to determine when they are no longer referenced, and the classes which use those objects should not be bothered with low-level memory-allocation issues. In addition, memory management done by the programmer is the source of two of the most common kinds of bug. If a programmer inadvertently frees up the space for an object which is still being referenced, a later call may find the memory in an inconsistent state. These so-called "dangling pointers" are often difficult to track down, because they frequently cause errors in code which is far removed from the offending statement. The second error is to forget to free up the space occupied by objects that are no longer referenced. This causes "memory leaks," in which a program uses up more and more memory until it crashes. This kind of error is also difficult to find and track down. Instead of user deallocation, Sather uses a "garbage collector."[12] This feature automatically tracks down unused objects and reclaims the space they use. To further enhance performance, the Sather libraries are designed to generate far less garbage than is typical in languages such as Smalltalk or Lisp.

The Implementation

The Sather compiler was itself written in Sather by Chu-Cheow Lim, and has been operational for about two years. It compiles into C code and is therefore easily portable to a wide variety of machines. The implementation is described in detail in Ref. 13. It is a fairly large program with about 30,000 lines of code in 183 classes. (This compiles into about 70,000 lines of C code.) As such, it provides a very nice example on which to test out concepts for structuring large programs.

The authors of Ref. 11 extensively studied the performance of the Sather compiler on both the MIPS and Sun Sparc architectures. Because the Sather compiler uses C code as an intermediate language, the quality of the executable code depends on the match of the C code templates used by the Sather compiler to the optimizations employed by the C compiler. Compiled Sather code runs within 10% of the performance of handwritten C code on the MIPS machine, and is essentially as fast as handwritten C code on the Sparc architectures. On a series of benchmark tests (including examples such as Towers of Hanoi, 8 Queens, etc.) Sather performed slightly better than C++ and several times better than Eiffel.

The Libraries

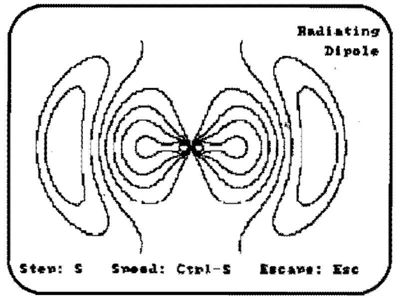
The Sather libraries currently contain several hundred classes, and new ones are continually being written. Eventually, we hope to have efficient, well-written classes in every area of computer science. The libraries are covered by an unrestrictive license, which encourages the sharing of software and crediting of authors, without prohibiting use in proprietary and commercial projects. Currently, there are classes for basic data structures, numerical algorithms, geometric algorithms, graphics, grammar manipulation, image processing, statistics, user interfaces, and connectionist simulations.

Many interesting issues have arisen in the design of efficient and widely usable abstractions. We will describe two sets of classes that are relevant to scientific computing. The

Physics Academic Software

PEER-REVIEWED EDUCATIONAL SOFTWARE

Published by the American Institute of Physics in cooperation with American Physical Society American Association of Physics Teachers



PHYSICS SIMULATION PROGRAMS

New

Robert H. Good
California State U.–Hayward

PHYSICS SIMULATION PROGRAMS is a set of eight independent programs that help introductory physics students learn about wave motion, chain reactions, Maxwell's demon, radiation from accelerating charges, and the optics of light through a thin lens The programs are suitable for lectures or nonstructured, exploration-oriented, individual study at various levels.

Includes 54-page User's Manual.

For the IBM PC and Apple II package includes both versions $49.95

$149.95 with a ten-copy site license
$39.95 for members of AIP Member Societies

TO ORDER

Order using Visa, MC, AmEx, check, or institutional PO. Specify 3.5" or 5.25" format. S/H $3.50 for first item ($7.50 foreign) and $.75 for each additional item.

TASL
Box 8202, NCSU • Raleigh, NC 27695-8202

Call Toll-Free (800) 955-TASL
(919) 515-7447/FAX (919) 515-2682

Ask for details of AIP's SATISFACTION GUARANTEE

Circle number 12 on Reader Service Card

first represents mappings from one vector space to another. Examples of such mappings are those produced by connectionist networks, linear least-squares fitters, and statistical nonlinear regression techniques. In the Sather library, each of these classes inherits from the class VECTOR_MAP. This class defines the interface for such operations as determining the dimension of the input and output spaces, computing the image of a vector under the mapping, and miscellaneous operations such as computing the mean square error for a set of training examples.

Once such an abstraction is defined, there are a whole set of "functorial" combining classes which may be defined to construct more complex maps. For example COMPOSITION_VECTOR MAP{M1,M2} represents the mapping which is a composition of two maps of types M1 and M2. Another class PRODUCT_VECTOR_MAP{M1,M2} forms the product of two maps, while SUBSET_VECTOR_MAP maps a vector to a permutation of some of its components, and CONSTANT_VECTOR_MAP represents a constant output vector. These classes may be combined like Tinkertoys to build up arbitrarily complex maps which still obey the defining interface.

We found a similar approach to be useful in the random-number generation classes. We wanted a class RANDOM which could produce random samples from a variety of different probability distributions (e.g., normal, binomial, gamma, Poisson, geometric). Such samples are generally produced by manipulating samples from an underlying generator that produces real valued random samples uniformly distributed in the unit interval. There are often differing requirements for such an underlying generator, however. For most applications, speed considerations dominate, and a linear congruential generator is sufficient.

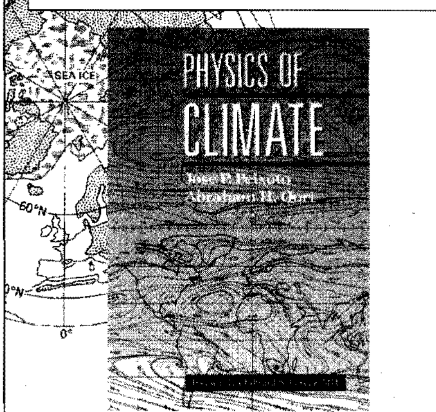
For certain critical applications, however, we do not care so much about speed, but we do require extremely high-quality samples. We therefore structured the library classes so that objects of type RANDOM have an attribute of type RANDOM_GEN, which holds the underlying generator. This is dispatched to retrieve uniform random variates. A variety of basic generators is provided. In addition, like vector maps, a variety of combining classes is provided to construct new generators. Examples include classes that generate new samples by summing the outputs of two generators modulo 1 and classes that form new generators by randomly permuting the outputs of other ones.

The Future

In this article, we have described aspects of the design and implementation of Sather, and how they are related to the goals of achieving code encapsulation, reusability, efficiency and portability. The Sather language provides powerful features such as parameterized classes and object-oriented dispatch, which are essential to achieving code encapsulation and reusability. The language is small, simple, and efficient. As we continue actively to develop classes and language tools, important new class abstractions become apparent. The current distribution includes X Windows user interface classes and a symbolic debugger. We are working on higher-level user-interface abstractions and extended language tools such as an interpreter.

PHYSICS OF CLIMATE

THE MOST COMPREHENSIVE AND CONTEMPORARY LOOK AT CLIMATE AS AN INTEGRATED PHYSICAL SYSTEM



PHYSICS OF CLIMATE

J. P. Peixoto, University of Lisbon, and A. H. Oort, National Oceanic and Atmospheric Administration

"A modern treatment of the nature and theory of climate."

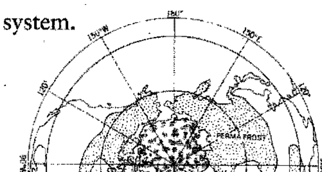
From the foreword by Edward N. Lorenz, MIT

AMERICAN INSTITUTE OF PHYSICS Marketing and Sales Division
335 East 45th Street
New York, NY 10017

The global upper air network. . .satellite data. . .nonlinear mathematical models. . . . Using the tools that have breathed new life into the study of climate, this ground breaking work demonstrates how environmental phenomena worldwide interact in a single unified system.

With more than 220 drawings, charts, and graphs, PHYSICS OF CLIMATE offers you the best current understanding of the Earth's climate system.

"A superb reference.... Belongs on the shelf of anyone seriously interested in meteorology and climatology."
—Curt Covey and Karl Taylor, *Physics Today*



For faster service call toll free 1-800-488-BOOK

To order, mail to: American Institute of Physics c/o AIDC • 64 Depot Road • Colchester, VT 05446

☐ Check enclosed (U.S. dollars only) ☐ Mastercard ☐ Visa ☐ American Express

Exp. Date Card No.

Signature (Required on all credit card orders)

Name

Institution (if applicable)

Street Address

City / State / Zip

Qty	Edition	ISBN	Price	Total
	Cloth	0-88318-711-6	$95.00	
	Paper	0-88318-712-4	$45.00	

Subtotal ________

Shipping: $2.75 for 1st book ($7.50 foreign), $.75 for each additional book. ________

Canadian residents add 7% GST. ________

9241 TOTAL ________

Another direction of research is the further extension of the Sather language to a parallel multiprocessor environment. The language "pSather"[14] is still being modified, but an initial version runs on the Sequent Symmetry and the Thinking Machines CM-S. As described in Ref. 15, the language adds constructs for synchronization, locking variables, and creating and manipulating threads. The issues that make object-oriented programming important in a serial setting are even more important in parallel programming. Efficient parallel algorithms are often quite complex, and so should be encapsulated in well-written library classes. Different parallel architectures often require the use of different algorithms for optimal efficiency. The object-oriented approach allows the optimal version of an algorithm to be selected according to the machine it is actually running on. It is often the case that parallel code development is done on simulators running on serial machines. A powerful object-oriented approach is to write both simulator and machine versions of the fundamental classes in such a way that a user's code remains unchanged when moving between them.

Acknowledgments

Many people have contributed to the development and design of Sather. I would especially like to thank Subutai Ahmad, Jonathan Bachrach, Jeff Bilmes, Henry Cejtin, Graham Dumpelton, Richard Durbin, Jerry Feldman, Ben Gomes, Chu-Cheow Lim, Franco Mazzanti, Heinz Schmidt, David Stoutamire, and Bob Weiner.

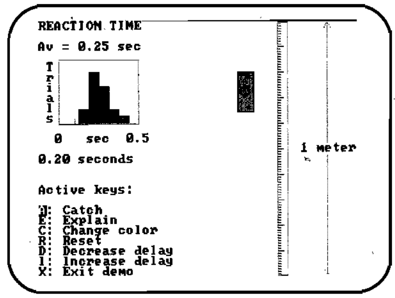
References

1. S. M. Omohundro, The Sather Language, Technical Report, International Computer Science Institute, Berkeley, CA, 1991. Included in the software distribution.
2. S. M. Omohundro and C. -C. Lim, The Sather Language and Libraries, Technical Report TR-92-017, International Computer Science Institute, Berkeley, CA, 1991.
3. H. W. Schmidt and S. M. Omohundro, Clos, Eiffel, and Sather: A Comparison, ICSI Technical Report TR-91-047, to appear in "The CLOS Book," edited by Andreas Paepcke, February 1992.
4. B. Meyer, *Object-Oriented Software Construction* (Prentice Hall, NY, 1988).
5. B. Meyer, *Eiffel: The Language* (Prentice Hall, NY, 1992).
6. B. Stroustrup, *The C++ Programming Language* (Addison-Wesley, Reading, MA, 1986).
7. B. J. Cox, *Object-Oriented Programming: An Evolutionary Approach* (Addison-Wesley, Reading, MA, 1986).
8. D. Unger and R. B. Smith, SELF: The Power of Simplicity, in *OOPSLA 1987 Conference Proceedings*, pp. 227-241, 1987.
9. A. J. Goldberg and D. Robson, *Smalltalk80: The Language and its Implementation* (Addison-Wesley, Reading, MA, 1983).
10. R. P. Gabriel, J. L. White, and D. G. Bobrow, CLOS: Integrating Object-Oriented and Functional Programming, *Communications of the ACM* **34** (9), 28, September 1991.
11. C. -C. Lim and A. Stolcke, Sather Language Design and Performance Evaluation, Technical Report TR-91-034, International Computer Science Institute, Berkeley, CA, May 1991.
12. H. Boehm and M. Weiser, Garbage Collection in an Uncooperative Environment, Software Practice & Experience, pp. 807-820, September 1988.
13. C. -C. Lim, S. M. Omohundro and J. Bilmes, The Sather Language Compiler /Debugger Implementation, Technical Report, International Computer Science Institute, Berkeley, CA, 1992 (in preparation).
14. J. A. Feldman, C. -C. Lim, and F. Mazzanti, pSather monitors: Design, Tutorial, Rationale and Implementation, Technical Report TR-9 1-03 1, International Computer Science Institute, Berkeley, CA, September 1991.
15. J. A. Feldman, C. -C. Lim, and T. Rauber, *The Shared-Memory Language pSather on a Distributed Memory Multiprocessor*, The Second Workshop on Languages, Compilers, and Run-Time Environments for Distributed Memory Multiprocessors, Boulder, CO, October 1992.

Physics Academic Software

PEER-REVIEWED EDUCATIONAL SOFTWARE

Published by the
American Institute of Physics
in cooperation with
American Physical Society
American Association of Physics Teachers



PHYSICS DEMOS

Julien C. Sprott
University of Wisconsin

PHYSICS DEMONSTRATIONS is a collection of ten computer simulations of the physics of motion and sound. Each simulation models a demonstration that a physics teacher in an introductory college or high-school course might perform.

You can vary parameters and freeze the action by pressing a key or clicking a mouse. Topics include reaction time, ballistics, oscillatory motion, waves, resonance, and the Doppler effect. The program can run on its own in a museum mode.

Includes 62-page User's Manual.

For the IBM PC **$49.95**

$149.95 with a ten-copy site license
$39.95 for members of AIP Member Societies

TO ORDER

Order using Visa, MC, AmEx, check, or institutional PO. Specify 3.5" or 5.25" format. S/H $3.50 for first item ($7.50 foreign) and $.75 for each additional item.

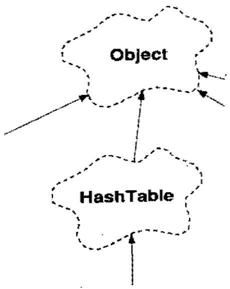
TASL
Box 8202, NCSU • Raleigh, NC 27695-8202

Call Toll-Free (800) 955-TASL
(919) 515-7447/FAX (919) 515-2682

Ask for details of AIP's
SATISFACTION GUARANTEE

Circle number 13 on Reader Service Card

COMPUTERS IN PHYSICS



C + + Programming Language...450

```
feature specification
    ring: BITS 32 -- point
    BPM: BPMS -- BPM
    STRMAG: STRMAC
    NoCelem: INTEGER
        -- # of cell
    fitTune (nux: DOUBI
        -- Fit betatr
    Tune (dir: INTEGER
        -- Get betat
        -- dir=0 for
    setCODeps (eps: DO
        -- Set COD
```

C Routines in Eiffel...456

Chemical Reaction Kinetics...494

COVER: Energy spreading versus initial energy, with color representing probability, reveals nonlinear resonance in Hilbert space of extreme Stark hydrogen. Paper begins on p. 483.

Contents

COMPUTERS IN PHYSICS (ISSN 0894-1866 coden CPHYE2) volume 6 number 5. Published bimonthly (6 issues per year) by the American Institute of Physics, 500 Sunnyside Blvd., Woodbury, NY 11797. Member subscription rates: $35.00 per year; foreign, $45.00; air freight to Europe, Asia and rest of Eastern Hemisphere, $55.00. Nonmember Individual subscription rates: United States and possessions, $45.00 per year; Canada, Mexico and rest of Western Hemisphere, $55.00; air freight to Europe, Middle East, North Africa, Asia and rest of Eastern Hemisphere, $65.00 per year. Nonmember Institutional subscription rates: United States and possessions, $195.00 per year; Canada, Mexico and rest of Western Hemisphere, $205.00; air freight to Europe, Middle East, North Africa, Asia and rest of Eastern Hemisphere, $215.00 per year. Back numbers, single copies, $35.00. Second Class Postage Paid at Woodbury, New York, and at additional mailing offices. POSTMASTER: Send address changes to Computers in Physics Subscription Fulfillment, c/o American Institute of Physics, 500 Sunnyside Blvd., Woodbury, NY 11797. Copyright © 1992, American Institute of Physics.

BPA

COMPUTERS IN PHYSICS

FROM THE EDITOR

Scientific Programmers Explore Benefits of Object Orientation

In three feature articles in this issue, scientific programmers explore the use and benefits of object-oriented programming (OOP). They find much to their liking.

In the first of these articles, Stephen Omohundro presents Sather, a nonproprietary OOP language, which the International Computer Science Institute made publicly available in June 1991 (see p. 444). Sather was derived from Eiffel and reportedly provides the benefits of OOP without sacrificing computational efficiency. Scott W. Haney and James A. Crotinger have used C++ to write a tokamak systems code. Starting on p. 450, they describe how the OOP paradigm helped them to implement their sophisticated distributed application. Hiroshi Nishimura has applied OOP to dynamic modeling for the control of accelerators such as the Advanced Light Source at Lawrence Berkeley Laboratory (see p. 456). The language Eiffel provides the adaptability and expandability needed for the creation of computer models that can evolve as practical accelerator experience builds up.

A related article on OOP will appear in the Nov/Dec issue. David M. Butler and Steve Bryson will present an object-oriented implementation of the vector-bundle visualization model, for which they claim broad scientific applicability.

Paul F. Dubois of Lawrence Livermore National Laboratory has served as guest editor for this special coverage of OOP. In The Last Word on p. 560, he derives some surprising lessons on the subject. We are very happy to announce that Paul has also agreed to serve as Department Editor for a regular column on scientific programming, beginning with the Jan/Feb 1993 issue. Introduction of this column is one of a series of changes that will, we hope, allow us to address the needs of working physicists more effectively. Other new columns will cover practical numerical algorithms and computers in experimental physics.

Lewis M. Holmes
Editor

LEWIS M. HOLMES
Editor
PHIL ROSE
Art & Production Editor
CHRISTINA BOEMIO
Assistant Editor
PARAM D. BAJWA
Editorial Assistant

LIN MILLER, CONNIE NEDOHON
Journal Section Production

BEVERLY PORTER
Director of Physics Programs
EDWARD P. GREELEY
Director of Advertising and Exhibits

DEPARTMENT EDITORS
A. JOHN MALLINCKRODT, SUSAN R. MC KAY
Book Reviews
HARVEY GOULD, JAN TOBOCHNIK
Computer Simulations
DENIS DONNELLY
Computers in Physics Education
WILLIAM H. PRESS, SAUL A. TEUKOLSKY
Numerical Recipes
MARGARET GJERTSEN
Product Reviews
ROBERT A. DORY
Spreadsheets
ROBERT S. WOLFF
Visualization

ALLEN ZEYHER
Correspondent

ASSOCIATE EDITORS, JOURNAL SECTION
BRUCE L. BRANDT
Florida State University
MARK FISCHLER
Fermilab
ALAN H. GLASSER
Los Alamos National Laboratory
HARVEY GOULD
Clark University
ROY LANG
NEC Corporation
CHARLES W. MISNER
University of Maryland
CLIFFORD A. PICKOVER
IBM Corporation
DON RAGAN
Wayne State University
DIETRICH STAUFFER
Cologne University

ADVISORY COMMITTEE
ROBERT R. BORCHERS, Chair
Lawrence Livermore National Laboratory
JOSEPH E. LANNUTTI
Florida State University
MYRON MANDELL
Maxwell Laboratories S-Cubed Division
RICHARD MATZNER
University of Texas at Austin
ELAINE ORAN
Naval Research Laboratory
JOHN S. RISLEY
North Carolina State University

EDITORIAL OFFICES

EDITOR AND JOURNAL SECTION:
American Institute of Physics
1630 Connecticut Avenue NW, Suite 750
Washington, DC 20009
Tel. (202) 745-1894
LH4@AIP.ORG
LH4@AIP.BITNET

PRODUCTION:
American Institute of Physics
500 Sunnyside Blvd.
Woodbury, NY 11797
Tel. (516) 576-2200
TRB@AIP.ORG
TRB@AIP.BITNET

ADVERTISING OFFICE
American Institute of Physics
140 E. 45th Street
New York, NY 10017
Tel. (212) 661-9260
Telex 960 983 AMINSTPHYS NYK