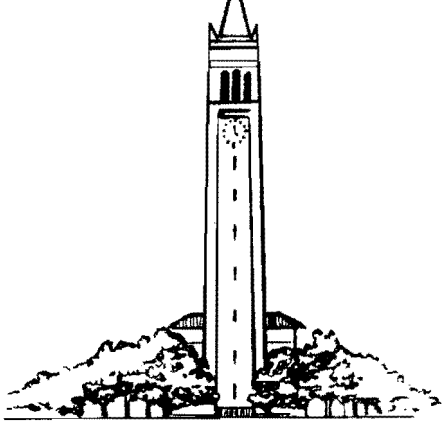

The Differences Between Sather and Eiffel

Sather is a new object oriented language derived primarily from Eiffel. Many design decisions must be made in the design of a language and different priorities give rise to different decisions. Every language is a compromised response to a large number of conflicting goals. This article focuses on the priorities which informed the design of Sather. Eiffel reflects a different set of priorities than those described here and may well be more appropriate for certain users.

The International Computer Science Institute is involved in a variety of projects that require the construction of complex software. Some examples include a general purpose connectionist simulator; a high-level vision system based on intricate geometric data structures; the support software for a high-speed parallel computing engine for speech recognition; and CAD tools for integrated circuit design. Each of these applications requires both complex data structures and very high efficiency. The complexity of these projects suggested that the benefit of switching from C to an object oriented language would far outweigh the startup costs.

Sather was initially developed in response to the needs of the vision project. Preliminary work with C++ and Objective C showed them to be insufficiently clean and modular for the needed tasks. The lack of garbage collection, parameterized types, and at the time, multiple inheritance prevented the development of software modules with the desired level of encapsulation and reuse. Work with CLOS showed it to be far too inefficient for the desired purposes. Eiffel was chosen as the language of choice and a system with about 150 classes was developed over a period of 18 months. This experience demonstrated the power of many Eiffel concepts, but also exposed many weaknesses in the design and implementation for our purposes. The primary shortcomings from our perspective were the inefficiency of the generated code and growing semantic complexity of the language. We were also beginning the design of object oriented constructs for parallel computers and needed a non-proprietary compiler base on which to build. For these reasons, we undertook the design of a more efficient, simplified language which still retained the desirable features of Eiffel. The design went through many iterations and many people made suggestions and refinements.

The fundamental language features were retained from Eiffel. The two primary forms of reuse in Sather are based on parameterized classes and object oriented dispatch. Sather is garbage collected and supports multiple inheritance. Classes provide modularity and encapsulation and all code is defined within some class. Class names are in a global namespace and class feature names are interpreted relative to the class in which they appear. These two levels of namespace hierarchy appear exactly right for projects with several hundred classes and several tens of features within each class. Aspects of Eiffel's clean syntax have been retained. Each construct is self bracketed by keywords and there is no need for "begin/end" constructs. The grammar is LALR(1) and easily fits on a page.

Added Features

Several features not found in Eiffel were added. Shared and constant features in Sather are variables that are allocated per class and may be directly accessed by any object from the class. Sather arrays are a part of the language rather than being merely a special class as in Eiffel. The memory for objects may have a variable sized array portion after the storage for features. Standard syntax is used to access and assign to array elements (eg. "a[5]:=3", "e:=b[17,c+d]", "f[7,2,3].foo"). The code generated for Sather array access often avoids a double indirection needed for an equivalent Eiffel array access.

Whither Sather?

The Sather Language gets its name from Sather Tower, a landmark at the University of California Berkeley campus which is shown above. The tower is actually a replica of the Campanile in Venice, Italy.

ICSI is a non-profit organization for basic research in computer science. It is a joint effort of the United States, Germany, Italy, and Switzerland to foster international cooperation in computer science. Funding comes from both U. S. government research grants and from the sponsor countries. ICSI consists of about 60 researchers about two-thirds of whom are visiting scholars. While separate from U.C. Berkeley, it is closely affiliated and several staff members hold joint appointments on the Berkeley faculty. The four main areas of research are AI (vision, language, speech recognition, knowledge representation), theory(complexity theory, parallel algorithms, etc.), realization (construction of both software and parallel hardware) and networking (protocols for gigabit networks).

Arrays are used extensively in the library code. Local variables in Sather may be declared at the point of use (as in C++) rather than in a special "local" section at the beginning of a routine. Class features may be declared "private" at the point of definition (instead of not appearing in an "export" list at the beginning of a class definition). A "break" statement to exit from loops and a "return" statement to exit from routines were added. The "switch" statement (analogous to the Eiffel "inspect" statement) allows arbitrary expressions in the target clauses. Conditionally compiled assertions and debug statements are named in Sather and an arbitrary subset of names may be activated on any given compilation. As in C++, Sather allows direct access to class routines, shareds and constants using the syntax "FOO::routine(5,6)".

The Sather type system has a major addition to that of Eiffel. In Eiffel, a variable may hold an object of any descendent class of its declared class. In Sather, the declaration "a:FOO" means that the variable "a" will hold an object whose type is "FOO", while "a:$FOO" means that "a" may hold any descendent of "FOO". This often allows the compiler to do better type checking and to generate far more efficient code than it otherwise could. It also introduces an extra level of specification which can clarify the semantics in situations with complex inheritance. In Sather, a descendent class need not define all of the features that its ancestors define. Such a requirement only holds for those features which are applied to "$" variables. Thus if "FOO" defines the routine "foo_rout", it's descendents need only define it if a dispatched access "a.foo_rout" appears applied to a variable declared as "a:$FOO". This solves a problem that has recently received much discussion. The class "POLYGON" might define a routine "add_vertex", which is inappropriate for the descendent class "RECTANGLE". As long as "add_vertex" is not applied to a "$POLYGON"variable, this causes no problems in Sather.

Sather uses the more natural "contravariant" rule for constraining the types of function arguments in inherited classes, while Eiffel uses a "covariant" rule. This means that in Sather the arguments of routines which are used in a dispatched fashion must have a type in a descendent which is a super-type of the type in the parent rather than a sub-type. This ensures that any code which has valid types on the parent will still have valid types on the child. Eiffel cannot make this natural choice because it would prevent many common uses of inheritance. Because Sather makes the distinction between dispatched and non-dispatched type declarations, and only applies the type constraints to dispatched usage, these common uses are still available in a contravariant framework.

Simplifications

Several Eiffel features were eliminated or simplified. Expanded classes and binary operator overloading were viewed as creating more complexity than they eliminated. Sather provides library classes to support exception handling rather than building it into the language. The complex exception handling in Eiffel is a major source of inefficiency. Eiffel "once" functions were eliminated because the "shared" attributes in Sather are simpler and may be used to perform the same function. The "like" construct was eliminated. The main use for this in our code was to declare something to have the type of the current class. A special declarator "SELF_TYPE" is used in Sather to declare such variables. The conditionally compiled statements in Eiffel ("check", "ensure", "invariant", "require", "variant", and "debug") were reduced to just "debug" for a conditionally compiled statement list and "assert" for a conditionally compiled boolean test.

The inheritance rules are much simpler in Sather than in Eiffel. There is no "rename" construct to access ancestor features under a different name. Eiffel has complex rules for multiple inheritance paths to the same ancestor class in the inheritance DAG. In Sather, the effect of inheritance is exactly as if the parent class were textually copied into the descendent class at the point at which the parent is inherited. The definitions of later attributes override the definitions of earlier attributes with the same name. In Eiffel, "Create" and "Clone" are different from every other function in that they change the value of a variable they are applied to. In Sather the only way to change the value of a variable is to assign to it. Sather adds the "type" feature that returns the tag of an object and is necessary to do old-style switch-based dispatch.

Sather is much smaller than Eiffel. Eiffel keywords which were eliminated in Sather include: "as", "BITS", "Clone", "Create", "deferred", "define", "div", "do", "ensure", "expanded", "export", "external", "feature", "Forget", "from", "implies", "inherit","infix", "invariant", "is", "language", "like", "local", "mod", "name", "Nochange", "obsolete", "old", "once", "prefix", "redefine", "rename", "repeat", "require", "rescue", "retry", "unique", "variant", "Void", "xor", and "^". In several cases features have been moved from the language into an appropriate library class.

The Sather Environment

The Sather implementation and programming environment is somewhat different from Eiffel's. More than one class may be defined in a file and the file name need have nothing to do with the class name. Class names may be of any length and are not restricted to file names

Sather Differences

(continued from page 13)

allowed by the operating system. Many times a natural conceptual unit consists of many small classes and it's convenient to put them in the same file. It's also convenient for keeping test code in the same file.

The Sather compiler is written in Sather. Like Eiffel, Sather compiles to portable C and easily links with existing C code. The compiler generates code that optimizes efficiency over code size. In particular, Sather does not attempt to reuse the binaries of ancestor code or other parameterized classes. This allows the compiler to compile in class specific knowledge. On the other hand, the Sather compiler only generates code for routines and classes which are actually used in a system. The dispatch mechanism in Sather is based on hash tables and caching and is extremely efficient in the most common situations.

Preliminary tests on the Sun Sparcstation I show Sather is 4 to 50 times faster than Eiffel in basic dispatching, routine access, and array access. Code generated by the current Sather compiler was slightly faster than C++ on a variety of test problems, including Towers of Hanoi, Eight Queens, etc. The special features of RISC machines (register windows, etc.) may exacerbate the efficiency problem with the generated Eiffel code.

Sather's garbage collector doesn't have any overhead when it isn't running, whereas the Eiffel collector extracts a cost on any pointer variable assignment. Eiffel's collector also must keep a stack of pointers to pointers on the stack. This keeps these variables from being put in registers, which can severely affect performance on RISC machines. Eiffel's ability to replace functions by variables in descendents has been eliminated. This allows most attribute access to be done without the overhead of function calls. Eiffel tries to ensure that type errors are impossible if a program makes it through the compiler. Sather has no such goal, though it does try to catch all common errors. Sather's interface to C is cleaner and more efficient than Eiffel's. Sather has compiler switches to enable array bounds checking, runtime type checking, and checking for dispatching from void variables.

An extensive library of Sather classes is being developed. As with the compiler, the design of the library focuses on efficiency and extensively uses the efficiency enhancing features of Sather. Algorithms and data structures based on highly amortized efficiency are used through out.

The Sather compiler and libraries will be made freely available with a license designed to encourage the development of widely available, well-written, reusable code, while not restricting the use of Sather for proprietary projects. A variety of Sather tools is under construction, including a GNU Emacs editing mode and environment, GDB-based debugger, browser, etc. A version of Sather for massively parallel shared memory machines is also under construction.

Sather is now in internal Beta Test. Outside Beta should commence soon. Full release is expected sometime this summer. Most EO charter subscribers should receive their Sather manual in April. Eiffel Outlook will be keeping a close watch on further developments. Expect to see regular Sather coverage in future issues.

Bertrand Meyer's Reaction To Sather

Ed. Note: As an extension of the Q&A interview, I asked Bertrand Meyer about his initial reaction to the October, 1990 version of the Sather Manual. His reply follows:

It's a welcome development. Although I tried to make Eiffel as small as I could, there is always room for subsetting and simplification. Look at Ada: the worst mistake made by the DoD was to forbid subsetting. It is quite possible that without this absurd policy decision, the lingua franca of the U.S. computing world today, instead of being C, would be some kind of micro-Ada -- and we would all fare better for it.

I do have two criticisms to address to the Sather designers, however. The first is a matter of form. When discussing the aspects of Eiffel they do not like, the Sather document unpleasantly confuses the language and a particular ISE implementation (2.2, I believe). To say that certain language features are inefficient is incorrect, unless you really mean to say that the feature *cannot* be implemented efficiently. Often the Sather Manual criticizes the Eiffel language where I believe the criticism should be reserved for a particular implementation.

The second has to do with Sather-Eiffel compatibility. If the authors of Sather disagree with some Eiffel constructs, and want to substitute their own, that's fine. But it serves no useful purpose to have different conventions on non-essential matters or constructs for which there is no conceptual disagreement. Why Sather should use the keyword "assert" where Eiffel has "check", or omit the keyword "do" to begin a routine, is beyond my understanding. If Sather is successful, then some people, especially students, are going to have to move between the two languages. Let's make their lives easier, not harder.

These problems are correctable and I hope Professor Omohundro and his colleagues will listen. I greatly appreciate their use of the Eiffel concepts and I hope their efforts are successful.

EIFFEL OUTLOOK

The Independent Source for the International Eiffel Community

Volume 1 Number 1, April 1991

In This Issue

'Eiffel' Trademark Transferred To Public Consortium

PARIS--3/11--NICE, the Non-profit International Consortium for Eiffel, has acquired the trademark 'Eiffel' from Interactive Software Engineering of Santa Barbara, CA. The consortium will be putting into place a validation scheme to enable anyone who writes a conforming compiler to use the trademark. The initial reference point for the language definition will be Dr. Bertrand Meyer's forthcoming book, Eiffel: The Language, Version 3.0. The manuscript for this book will be sent to Prentice Hall this month with publication to follow shortly.

NICE also announced that it plans to maintain close contact with the Eiffel user community to ensure that any decisions made by NICE reflect real user needs. Eiffel Outlook will be one of the prime communication channels for keeping users informed of its' activities and for soliciting input from users on techincal and other issues. The first NICE Report appears within starting on page 3.

PARIS--3/8--The Eighth International Eiffel User Conference took place with 55 attendees. The lively session including many interesting talks such as Michael Schweitzer speaking about the new Eiffel/S System of SiG Computer, and Kim Walden and Jari Koistinen speaking about EGL, an Eiffel class library for 3D graphics. Other presentations concerned graphical toolkits, Eiffel applications, language standards and reusability. Bertrand Meyer gave the Keynote Speech which was about current developments in Eiffel. We will publish more detailed summaries of these activities as they become available to us.

HANOVER--3/12--SiG Computer of Braunfels, Germany debuted their Eiffel/S system at CeBit this week. This is a new implementation of Eiffel for DOS computers. The system follows the new 3.0 language definition, but differs from previous Eiffel compilers in that it produces executables directly without the use of intermediate C code. The system is claimed to be fast and reasonably priced. Details are not yet available.

Freider Monninger of SiG Computer said the system is going into Beta test in April with deliveries expected by June. The main hold up at this time is the availability of Dr. Meyer's new book which will serve as an important component of the documentation. Eiffel/S will be available through ISE's distribution channels. At this time SiG and ISE are in negotiations over Eiffel libraries. SiG will supply some libraries which were developed internally, but are also considering inclusion of portions of ISE's standard libraries. The initial release will not include a graphics library.

SiG Computer has agreed to allow Rock Solid Software to evaluate the Eiffel/S system in the near future. Subscribers to Eiffel Outlook should see the results of this evaluation in the next issue.

SANTA BARBARA, CA--3/25--ISE announced today that it has ported its Eiffel software to the VAX/VMS operating system. This is a complete port of the version 2.3 system including the standard libraries and tools. ISE also claims that all Eiffel custumers currently under contract have been shipped version 2.3.

Q & A
with Bertrand Meyer

This interview with Bertrand Meyer was conducted via email and completed just as we went to press.

First, let me thank for agreeing to serve on the Board of Editors for Eiffel Outlook.

Let me in turn thank you for taking this timely initiative. I am sure that Eiffel Outlook will be a great success and I wish you the best.

What do you see as the most important developments for Eiffel and/or ISE in the last 12 months?

Internally, we released version 2.3, which our users universally welcomed as more complete and more robust. But the most important news was external: the creation of the NICE consortium, transferring control to a much wider body; the upcoming availability of other implementations; the launching of the 1992 ACM Eiffel conference in Germany; and the explosion of new interest in Japan.

What was the biggest surprise?

Learning about the MS-DOS implementation from SiG Computer. was planning to buy opera tickets for the Staatstheater in Munich and was told that I should have a business dinner instead. The purpose of the dinner, it turned out, was to tell me that a DOS version would soon be ready.

What was the biggest disappointment

Missing the opera. They could have told me afterwards.

(continued on page 15)