Practical Experiences with the Sather Language

Kresten Krab Thorup
Aalborg University*

July 17, 1992

Abstract

This paper describes experience using the Sather language in a 6th semester Computer Science project at University of Aalborg, Denmark. The concrete aim of the project was to design and implement a small programming language as part of the basic c.s. education. This paper presents the rôle of the Sather programming language in this project. It is concluded, that compared to C++ and Eiffel, Sather is indeed attractive as a programming language.

Introduction

The purpose of this paper is to communicate the experiences with the Sather Language, as being used for an undergraduate Computer Science project. To get the most out of it, it is worth reading Omohundro [6] to get a feeling of the Sather language and Nørmark [1] to know more about the background for this paper.[1]

The project was carried out by a group of 6 students, of which 2 were Computer Engineering students and 4 were studying Computer Science. The purpose of the project was to design and implement a programming language. The language was implemented both as a compiler generating Motorola 68000 assembler and an interpreter to be used for debugging purposes. As the reader may have guessed by now, The Sather Language was chosen for the implementation, and the entire implementation is around 6000 lines of Sather code, with some few hundred accompanying lines of C code.

An introductionary programming course in Eiffel had been carried out at the prior semester, so all of the participants had had quite a lot experience with Eiffel as described in Nørmark [1].

The paper will fall in to major

*Author's address: Dept. of Mathematics and Computer Science, Fredrik Bajersvej 7E, DK-9220 Aalborg Ø, Denmark. Email: internet `krab@iesd.auc.dk`.

[1] These are available by anonymous ftp to `iesd.auc.dk` in the directory of `/pub/sather`

parts. First, a chronological description of the entire course, and secondly some more specific notes on the Sather Language and it's implementation and available environments.

Choosing Sather

Compared to our earlier experiences with Eiffel we soon found out that C++ was much too complex for our purposes. First of all, we agreed, that the simplicity of Eiffel had made it much easier to cooperate in a group of developers. Trying C++ we saw, that we all developed different 'styles' of programming, while the simple Eiffel syntax had made it much easier to read the source code which other participants wrote. Next, the complex syntax and semantics of C++ was quite a problem for the inexperienced members of the team.

At first impression Sather seemed like a neat little language, which had several big gains: it was simple, had Eiffel-alike syntax, and had Garbage Collection; and at the same time it was much more handy than Eiffel (as described in the next sections). Also, Sather allows parameterized classes (known in C++ 2.1 as templates), see Stoustrup [3]. At this point a C++ compiler with templates fully implemented is not yet available to us.

Also, as we were going to write a compiler, it was of some interest, that the source for the Sather compiler itself (written in Sather) was available. As Sather was still developing, we were prepared to face minor bugs, inconsitensies and missing features in the implementation, and libraries, of which we also found some, as will be described later.

Using Sather

Writing Sather programs is really simple, and, in the light of our experiences with Eiffel, very easy to learn.

Compared to both Eiffel and C++ the group found, that Sather was really pleasing to work with. The simple language made it very easy for the participants to read and understand each other's code.

The simple language, with garbage collection and a strong type check allows the programmer to worry about the important parts of a program—the structure of it, and the algorithms used in it. This is opposed to C++, where it is our experience, that the programmer will be using a considerably amount of time locating semantic errors and memory leaks (unallocated memory).

Compared to Eiffel, Sather is a charm. Eiffel is really a drag on the programmer. There is quite a number of points worth mentioning here. Most of them will be explained further in the following section.

- The sematics of many language constructs if much simpler in Sather. Most noteworthy, perhaps, is the semantics of inheritance, which is simply 'textual replacement'. We know this point is a hot potato, but again, it is our experience, that the simple sematics makes the language easier to use, especially in a project with more participants. For further reference see [6].

- In Sather, you can have more class descriptions on one file. This makes it *much* easier to get the big picture in a complicated application.

- In Sather, you can access features directly (a'la C++ static) without nessecarily having an instance of the class. Also the Eiffel 'once' functionality have been replaced by a more C++ static alike mechanism called 'shared'.

- The Sather compiler is both faster than Eiffel's (ISE v2.3) and it produces truly faster code. As described in Omohundro [6] tests have actually shown that it is slightly faster than C++.

- Sather's C interface is very simple and easy to use compared to Eiffel. One may simply call functions/methods and share data between C and Sather.

- Sather has a real source level debugger, Eiffel has not.

On the other hand, a number of important features from Eiffel have also been removed. These are mainly the 'religious' point, which may seem somewhat hard to use.

- Invariants have been removed. This is a construct, which cannot easily be 'emulated' in Sather, but the project group felt no big need for it. It's nice to have, but easy to do without. Especially as we were doing dedicated programming and not general library classes.

- Assertions have been simplified, so that both `require` and `ensure` construct have been replaced by a single `assert` construct.

Besides those thing, Sather is a little simpler syntactically spoken. E.g. arrays are indexed using traditional `[]` brackets.

The Sather Implementation

Sather is implemented as a compiler, written in Sather itself. Additionally a very comprehensive environment for GNU Emacs is included in the distribution. More over, there is a source level debugger based on GNU's debugger, gdb.

The compiler is implemented in Sather itself. It produces K&R style C code.

Like Eiffel, Sather takes care of handling incremental compilation, but it is not fully implemented, and it often seems like the compiler recompiles far too much. In C++ the programmer is supposed to handle class interdependencies by maintaining a make file. I have made a simple filter which will split up the Sather sources into single classes, and then produce a new system description file (`.sather`) based on the present. This speeds up compilation somewhat, but still, it isn't good enough.

As noted earlier, we found some errors in the implementation. These were nothing important to the use of the compiler at all. The problems found were among other that if a non-boolean expression was used in boolean context, the compiler sometimes 'produced' a segmentation violation. Another tjing was that the compiler couldn't handle multi line strings. Some more such minor bugs were found around exception handling, but it didn't seem as any problem. (Besides, the author had fun correcting them.)

The interface to C

A high level programming language like Sather and Eiffel needs a good interface to C. This is because *any* dedicated library or programming tool in general will be available for C. In this specific project, `Yacc` has been used for the parser construction, and it is dedicated to C.

Sather has a very simple interface to C. Much like in C++, you 'declare' the 'external' functions and/or variables, by declaring them as features of the class C:

```
class C is
   func(arg1:INT):CHAR is end;
end;
```

would be declared in C++ like this:

```
external "C" {
   char func(int arg1);
}
```

Calling the 'other way' from C to sather, is accomplished by declaring a Sather feature to be having a specific name in the resulting C code, and then simply calling that feature from C, with a reference to the actual instance as the first argument.

Is however somewhat complicated to assure that all names match, but I have made a simple filter allowing Sather function calls to appear directly in the C code. This program will scan the existing sather source, and resolve virtual names in order to assure full type check. This program allows you to write function calls like the following in the C code:

```
MYCLASS::obj.feat(args...);
```

which will be transformed into something like:

```
(cast)_MYCLASS_feat(obj, args...);
```

in the resulting C code. This tool has been heavily used to interface from the `Yacc` specification to the Sather part of our compiler/interpreter. The tool called `ctos` is described further in [7].

Compared to C++, the C interface is of cause lousy. But compared to Eiffel, it is really a win—it is easy to use and understand, and that what's important.

Sather environment, and supporting tools

The Sather distribution comes with a comprehensive environment mostly based on GNU Emacs.

The Sather compiler

The Sather compiler is a command line program, which is controlled by specifying flags (or options) and by maintaining a system description file like Eiffel.

The flags which the compiler accepts are in some cases somewhat confusing. Most of them are not well documented, and some of them actually makes no difference.

The system description file is much like the one of Eiffel's, and is used for specifying things, like the files included in the system, specifying an alternative C compiler, or flags for it, to be used. Also the system description file is used for specifying specific C names of features in the generated code. In general, the notion of a such description seems like a good idea, which we liked. Although this is only convenient for projects which are not too small.

The Emacs programming environment

The GNU Emacs environment is *very good.* Most important, it is very good at indenting (also known as pretty printing) then source. The environment has hot keys for often used construct like classes and features. Also, it allows you to both compile and debug a program form *within* Emacs.

Also, there are quite nice browsing facilities. These will let you do manipulations like 'go to the definition of class `Y`' or giving listings much like Eiffel's short does it—That is a listing of all exported features of a given class, together with it's documentation.

As a part of this environment, there is an program written for Emacs, which can convert C structures to corresponding Sather constructs.

There is however a few annoying things. When you insert a class construct, it will insert a header for the file which among other things specifies that the file is copyrighted someone at ICSI. Also, as it is based on `language-tools` it inserts annoying messages specifying the last modification time whenever you save a file. This also influences some other modes. Inserting funny copyright notices, and the like has nothing to do with Sather!

The source level debugger `sdb`

A complete source level debugger is part of the distribution. It is quite good, but has some problems handling automounted directories. It is simply an extension of gdb (GNU's debugger), allowing one to visit and inspect objects at run time as well as single stepping.

Also it seems quite unpredictable when it will be able to list Sather sources. At compilation time a special name-mapping file is generated for the debugger, but it seems like the generation of that one is somewhat unstable.

The Libraries

Sather comes with a well suited set of library classes. As the language has parameterized classes many of the data structures provided can be used directly out of the bag.

There are for instance nice implementations of list, queue's etc.; and also some quite good abstractions over things like files and UNIX interaction in general.

Sather does however not have the notion of a real binary library, which can be reused between programs. This is really a kludge, as the source code for all the libraries will be compiled for every single program. This takes up quite a lot of disk space. Introducing something like that would be a *very big* improvement.

Conclusion

Sather is a programming language under development. Hence it will, at least for some time, still be changing continiously, and have minor bugs and inconsistencies. If you are planning to use Sather, you should be prepared to face these, as the only form of 'support' you can have is what you can get on the USE-net (i.e. news).

Apart from this single black point, we find that Sather is a very good language, especially for applications of a certain size (approx. 2000 lines). Sather look very much like Eiffel, and to get the most out of it, you will have to read Eiffel [4] and/or [5], as the documentation for Sather itself is based on those. Sather is easy to use and get started with. It is more simple and than Eiffel, and much much more clean than C++. As we like to say:

> Sather is Eiffel for programmers.

References

[1] Kurt Nørmark. Experience with Object-oriented Programming and Eiffel in an Introductory Programming Course. Unpublished, Aalborg University 1992.

[2] Bjarne Soustrup. The C++ Programming Language.

[3] Bjarne Soustrup. The annotated C++ Reference Manual.

[4] Bertrand Meyer. Object Oriented Software Construction. Addison-Wesley, 1991

[5] Bertrand Meyer. Eiffel, the language. ISE, 1991.

[6] M. Omohundro. The Sather Programming Language. ICSI, 1989

[7] Kresten K. Thorup. The `ctos` utility. (To appear)