SETL NEWSLETTER # 72 17 April 72

D. Shields

An introductory explanation of SETL, a status review, and a profile of SETL User-Group

In this newsletter we present a brief profile of the current status of the SETL project. It is hoped that this profile will indicate the overall relations between the various groups currently at work; indicate to the NYU systems group how the SETL project is currently using the oprating system, and how the operating system may be expected to be used in the near future.

The goal of the project is to implement the programming language SETL. In brief, SETL is a language of very high level which has sets as its fundamental data-type, these sets have as members the standard "atoms" of most programming languages -- integers, booleans, character strings, etc. Sets may themselves contain sets; and sequences, or "ordered sets" are provided. Sequences roughly correspond to the "vectors" found in many programming languages.
The language provides the standard set-theoretic operators (addition/deletion of object in set, test of set membership, a set former), and the standard operations on the primitives (addition of integers, catenation of character strings, etc.) Syntactically, the language is of the level of Algol or PL/1, except that no declarations are necessary since object "types" (set, atom, sequence) may vary dynamically. Storage allocations is dynamic and is handled by the system, which thus must provide a garbage-collector.

SETL is intended to provide an abstract, set-theoretic, executable language for algorithm specification, i.e., one in which the programmer defines his algorithm abstractly without worrying about particular implementation details relevant to efficiency, this abstract description is based on set-theoretic notions, and the description is executable in that there exists facilities for "running" the algorithm

to determine its correctness, execution properties (time, memory, etc.). In this abstract view, a program is an implemented algorithm -- the algorithm contains the description of the necessary abstract objects which must be formed (which sets must be computed, as for example, a "symbol table") and which operations must be performed (e.g., when something must be "added" to a symbol table), and the implementation of the formation of the objects and the opeations on them so that the algorithm can be executed. For example, we view a FORTRAN compiler as consisting of two fundamental pieces: a description of all the operations that must be done to recognize and compile FORTRAN -- input of cards, test of statement types, compilation of code which may involve register allocation, etc.; and the binding together of these abstract pieces into something executable (choice of data structures to be used by the compiler, the conventions for linking with the operating system, etc).

A much more complete description of the underlying assumptions behind SETL is contained in the introduction of the SETL NOTES.

How then is SETL implemented? An implementation may conveniently be divided into three parts:

(a) a translator which accepts SETL source and produces output in some lower level language, which we shall denote as the LL.

(b) A run time library, or RTL, of routines written in LL which support the basic set operations on the data structures chosen to represent the sets, sequences and atoms of SETL.

(c) A facility for executing the LL-translation of a SETL program.

This division is convenient in that it allows parallel development of the necessary parts; moreover, the parts are independent in that different parts may be implemented in different ways without affecting substantially the design of the other parts. For example changes in the syntax of SETL would of course require a change in the translator (a), but little, if any, modification of the executor (c). The above division serves to define a plan of attack for creating a particular realization of SETL, and

helps to clarify the design- and implementation-tradeoffs involved.

For example, suppose we want to "implement" in an installation where only PL/1 is available. We could proceed as follows:

(a) Realize the translator by writing a PL/1 preprocessor which accepts SETL-like source and produces PL/1, or eliminate the need for a translator by requiring the programmer to express his SETL algorithm in PL/1 using a combination of "pure" PL/1 and calls to routines in the RTL.

(b) Write the RTL in PL/1; i.e., decide on how sets, sequences, etc. are to be implemented, and code the necessary library routines in PL/1.

(c) Execute our SETL programs by compiling and then executing using the available PL/1 compiler.

The previous approach may conveniently be characterized as the "subroutine-library" method, a common software implementation method often used to implement statistics or linear-programming packages. Another approach, the "interpreter approach" is as follows:

(a) Design an abstract SETL-like machine which has as its basic operations the basic operations of SETL -- set membership, iteration over sets, etc. Assembly language for this machine thus has the flavor of SETL programs in which the programmer has reduced all the complicated forms of the SETL syngax himself. Call this language the SIL, for set implementation language.

(b) Construct an interpreter for SIL, thus providing the execution facility (c).

(c) Obtain a translator by hand-translation of the SETL to SIL translator written in SETL to SIL, thus providing (a).

(d) Similarly obtain the RTL by hand-translation of the RTL written in SETL, this task may be simplified by hand translating only the parts necessary to obtain a "minimal" system, and then treating the rest of the RTL as just another program to translate into RTL.

As a historical note, observe that an approach commonly used in the past corresponds to writing (a), (b), and (c) in assembly language; this approach may be characterized as the "manufacturer's" method.

Currently (Spring 72) the SETL group is working on two SETL implementations:

SETLB - consisting of

(a) A FORTRAN-written translator accepting as input a simplified subset of SETL, and producing as output BALMSETL text.

(b) BALMSETL, consisting of an RTL library for SETL written in BALM, and a modification of BALM which in effect replaces the standard BALM syntactic forms by calls to the appropriate procedures in the RTL.

(c) BALM,an extensible language system already implemented by Malcolm Harrison. The BALM system uses the "interpreter" approach previously mentioned with an interpreter written in FORTRAN, while the BALM RTL and translator were obtained by using a previously existing version of BALM written in a combination of FORTRAN and assembly language.

SETL/LITTLE, consisting of

(a) a parser, originally written in SETL which is being hand-translated into LITTLE, making use of the RTL being written in LITTLE. This parser outputs LITTLE.

(b) An RTL, written in LITTLE, based on an RTL written in SETL in which sets are represented as hash addressed vectors of lists. (This is also the RTL used in BALMSETL).

(c) A compiler for LITTLE, a low-level FORTRAN-like language currently being implemented, which produces relocatable binary code for the 6600.

SETLB is intended as a tool for verifying algorithms already written in SETL, both for purposes of publication and implementation, and as a means of discovering some of the issues and problems that will become important when SETL is available (for example, preliminary use of SETLB indidates that some thought should be given to providing a variety of input-output routines,

so that the user may display a set in the same way as he is thinking of it, as for example, a tree or graph).

SETL/LITTLE has as its goal the description of a complete SETL system expressed in a low-level language which may conveniently be transported to other operating-system / hardware configurations. That is, this system will consist of a parser and RTL written in LITTLE, and the LITTLE system itself, which will include a compiler written in LITTLE for LITTLE and the facilities for transporting LITTLE to other machines and operating systems. We remark that the BALM system also has the facilities (and goals) for transportation to other environments, but that the SETL group feels that, due to the high level of both BALM and SETL, a portable SETL system must itself be expressed at a fairly low level.

The LITTLE language is intended as a low level machine-independent language for the specification of systems programs. Developmental work on this language was begun at N Y U in 1968. The original design intent for LITTLE was for simplification of the process of transporting the LITTLE compiler to various computers; however, the problems posed by using LITTLE to transport SETL have both intensified the pressures to obtain a locally working LITTLE compiler so that debugging of the SETL implementation may proceed, and have shown that LITTLE itself must be extended in some way toward the level of a "systems-programming" language -- for example, the I/O facilities to be provided by LITTLE are currently being extended, and those optimizations of code particularly relevant to SETL are being incorporated in the LITTLE compiler.

In the preceding few paragraphs we have briefly outlined the main design goals and the thrust of the current efforts toward achieving an implementation of SETL. We now give a brief description of the current status of the major program products being developed.

We begin with the current status of SETLB. As has been outlined, the SETLB system consists of a preprocessor, which produces BALMSETL, the BALMSETL system itself, and the BALM system already available at NYU. The SETLB preprocessor consists if about

80000 lines of FORTRAN code, is in the final stages of debugging, and should be available for use within a week or so. The BALMSETL system has been available for about two months, and is being revised slightly to mate more smoothly with the pre-processor. The BALM system in its current form has been available for about a year, and is currently being extended to produce machine-level code for the 6600, instead of interpretative-code for a generalized BALM machine.

The SETL-LITTLE system consists of a SETL parser currently being converted into LITTLE from the original source in SETL, the BTL, and the LITTLE compiler. About 3000 lines of LITTLE code for the parser, and approximately 3000 lines of code in LITTLE for the RTL have been completed; each of these pieces is perhaps 30 to 40 percent complete in terms of code-volume, and perhaps 80 to 90 percent complete in terms of design. The LITTLE compiler consists of a lexical front end, consisting of about 4000 lines of FORTRAN code, which is debugged and available (the front end includes a macro-processor and a cross-reference feature), together with a parser, consisting of some 2000 lines of FORTRAN code (the parser proper is itself generated by a FORTRAN written program which accepts a Backus description of LITTLE and produces the parser in FORTRAN); the semantic verification and code generation routines have been expressed in FORTRAN and are in debugging.

The last phase of the LITTLE compiler, the assembler which produces relocatable CDC 6000 series machine code is currently being translated into FORTRAN and is in the initial stages of debugging. (We note that the LITTLE compiler was originally written in LITTLE, and then translated into assembly language macros using a FORTRAN-written translator; the compiler has now been translated into FORTRAN from LITTLE, using an adaptation of the lexical processor for LITTLE to produce FORTRAN).

Efforts in the near future will be directed toward completing the SETL/LITTLE system just outlined; and the support of the SETLB user group, consisting of some people in the SETL group proper, and others - both students and interested users - who will be exploring the possibilities offered by language much

like SETL. Note that the completion of SETLB in a few weeks, and the availability of a working version of LITTLE in a few months, will place upon the SETL group the additional burden of supporting its own user group.

In the preceding few pages we have outlined briefly the scope of the SETL project and the current status of the development work. We now proceed to give some details which may help to outline the demands of the SETL group on the CIMS 6600 facility, and the shape these demands may assume in the next few months. We begin by noting that the SETL project is a substantial software development effort, that the SETL group forms one of the main users of the CIMS computing facility, and that the nature of the SETL effort requires a fairly close (and hopefully friendly) relationship with the CIMS systems group. It is our hope that the SETL group can also provide useful "input" to the systems groups, both by pointing out the deficiencies and merits of the current operating system, and by indicating some directions which operating system development effort should take. Indeed, the main goal of this report is to begin a definition of the SETL group as a "coherent user", so that the systems group may both anticipate the problems which may arise from members of the SETL group, and the ways in which the two groups may interact.

The main factors of interest to the systems group would seem to be as follows:

(a) A profile of the various subprojects currently under way: this profile to include the people involved, the nature of a typical job, the parts of the operating system used most often by these jobs, and any special hardware demands on the jobs.

(b) Critical parts of the operating system as far as the SETL group is concerned; i.e., those parts of the operating system used by all subprojects, in that an error or improvement here will be felt by all members of the SETL group.

(c) File maintenance - the creation and maintenance of files. This area is in fact the most critical component of (b) above, and is treated separately due to its importance, and also due to the "file oriented" nature of the operating system.

(d) A profile of the users of the program products which the SETL group develops. Only a brief description is possible at this time, since most of the efforts to date have been internal to the group and it is only recently that some programs have become sufficiently debugged and polished so that they can be used by epople not directly involved in the SETL implementation effort. This user group will probably develop first for SETLB, and will consist of many users not too experienced with the CIMS operating system, so that some efforts may have to be directed toward creating a "hospitable" environment for them in which these users may conveniently use SETLB without adversely affecting the operating system.

We now proceed to give more details in each of the above areas; each area will be described on a separate page, in the hope that this description can be conveniently kept up to date without needless textual rearrangement.

PEOPLE-Profile

The members of the SETL group who now or soon will be major users of the computing facilities are as follows (these are the people to see if questions about a particular project or program arise):

Jack Schwartz - director of SETL project; design and programming of LITTLE front-end, and the preprocessor for SETLB.

Bob Abes - LITTLE I/O, extensions of LITTLE, and LITTLE implementation.

Kurt Maly - SETL parser and its implementation in LITTLE.

Elie Milgrom - BALMSETL and SETLB/BALMSETL interface.

Bob Paige - development of BALM translator to produce machine level code for BALM (this effort promises a substantial speedup in BALM, and is critical if the users of SETLB are not to swamp the operating system); partial responsibility for file maintenance.

Dave Shields - "polishing" of LITTLE front-end and SETLB preprocessor for efficient match with operating system; LITTLE optimization and implementation.

Aaron Stein and R. Bonic - LITTLE implementation, in particular, the development of LITTLE compiler.

Hank Warren - construction of RTL in LITTLE and original design of RTL in SETL; overall responsibility for file maintenance.

JOB-Profile

The typical job of a SETL user has the following properties:
a) No tapes are used; due to the number of people in the project, the problems of maintaining often used files on tape, and the recognition that frequent use of tapes would further narrow a critical system bottleneck.
b) Free use of permanent files, since such files are frequently accessed and can conveniently be maintained.

c) A job time limit of 10 100_8 seconds, except for BALMSETL jobs.
d) A memory requirement of less than $110k_8$ of memory, except for BALMSETL jobs; typically $60k_8$.
e) Perhaps 5 to 20 pages of output, except for listings of source files and occasional debugging rungs which may produce 100-200 pages of output.
f) No punched output.

SETLB and BALMSETL jobs are compute-bound, and currently have an average job time limit of 250_8 seconds and $130K_8$ memory; when the new version of BALM is available, typical requirements might be 100_8 seconds and $150k_8$ memory.

SOFTWARE demands of SETL group

The key pieces of software used by almost all members of the SETL group are as follows.

UPDATE - this program is used to maintain the source cards for all major programs; the heavy use of UPDATE is due to the convenience and good design of the program, and the knowledge that UPDATE is used to maintain the operating system itself, so that any error in UPDATE must be viewed as a high priority critically important systems problem. Most users in the SETL group use UPDATE in a standard, straightforward manner; since the deficiencies in UPDATE documentation preclude sophisticated use.

RUN FORTRAN COMPILER - since FORTRAN is in fact the highest level reasonably efficient language available in the current operating system, most of the first level or bootstrap routines for LITTLE and SETL are being written in FORTRAN. RUN is used primarily to allow convenient interface with necessary assembly language routines written before the days of FTN, and to avoid the use of more than one FORTRAN compiler in what is, hopefully, a temporary situation in which existing languages such as FORTRAN must be used for a SETL implementation.

PERMANENT FILES - known to the SETL group as "problem" or "passing" files. These files are used as the main vehicle for inter-user communication e.g. when a SETL program product is completed, it is made avaiable to other uses as a permanent file.

BALM - this is a language system developed at NYU by Malcolm Harrison. BALM is the target language for SETLB (via BALMSETL). We remark that BALM requires the use of FTN, and uses MACE I/O, and that the SETL user group is currently the major user group for BALM.

SETL 72-12

FILE MAINTENANCE for SETL Group

Problems related to file maintenance are particularly important to the SETL group for two reasons:
a) The files (source libraries, documentation, etc.) produced by the SETL/LITTLE groups in the past few years represent many man years of effort and several hundred thousand dollars of research money.
b) The lack of facilities in the current operating system for providing "permanent" files on disc, archival storage facilities, and absolute backup facilities for file protection. Although it may be argued that such facilities exist, for an individual user, the description of the use of these facilities to the members of the SETL group may involve a much more detailed knowledge of the operating system than is necessary for work in the SETL implementation (e.g. use of multi-file tapes, use of FET to label tapes, etc.).

The problems encountered by the SETL group in maintaining files have resulted in the expenditure of at least four man months of effort in developing our own permanent file save-to-tape system; the development of an archive system to save files on tape; and the implementation recently of a policy of punching critical files at regular intervals and storing these files at off-site locations.

In summary, the SETL group has had severe problems in maintaining important files and has had to develop its own methods for file maintenance; it is felt that these methods both indicate limitations in the current operating system, and define procedures which should perhaps be used by other users who have expended substantial effort and money in creating files which should be protected.

Relevant Areas for Systems Group - SETL Group INTERACTION

We conclude by further amplifying previous remarks about the possibilities for and benefits of interaction between the SETL group and the systems group:

a) The SETL group has its own well defined internal goals which include the development of a substantial software product, numbering several tens of thousands of lines. Thus the SETL group serves to define to the systems group some problems in using the current operating system to implement a very large software package.

b) Systems groups are too often forced to operate in a void in which the user community is seen as several hundreds of isolated users making various demands on the operating system. By providing a coherent description of a group of several users and by providing the means for interaction, it is hoped that the systems group can more precisely define the areas in which improvements and research are relevant.

c) Since SETL ultimately hopes to obtain an implementation of various machine/operating system configurations, it is hoped that the systems group can help to point out some of the problems that may arise when SETL attempts to leave CIMS and survive at other installations.

d) Since a fundamental goal of SETL is to provide a convenient, expressive tool for describing algorithms, it is hoped that such considerations may help give the systems group the opportunity to step away from day-to-day considerations and to approach the problems arising from the effort to create an abstract, machine independent realization of a set-theoretic based programming language.