THE DEVELOPMENT OF JOVIAL

Jules I. Schwartz
Computer Sciences Corp.

1. THE BACKGROUND

1.1 The Environmental and Personnel Setting

The time was late 1958. Computers that those who were involved with the first JOVIAL project had been using included the JOHNNIAC (built at Rand Corporation), the IBM AN/FSQ-7 (for SAGE), 701, and 704. A few had worked with board wired calculators. Some had never programmed. For those using IBM, the 709 was the current machine. There were various other manufacturers at the time, some of whom do not exist today as computer manufacturers.

The vacuum tube was still in vogue. Very large memories, either high speed direct access or peripheral storage, didn't exist, except for magnetic tape and drums on some computers.

The first large scale system which was built and maintained by thousands of people was in the process of being installed after a considerable amount of effort, expense, and technological innovation. Called SAGE, it was a large real-time system (very large by 1958 standards). It had, among other things, an interesting and very valuable (even by today's standards) utility system for assisting in development. This included the Communication Pool. The Comm Pool's purpose was to permit the sharing of System Data among many programs by providing a centralized data description. Programming for SAGE had been done in machine language, as was almost all programming up to that time.

FORTRAN existed and had been in use for a few years, but use of higher level languages wasn't nearly as common as the use of such languages today. Certainly the majority of people involved in compiler efforts at that time, and in particular the initial JOVIAL language development and compiler effort, had almost no experience with such activities. I had had experience with the language and compiler called PACT (Project for Automatic Coding Techniques) which was developed and described in 1954-1955 (Melahn 1956). But actually this experience, largely because of the nature of that language and the computer the first version ran on (the IBM 701), was not particularly valuable for the development of languages in the ALGOL class.

One of the significant things that got the JOVIAL language and compiler work started was an article on Expression Analysis that appeared in the 1958 ACM Communications (Wolpe 1958). The fact that this was actually quite a revelation to us at that time seems interesting now. Since that day, of course, there have been many developments in the parsing of mathematical and logical expressions. However, that article was the first exposure many of us had to the subject. Some of us who had just finished work on some other projects (including SAGE) began experimentation with the processing of complex expressions to produce an intermediate language utilizing the techniques described. There was no compiler language at the time in our plans, but the idea of being able to understand and parse complex expressions in itself was of sufficient interest to motivate our efforts.

Another article which was to have great influence on our future development was also published in the ACM Communications in 1958. This was the description of what was then called the International Algebraic Language (IAL, later called ALGOL), which had been defined by what became (and for the most part had been) an esteemed group of computer people (Perlis and Samuelson 1958).

1.2 The Economic and Situation Stimulus

JOVIAL really got its beginning because of the launching by the Air Force of another large system. This one followed SAGE and it was called the SACCS System. SACCS was to be developed from scratch. This meant new computers, a new system, new programming techniques, and a new operating (executive) system. All of these had to be developed before the operational program could be developed. The main computer was the IBM AN/FSQ-31. Other Communications Computers (built by ITT) were also utilized.

1.3 The Technical Base for JOVIAL

The two major influences on the language of JOVIAL were, first, the International Algebraic Language, which served as the language architectural base, and SAGE, which contributed to the knowledge of many of the problems and ideas needed to solve the large

programming system problem. IAL was chosen as the base language for several reasons. One was that at the time it appeared as if it would become the commonly accepted standard for languages. Secondly, it seemed like a better technical basis for a new language than FORTRAN, which was the only other possibility. SAGE influenced such matters as data handling, Communication Pool, and the need for a variety of types of variables (items or elements). These things were deemed essential to the programming of SACCS and served as the basis for the first definition of JOVIAL.

1.4 Organizational and People Beginnings

The first actual contribution to the language effort that eventually led to JOVIAL was the small study which took place at SDC in California in the latter part of 1958. As stated before, it started with experimentation with the analysis of logical and mathematical expressions. And, although the intent was not, at the time, to start developing a language, that was not completely out of the realm of possibility. (But it should be remembered that up until then language development was not a wide-spread phenomenon. Within a year or two, it would become as wide-spread as certain viruses.) Key members of this study were Erwin Book, Harvey Bratman, and myself. During the latter months of 1958 I was transferred to New Jersey to work on the SACCS project. This transfer came shortly after the publication of the description of IAL in the ACM Communications that was referenced above. Based on the experience with SAGE and the reading of the IAL description, I recommended to the SACCS development managers, prior to my transfer, the use of a higher level language to program the system. To my and a number of other people's surprise, this recommendation was accepted.

Actually the final recommendation was made in New Jersey where the SACCS program was to be developed. This written recommendation was done with a paper typed on about seven or eight onion-skin pages around December 1958. The title of it was "OVIAL - Our Version of the International Algebraic Language". It was a brief description of some of the language concepts which could be implemented in order to develop the SACCS program. This paper was accepted as the beginning of the project, and the first work was begun. (I realized some years later that I had not kept that paper. There were not many copies, and it was never a formally issued document. At the time it seemed like a relatively small item and not likely to be continued much in the future, at least outside of the particular SACCS development environment. So the paper didn't seem very important to me. Now it would be of great interest to me for several reasons. One is its historical interest to me and perhaps some others. Secondly, I am quite curious what my first ideas were at that time for the language I called OVIAL.)

The project that was formed to develop this language and compilers for it was called the CUSS (Compiler and Utility System for SACCS) project. It was part of the SACCS Division of System Development Corporation which at the time was serving as a subcontractor to IEC. IEC was part of the ITT complex, formed specifically to develop the SACCS system.

During this period of the organization and beginning of the JOVIAL effort in New Jersey, the other members of the team who did some of the initial language investigation also started a project to develop a language. Eventually this language was called CLIP (Compiler Language for Information Processing) (Bratman 1959), (Englund and Clark 1961). These two activities, the one in New Jersey and the one in California, were rather independent efforts, the one in California being somewhat research oriented, while the one in New Jersey on JOVIAL was operationally oriented - and in fact with a fairly difficult schedule (which wasn't met).

The people on the CUSS project were primarily implementors. With the exception of my own experience with the language PACT in 1954 and 1955, nobody else on the project had any experience with compilers or languages. Some had had experience in the development of the SAGE system. For some members of the project, this was their first programming experience. The list of people who served at least sometime on the CUSS project (and their current, or last known affiliation, where known) follows:

Jules Schwartz (Manager)
Moe Spierer (Manager 709 JOVIAL)(CSC)
Hank Howell (Manager Q-31 JOVIAL)(SDC)
Paul McIsaac (Data Vantage)
John Rafferty (IBM)
Patricia Weaver (Unaffiliated)
Emmanuel Hayes (Century Data Systems)
Lynn Shirley (Consultant)
Jack Friedland (Consultant)
Donna Neeb (CSC)
Richard Brewer (Litton Systems)
Stan Cohn (Unknown)
Mike Denlinger (Unknown)
John Bockhorst (Northrop)
Phil Bartram (Unknown)
Frank Palmer (SDC)
Ed Foote (IBM)
John Dolan (Unknown)
Patricia Metz (Unknown)

Some of the members of the CLIP Project, in addition to Book and Bratman mentioned previously, were:

Howard Manelowitz
Don Englund
Harold Isbitz
Ellen Clark
Ellis Myer

1.5 History of the Name

Since it has been stated that this paper on JOVIAL will serve as some sort of permanent record of the language, this seems like the right place to describe the origin of the name. For many people this seems to be the most

important and best known part of the language. I have over the years met many people who know the name (and that Jules is part of it) but know nothing about the language. There is a discussion of this in Sammet's book on languages (Sammet 1969), but the following expands somewhat on that.

As stated above, the name OVIAL was the one originally recommended, which meant Our Version of the International Algebraic Language. In the late 1950s, society wasn't quite as free thinking as it is today. The name OVIAL seemed to have a connotation relative to the birth process that did not seem acceptable to some people. So, during a meeting, held approximately January 1959, at which a number of technical issues were discussed--a meeting attended by members of the staff of the SACCS Division Management, the CUSS project, and IEC personnel--the subject of an acceptable name for the language was initiated. Someone in the group, and it's not known by me who it was, suggested the name JOVIAL. This seemed like the easiest transition from OVIAL. The question then was the meaning of the "J". Since I was standing in front of the room conducting the meeting at the time, somebody--perhaps somebody other than the one who suggested JOVIAL--suggested it be called Jules' Own Version of the International Algebraic Language. This suggestion was met with laughter by the assembled group. The meeting ended shortly afterward without actually finalizing the discussion of the name.

I left town for a trip soon after. Upon my return I found that the IEC people had put in the performance contract an obligation for SDC to develop the language called "JOVIAL (Jules' Own Version of the International Algebraic Language) and compilers for this language for the IBM 709 and AN/FSQ-31." The AN/FSQ-31 computer wasn't due to be installed for quite some time, but the 709 was available and was to be used by user personnel for a variety of kinds of work. (I didn't remember who the IEC people were, but was told in 1976 by Joe Ceran, now of CSC, who was then one of the IEC personnel at the meeting.) Thus, along with the dates for various deliveries, the name JOVIAL, which included my first name, became official terms.

1.6 Objectives

Both compilers were to be delivered to the Strategic Air Command in Omaha and to be used for a variety of functions. The AN/FSQ-31 Compiler was intended for the programming of the SACCS System, the 709 for a variety of other ongoing SAC functions as well as the early development of the Q-31 Development System prior to the availability of the Q-31. The JOVIAL language was to be the same for both the 709 and Q-31. The compilers were to be programmed in JOVIAL.

1.7 Schedules and Allocated Manpower

I have not been able to find the original contract statements containing the actual schedules that were required. Whatever they were, it is quite certain that they didn't get met. As I recall, the first version of the compiler for the language was scheduled something like six months after the beginning of the effort on the IBM 709. Since the AN/FSQ-31 wasn't due to be installed for quite some time afterwards, that had a much later scheduled delivery. The number of people on the project soon after it got started was around nine and it grew to a maximum of about 15 people as the CUSS project continued. The actual first users of JOVIAL, aside from the compiler developers themselves, were to use the first processor, which was an interpreter, not a compiler, for a very simple subset of the language, which was available in about 12 months (around January 1960).

People within the SACCS development effort began to use the language for very simple things (e.g., short subroutines for conversion, calculations, expression analysis). There were somewhere around eight man-years' effort before the language began to be used. About 25 man-years of effort were expended before very heavy use of the 709 Compiler took place. The first programs to make use of the compilers (outside of the compiler builders) were the utility and executive systems that were to be used for the further development of the SACCS system. The compiler development itself utilized the early compilers. Once the earliest subset of the language was available in a machine-coded compiler, the compiler was immediately reprogrammed in that language. From then on, all versions were programmed in some version of JOVIAL.

1.8 Planning and Procedures

Today there are a variety of concepts for orderly development, including structured programming, structured design, top-down development, continuously evolving specifications, and in general rigorous planning and disciplined effort. These ideas were unknown to the original JOVIAL development. The language design actually proceeded largely in parallel with the implementation of the compilers. There were some basic concepts (e.g., IAL, SAGE data definitions) and enough definition to get started with the development, but a firm baseline never existed, and changes were made almost daily. The early documents were limited and essentially working papers, subject to considerable change. No reasonably complete "final" documents were published until the end of the major work.

The main "management technique" used was frequent interaction among all personnel, including the project leaders, where changes, techniques, and progress were discussed on an individual and group basis. Although not to be recommended as the major way to run a project, it seemed to work for this project (where most personnel, although rather inexperienced, were quite capable and hard-working). No detailed plans or schedules or systematic growth from design through testing were ever written or followed. The only real test case was the compiler itself.

The first official document on JOVIAL was published in April 1959 (Schwartz 1959) and the second in May 1959 (Schwartz 1959). These described a number of the

concepts and details of the language. Additional documents came out on the state of the language over the next six months.

Since one of the major emphases of this language was to be the development of the compiler in the language itself, many of the ideas for the language came from those who were actually in the process of developing parts of the compiler. Changes were suggested by those people as they programmed.

Many of the important parts of the language were developed in an incredibly short time. A good example of this is the Data Definition Capability, which was the original heart of JOVIAL. It was not well-defined when we embarked on the development effort. It had to be developed as the compiler development proceeded. When it was determined that we couldn't postpone the data definition any longer, it was developed in about 30 minutes. One other person (H. Howell) and I examined each possible type and structure and developed the syntax for them immediately (Howell 1960). Those who know the original JOVIAL will readily agree that that part of the language could have been developed in only 30 minutes. Many of the syntactic improvements over the years have been in that area, although the capability provided has always been well received. Other language features (such as the CLOSE Routine and Item Switch - discussed below) were decided on and added in minutes. Rarely did language features require more than a day to determine both need and structure. STRING Items (discussed below) were one exception, mainly because of objections by the compiler producers.

1.9 Documentation and Early Versions

As stated previously, the first document describing the language was published in April 1959 (Schwartz 1959). It was superseded by another in May (Schwartz 1959). The next two language documents appeared in September (Bockhorst and Reynolds 1959; Schwartz 1959). In early 1960 work began in California on JOVIAL itself in addition to CLIP, so documents began to appear in the spring of 1960 (Shaw 1960) on the language JOVIAL and some plans for JOVIAL on computers other than those being developed in New Jersey. However, those versions of JOVIAL were not identical to the East Coast version.

The early use of the interpreter (for language version J0) occurred in January of 1960, about a year after the system was started. The first compiler for the language version called J-1 (J minus one) was available for the compiler developers in the fall of 1959. That compiler was used to program the next version of the language--called J1--which was available in the winter of 1960. This was used to produce the J2 version. The J2 version was delivered in March of 1961 (Schwartz and Howell 1961). It ran on the IBM 7090, which had replaced the 709 by that time.

The AN/FSQ-31 version of J2 was delivered some months later. Usage of the 709 version really began back with the J1 version of the compiler for the development of utility systems. In Omaha, at SAC Headquarters, it began with the delivery of the 709 J2 version in March of 1961.

What eventually became a standard was the J3 version of JOVIAL. That was originally described by Chris Shaw (Shaw 1960; Shaw 1964). This version was developed in California. Chris Shaw was responsible for most of the early formal descriptions of JOVIAL.

The first formal presentation on JOVIAL was given in the early 1960s at the International Symposium on Symbolic Languages in March of 1962 (Schwartz 1962).

2. RATIONALE OF THE CONTENT OF THE LANGUAGE

2.1 Language Objectives

The actual major objective of the original JOVIAL was a language for programming large systems, although it has typically been referred to as a Command and Control Language. These were systems which required the contributions of many people. Also, they would not be constrained to utilize limited sections or instructions of the computer, nor to work with strictly computational problems. These properties certainly seem to serve the Command and Control problem, but serve equally well the programming of a compiler or operating system.

A natural result of this basic objective was the requirement for flexibility. The language had to be able to handle a variety of situations. The necessity for people to use machine language often would not be considered satisfactory, even in those cases in which machine language had always been assumed necessary in the past.

Another language objective was a reasonable level of machine independence. The fact that two compilers for two totally different computers was the original contractual objective of the language helped to achieve this latter objective (Wilkerson 1961).

2.2 Things Which Were Pretty Much Ignored In The Language

One thing which was given little attention until the first versions of the compiler began to be used to compile themselves was compiler speed. Capabilities were added to the language independent of their effects on compilation speed. This was not true of some other languages being developed at that time, one of which was NELIAC, which had similar overall objectives but tended to emphasize compilation speed to a much greater extent.

Another thing which was ignored was language elegance. An example of this was given previously in the discussion of how the Data Description part of the language was developed. Almost everyone on the project had no foundation in language design, including the managers. Consequently, although the example of IAL existed for

a relatively formal description of a language, the formality of JOVIAL tended to be ignored. This could not have been completely bad, I think. It certainly had its weak points, but in fact JOVIAL actually turned out to be very natural for people to use. Many language features were developed as actual needs were recognized, leading to useful, if inelegant, results. The Data Definition area easily fits here. It contained quite a bit of flexibility and power, along with reasonable programming transparency, but syntactically was a string of hard-to-memorize characters required in specific sequence. Long string constants were defined in a very awkward way (e.g., 18H (THIS IS AN EXAMPLE)), requiring considerable counting and recounting for changing. The use of the symbols "($" and "$)" for liberally used brackets was a bad choice. Imprecision in semantic definitions did not help in making all versions totally compatible, even when there were attempts at compatibility.

Another thing which was ignored was standardization with other efforts. Even other efforts within SDC at that time were ignored--partially because of the need for (if not the achievement of) meeting of schedules on the SACCS effort. It was also partially because of the physical distance between the JOVIAL efforts (For much of the time period of the Project, the coast-to-coast trip was by propeller-driven aircraft.) and the not unusual tendency for people to ignore other people's work. And the initial inspiration of IAL didn't lead to continued following of it. No further attention was paid to IAL (or ALGOL, as it was eventually called) after the initial design.

2.3 Things Excluded in the Language or Given Little Attention

One of the major initial exclusions of the language was input/output. This can be attributed to SAGE. In the SAGE system no program except the central control program did any input/output. Consequently, it was felt that the great majority of programs would not require input/output for the SACCS System. JOVIAL thus did not include it initially. Eventually, subroutines were written and called through Procedures (Bockhorst 1961). One of the original omissions (and still omitted) is interrupt handling or any other obviously real-time instruction or capability. Instead, JOVIAL gave access to parts of the machine through various operations and modifiers. These included access to parts of words in arbitrary groupings of bits or bytes, which allowed one to inspect interrupt or message registers, therefore obviating the absolute need for more direct commands in this area. In the initial versions of the language no real debugging facilities were added. But the Communications Pool provided capability for supporting systems to give considerable debugging capability at both the individual program and system level. Also, no data allocation statements were put into the initial version of the language.

One thing which was allowed in the language but somewhat ignored in the compiler were multi-dimensional arrays. The language provided for them, but the efficiency of handling anything more than a single subscripted variable was not stressed in the compiler. (However, single-subscripting was handled quite well.) This resulted in relatively little interest in JOVIAL for programming multi-dimensional matrix problems. It was felt, in justification of the latter, that the use of multi-dimensional arrays for the kinds of systems for which JOVIAL was to be used would be minimal.

2.4 Major Language Features

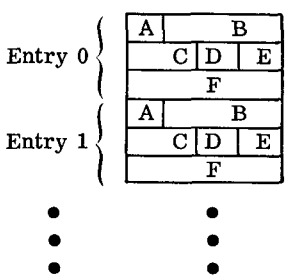
The basis of JOVIAL taken from IAL included such things as compound statements. The use and syntax of compound statements was the same as defined in IAL. The major operators were generally the same, including a switch which was similar to the IAL switch. The statement structure, labelling, and ending were the same. Procedure calls, at least initially, were similar to IAL's, and the general loop structure and FOR statements were similar.

The biggest departure (and the original one) from IAL was in the Data Definition area. Where IAL provided primarily for floating point arrays and some integer values, the JOVIAL language included at the lowest data description level entities which were called "items" (based on the SAGE term) which had a variety of types. The types included floating point, fixed point (where the scaling was specified to be part of the item), Hollerith and Standard Transmission Code character strings (in the initial version, these were limited to reside within one computer word), and status valued items. These latter represented with symbolic values finite states such as GOOD, FAIR, and POOR, or ON and OFF, or any predefined set of values. In the JOVIAL program these are referred to by the symbolic names (e.g., IF WEATHER = V(CLOUDY)), although internally they are assigned integer values by the compiler.

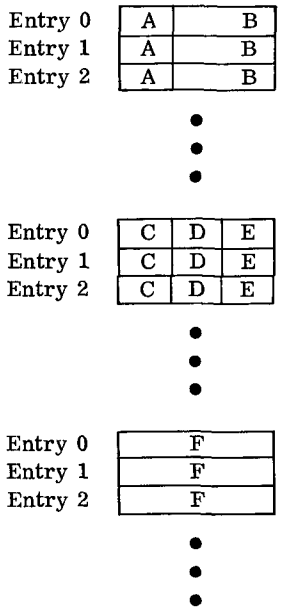
Items were assigned to the next higher level of hierarchy, which was an "entry" in JOVIAL. Each entry contained all of the elements or the items that were necessary to describe a particular object (e.g., Aircraft, Persons, etc.). All entries for an object were contained in a "table", the top level of the hierarchy.

Tables could take on several forms. These were serial, parallel, and variable.

In the serial table, all words containing an entry were contiguous. As an example, a serial table containing the items A, B, C, D, E, and F in a three-word entry would appear as follows:



A parallel table containing the same items could appear as in the following:

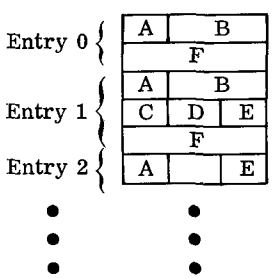


Indexing in both cases with JOVIAL was independent of the entry size or parallel/serial format. For example, the sequence:

```
FOR I = 0, 1, 10$
D($I$) = 5$
```

would set D in the first 11 entries to the value 5 for either of the above (and would work in the serial case if fewer or more words were in an entry).

It was also possible to utilize a variable format which might have a different structure for each entry, as follows:



In the variable case, there would normally be some item in each entry (say A in this example) which would provide assistance in stepping from one entry to the next. This was not done automatically by the compiler. A sequence of the following kind would be possible:

```
          FOR  I = 0$
BEGIN X1. IF   A($I$) = 3$  D($I$) = 5$
          I.= I + A($I$) $
          IF   A($I$) = 0$ STOP $
          GOTO X1$
END
```

This sequence would set all existing Ds to 5 until the end of the table, where only three word entries contained D, and the end was signaled by an A equal to 0.

Another data type was added to the language--much to my chagrin because of its awkwardness--during its early implementation. This was called STRING and it provided for a rather complex combination of character string elements in a variety of formats within an entry. This went beyond the original concept of items and character string items. (This was one of the only "committee type" additions to the early language.)

These data types and data structures provided for a comprehensive capability for access to almost any configuration of logical or arithmetic values. Items could be either full or part words. (No item could be more than one word. This was changed in later versions of the language.) Also, access to parts of items (discussed below) was provided. The point of this was not only the definition of data structures. It was to provide it in such a way that the programmer would not need to know where the item was placed within a word, which word in an entry it was in, and in most cases the size of the items and other facts about it. In some cases, the type itself did not have to be known.

Thus, the JOVIAL program could remain largely unaffected by changes in the data description. That was a crucial part of the language. Also, in some ways this capability was used to show that in fact JOVIAL would produce more efficient code for programming systems than the machine language as used for SAGE. The following type of example was used as an illustration.

In SAGE, "pseudo-instructions" (macros) were used in machine language when referring to Communication Pool items. The purpose was to prevent any assumptions about the size or position of items by programmers while producing reasonably good code for specific situations. (Programs had to be reassembled when the Communication Pool changed.) Four such pseudo instructions were:

ETR	Perform an AND to the Accumulator with a mask the size and position of the item.
POS	Shift the accumulator so that the least significant bit of the item is in the least significant bit of the accumulator.
RES	Shift the accumulator left so that the least significant bit of the accumulator moves to the least significant bit position of the item.
DEP	Deposit into the word containing the item from the proper bit positions of the accumulator.

At least one instruction was generated for each pseudo-instruction.

Thus, to add two items and store in a third, the following sequence had to be used.

```
CLA ITEM1     Move ITEM1 to accumulator
ETR ITEM1
POS ITEM1
STO TEMP      Save ITEM1
CLA ITEM2     Move ITEM2 to accumulator
ETR ITEM2
POS ITEM2
ADD TEMP      Add the two items
RES TEMP3
DEP TEMP3
```

This sequence generated a minimum of 10 instructions (on the SAGE computer). In JOVIAL, the same operation would appear in the following statement:

```
ITEM3 = ITEM1 + ITEM2$
```

Depending on the definitions of the items, this statement could generate as few as three instructions because of the capability of the compiler to analyze the entire statement.

The Communication Pool is essentially the Data Definition for all items and tables to be used by the subprograms of the system. In some versions of JOVIAL compilers the Communication Pool is syntactically identical to the Data Definition section within a program. In other versions, however, the Communication Pool actually has a different format than the Data Definitions and has more information than the JOVIAL programs themselves permit. Such things as allocation information can be put into the Communication Pool for both programs and data. The Communication Pool is used for more than compiling programs. It can also be used for such things as simulating data for testing. It allows for the entry of the item name and its value without the programmer having to specify where to place the value or its internal form. The Communication Pool also can serve as a tool for data reduction after tests are run. This is because the Data Reduction Program can examine the information as it exists after a run and use the Communication Pool definition to translate it to a readable form for the user or the tester (Fjomsland 1960).

Another significant feature of the JOVIAL language from its earliest version was the provision for access to bits and bytes of items. Modifiers for this provided access to a variable number of bits or bytes of the item, starting from any position within the item. The expressions are BIT(i, j)(item) and BYTE(i, j)(item), where i is the starting bit or byte and j the number of them. These provide a good deal of power. However, the efficiency of this form when i and j are both variable is at best adequate, but normally much worse. The simpler expression where only one or a constant number of bits or bytes was referenced can produce more efficient code.

Access to entries of tables also was given in JOVIAL. Thus, one can move whole entries other than on an item-by-item basis. A modifier was provided which provided access to the count of entries in a table for use or modification. For example, the statement NENT(TABLE) = NENT(TABLE) + 1$ increments the count of entries in TABLE.

Some other rather unique forms were added to the early JOVIAL to resolve certain problems. One was the CLOSE Statement. This provided for a closed subroutine with no entrance or exit parameters, but it maintained and could use the value of a subscript when it was within its domain of definition. The LOC modifier was added. This modifier, which stands for "(absolute) core location", provided the capability of getting to the location of a part of memory. The expression LOC (TABLE or ITEM) enabled one to reference relocatable information. Today, pointers can be used for this without some of the awkwardness and inadequacy available only with LOC. Another kind of flexibility which was provided (and which created problems for compiler writers) was the fact that the step-factor in the FOR statement (For I = A, B, C; with B being the increment or the decrement for I) was allowed to be a variable. It could be either negative or positive, and in fact during the loop was allowed to change sign. Of course, that was not the normal case, but the need for providing for it in the compiler was one of those things which created some of the early compiler problems in achieving efficiency and speed.

NENT, BIT, BYTE, LOC, ENT, MANT and other terms are examples of functional modifiers which were added during the early development of JOVIAL. These were utilized to satisfy a number of the problems which required language solutions. They can normally be used in "right" or "left side" expressions. For example, the statement

```
BIT ($I$)(ITEM) = 1$
```

causes the Ith bit of ITEM to be set to 1. (When the number of bits or bytes is 1, it doesn't have to be specified.)

Although the purpose of the language was to provide for machine oriented programming without the use of machine language, the originators were realistic enough to know that there would be a requirement for the use of machine language, at least occasionally. Particularly in the early versions, at a minimum, access to input and output required some machine language operations.

Access to machine language is easy in JOVIAL. The operator DIRECT signals the beginning and the operator JOVIAL the end of machine language. Within the machine language, it was expected that users would need access to JOVIAL (and Comm Pool) defined items and tables. Thus, an operator was defined to give

access to these from within the machine language part of the program. This operator was called ASSIGN and it permitted one to place an item in the accumulator or the accumulator to an item. The compiler automatically took care of the shifting of the item as it would in normal JOVIAL statements. (The scaling of the accumulator was specified by the programmer in the ASSIGN statement.) Actually, the ASSIGN combined the functions available in the pseudo-instructions of SAGE shown above. Another capability of interest was called the Item Switch. A Numeric switch is provided as in IAL for a series of branches based on normally continuous integer values of an item. But this was not sufficient for things like status items which were, first of all, symbolically named where users could not assume values for the names. Also, the Item Switch could be used for other item types such as character strings, which can have quite random values.

2.5 Language Design Methodology

The primary technical driving forces for JOVIAL are quite clear. Given that the basic structure would be that of IAL, the major question was what would have helped in the programming of SAGE, since it was assumed that the programming of SACCS would have a similar set of problems. These included the use of a large number of people, different types of data, more logic operations than calculation, need for item packaging to save space and access certain machine registers, etc. For example on the SAGE computer there was no floating point arithmetic. Everything was fixed point. So, although the SACCS computer had floating point, it was assumed that fixed point arithmetic would be useful for it as well. Other major influences were those things found necessary to program the compiler itself. That was the source of most of the early modifications to the language. Other ideas came from early users who were beginning to use the language. They discussed and wrote about problems that they had found in using the language.

It would be quite an extreme exaggeration to say that a strict methodology or control system was used in developing the JOVIAL language. As stated earlier, much of the development came from innovation when needs were recognized. Most problems were resolved by adding to or modifying the language. Many problem resolutions had as a goal the obviation of the need to resort to machine language. This mode of working led to some extremely fast decisions, some awkward syntax, some difficulties for the compilers, but a quite practical and usable language.

The basic approach was essentially as described before. The base language was chosen, and it was initially implemented in a variety of processors. Compiler work was begun with a minimal form of the language. Data Definitions were added for the total range of data that was to be needed both for the compiler and for the application systems. As time went on, modifiers and other forms were added as the need arose. The language continued to evolve for some years thereafter, both within the original effort and in California and other places as the language was used elsewhere.

3. A POSTIERI EVALUATION

It is always difficult to provide an accurate objective account of a product's success or failure (even when one's name isn't directly involved). This is true for many reasons, including the fact that nothing is ever absolutely good or bad. But the following attempts to present a reasonable assessment of JOVIAL's effect on the world.

3.1 Meeting of Objectives

In general, it seems that JOVIAL met its goals. There have been a large number of systems of many types produced using JOVIAL. Most of these have been for areas which up until the time when JOVIAL was developed were not thought programmable in other than machine language.

For the first decade of use there were really never very many serious complaints about the language capability (after the first year or two of development). The difficulties and the complaints tended to be largely the result of early implementations. The early compilers were, to put it mildly, not very reliable. They were extremely slow. The first compiler on the 709 took eight hours to compile itself. In those days the probability that one could compile for eight straight hours and get a properly punched binary deck for output was less than .5. Of course this slowness was also recognized when compiling other sizeable programs. This, combined with the fact that the compiler might produce erroneous results, could dampen even the most ardent enthusiasm. But even with these kinds of problems, there was little drive on the part of early users to disassociate themselves from the use of the language. Rather strong statements were used to point out compiler problems, but once many of the early difficulties were overcome with the compiler the language was considered quite satisfactory.

Another interesting fact was pointed out by someone responsible for teaching early users how to use the language (Rizzo 1962). The initial capability didn't provide much tutorial assistance, so that learning of the language was not easy and the documentation, although it improved with time, wasn't the best for beginners. But those who were responsible for training and criticized this lack of good teaching aides also pointed out that once people got over the hurdle of learning JOVIAL there were very few questions and very few problems in using what they learned. Most people found the language relatively easy to retain and use for everyday work.

The fact that the SAC organization under Colonel William Shirey in Omaha (the first non-SDC users) continued to use the language on a fairly heavy basis, despite some early tough going, was critical in keeping

the JOVIAL language alive. This led to an eventual spread of use in the Air Force. It is also true that its use spread without support from aay major manufacturer. Almost all the JOVIAL compilers that were produced until the mid-late 1960s were done under contract to a government agency for a specific system. Manufacturers were evolving COBOL and FORTRAN at that time. Of course a number of other languages were also in development in the period from 1960-1965, such as MAD, NELIAC, ALGOL 60, SNOBOL, JOSS, and hundreds of others. But JOVIAL maintained its progress during that period.

The objective of moving JOVIAL-coded programs and the compilers from computer to computer was partially met. The initial effort in New Jersey provided compilers for two computers for the same language almost simultaneously. When one moved a program from a computer with six characters in a word to another which had eight-character words, the fact that the early compilers didn't provide for character string items of more than one word in length certainly made the portability imperfect. Similar results occurred because of the restrictions on scaled fixed point items. However, considering the type of programming which was permitted (much more variable than possible with FORTRAN), portability was reasonable. I'm not sure that it had too much effect on the ultimate success or failure of the language. Most users tended to program for a machine and stay with it on that machine. It is true that very few JOVIAL compilers handled the same version of the language. After the J2 effort, most others produced different subsets of J3.

The objectives for JOVIAL that were set originally were not changed. The flexibility and power of the language continued to be objectives throughout and were reasonably well satisfied. In fact, as people developed the language further there were actually some restrictions put in that did not exist in the early version. These included restriction to equal types for formal and entrance parameters of procedures. Restrictions on the step-factor of the FOR Statement were implemented in some later versions.

3.2 Contributions of the Language

It is hard to say what contribution to programming technology in general or to other languages JOVIAL actually made.

There are a number of languages which contain some of the capabilities or characteristics of JOVIAL. Most often they are in a different format and except for certain situations it's not possible to directly relate that language to JOVIAL itself. A good example of this is PL/I. JOVIAL knowledgeable reviewers of PL/I when it was first developed pointed out that many of the characteristics of JOVIAL had been adopted in PL/I. However, there's no official acknowledgement of the use of JOVIAL as one of the bases for PL/I to my knowledge.

Nevertheless JOVIAL had things to offer and, whether directly or indirectly, should have contributed something to the technology. It was one of the first system programming languages and helped show that languages were capable of serving that purpose. It was one of the early, if not the earliest, compilers coded completely in its own language. It provided for data structures, types, and operations that permitted programming outside the realm of the strictly computational. It was even possible to use it for commercial programming. Although certain problems occurred in the early versions with rounding and report formatting, JOVIAL was used for the programming of a wide variety of systems. It provided for access to subparts of elements and words, which in early computers with small memories was very important, and is still valuable today for a variety of things. It was a language really aimed at flexibility and non-constraint on the program. These were all contributions and valuable factors for the people who used it. They have helped maintain the language's use for a lengthy period, although the initial intent of the language was actually intended for only one specific set of users and a single system.

There have been quite a few versions of JOVIAL. I don't know them all and within each version there were variations with different implementations, so that there is unlikely to be a comprehensive list of all the capabilities that actually were in each version. A list of the names for different varieties or versions of JOVIAL includes: J0 (Olson, Petersen, Schwartz 1960), J-1, J1, J2 (Schwartz and Howell 1961), J3 (Shaw 1961; Shaw 1964), and J3B (Department of the Air Force 1967). This was the original Air Force standard version, J3 being the version developed in California in parallel with the development of J2 in New Jersey for the SACCS contract. Other versions included J4, J5, and J5.2, the latter two being outgrowths of a version called Basic JOVIAL (Perstein 1968), which was actually an attempt to provide a working version with a fast compiler. (For example, it didn't include fixed point arithmetic). This was developed because of the criticism of the slowness of the compiler. Basic JOVIAL was itself derived from an experimental version called JX2 (developed in 1961), which rivaled other compilers in speed but did not have enough language. When timesharing came in the early 1960s, another interpretive version of a subset of JOVIAL was produced. It had language capability similar to JOSS (if not the elegance) in that it provided for arithmetic, formatting, simple terminal input/output, and other capabilities useful for interactive users. That was a version of JOVIAL which had quite different objectives than the original system programming language. This version was called TINT and was described in a paper at the 1965 IFIP Conference (Schwartz 1965).

Although the preceding references to the various versions of JOVIAL may appear rather mind-boggling, it is actually probably an understatement. With the exception of some Air Force sponsored versions of J3, little control was exercised over the language. So, for

reasons of implementation pressure, computer idiosyncrasies, particular needs or lack of needs, and individual choice, one will note differences within "versions" as well as among different ones. New versions were sometimes labeled after compilers were developed. However, the major aspects generally were implemented, and there was "near-compatibility" in most versions.

JOVIAL has served as a direct contributor to at least several languages. One language which is in use today in Europe is CORAL (Woodward and Wetherall 1970). Many of the initial concepts for CORAL were taken from the JOVIAL of the early 1960s. A language which has been used by Computer Sciences Corporation for much of its systems programming work is called SYMPL. It consists of elements of JOVIAL and FORTRAN and some other properties developed at CSC. SPL, a Space Programming Language which was developed by the Air Force Space Command some years ago, was based to a large extent on the experience with JOVIAL. Others, such as CMS II used by the Navy and perhaps PL/I as stated before, have had either a direct or some indirect relationship with JOVIAL.

In the early 1970s a group was formed by the Air Staff under Leo Berger to produce a new standard version of JOVIAL. They developed a specification for a language called J73 (Department of Defense, date unknown). There have been several implementations of it, and it is being used on a variety of computers at the present time. This is the first version which, by design, is not upward-compatible with other versions. Some version of JOVIAL, or several versions of JOVIAL in some cases, have existed on just about every major manufacturer's computers and some less well-known computers over the years. A minimum of ten companies have produced JOVIAL compilers.

JOVIAL was adopted in the mid-1960s as a standard language (Department of the Air Force 1967) for the Air Force and that's where most of its use has been. It also was for a short period of time adopted by one part of the Navy (NAVCOSSAT). This adaptation of JOVIAL for the Navy was done on the basis of a comparative study comparing JOVIAL, a number of other languages, and machine language. However, there were a number of criticisms of the particular study, and the future of JOVIAL in the Navy became somewhat tenuous. (An interesting side-light to that particular study was the fact that the first time it was run the machine language version of the program was assumed to be the base with which to compare other languages' efficiency and size. But the machine language program actually came out worse in these respects than some of the higher level languages involved. This caused the people responsible for the study to rerun it. Machine language fared better the second time.)

In 1961, members of the Rand Corporation conducted a study to compare JOVIAL with FORTRAN to see if JOVIAL or any language really merited being used as a Command and Control language at the time. The actual test consisted of seven problems, five of which utilized multi-dimensional arrays within several statements. The basis of the test was to see how well these would be compiled. FORTRAN won handsomely on all of the array problems with respect to generated code.

JOVIAL compared favorably on the non-matrix problems. JOVIAL was much worse in compilation time, which didn't help its cause. In any case, based on that experiment, which seemed inadequate for the choosing of a Command and Control language, the people doing the study recommended that no language be adopted for Command and Control use at that time. The main thing proven by some of this is that we haven't mastered the art of comparative studies. The conclusion may have been right, but it always seemed to me that if so, it was in spite of the method used to reach it. (Haverty and Patrick 1963.)

JOVIAL has been used by a variety of services and some other organizations, including the FAA, the Air Force (of course), the Navy (for some things and a variety of research and system projects).

Since the earliest versions of JOVIAL, the most significant changes that have been added are in the area of input/output. There are now a number of input/output commands in the language. Dynamic control of data allocation has been entered through data structure definitions, and pointers have been added in J73. Some syntax improvements have been provided in the language, particularly in the data definition area and in some of the modifiers. Some types have been purified to a certain extent. Originally, arrays and tables were separate. For example, when one defined an array he couldn't include items, and arrays couldn't be part of tables. That has been changed. There are a number of other changes which have taken place. Many of the key ones, of course, are being implemented in J73.

3.3 Mistakes and/or Omissions

The first major omission which constrained the use of JOVIAL considerably was its lack of input/output. This of course was actually a planned omission from the beginning. But its use for systems other than large systems where it is assumed that just one or several routines would service all input/output is of course diminished by this lack of input/output for individual programs. The lack of formatting and the ability to output or input by format or other convenient means (which is still an official omission from the language) has kept the language at arms length for many applications. Actually formatting has been implemented in at least a few versions. One which included formatting was the interpretive version called TINT (Schwartz 1965). That was essential for it to operate as an interactive user oriented language. JOVIAL syntax awkwardness in certain areas probably didn't affect its use from the point of view of those who started using the language but helped keep certain communities uninterested in the language. The early implementation problems, particularly the speed and size of the initial compilers, which

were 50,000 or more instructions, helped keep down any overwhelming use that might have occurred otherwise. To a great extent these implementation problems were due to the fact that features which made compiling difficult were sometimes added with reckless abandon. And, of course, the lack of efficiency in certain areas--particularly multi-dimensional arrays--restricted its application by programs which required them. It is unlikely that much interest would have been generated in JOVIAL for programming outside of its major areas of current use even if it had been much improved in these areas. Other languages such as FORTRAN and COBOL are accepted for programming these problems.

3.4 Problems and Trade-Offs

The hurry-up mode in the early versions created incompatibilities among different versions, particularly those separated geographically and those developed by different groups. And, of course, this detracted from portability among versions. The overall emphasis was for flexibility and generality of the language, which in turn led to a variety of implementation problems. This caused reliability, speed, and code generation difficulties. But the trade-off was made. There is a question whether in the long run it was a bad trade-off. The flexibility and the generality were probably worth some initial difficulties. Time for implementation was always a problem in this development. The language was defined rapidly, and the compilers were written quickly. There wasn't a lot of planning by the individuals and the managers concerned, but I think to some extent that this hurrying was as much a question of personalities as pressure. The people who were involved liked to do things in a hurry, were not very academically or research oriented, and they liked to see things run (typical system programmers). So I sometimes wonder if there hadn't been imposed deadlines (which in many cases of course were missed anyway) whether the people would have done a significantly more studious effort. (I'm certain that the tendency to "produce" rather than "study" is true for me, who was responsible through J2.)

4. IMPLICATIONS FOR CURRENT AND FUTURE LANGUAGE

Certainly, whether or not directly a stimulus, the work that went on in the early JOVIAL is now represented in a variety of ways. System programming languages are quite common. Complex data definition capability in a variety of languages certainly is not unusual today. Much of the system programming language work today is based to a large extent on PL/I, but it is also based on other languages as well. This concept within JOVIAL did prove feasible and valuable.

The original JOVIAL work was not a committee effort. There were relatively few people who actually helped design the language. Most of the people involved hadn't done this kind of work before and were basically implementors, or programmers. The fact that it wasn't a committee effort in many ways created the positive characteristics of the language as well as some of its negative ones. The language did maintain its original objectives, and its major characteristics remained fairly consistent throughout its early development. Perhaps the influence of others who had some better or more experience in this area could have been valuable, but it is also quite possible that with a variety of other contributors, it would have lost some of its flavor. For initial efforts on language design, an individual driving effort is probably a good idea.

I'm not certain what the future of JOVIAL is. The Department of Defense is trying to standardize once and for all on a language. This is one of those efforts which is going to take a long time to resolve. Meanwhile JOVIAL itself is being implemented and used for a number of projects. It has of course much more competition now in the areas for which it was intended. Also, competition from entrenched languages like COBOL and FORTRAN for things outside of what are typically JOVIAL applications, or considered to be JOVIAL applications, is much too intense to assume that it'll ever go beyond that. It's probably more likely that it will continue to have about the same percentage of use in the world as it has had until now. Hopefully, it will have influenced the Department of Defense standard language which is eventually chosen to some degree.

BIBLIOGRAPHY

Bockhorst, J. and Reynolds, J. 1959. Introduction to "JOVIAL" Coding. Lodi, NJ: System Development Corporation SDC Report FN-L0-139.

Bockhorst, J. 1961. JOVIAL I/O (7090). Paramus, NJ: System Development Corporation SDC Report FN-L0-34-3, S1.

Bratman, H. 1959. Project CLIP. Santa Monica, CA: System Development Corporation SDC Report SP-106.

Department of the Air Force. 1967. Standard Computer Language for Air Force Command and Control Systems. Washington, DC Air Force Manual AFM 100-24.

Department of Defense. Military Standard, JOVIAL (J73/I). MIL-STD-1589.

Englund, D. and Clark, E. 1961 January. CLIP Translator. Communications of the ACM 4(1): 19-22.

Haverty, J. P. and Patrick, R. L. 1963. Programming Languages and Standardization in Command and Control. Santa Monica, CA: Rand Corporation Memo RM-3447-PR and DDC Document AD-296 046.

Howell, H. L. 1960. JOVIAL - Variable Definition For the 709 Translator, Paramus, NJ: System Development Corporation SDC Report FN-L0-34-2-52.

Kennedy, P.R. 1962. A Simplified Approach to JOVIAL. Santa Monica, CA: System Development Corporation SDC Report TM-555/063/00.

Melahn, W.S. et al. 1956 October. PACT I (A series of 7 papers). Journal of the Association for Computing Machinery 3(4): 266-313.

Olson, W.J., Petersen, K.E., and Schwartz, J.I. 1960. JOVIAL and its Interpreter, a Higher Level Programming Language and an Interpretive Technique for Checkout. Paramus, NJ: System Development Corporation SDC Paper SP-165.

Perlis, A.J. and Samuelson, K. 1958 December. Preliminary Report - International Algebraic Language. Communications of the ACM 1(12): 8-22.

Perstein, M.H. 1968. Grammar and Lexicon for Basic JOVIAL. Santa Monica, CA: System Development Corporation SDC Report TM-555/05/01A.

Rizzo, M. 1962. Critique of the JOVIAL User's Manual, FN-L0-34-3, Paramus, NJ: System Development Corporation SDC (internal use only) Report N-L0-2109/000/00.

Sammet, J. 1969. Programming Languages: History and Fundamentals. Englewood Cliffs, NJ: Prentice-Hall.

Schwartz, J.I. 1959. Preliminary Report on JOVIAL. Lodi, NJ: System Development Corporation SDC Report FN-L0-34.

Schwartz, J.I. 1959. JOVIAL - Report #2. Lodi, NJ: System Development Corporation SDC Report FN-L0-34-1.

Schwartz, J.I. 1959. JOVIAL - Primer #1. Lodi, NJ: System Development Corporation SDC Report FN-L0-154.

Schwartz, J.I. 1960. JOVIAL - A Description of the Language. Paramus, NJ: System Development Corporation SDC Report FN-L0-34-2.

Schwartz, J.I. 1960. JOVIAL - Clarifications and Corrections for FN-L0-34-2. Paramus, NJ: System Development Corporation SDC Report FN-L0-34-2-51.

Schwartz, J.I. 1962. JOVIAL: A General Algorithmic Language. In Proceedings of the Symposium on Symbolic Languages in Data Processing. pp. 481-493. New York: Gordon and Breech.

Schwartz, J.I. and Howell, H.L. 1961. The JOVIAL (J-2) Language for the 7090 Computer. Santa Monica, CA: System Development Corporation SDC Report FN-6223/100/00.

Schwartz, J.I. 1965. Programming Languages For On-Line Computing. In Proceedings of the IFIP Congress 1965. Vol. 2 pp. 546-547. Washington: Spartan Books.

Shaw, C.J. 1960. The Compleat JOVIAL Grammar. Santa Monica, CA: System Development Corporation SDC Report FN-4178.

Shaw, C.J. 1960. The JOVIAL Lexicon: A Brief Semantic Description. Santa Monica, CA: System Development Corporation SDC Report FN-4178, 51.

Shaw, C.J. 1960. Computers, Programming Languages and JOVIAL. Santa Monica, CA: System Development Corporation SDC Report TM-555, Part 1.

Shaw, C.J. 1961. The JOVIAL Manual, Part 3, The JOVIAL Primer. Santa Monica, CA: System Development Corporation SDC Report TM-555/003/00.

Shaw, C.J. 1964. Part 2, The JOVIAL Grammar and Lexicon. Santa Monica, CA: System Development Corporation SDC Report TM-555/002/02.

Tjomsland, I.A. 1960. The 709 JOVIAL Compool. Paramus, NJ: System Development Corporation SDC Report FN-3836.

Wilkerson, M. 1961. JOVIAL User's Manual. Subtitled JOVIAL Language Specifications for 7090 and MC Compilers. Paramus, NJ: System Development Corporation SDC Report FN-L0-34-3.

Wolpe, H. 1958 March. Algorithm for Analyzing Logical Statements to Produce Truth Function Table. Communications of the ACM 1(3): 4-13.

Woodward, P.M. and Wetherall, P.R. 1970. The Official Definition of CORAL 66. Her Majesty's Stationery Office.