SOFTWARE DEVELOPMENT METHODOLOGY AND TECHNOLOGY for the TALON COMBAT SIMULATION MODEL

Henry Kleine
10045 Wisner Avenue
Mission Hills, California 91345

The subject of this paper is a methodology and technology used for the program development and documentation of the TALON Combat Simulation Model. TALON was developed for the Air Force at Nellis Air Force Base, Nevada. It is a broadly scoped model encompassing many functional elements such as Ground Operations, Air Operations, Reconnaissance, Intelligence, etc. Because of the size and scope of the model, heavy emphasis was placed on developing and using state-of-the-art methodology for program design, coding, documentation, and configuration management. As a result of this emphasis, many new methods and techniques were developed and implemented in the TALON project with satisfying results. This paper contains examples taken from the TALON model but, it should be noted that the methods and techniques themselves can be applied generally to any software development task.

1. INTRODUCTION

This paper deals with software development methodology used to construct a large, multi-faceted simulation model. The subject of software development methodology for large scale models is important because with the great strides being made in increasing machine capacity, software development technology has become the constraining force on our abilities to construct such models.

The program for which this methodology was designed and used is the TALON simulation model. The TALON model was developed for the purpose of studying the effects of tactical air power on ground combat operations. In its current configuration, TALON is comprised of fifteen diverse functional elements such as command-control-communications, etc. A requirement of the program design was that each of these functional elements had to be de-coupled to the greatest extent possible in order to support independent development and, where feasible, independent operation. The input data required to support each of these elements are large, complex and varied. In addition to these data the program supports user interaction during execution and graphic display of data and battle scenes. The program consists of more than twenty thousand lines of code written in the SIMSCRIPT II.5 programming language.

Each of these characteristics of the TALON program contributed to the difficulty of building a model which can continue to evolve and remain viable throughout every stage of its development. This paper deals with the techniques and tools used to meet these and other problems which had to be overcome to meet this objective.

2. SOFTWARE DEVELOPMENT TOOLS

Two software development tools, The SIMSCRIPT II.5 Programming Language[2] and the Software Design and Documentation Language (SDDL)[3] together formed the basis for the techniques and methodology used in the project. SIMSCRIPT is a well known simulation programming language which has been in existence, in various forms, since 1963. SDDL is a relatively new general purpose software development tool. It was initially intended to be used to express and document program designs but its scope has since evolved to include any subject which requires structured organization for effective expression and documentation. It is especially useful for processing programming language documents (code), enhancing the information content of these documents by adding table of contents, cross reference tables, indented formatting to emphasize structure, flow lines to indicate structure escape statements (e.g. exit loop) and module invocation statements (e.g. call). It also diagnoses incorrect structures and provides appropriate error messages. Several of these features are now being added as standard capabilities to programming language compilers and more can be expected in the future. Since the concepts of Structured Programming[1,6,7] are fundamental to SDDL, it is imperative that it be used with a programming language which also supports these concepts. SIMSCRIPT and SDDL are ideally suited for each other, not only for their compatibility in this regard, but additionally because they both allow redefinition of key words allowing an excellent match between the two languages. In the case of SDDL, keyword definition is a basic part of the language, and in the case of SIMSCRIPT, the simple replacement macro provided by the language is sufficient to match the two systems with only minor alteration to the SIMSCRIPT standard language.

3. SOFTWARE DEVELOPMENT METHODOLOGY

3.1 Program Modularization

An effective and often used method for modularizing a program is to divide it into program functional elements such as data declarations (and structure), initialization, input data handling, events (in the case of simulation models), output processing, support routines, and low level utility routines. This division technique was followed in the TALON development, but in addition, a higher level modularization was needed because of the many distinct operational (i.e., military mission) functions in the TALON model. To meet this requirement the model was modularized with respect to these missions while retaining the more commonplace subdivisions within each of these modules. Thus the structure, as exemplified below in an abbreviated list, Figure 1, consists of a two level hierarchy of modules.

```
General program system functions
   System related data declarations
   System initialization
   System input processing
   Driver and initial event schedulers
   Output routines (user greeting and other start up info.)
   Support routines general to all modules
   System input data

Ground movement module
   Summary description to ground movement functions
   Ground movement data declarations
   Initialization
   Input data processing
   Events related to ground movement
   Support routines for ground movement functions
   Ground movement input data

Artillery module
   Summary description of the artillery functions
   Artillery data declarations
   Initialization
   Input data processing
   Events related to artillery functions
   Support routines for artillery
   Artillery input data
```

TALON MODULARIZATION EXAMPLE
Figure 1.

The modularization hierarchy shown in Figure 1 is a schematic conceptualization which includes program descriptions, data declarations, instruction modules and initialization data. This information was developed and organized in separate files for each mission module. In practice, these data require different organization for the purposes of SIMSCRIPT compilation, SDDL processing, and data input for running the model. This was accomplished by providing an automatic mechanism to extract the appropriate data required for each operation and merge them into a single file for processing. For processing the source code with SDDL the data are input as shown in the schematic, producing a document organized by mission function. The required extract and merge operations for the purposes of SIMSCRIPT compilation and input data preparation for program execution are shown in Figure 2.

```
SCHEMATIC OVERVIEW                    COMPILATION LAYOUT

System functions (SS)                 SIMSCRIPT PREAMBLE
  Data declarations                     (SS)Data declarations
  Initialization                        (GM)Data declarations
  Data loading                          (AT)Data declarations
  Driver
  Output routines                     SYSTEM ROUTINES
  Support routines                      (SS)Initialization
```

```
 System input data

Ground movement module (GM)
  Function description
  Data declarations
  Initialization
  Data loading
  Events
  Support routines
  Input data

Artillery module (AT)
  Function description
  Data declarations
  Initialization
  Data loading
  Events
  Support routines
  Input data
```

```
  (SS)Data loading
  (SS)Driver
  (SS)Output routines
  (SS)Support routines

GROUND MOVEMENT
  (GM)Function description
  (GM)Initialization
  (GM)Data loading
  (GM)Events
  (GM)Support routines

ARTILLERY
  (AT)Function description
  (AT)Intialization
  (AT)Data loading
  (AT)Events
  (AT)Support routines
```

```
PROGRAM INPUT DATA LAYOUT
  (SS)Input Data
  (GM)Input Data
  (AT)Input Data
```

TALON Data Extraction and Merge Schema
Figure 2

3.2 SDDL - SIMSCRIPT Interface

SDDL is a computer supported language processor with built-in knowledge and capabilities specifically related to the software development process. The foundation of SDDL is based on the concepts of structure and abstraction as exemplified in the technology of Structured Programming. Since one of the capabilities of SDDL provides for user definition of the keywords which dictate the production of the indentation and flow lines which are used to display structures, it can be adapted to most programming languages which support Structured Programming. In some cases this adaptation requires some minor alteration of the programming language in order to obtain the best interface, and since SIMSCRIPT provides a simple word replacement macro capability the two systems are ideally matched. Furthermore, SIMSCRIPT and SDDL both allow punctuation characters to be used as part of the name of a variable. Some examples of the TALON SIMSCRIPT code processed through SDDL are shown below. Note that in the SIMSCRIPT source file the program statements are all left justified with no indentation as shown in Figure 3.

```
ROUTINE SS.INITIALIZE.THE.PROGRAM
"FIRST, ADDRESS GCS (FORTRAN) INTERFACE
LET SS.GRAPHICS.INDICATOR = 0 "OFF"
IF DIM.F(PARM.V(*,*))) GREATER THAN 0
FOR I = 1 TO DIM.F(PARM.V(*,*)
WITH PARM.V(I,1) = "GCS" AND PARM.V(I,@) = "YES"
DO
LET SS.GRAPHICS.INDICATOR = 1 "ON"
CALL USTART
CALL USET("HARDWARE")
CALL USET("INCH")
LEAVE
REPEAT "ONLY ONCE"
ALWAYS
RESERVE GX.GCS.PRINT.BUFFER(*) AS 8
"NEXT, INITIALIZE I/O FILE VARIABLES
LET SS.INPUT.DATA.FILE = 1
        ...
```

Program Source Data File (Partial listing)
Figure 3

Examples of the formatting introduced by the SDDL processor can be seen in Appendix A, page 18 (page numbers in Appendix A are not numbered consecutively since most of the pages of the original document were omitted.) Note the indentation used to delineate the structures (IF - ALWAYS, DO - REPEAT), the flow lines to the left for structure escape statements (LEAVE), and the flow lines to the right for subroutine calls. The numbers shown in parentheses on the right margin provide the document page number where the module being called can be found. In addition to this formatting, the SDDL processor automatically provides a Table of Contents, Module Forward Reference Tree, Module Cross Reference Table, and miscellaneous user controlled cross reference tables. The line numbers at the left margin correspond exactly to the line numbers on the input file in order to facilitate data base editing.

3.3 TALON Coding Conventions

Comment Structure. A typical program subroutine performs several major functional steps in succession. These major function blocks require a special comment to set them off from the rest of the program structure and thus clearly display to the reader the important, high level functions performed by the subroutine. This requirement was implemented in the TALON model by means of the FIRST - NEXT Comment Construct. The first of these comments begins with the SDDL defined keyword FIRST and subsequent comment lines begin with the keyword NEXT. These comment lines are designed to be ignored by the SIMSCRIPT compiler while being recognized in an important way by the SDDL processor. SDDL recognizes the keywords FIRST and NEXT and provides block indentation for the included lines just as in the case of the IF - ALWAYS, and DO - REPEAT block constructs. This comment structure is superior to ordinary comment lines because the indentation clearly delineates the scope of the comment. An example of the comment structure can be seen in Appendix A, pages 18 (lines 648, 663, 672), and 423 (lines 651, 661, 665). Note how the comment lines describe the high-level actions being performed and how the indentation delineates the scope of the comment.

Section Titles and Program Description Paragraphs. The SDDL processor can be directed to enclose lines of text within a box formed by a specified punctuation character in order to provide title pages and highlighted comment paragraphs. This capability has been used in the TALON program to place titles at the beginning of both levels of program modules (Appendix A, page 10). These titles are automatically entered in the table of contents with indentation provided to show the subordinate relationship (Appendix A, table of contents). Other boxed in text segments (i.e., not title pages) are used for the description and explanation at the top of each routine or function (Appendix A, page 423, lines 632-637).

Prefix Notation for Variable Names. A program the size of the TALON model requires a very large number of variable names, many of which refer to similar physical data. To improve the readability of the program the names of all global parameters were prefixed with a two letter abbreviation of the mission function to which they were related. This prefix is very useful for assuring the uniqueness of variable names, and providing clarifying information for the reader. With very little practice the reader becomes accustomed to ignoring the prefix except when needed. Examples can been seen on several pages and in the cross reference tables of Appendix A.

SDDL Cross Referencing Capability. SDDL provides a user controlled cross referencing capability based on the use of selected punctuation characters within the text of the document. SIMSCRIPT allows the period (.) to be used in a similar manner to form a single identifier out of several words joined by periods. E.g., THIS.IS.A.SINGLE. IDENTIFIER and HERE.IS.ANOTHER. The combination of these capabilities provides a means for obtaining a cross reference listing of all identifiers which include a period in their name. Note that since the cross reference tables are alphabetized, the variables associated with a particular system mission will be grouped together since they will all have the same prefix.

In addition to this cross reference table, others can be produced by means of using other punctuation characters within SIMSCRIPT comments. This will result in the identifiers being ignored by SIMSCRIPT and caught for cross referencing by SDDL. Examples of this usage include revision notation, program portability considerations, and debug statements. SDDL also provides a means for causing portions of a line to be shifted to the right margin. This is especially useful to prominently display the revision marks mentioned above. See Appendix A for examples.

3.4 Data Base Preparation

Generation of the voluminous, varied and complex TALON program data base requires the skill and knowledge of experts in several mission function areas, such as, ground, air, artillery, intelligence, command and control, air defense, reconnaissance, etc. Since the data have to be presented in a large variety of formats and organizations in accordance with the requirements of the simulation model, methodology was developed to make the task more manageable for the data preparers. This methodology is comprised of (1) an Input Format Specification Document (IFSD) which presents the rules for data preparation, (2) a means for inserting commentary directly into the data base, (3) a procedure which automatically retrieves that data from separate files and merges them into a single file for input to the program, and (4) conventions for identification and selective input of each data set. Appendix B provides an example of this data base preparation concept. It is comprised of a few selected pages taken from the TALON IFSD and a test data base. The first part of the document contains excerpts from the IFSD and the second has excerpts from the sample data base. Note the automatically supplied page references between the two sections on pages 35 and 72.

Input Format Specification Document. The IFSD is a collection of data preparation algorithms, (i.e., a "program") which provide step by step descriptions for preparation of the TALON data base. These algorithms were developed using the same techniques used to produce a computer program design. The algorithms are presented in a top-down, hierarchical structure. At each level the step by step instructions are given in terms of sequences, iterations, and conditional branches. In addition to these data sequencing and formatting rules, the IFSD provides data attribute information such as, mode, units, range, and allowable values. The SDDL processor provides a means to set off this data attribute information by automatically right shifting so as to line it up at the right margin of the document. A special mark, ($$), is also placed in this position to emphasize and differentiate between rules explaining how to layout and organize the data and actual data input requirements.

Data Base Commentary. To enhance the readability of the actual data, a means to insert comments in the data base was provided. This was accomplished by adopting the SIMSCRIPT comment delimiter convention of two consecutive primes ("), and writing a small editing program to prescan the data and remove these comments. Data commenting techniques were developed to take advantage of the SDDL formatting capability, so that by processing the data base through SDDL it could be displayed in a well structured, formatted manner including table of contents and cross reference tables. Finally, additional comment lines were added so as to reference information from IFSD to data base and vice-versa. With this last step it was possible to process the IFSD and the data base through SDDL together thereby merging them into a single coherent document.

Data Retrieval and Merging. The task of prescanning the data base to delete comments and merging the remaining data into a single file was accomplished by a simple computer job control and system editor macro.

Mission Data Set Identification. The TALON data base is composed of several (currently fifteen) distinct, independent data bases which must be merged for input to the program. Furthermore, the program design permits the user to select or deselect entire mission functions which are not needed for a particular study. To fill this requirement a separate banner or label was supplied for each mission data set for the TALON program to key on while reading data.

4. SUMMARY

The software development methodology described above is aimed at reducing the difficulties of producing large scale simulation models. These methods were enabled by the combined use of the Software Design and Documentation Language and the SIMSCRIPT II.5 Programming Language. SDDL can be used effectively with other programming languages which support structured programming concepts (e.g., PLI, PASCAL) but it works best with SIMSCRIPT. SIMSCRIPT is an excellent simulation language and blends well with SDDL. Together they form a symbiotic relationship which provides a basis for the methodology described above and for future expansion as well.

ACKNOWLEDGEMENTS

Mr. Richard G. Ransom was responsible for the implementation of these techniques and methods. His suggestions and critique plus his devotion to the successful completion of the project were significant comtributions.

Major John C. Callahan also deserves recognition for perceiving the need for this work, initiating the action, and providing the necessary support.

APPENDIX A: EXCERPTS FROM THE TALON SIMULATION MODEL

Note: The attached document is an excerpt taken directly from the TALON program document which contains more than six hundred pages, therefore, the pages are not numbered consecutively but instead correspond directly to the pagination of the original document.

APPENDIX B: EXCERPTS FROM THE TALON INPUT FORMAT SPECIFICATION DOCUMENT & TEST DATA BASE

Note: The attached document is an excerpt taken directly from the TALON program's combined IFSD and data base document which contains more than seventy five pages, therefore, the pages are not numbered consecutively but instead correspond directly to the pagination of the original document.

REFERENCES

Dijkstra, E.W., "Notes on Structured Programming," in Structured Programming, Academic Press, NY, 1972.

Kiviat, P.J., et.al., "The SIMSCRIPT II.5 Programming Language," CACI,Inc.-Federal, Arlington, Va.

Kleine, H., "Software Design and Documentation Language," JPL Publication 77-24, Revision 1., National Aeronautics and Space Administration, Jet Propulsion Laboratory, 4800 Oak Grove Dr., Pasadena, Ca. 91103, August 1977.

Kleine, H., "A Vehicle for Developing Standards for Simulation Programming," Proceedings of the Winter Simulation Conference, Highland, Sargent, and Schmidt, eds., 731-741.

Kleine, H., and Morris, R.V., "Modern Programming: A Definition," SIGPLAN Notices, Vol. 9, No. 9, Sept. 1974, pp.14-77.

Mills, H.D., "Top-Down Programming in Large Systems," in Debugging Techniques in Large Systems, Edited by R. Rustin, Printice-Hall, Inc., Englewood Cliffs, NJ., 1971, pp. 33-45.

Mills, H.D., Mathematical Foundations of Structured Programming, IBM Doc. FSC 72-6012, IBM Federal Systems Division, Gaithersburg, Md., Feb. 1972.