1981

Restruct: A Specification-Driven File Transformation Tool

James D. Arthur

Douglas E. Comer
Purdue University, comer@cs.purdue.edu

Report Number:
81-386

RESTRUCT: A Specification-driven File Transformation Tool*

James D. Arthur

Douglas Comer

Computer Science Department
Purdue University
W. Lafayette, IN 47907

CSD-TR-386

ABSTRACT

RESTRUCT is a software tool that reformats sequential files. It can operate as a stand alone process or function as a filter in a sequence of operations. Using RESTRUCT, one can express simple file transformations in a simple way. It is driven by a specification-based data description language that includes record description at three levels of detail, record redefinition and field conversion capabilities.

* This work was supported by R. R. Donnelly & Sons, Chicago, Illinois.

RESTRUCT: A Specification-driven File Transformation Tool

1. Introduction

File manipulation is fundamental to most programming environments. Software ranging from operating systems to report generators manipulate, use, and transform information stored in files. Although there have been efforts to standardize data representation [7,8] many distinct forms currently exist. Each vendor has chosen data characteristics and file access methods best suited for their operating system and machine architecture, thus perpetuating diverse file and data representations. Consequently, interest in file translation tools has increased significantly in the last few years.

Much research has been directed toward database transformations [1,2,9,12,13] and the transfer of files between dissimilar environments [3,10]. However, there is also a need for tools, designed for the nontechnical user, that provide transformation capabilities for files with simple structure (e.g, sequential files). Current translation and restructuring tools are either difficult to use because they require excessive syntax for simple operations [4,5] or they are so specialized that they lack many primitives necessary for general file transformations (e.g., field reordering, character insertion, etc.) [6].

Our research will investigate data transformations in a broad sense. We plan to construct a set of tools or primitives for translations and restructuring, and an environment that automatically selects and combines appropriate tools to perform transformations based on interaction with a user.

This report discusses the design and implementation of the primitive RESTRUCT: an easy-to-use tool for data reformatting. Section 2 provides definitions for terms used throughout this report. Section 3 states the primary objectives that should be satisfied by any file transformation tool. Section 4 discusses the

design alternatives for transformation tools. Section 5 outlines the implementation of RESTRUCT and section 6 provides two simple, but detailed examples that illustrate the simplicity and functionality provided by RESTRUCT.

2. Definitions

A *file* is a set $F=\{r_1, r_2, \ldots, r_n\}$ of n records. A *sequential file* is an ordered file. Each record is divided into k fields $f_1, f_2, \ldots, f_k$. Field f_1 is called the *record header*.

For purposes of this discussion, data is measured in 8-bit quantities called *bytes*, the set of all possible bytes is denoted Σ. The *length* of any data item d, denoted $|d|$, is the number of bytes in the item. Usually $|f_i|$, $1 \le i \le k$, is not constant throughout all records in F.

A *format* for file F is an unambiguous specification of the arrangement and lengths of fields and records in F. A format for a field f_i specifies the physical interpretation for each byte in the field. Often, all records in a file have the same arrangement of fields, so the format of a file can then be given by specifying the arrangement of a single record. In such cases we refer to the *record format*. A record format is *fixed length* if $\exists$ c>0 such that $\forall i\ |r_i|=c$, and *variable length* if $\exists$ i,j, $1 \le i,j \le k$, such that $|r_i| \neq |r_j|$.

Let $f_i, r_i \in \Sigma$. A *byte translation* is a mapping: $\Sigma \rightarrow \Sigma$; when a file F' is produced from a file F by applying a byte translation to every byte in F, we say that F' is a translation of F. An example is the translation of all lower case letters to upper case. A record is *restructured* by inserting fields, deleting fields, or reordering fields. When a file F' is produced by restructuring every record of file F by applying the same set of reordering, insertion, or deleting operations, we say F has been restructured into F'. A *transformation* of file F to F' is a sequence of files $\{F_0, F_1, \ldots, F_n\}$ where $F = F_0$, $F_n = F'$, and each intermediate

file F_i, $1 \leq i \leq n$, is the result of applying a translation or restructuring to F_{i-1}. *Intra-record* transformations are translation and restructuring operations performed on fields within a particular record.

3. Objectives

The primary objective of this research is to examine automated methods for translating and restructuring files.

The plan is to construct a set of independent, orthogonal tools, or primitives, and an environment for connecting them together. The environment will interface with users on one side, and the set of tools on the other.

RESTRUCT is the first restructuring tool devised. It works on sequential files and allows the addition, deletion, reordering and modification of fields within a record. Only intra-record transformations are handled by RESTRUCT. This was a conscious design decision, made to restrict the syntax specifications: the goal in RESTRUCT is to make it easy to specify simple reformatting. Other tools will handle more complex related tasks.

Additionally, we worked toward a specification-based user interface that included a simple data description language (DDL) and adequate diagnostic capabilities. The result is a sequential file data description language (SFDDL) and a history file that contains diagnostic messages, a table of field and record attributes, and a detailed description of the transformations performed by RESTRUCT.

4. Approach to Data Reformatting

A transformation tool is a program that accepts a source file F, a set of transformation specifications S, and produces a new file F(S). Three different approaches to data restructuring are possible. The first method is to build a new program for each application. The result is a different file translator for

each type of file structure. While this method is the one most frequently used, it has the highest overhead cost in terms of manpower and system resources. The second approach is to build a single program for all applications. Program complexity alone is enough to discourage this method. We chose the third alternative: constructing a set of independent tools that can be combined. RESTRUCT is a tool that performs many common transformation primitives such as record restructuring, field conversion and so on. It functions as a filter so it can be used with other tools tools.

The advantages of a tools approach are obvious: ease of documentation, development, testing and maintenance. Additional flexibility and transformational power are gained by selecting and executing the appropriate set of tools.

4.1. A Data Description Language for RESTRUCT

For a transformation tool to be effective, it must include a file description mechanism. One would like the description mechanism to be as simple as possible, yet powerful enough to handle the task. Many file translators are driven by a parameterized language which uses an inflexible fixed-column syntax that describes little more than data types and field lengths [5]. Fixed-column syntax and rigid declaration imposed ordering place unnecessary restrictions on specifications since a user must insure proper column alignment as well as the correct sequence for translation primitives. Although there are DDLs that avoid these restrictions their syntax structure is either very complicated [4,5] or lacks general flexibility [6]. The design of SFDDL includes a syntax structure that is essentially free format, keywords whose mnemonics are short but descriptive, and permits record description at three levels of detail.

4.2. Error Detection

RESTRUCT checks for syntax violations, conflicting specifications, and inaccurate file description. Syntax problems are immediately reported to the user and cause program termination. Conflicting specifications are resolved according to a simple set of precedence rules. RESTRUCT warns the user that assumptions have been made and then continues processing. RESTRUCT's execution history file is its most informative error detection and correction mechanism. The history file contains not only syntax and specification error messages but a record-by-record execution history of the translation process. Records that do not conform to the prescribed format are reported here.

5. Implementation of RESTRUCT

RESTRUCT consists of 3 modules, a lexical analyzer (scanner), a syntax analyzer (parser), and a transformation program. All three are implemented in C using lex and yacc, compiler writing tools available on the UNIX[1] operating system [11]. After the user specifications have been parsed successfully, the transformation program T is executed. T has access to the transformation specifications collected during lexical and syntactical analysis.

The implementation of RESTRUCT includes several novel ideas. First, it is driven by a non-procedural specification language, SFDDL, whose syntax structure is essentially free form. The following illustrates the structure of an SFDDL specification:

```
input [record-specs] [global-field-specs]
  field-name_1    input-field-specs
  .
  .
  .
  field-name_n    input-field-specs
```

[1] UNIX is a trademark of Bell Laboratories.

output [record-specs] [global-field-specs]

field-name$_x$ output-field-specs

.

.

.

field-name$_y$ output-field-specs

$1 \leq x,y \leq n.$

The input section describes the source file and associates a name with each field. These names appear again in the output section, which describes the format of the target file. The symmetry between the input and output syntax simplifies use.

RESTRUCT provides specification capabilities at three levels of detail. Record-specs denote attributes associated with each record (e.g., record length) and gives its format. Global-field-specs, the second level, ascribes properties to every specified field in a record. The third and lowest level of detail, field-specs, specifies those properties of a single field that differs from the properties of all other fields. To resolve specification conflicts the lower level specifications have precedence over the higher level specifications. An example will illustrate this idea.

Suppose we are to describe a file whose records are 20 bytes long and each byte is represented in ascii format except bytes 12-15 which form an integer (represented in binary). In RESTRUCT, one describes such a file by writing:

```
input   rlen=20   ffmt=a   # ("a" indicates ascii)

 f1     len=11
 f2     len=4     fmt=i    # ("i" indicates integer or binary)
 f3
```

The first line describes the input records as 20 bytes long with all fields being ascii. Successive lines describe each field in the record. The format specification for field f2 overrides the global specification, and makes its format integer (fmt=i).

The primary advantage of multi-level specification is that the user can chose the level of specification suitable for each application. One is not forced to describe high level features in low level terms.

A third concept implemented in reform concerns variable length fields and variable length records. Many transformation tools limit their processing capabilities to files containing only fixed length records [3,4,10]. RESTRUCT accepts both fixed length and variable length records. For example, text editors such as xe[2] and vi[3] normally suppress trailing blanks when saving text files. A transformation tool that reads a file created by one of these editors must handle variable length records (lines) and recognize the end of line character.

RESTRUCT includes a method for associating multiple transformation specifications with the same data (i.e., one can think of templates positioned on the input record, some of which may overlap), translation capabilities for fields represented in internal machine format, and directives that permit replicated field descriptions. These features along with those previously mentioned provide specification flexibility and transformation power lacking in many transformation tools. Appendix 1 provides a brief description of the input and output transformation directives.

6. Using RESTRUCT

RESTRUCT requires the following information: (1) a set of reformatting specifications, (2) an input file and (3) an output file. The UNIX version can be invoked by typing

RESTRUCT -f filename < infile > outfile

where filename is a file that contains a description of the input file (infile) and

[2] Xe is a version of the RAND screen editor E used at Purdue.

[3] Vi is a screen editor developed at the University of California at Berkeley.

formatting directives for the output file (outfile). (The symbols "<",">" are UNIX shell notations that denote input and output file assignments respectively.) If the user wishes to enter the reformatting specifications "in-line", The command is:

RESTRUCT -s "format specifications" < infile > outfile.

UNIX pipes, denoted "|", allow one to use RESTRUCT as a filter that takes the output from program A, reformats it, and sends the result to program B:

A | RESTRUCT -f filename | B.

The following two examples illustrate some of the uses of RESTRUCT. The applications shown are conceptually simple but provide an insight into the functions and versatility of RESTRUCT.

Example 1

The source file contains student records with the following format:

field name	position	length	format	delimiter
SSN	1 - 9	9	ascii	
Last name	10 - 19	10	ascii	
First Name	20 - 29	10	ascii	
Middle Init	30	1	ascii	
Address 1		variable	ascii	$
Address 1		variable	ascii	$
Address 2		variable	ascii	$
GPA		4	float	
Curr Course 1		5	ascii	
Curr Course 2		5	ascii	
Curr Course 3		5	ascii	
Curr Course 4		5	ascii	
Curr Course 5		5	ascii	

The target is a "human readable" file that contains the student's last name, first name, middle initial, SSN, and grade point average (GPA) in the order stated.

One could write all the details about length and format for each field of the input. Using defaults for format (ascii) and RESTRUCT's higher levels of

specification, however, leads to this short description:

```
input
 SSN     len=9
 Name    len=21
 Addr    len=v    dim='$'   rep=3
 GPA     fmt=f
 CC      len=25

output
 Name
 SSN
 GPA
```

Here, the student name has been described with only one field. Since the current course loads (CC1 - CC5) are not needed they are described as one field. The "rep" (repetition) specification on the address field specifies that Addr and its associated attributes occur three times. Observe that the input and output formats for GPA differs. This implies a conversion from internal machine format (float) to ascii (human readable) format.

Example 2

Suppose we are given the same student file but different output requirements. The requested output form is: the SSN with the appropriate "-"s inserted, last name followed by a comma and a blank, first name followed by a blank, middle initial followed by a period and a blank, and each of the five current courses separated by a blank. A simple set of transformation specifications is:

```
input
 SSN3    len=3
 SSN2    len=2
 SSN4    len=4
 Lname   len=10
 Fname   len=10
 Minit   len=1
 Addr    len=v    dim='$'   rep=3
 GPA     fmt=f
 CC      len=5    rep=5

output
```

```
SSN3     len=3    char='-'
SSN2     len=2    char='-'
SSN4     len=5
Lname    len=11   char=','
Fname    len=11
Minit    len=2    char='.'
CC       len=6    fill=l
```

This example introduces the "fill" specification. If the output length of a field is greater than its input length, left or right filling will occur. By default, filling to the right is assumed. Every output field uses default filling except CC, which is left filled. Note the form of SSN4; no fill directive nor fill character is specified. Finally, we examine the last field: CC. RESTRUCT is instructed to fill to the left with the default character.

7. Conclusion

We have begun to explore reformatting tools, and ways to combine them. Our first tool, RESTRUCT, is a simple user-friendly tool that provides file reformatting capabilities. It accepts input and output file specifications that describe the reformatting process at three levels of detail. Default attributes have been provided to minimize the amount of user description. The reformatting directives are concise in their descriptions and provide a powerful yet flexible set of capabilities. Even though RESTRUCT executes in a UNIX environment, it provides a mechanism to successfully describe, read and process files originating from other environments.

More work is needed in two areas: tools and environments for using them. We expect to develop more powerful restructuring tools, and an environment which selects sets of tools to solve restructuring problems.

Appendix 1: A Quick Reference to RESTRUCT

Record Descriptions

rdlm='delimiter'

Rdlm specifies the delimiter that terminates a record.

rfmt=format-spec

Rfmt describes the physical format of a record. The default is "a", ascii.

rlen=number|v

Rlen denotes the length attribute associated with a record. If a number is specified, it is assumed to be the length. If a "v" is specified the record is assumed to be variable in length. For variable length input records, the actual record length is obtained from the first 4 bytes.

Global Field Descriptions

These field descriptions apply globally to all specified fields.

fchar='character'

Fchar denotes both the padding and filling character. The default is blank.

fdlm='delimiter'

Fdlm specifies the field delimiters. The default is ",".

fill=r|l

If the length of an output field is greater than its input length then filling occurs. Ffill denotes whether filling will occur to the left (l) or to the right (r).

ffmt=format-spec

Ffmt describes the physical format of each field. The default is "a", ascii.

flen=number|v

Flen denotes the length attribute for each field. If a number is specified, it is assumed to be the field length. If a "v" is specified, a variable length field is assumed.

ftrunc=r|l

If an output field length is less than its input length then the field is usually truncated. Ftrunc denotes whether truncation will occur to the right (r) or to the left (l). The default is right (r).

Single Field Specifications

These specifications apply only to the associated fields with which they are defined.

char='character'

Same as "fchar"

dlm='delimiter'

Same as "fdlm"

fill=r|l

Same as "ffill"

len=number|v

Same as "flen"

pad

Pad is a reserved word that enables the insertion of characters into an output record. The default pad length is 1; the default pad character is blank. These defaults may be changed by specifying a length (len=) and character (char=) on the same description line.

pos=number

Pos specifies the starting position of a field in an input record. the default is the current position in the record.

rep=number

Rep permits multiple field definitions with the same attributes.

trunc=r|l

Same as "ftrunc"

References

[1] Shu, N. C., Housel, B. C., Lum, V. C., "CONVERT: A High Level Translation Definition Language for Data Conversion", Proceedings 1975 ACM SIGMOD International Conference On Management of Data, 1975.

[2] Shu, N. C., Housel, B. C., Taylor, R. W., Ghosh S. P., Lum, V. Y., "EXPRESS: A Data EXtraction, Processing and REStructuring System", ACM Transactions On Database Systems, March 1977.

[3] Bakkom, D. E., Behymer, J.A., "Implementation of a Prototype Generalized File Translator", Proceedings 1975 ACM SIGMOD International Conference On Management of Data, 1975.

[4] "EASYTRIEVE/IMS System for IMS and DL1 Databases", EASYTRIEVE Reference Manual, Pansophic, Oak Brook, Illinois, 1980.

[5] "AIIRS: Arthur Anderson Information Retrieval System", AIIRS Reference Manual, Arthur Anderson Inc., Chicago, Illinois, 1974.

[6] "TR - Translate Characters", UNIX Programmers Manual, Bell Laboratories, 1978.

[7] Steele, T. B., "Data Base Standardization - A Status Report", Proceedings 1975 ACM SIGMOD International Conference On Management of Data, 1975.

[8] SPARC: "Outline for the Preparation of Proposals for Standardization", document SPARC/90, CBEMA, Washington, DC, 1974.

[9] Su, S. Y. W., Liu, B. J., "A Methodology of Application Analysis and Conversion Based on Database Semantics", Proceedings 1977 ACM SIGMOD International Conference On Management of Data, 1977.

[10] Briss, E. W.,, Fry, J. P., "Generalized Software for Translating Data", Proceedings AFIPS NCC, 1976.

[11] Ritchie, D. M., Thompson, K. L., "The UNIX Time-sharing System", CACM, July 1974.

[12] Nestor, J. R., Wulf, W. A., Lamb, D. A., "IDL - Interface Description Language", Technical Report CMU-CS-81-139, Computer Science Department, Carnegie-Mellon University, August, 1981.

[13] Shneiderman, B., Thomas, G., "Path Expressions for Complex Queries and Automatic Database Program Conversions", Proceedings 1980 Sixth International Conference on Very Large Data Bases, 1980.