A HIGH LEVEL DATA FLOW PROGRAMMING LANGUAGE FOR ULTRASONIC DATA ANALYSIS

Roger G. Carter and
Lee H. Pearson
Morton Thiokol, Inc.
P. O. Box 524
Brigham City, Utah 84802

INTRODUCTION

Developing and implementing data analysis methods can consume a significant fraction of the time and effort required for Ultrasonic NDE technique development. This is especially true if the techniques under development involve either specially designed hardware or innovative waveform analysis methods. With computer based data acquisition systems, this situation often requires the researcher to write the needed computer software before the technique can be tested.

We found that implementing computer software required the majority of our time and often delayed us for a day or more before we could begin experimenting with a new technique. A new software system was needed that would speed up the process of implementing new data analysis techniques and remove this "software bottleneck" as well as providing the necessary scanning control and data acquisition functions.

ANALYSIS OF REQUIREMENTS

Any implemented software system must meet the following requirements:

- Data acquisition, scan control, and graphics/image display functions must be part of the software package
- Implementation of new data analysis methods must be easy and fast
- Implementation of all current signal analysis methods must be possible
- Software must be easily expandable

The first step in planning this software was to carefully examine our current data analysis methods. We use both time domain and frequency domain analysis, including analytic signal evaluation [1]. Careful examination of these methods revealed that:

- All of our analysis methods transformed each ultrasonic waveform into a single value to produce a gray scale or false color image
- The waveforms in the scan were operated on one at a time
The exact same operation was repeated on every waveform in the scan. The analysis on each waveform could be represented in every case by arranging a small number of operators in the proper order.

The following example clarifies this further. Suppose we are doing a pulse-echo scan on a test specimen and want to make an image based on the magnitude of the 450 kHz component of the signal received between 60 and 75 microseconds after the pulse was produced. This data analysis method may be constructed from three simpler operations: a window operation that keeps only the points within a specified region, a Fast Fourier Transform (FFT) operation, and a magnitude detection operation. The addition of two steps to handle the necessary input/output completes the description of this process:

- Digitize the ultrasonic waveform.
- Window out the sample points recorded between 60 and 75 microseconds.
- Perform an FFT on these points.
- Window out the magnitude point that corresponds to 450 kHz.
- Display the image pixel that corresponds to this waveform.

This set of operations is then performed exactly the same way on every waveform in the scan.

All of the data analysis methods used or anticipated to be used lend themselves to this type of description. They all can be described by a short list of simple operations just as the above example. Perhaps more important, the number of simple operations needed is relatively small. Most of our data analysis methods could be described with about a dozen simple operations combined in different sequences. This is not unlike mathematics where the five simple operations of addition, subtraction, multiplication, division, and exponentiation may be combined to form many powerful and complicated operations. By analogy, it is not surprising that a few basic operations can be combined to represent many different data analysis methods.

DATA FLOW REPRESENTATION

We now can represent our data analysis methods as lists of simple operations. However, lists are still somewhat clumsy and imprecise. An easy way to represent these lists was needed. We chose to use the Data Flow Diagram.

Data Flow Diagrams are often used in computer software design [2]. They represent a system as data flowing through a set of operators, each of which may change the data as they pass through. They are commonly depicted graphically by representing each operator with a circle and using arrows to represent the data flow paths between operators. This concept fits very well with our ultrasonic data analysis methods. A data analysis method is broken down into a set of simple operations, each of which becomes an operator in a Data Flow Diagram. The data, which are ultrasonic waveforms, then flow through these operators while they are being processed. The Data Flow Diagram for the previous example is shown in Figure 1.

Since the Data Flow Diagram is represented graphically, it is a very easy, intuitive, and flexible means of representing an operation. It is a high-level representation in that it does not include the details of
implementation but only the operations to be done. The following examples are included to provide a better feel for how this representation may be used.

Consider the example shown in Figure 1. Suppose the digitized waveforms are saved on the computer disk as we scan, and that we want the image to reflect the highest magnitude frequency component. The new Data Flow Diagram is shown in Figure 2. In Figure 3, the Data Flow Diagram has been further modified to create a second image based on the bandwidth of the signal and to save on disk only the portion of the signal that is of interest.

Figure 1. The Data Flow Diagram representing the example data analysis discussed in the Analysis of Requirements Section.

Figure 2. A Data Flow Diagram representing an example data analysis method.
Figure 3. A modification of the Data Flow Diagram shown in Figure 2, including the creation of a second image and a different save method.

Figure 3 is seen to be easier to understand than the written description. Hopefully, this provides some appreciation of the power of this representation. One simply draws a picture of the desired operations.

IMPLEMENTATION

Now we have an easy, flexible method of describing a data analysis procedure that is desired. The next step is to get the computer to execute what we have described. Ideally, the computer would execute the Data Flow Diagrams directly without any changes or translation. In other words, we want a high-level Data Flow Programming language that allows us to program the computer with Data Flow Diagrams just as we would program it with any other computer language.

Three steps were required to accomplish this:

1. A Data Flow Diagram graphical editor (called DFDRAW) was implemented. This editor allows the user to draw the Data Flow Diagram on the computer screen. The editor saves the Data Flow Diagram in a special file.
2. A FORTRAN subroutine was written for each needed operator. A standard calling interface was used for passing the waveforms into and out of the subroutine.
3. An interpreter (we called it DFSCAN) was implemented that reads the Data Flow Diagram files and executes them by calling the appropriate subroutines.

Once this was done, the user interface is very simple. The user runs DFDRAW, draws the desired Data Flow Diagram on the computer screen, and then runs DFSCAN to execute it. The user only sees his Data Flow Diagram; all other operations are transparent to the user. No traditional computer programming is needed, and the Data Flow Diagram is not translated into another form. For example, Figure 4 shows what the screen of the graphical editor (DFDRAW) looks like after the Data Flow Diagram shown in Figure 3 has been entered.
The original implementation of this system was on an IBM/PC-AT. We now have it running on a MicroVAX 2000 computer under VMS, and we are using a LEXIDATA 2410 for graphics output. We are developing a second version on the VAX that will make each operator a separate process and handle data by passing it through VMS mailboxes. This will allow greater I/O and computational parallelism and should result in better usage of CPU resources. It will also remove some of the limitations of the current implementation and allow distributing the operators of different systems in a network so that multiple CPUs may work on the same problem.

```
  <ROOT>SCOPE
  | C2 0,8,0,512,1,4,0,4
  | 8,2,4,0,48,
  +-----------+--------------+
  | WINDOW     | SAVE         |
  | 0,60E-6,0,15E-6, | SCAN1.DAT |
  +-----------+--------------+
  FFT        |
  | 1,         |
  +-----------+--------------+
  PEAK       |
  |            |
  +-----------+--------------+
  BANDWIDTH  |
  |            |
  +-----------+--------------+
  IMAGE      | IMAGE        |
  | 2,0,100,WR | 3,0,1E4,WR |

Q=quit  R=rewrite screen  O=change operator  F=change data flows
```

Figure 4. The screen of the graphical Data Flow Diagram editor, DFDRAW, after the Data Flow Diagram shown in Figure 3 has been entered.

A SPECIFIC EXAMPLE

As an example of the application of our Data Flow Programming language, we have included images from an ultrasonic scan of a specimen representative of a large solid rocket motor (SRM) segment end which is referred to as the inhibitor. The inhibitor is a layered, bonded system including an insulator, adhesive liner, and propellant. The liner thickness is inherently variable as a result of the application process which causes artifacts to appear and dominate the images obtained by conventional pulse-echo ultrasonic NDE used to inspect the bondlines. Our inhibitor sample consisted of a layer of rubber insulator approximately 0.25-inch thick, a layer of liner with thickness ranging from approximately 0.1 to 0.3 inch, and approximately 2.0 inches of inert solid rocket propellant. Figure 5 shows a photograph of a cross-sectional view of a section of the sample, showing the liner thickness variation.
Figure 5. Cross-sectional view of a piece of the inhibitor sample. The liner thickness variation is clearly visible.

Figure 6 above shows a conventional ultrasonic pulse-echo c-scan image taken at 1.0 MHz of the inhibitor sample in the region of two knife cuts made in the bondline. Figure 7 shows the corresponding Data Flow Diagram for the process. Locating the bondline defects in the image is difficult because of the presence of image artifacts which are caused by phase cancellation and beam redirection in regions where there are variations in liner thickness. Our objective was to use the simplicity of the Data Flow Diagram software package to aid in the development of an ultrasonic NDE technique to image bondline defects without having them obscured by the presence of phase cancellation artifacts.

Figure 6. 1.0 MHz pulse-echo c-scan of the inhibitor sample. Image was produced by peak detecting the reflection from the liner-propellant interface.
Theoretical evaluation of the problem indicated that a reduction of the frequency of the ultrasonic wave to less than 100 kHz would effectively make the liner layer thin enough relative to wavelength so that thickness variations would become relatively unimportant. It was also determined that a Fourier analysis of the waveforms would enhance the ultrasonic images. To test these ideas, we set up a scan using the Data Flow Diagram shown in Figure 8. The waveform was digitized, Fourier transformed and the magnitude at 67 kHz was obtained and converted to a color scale for the image. The results are shown in Figure 9. The two bondline voids are clearly seen above the liner thickness background signal variation.

Programming this scan required approximately 20 minutes. When we implemented a similar analysis on our previous system, it required nearly six hours to program and debug.
CONCLUSIONS

The greatest advantage of Data Flow Programming is for the user. The desired computations are simply diagrammed and entered into the computer in a graphical fashion that closely corresponds to this diagram. The use of traditional computer languages is not required, and yet the approach has the flexibility generally associated with high level programming languages. This is very important in a research environment. A new analysis method can be designed and tested in a matter of minutes instead of days with the conventional approach. Another advantage is that the system is easily expandable and adding new operators is a straightforward process. Since the operators perform simple functions, they are generally easy to write and debug. Since the operators run totally independent of each other, it is very easy to utilize multiple processors or multiple computers networked together.

The use of a high level Data Flow Programming language has proven to be very advantageous in a research environment. Although the programming language is very different from traditional computer languages, it is intuitive and easy to learn. It has greatly increased the efficiency of the research personnel using it.

As ultrasonic NDE techniques become more sophisticated, more and more computer resources will be needed along with better means of effectively utilizing them. New programming methods like Data Flow Programming and graphical computer languages should be incorporated into future systems.

REFERENCES