NASA Contractor Report 172210

NASA-CR-172210
19830027409

Development of a Prototype Multi-Processing Interactive Software Invocation System

William Joseph Berman
Advanced Programming Techniques, Inc.
704 Village Road
Charlottesville, VA 22903

Contract NAS1-16985
September 1983

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

Summary

During NASA Contract NAS1-16985, the Interactive Software Invocation System (NASA-ISIS) was first transported to the M68000 microcomputer, and then rewritten in the programming language Path Pascal. Path Pascal is a significantly enhanced derivative of Pascal, allowing concurrent algorithms to be expressed using the simple and elegant concept of Path Expressions.

The problem of transporting NASA-ISIS to the M68000 was solved by modifying the standard M68000 implementation tools to overcome their size limitations. The sequential version of NASA-ISIS was operational on the M68000 within the first several months of this contract.

The major effort on this contract was that of converting the sequential code of the original NASA-ISIS to Path Pascal. This necessitated the complete reorganization of the software in order to take advantage of the concurrent processing capabilities of Path Pascal. The resulting system is much easier to understand and to modify than the original Pascal-written system; furthermore, in writing the Path Pascal version, several programming paradigms were identified that are now considered to be standard Path Pascal coding techniques.

The principle problem encountered during this contract was the fact that the Path Pascal compiler/translater was being enhanced and debugged by the Government in parallel with its use on this contract. As new constructs were identified as necessary and as errors were detected, the Government would make the appropriate modifications. This interaction lasted the duration of the contract, and has resulted in a Path Pascal system that is robust and well-tested; the frustrations and delays during the contract period have been rewarded by the fact that Path Pascal is now available for use in future Government projects.

The primary result of this contract has been to verify the viability of Path Pascal as a system's development language. The NASA-ISIS implementation using Path Pascal is a prototype of a large, interactive system in Path Pascal. As such, it is an excellent demonstration of the feasibility of using Path Pascal to write even more extensive systems. It is hoped that future efforts will build upon this research and, ultimately, that a full Path Pascal/ISIS Operating System (PPIOS) might be developed.

Introduction

The development of the National Aeronautics and Space Administration's Interactive Software Invocation System (NASA-ISIS) was begun in April 1977 under Contract NAS1-14862 with the University of Virginia in Charlottesville, VA. This effort was continued after September 1979 under Contract NAS1-15935 with Dr. Joseph Berman. The principle accomplishments of Contract NAS1-15935 were to complete the development of NASA-ISIS and to demonstrate its portability across a range of systems using a highly-portable subset of the programming language Pascal.

Starting in February 1982, Advanced Programming Techniques, Inc., with Dr. Berman as president, was awarded NASA Contract NAS1-16985 to continue researching the NASA-ISIS "shell" concept, but moving the emphasis from portability to obtaining various efficiencies by rewriting the software in the programming language Path Pascal. Path Pascal was developed by Dr. Roy Campbell at the University of Illinois under a NASA grant [1]. It allows the simple and elegant specification of highly concurrent algorithms.

Background

The research done under this contract involved the merger of two powerful technologies -- NASA-ISIS and Path Pascal. In the following sections, the important characteristics of these systems are presented.

NASA-ISIS

The primary motivation for NASA-ISIS has been, and continues to be, to support the orderly development of flight software. Early in the development of NASA-ISIS this software was analyzed and characterized by:

-- long life-cycles, involving many users and facilities as the development proceeds from specifications through coding, testing and maintenance

-- increasingly complex programs, as flight computers become larger and faster, and projects more ambitious

-- high reliability, requiring large amounts of information including requirements, source code, test data, reports and documentation

-- tight schedules, implying high programmer productivity

The basic concept for NASA-ISIS is that it is a software system that provides an integrated, interactive software development environment for all phases of software development. This is not to say that NASA-ISIS was designed to perform all of the various specialized tasks of the many possible software tools; rather, NASA-ISIS provides a systematic interface to the host computer system and the various tools. NASA-ISIS is the toolbox and the workbench, and is based upon the following major concepts:

-- NASA-ISIS is a "shell language"

-- NASA-ISIS is a PASCAL-like language

-- NASA-ISIS has integral text editing and file management

-- NASA-ISIS can be used to "invoke" tools

Path Pascal

The standard definition of the programming language PASCAL includes no constructs for separate compilation, data encapsulation or concurrent processes. Path Pascal is a derivative of PASCAL that extends PASCAL by providing these necessary constructs.

Separate compilation was recently added to Path Pascal by Dr. Ed Foudriat of NASA LaRC. A large program may be divided into "MODULEs", each of which can be separately compiled and later "linked" to the other MODULEs in order to create the full program. Each MODULE includes an "interface", specifying information that is available to other MODULEs in the system, and in "IMPLEMENTATION", specifying information that is private to that MODULE.

Data encapsulation is provided in Path Pascal using the OBJECT construct. An OBJECT is a new Pascal "TYPE constructor" that allows the declaration of CONSTs, TYPEs, and VARiables that are hidden within the OBJECT. To gain access to these hidden VARiables, the OBJECT must include "ENTRY" subroutines that may be called from outside the OBJECT: only these ENTRY subroutines may directly use the OBJECT's variables. Thus, the ENTRY subroutines provide a well-defined set of operations upon the encapsulated variables such that the code using these operations can be written knowing only the effects of the operations, not their implementation.

Concurrency is provided in Path Pascal by allowing the specification of PROCESS subroutines. Calling a PROCESS subroutine causes the PROCESS to be allocated its own run-time stack and both the calling routine and the PROCESS to proceed to execute in parallel (pseudo-parallel on a single-processor machine). Communication and synchronization among the various concurrently executing PROCESSes is achieved by having OBJECTs act as "monitors" to control concurrent access to data according to a set of rules expressed as an OBJECT's "Path Expression". A Path Expression is a special notation for specifying the strategic placement of counting semaphores within the ENTRY routines of an OBJECT. A Path Expression can specify concurrent access using the comma (","), synchronized access using the semicolon (";"), and "burst" access using square brackets ("[" and "]").

Research Performed

Research under this contract lasted for 15 months. There were three distinct phases -- implementation of NASA-ISIS on the M68000, study and experimentation with Path Pascal, and implementation of NASA-ISIS in Path Pascal. These phases are reviewed in the following sections.

NASA-ISIS on the M68000

The first three (3) months of this contract were spent transporting NASA-ISIS to the M68000. The main problems were the restrictions imposed by the standard software development tools for the M68000. By modifying these tools, it became possible to implement the large NASA-ISIS system.

Additionally, NASA-ISIS on the M68000 was extended with character processing capabilities to allow direct interaction with the terminal from NASA-ISIS. These capabilities are sufficient to implement sophisticated user-interactions that are not possible on older computer systems such as the CDC CYBER machines.

Path Pascal Experimentation

The next seven (7) months of this contract were spent studying and experimenting with Path Pascal. Since Path Pascal is a new language, it took a substantial effort to understand how to use its powerful constructs.

In addition, the Contractor spent significant effort working the Dr. Ed Foudriat of NASA LaRC in designing enhancements to the Path Pascal language, in implementing various aspects of the system, and in testing the resulting Path Pascal system. Several major contributions were made to this effort by the Contractor, including process/procedure tracing in the run-time system, full interrupt handling capability, and sample programs that thoroughly test various aspects of the system.

NASA-ISIS in Path Pascal

The final five (5) months of this contract were spent implementing NASA-ISIS in Path Pascal. Although a significant amount of the NASA-ISIS software could be directly transported to Path Pascal, it had to be entirely reorganized in order to take advantage of the OBJECT/PROCESS capabilities of Path Pascal. This reorganization was complicated by several errors in the Path Pascal translater that required many hours to pinpoint and equally as long for the Government to correct.

By using several MODULEs developed during the Path Pascal Experimentation phase described above, the implementation of NASA-ISIS in Path Pascal was relatively straight-forward. The resulting system demonstrates the concurrent processing capabilities of Path Pascal by allowing a single user to control two NASA-ISIS sessions simultaneously from a single terminal. The user sees the top half of the CRT as one NASA-ISIS system, and the bottom half as another, independent, NASA-ISIS system. By typing the sequence "ESC ESC A", the user may communicate to the top system; "ESC ESC B" accesses the bottom system.

Results

The main accomplishment of this research effort has been the demonstration of the feasibility of using Path Pascal as an implementation language for complex software. While the software delivered to NASA-LaRC under this contract is interesting, it is the fact that it was written using this new, untried language that is most important. Furthermore, by using Path Pascal to develop such a large system, several important aspects of Path Pascal have been found.

Standard OBJECTs/Path Expressions

While the Path Expression concept of Path Pascal can very succinctly state a complex interaction among many PROCESSes trying to use the data contained in an OBJECT, it is not easy to design these all-important Path Expressions. Fortunately, however, it has been found during the course of this research that most OBJECT/Path Expressions conform to a limited number of "paradigms".

Buffer OBJECTs

A common use of an OBJECT and its Path Expression is to act as a buffer between two PROCESSes. Such an OBJECT acts as a "pipe" between the PROCESSes by allowing one (the "supplier") to place information into the buffer without having to wait on the other PROCESS, and by allowing the other (the "consumer") to extract information from the buffer as it is ready to process it. This standard OBJECT/Path Expression was first described by Roy Campbell in his original paper on Path Expressions [2]:

```
PATH
   buffersize:(store;fetch),
              1:(store),
              1:(fetch)
END
```

This same Path Expression can also be used to solve some of the simple versions of the many-to-1 buffering problem. If the suppliers are competing "fairly" to send information to the consumer, and if the consumer can process the information in any order, this standard OBJECT/Path Expression is sufficient.

Controlled-Interleaving Buffer OBJECTs

Another standard OBJECT/Path Expression is used to handle the many-to-1 situation in which a given supplier needs to send an un-interleaved sequence of data to the consumer. For example, in a CRT system supporting multiple windows, it is desirable to "reserve" the buffer (if necessary, waiting until it is available), to position the cursor to the appropriate position on the screen, to write a sequence of characters, and to "release" the buffer. Thus, there is a controlled interleaving of characters being routed to the windows on the screen. This, and many similar situations, can be solved by using the standard OBJECT/Path Expression:

```
PATH
          1:(hold;free),
  buffersize:(store;fetch),
          1:(store),
          1:(fetch)
END;
```

In this case, each supplier first calls "hold" to gain control of the buffer, then uses one or more "stores" to place the information into the buffer, and, finally, calls "free" to allow another supplier access to the buffer. Note that, unfortunately, there is no way to guarantee that this protocol will be followed by each supplier; if the protocol is not followed, unacceptable behavior will result.

1-to-Many Processing

While the many-to-1 situation can be solved simply by most languages that support concurrency, it is important to note that Path Pascal has sufficient power to easily solve the 1-to-many problem. In this case, a supplier is producing information and, based upon some criteria in the data, will select a particular consumer to receive that data. Path Pascal solves this problem by allowing both ARRAYs and Linked Lists of OBJECTs, making it possible to use the supplier's data in conjunction with a search procedure through such a data structure.

An example of a 1-to-many situation is the processing of a terminal's input stream as it enters a multi-tasking environment. The user must be able to direct the characters he types to the appropriate task. This can be solved by having a standard buffer OBJECT convey the characters to a "routing" PROCESS that examines the input stream for a particular sequence of characters (e.g., "ESC ESC") that indicates that the user wishes to communicate to a different task. When this sequence is found, the linked list of OBJECTS corresponding to the terminal input for the various tasks is searched to find the appropriate OBJECT (also a standard buffer) and subsequent input is channeled into this OBJECT.

"Remote Procedure Call" OBJECT

There have been several proposals made for extending systems such as Path Pascal that depend upon shared variables for describing and controlling concurrency to operate in a distributed computing environment. One suggestion that has received considerable interest is that of "remote procedure calls". As described in [3], a remote procedure call can be viewed as a two-message exchange: the caller sends a message, then waits for the called routine to return a message (such as a record from a shared, distributed database).

Even though it is not a message-based system, Path Pascal can achieve the effect of a "remote procedure call" by using a "four phase" OBJECT/Path Expression:

```
PATH
    1:(caller_calls; message_handler_receives),
    1:(message_handler_returns; caller_receives)
END;
```

Thus, the caller uses the routine "caller_calls" (limited to VALUE parameters) to send its data to the message handler and then calls "caller_receives" (limited to VAR parameters) to wait for a response. Meanwhile, the message handler is a PROCESS that uses the appropriate protocol to send the data to another part of the distributed system. The receiving message handler then uses a similar OBJECT/Path Expression to convey the data to its destination, and then waits for a reply. When the reply occurs, the message handler uses the network to return the data to the original message handler that, finally, uses the OBJECT/Path Expression to transfer the information to the caller.

Physical/Logical Input/Output

In addition to studying Path Expressions and identifying various standard OBJECT/Path Expressions, the design and implementation of a CRT input/output mechanism was an important contribution of this research. Once again, however, the details of the code produced during the contract period is not nearly as important as the concepts that are embodied in that code.

In designing and implementing the CRT interface, extensive use of buffer OBJECTs was made in order to isolate the various aspects of the information processing. This approach allows the substitution of PROCESSes between these buffers to perform specified processing. These PROCESSes have the functions of managing the input/output controller, device, sharing and buffering.

Controller Input/Output

All interrupt-driven code in a computer system is directly related to specific input/output controllers. Each controller is responsible for interfacing a device to the computer's bus structure and, unfortunately, most controllers use their own unique protocol for communicating to and from their device.

In Path Pascal, interrupts are handled using the DOIO statement. For input, an INTERRUPT PROCESS is a loop that waits for the interrupt to occur, examines the status/data associated with the interrupt and passes the appropriate information into a buffer OBJECT for further processing. For output, an INTERRUPT PROCESS is a loop that waits for data via a buffer OBJECT, initiates an output operation, waits for an acknowledging interrupt and examines the status associated with the interrupt for any errors in transmission.

It is critical that the loop of an INTERRUPT PROCESS be very efficient, especially for processing input interrupts. Data might be lost if the PROCESS cannot perform the DOIO statement in time. The buffer OBJECT associated with the INTERRUPT PROCESS must be sized to take into account the speed of the device and the anticipated load upon the entire system. It should be large enough to handle a burst of data to or from the device.

Device Input/Output

Each input INTERRUPT PROCESS places its data into a buffer OBJECT that is read by a "device input PROCESS"; each output INTERRUPT PROCESS receives its data from a buffer OBJECT that is written by a "device output PROCESS". These "device PROCESSes" are responsible for transforming between internal standard

requirements and external idiosyncratic behavior. For CRT devices, the most common requirements of the device PROCESSes are to interpret special keystrokes (e.g., cursor control) as standard internal codes, and to transform standard internal codes into appropriate escape sequences to perform special CRT operations (e.g., clear screen). If a new CRT is attached to the system using an existing input/output controller, only the device PROCESSes need be changed.

Shared Input/Output

At the controller and device levels, the input/output is simply treated as a sequence of input/output requests. In a multi-processing environment such as that created by Path Pascal, there may be many PROCESSes making input/output requests. It is the responsibility of the "I/O Sharing PROCESS" to accept these requests and to satisfy them in an order consistent to some pre-specified strategy. For instance, a disk-sharing PROCESS might reorder read/write requests for purposes of optimizing head motion, or a printer-sharing PROCESS might route all print requests except the process currently "owning" the printer to a spooling device.

Buffered Input/Output

If a PROCESS's input/output requests are for data transfer in the sizes other than those specified by the underlying input/output system, there is a need for a "I/O Buffering PROCESS". Thus, if a PROCESS makes a series of requests to read single characters sequentially from a disk file, the disk-buffering PROCESS would make a single request to read the appropriate block from the file and then use the characters in that block to respond to the requests for single characters.

Summary and Conclusions

The most important result of this research effort has been the verification that Path Pascal can be an excellent tool for building highly-concurrent software. Even if a high degree of concurrency is not required, Path Pascal's MODULEs and OBJECTs are powerful tools for creating reliable, readable software.

The program produced under this contract is interesting; it contains a few novel features. However, its primary contribution has been as a vehicle for trying new ideas and learning how to effectively use Path Pascal. It is hoped that this knowledge will be used in future research efforts to go beyond the prototype/demonstration phase and to build full-featured, production systems in Path Pascal.

References

1. Campbell, R.H. and Kolstad, R.B., "PATH PASCAL User Manual", SIGPLAN Notices, Vol. 15, No. 9, September 1980.

2. Campbell, R.H., "Path expressions: A technique for specifying process synchronization", Ph.D. disseration, Computing Laboratory, University of Newcastle upon Tyne, August 1976.

3. Andrews, G.R. and Schneider, F.B., "Concepts and Notations for Concurrent Programming", Computing Surveys, Vol. 15, No. 1, March 1983.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR 172210		

4. Title and Subtitle	5. Report Date
DEVELOPMENT OF A PROTOTYPE MULTI-PROCESSING INTERACTIVE SOFTWARE INVOCATION SYSTEM	6. Performing Organization Code September 1983
7. Author(s) William Joseph Berman	8. Performing Organization Report No. N8301
9. Performing Organization Name and Address Advanced Programming Techniques, Inc. 704 Village Road Charlottesville, VA 22903	10. Work Unit No. 11. Contract or Grant No. NAS1-16985
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546	13. Type of Report and Period Covered Contractor Report 14. Sponsoring Agency Code

15. Supplementary Notes

Langley technical monitor: Ralph Will
Final Report

16. Abstract

During NASA Contract NAS1-16985, the Interactive Software Invocation System (NASA-ISIS) was first transported to the M68000 microcomputer, and then rewritten in the programming language Path Pascal. Path Pascal is a significantly enhanced derivative of Pascal, allowing concurrent algorithms to be expressed using the simple and elegant concept of Path Expressions.

The primary result of this contract has been to verify the viability of Path Pascal as a system's development language. The NASA-ISIS implementation using Path Pascal is a prototype of a large, interactive system in Path Pascal. As such, it is an excellent demonstration of the feasibility of using Path Pascal to write even more extensive systems. It is hoped that future efforts will build upon this research and, ultimately, that a full Path Pascal/ISIS Operating System (PPIOS) might be developed.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
interactive computing, concurrent algorithms, programming languages	Unclassified-Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	11	

For sale by the National Technical Information Service, Springfield, Virginia 22161

End of Document