Improving a Data-Acquisition Software System With Abstract Data Type Components

S. D. Howard
Communications Systems Research Section

Abstract data types and object-oriented design are active research areas in computer science and software engineering. Much of the interest is aimed at new software development. In this experiment, abstract data type packages developed for a discontinued software project were used to improve a real-time data-acquisition system under maintenance. The result saved effort and contributed to a significant improvement in the performance, maintainability, and reliability of the Goldstone Solar System Radar Data Acquisition System.

I. Introduction

Software components based on abstract data type design and initially developed for a new software system have been used to improve the performance and quality of a data-acquisition software system under maintenance. Although object-oriented design and abstract data types (ADTs) are active research areas in software engineering and computer science, much of the attention has been devoted to the employment of these concepts in the creation of new software systems; incorporating ADTs into software systems under maintenance has been less widely explored.

ADT packages may be thought of as extensions to high-level programming languages. (A more complete discussion of this idea can be found in a paper by Liskov and Zilles [1].) Most widely used procedural languages provide facilities for declaring and manipulating basic data types. Integers, arrays, characters, and booleans are examples of basic data types available in high-level languages. Each

type has a set of operators that form part of the definition of the type. ADT packages provide additional data types and encapsulate the valid operations on those types. Type packages can be used to capture the domain-specific data abstractions that a general high-level language cannot provide. Integers, arrays, characters, and booleans are quite reusable in high-level languages. Similarly, ADT packages have potential for reusability in related applications.

The abstract data type components used in this experiment were developed for the Goldstone Solar System Radar Data Acquisition System. The architecture for the software system was based on an ADT component library from which specialized data-acquisition systems could be quickly assembled to support established science objectives and new experiments. The software was designed to replace prototype data-acquisition systems that had been used to collect radar returns for the planetary radar astronomy program since the inception of the High Speed Data Acquisition System. Reference 2 gives an overview of the High Speed Data Acquisition System.

The data-acquisition software ADT library was never completed. Work stopped in May 1988. To support observations of Mars during the fall 1988 opposition, effort was redirected toward the modification and enhancement of the Binary Phase Coded Ranging prototype system.

II. Assessment

The successful modification of the ranging prototype for the 1988 Mars opposition did not resolve all of the limitations of the prototype system. The performance of the ranging prototype had always fallen far below the estimate of possible performance. An interprocess communication mechanism based on polling, chosen for ease of implementation in the prototype system, was inadequate for operational use. Conversion to event-driven processing was a likely solution to the performance problem. However, the modification involved replacing parts of the real-time interprocess communication structure; a similar change to the ranging prototype had been attempted in the past without success.

This situation opened the opportunity to consider the inclusion of previously developed ADTs. Reliable components reduce the difficulty of modifications by minimizing errors introduced into the system. Two high-quality ADT-based packages managing event signals and interprocess message passing had been among the first deliveries to the ADT software library. After assessment of the modifications that would be necessary to the structure of the prototype software to support event-driven real-time processing, it appeared that the new ADT packages could support the changes.

The benefits of using the new ADTs were clear. In addition to encapsulating data types and operators, ADT packages also encapsulate the important software quality attributes of maintainability, robustness, and reliability. The type packages that had been developed for the ADT software library were of high quality. Reliability was recognized in the initial work assessment to be a very important factor in the success of the enhancement to the ranging prototype. Errors in the interprocess communication mechanism of a real-time system can cause the system to transition to an unknown state and fail. The circumstances of these failures can be difficult to trace. The prototype software had not been designed for operational reliability. The maintenance history of the prototype indicated that the system was sensitive to change and prone to side-effect errors that were difficult to trace. Introducing unreliable new work into the interprocess communication mechanism had the potential for degrading the system to the point of unusability.

Successful use of the ADTs could also reduce the amount of time necessary to complete the ranging prototype system upgrade. One software engineer (the author) was available to complete the work. Without the implemented type packages, event signals and interprocess message passing would have to be designed, implemented, and tested for the prototype revision. It is estimated that four weeks of calendar time were required for the development of each ADT. That estimate includes work by two software engineers on specification, specification checking, test plan specification, ADT implementation, test implementation, and unit testing.

The most serious difficulty associated with using ADTs was that much of the ranging prototype had been implemented in FORTRAN and the ADTs were implemented in Pascal. A mechanism for invoking both of the Pascal type packages from FORTRAN had to be established. The greatest risk, of course, was that the incorporation of ADT components into programs already designed and implemented had not been attempted before. Modifications to the prototype that were done to support the 1988 Mars opposition involved replacing two processes in the prototype software with processes that had been rewritten and redesigned to include abstract data types. Designing programs to incorporate ADTs was a relatively well-understood task. Including ADTs in program structures that were not designed to accommodate them was not a familiar task; it was difficult to assess the scope of

changes that might be necessary to support the inclusion of ADTs. Underestimating the scope of modifications could result in the loss of approximately 3 to 4 work-months of engineering time.

III. Implementation

It was possible to gather more information about the feasibility of using the ADTs in the ranging prototype by investigating the parameter-passing mechanisms that would be necessary to call Pascal type packages from FORTRAN. The creation of two FORTRAN test shells quickly demonstrated that both of the candidate type packages could be cleanly invoked from FORTRAN.

The process in the real-time ranging prototype that managed the operator interface had been written in FORTRAN. Polling for interprocess messages in the operator program consumed excessive CPU time. Calls to the ADT procedures were added to the FORTRAN operator program; only minor modifications to the program structure were necessary to convert from polling to event-driven processing using the ADT-based operators.

Because the prototypes and the ADT packages used VAX/VMS operating system facilities, experimentation demonstrated that the new ADT interprocess communication structures were compatible with the older facilities in the prototypes. This was an unexpected advantage that allowed modification and testing of one process in the prototype at a time. Modification of the operator process was commenced in November 1988 and completed in January 1989. The demonstration of the event-driven operator program communicating successfully in the ranging system marked the beginning of the use of ADTs in software for which they had not been originally designed.

IV. Evaluation

The inclusion of the ADTs in the ranging prototype was remarkably free of difficulty. The availability of the components saved development and testing time. Considerable time could have been lost if changes and corrections to the ADTs had been necessary. No changes or corrections have been required. Instead, the time was available to spend on other tasks that needed attention in the prototype modification. Well-crafted and carefully checked ADT components improved the overall system maintainability and reliability.

Software design based on ADTs has interesting consequences for both software development and software maintenance. All of the software products that had been developed for the discontinued ADT software library are now potentially usable. Furthermore, this work established the concept of using previously developed ADT components in the maintenance of systems for which they had not been designed.

It is important to note that this approach is not a simple solution to software development problems. ADT-based software design, like many modern software engineering techniques (Fagan inspections, for instance), can cause "front-end loading" in the software development project; there is often a long preliminary phase while the type library is being constructed, during which results are not visible to those outside the software development team. Reliability and reusability are obtained only by careful initial work. Good design decisions in the definition (or specification) of ADT components are not achieved mechanically. Reliability is gained from careful craftsmanship, static program verification, and well-considered test strategies.

Care must be exercised in the generalization of these results. No attempt was made to control this experiment or remove bias; the primary objective was to successfully convert the prototype to event-driven processing. The prototypes and the ADT components were designed by the same software developer. It is difficult to determine how much this might bias the result. Further, this experience was limited to a small number of ADTs. Confidence in continuing to use ADTs in maintenance derives from an assessment that the circumstances under which this work took place were typical.

V. Conclusion

Successful use of ADT-based software components in the prototype system played a critical part in achieving a seven-fold improvement in ranging data throughput. This capacity was used as soon as it was available on radar interferometry observations of Mercury in the spring and fall of 1989. The ADTs remain unchanged since their delivery to the data-acquisition software ADT library in 1987; no errors have been detected in these components.

The satisfactory results of this work have led to further exploration. Recently, a new ADT-based software component was created specifically for adaptive maintenance in the ranging prototype. These experiences with ADT-based components have created another software maintenance option for the improvement of the Goldstone Solar System Radar Data Acquisition System.

Acknowledgment

The abstract data type-based software architecture of the data-acquisition system was the work of Dr. J. L. Robinett. Dr. Robinett and the author collaborated on the production of the two abstract data type packages that were described in this article.

References

[1] B. H. Liskov and S. N. Zilles, "Programming with Abstract Data Types," *ACM SIGPLAN Notices*, vol. 9, no. 4, pp. 50–59, April 1974.

[2] L. J. Deutsch, R. F. Jurgens, and S. S. Brokl, "Goldstone R/D High Speed Data Acquisition System," *TDA Progress Report 42-77*, vol. January–March 1984, Jet Propulsion Laboratory, Pasadena, California, pp. 87–96, May 15, 1984.