Implementation of an Ada* run-time environment

by HERMAN FISCHER
Litton Data Systems
Van Nuys, California
and
EDGAR H. SIBLEY
Alpha Omega Group, Inc.
Silver Spring, Maryland

ABSTRACT

The Ada Programming Support Environment (APSE) has been introduced1 as a set of tools to support program development systems. This paper introduces the idea that the concepts and facilities of APSEs are valuable not only to host systems (those used to develop software), but also to certain Ada run-time environments (ARTEs) (those in which applications execute) and examines the implementation of large database transaction oriented systems in such an environment. Two examples of actual systems are used to show the benefits gained by using the Ada environment. A cost/benefit analysis for such a transition is also outlined.

*Ada is a registered trademark of the Department of Defense.
INTRODUCTION

The benefits of using an Ada run-time environment (ARTE) based on the Ada Programming Support Environment (APSE) model are basically the same as the APSE's benefits for the programming host: transportability and interoperability of system components in a hardware-independent manner that was not previously realizable. These terms, though in common use, are not always used in the same way and must first be defined. The KITIA have agreed that "interoperability" is the ability of support environments to exchange database objects and their relationships in forms usable by tools and user programs without conversion, and that "transportability" is the ability to install a function in a different environment, without reprogramming, to perform with the same functionality.

The ARTE/APSE Concepts

It is our belief that the STONEMAN model, though defined for host/target environments, will be applied to computing environments not intended for program development. In STONEMAN, a kernel APSE (KAPSE) is surrounded by a Minimal APSE (MAPSE) toolset that, when extended with a comprehensive set of tools, becomes a full APSE. This is illustrated in Figure 1, with the KAPSE at the center of a set of rings (embodying the hardware, operating system, standardized interprogram communications mechanisms, and its "database"), and with the surrounding wedges representing the minimal toolset. Surrounding the outside of the wedges are applications-specific tools and programs; these can also occupy "open" (MAPSE) tool spaces in the wedges ring.

It is proposed that the layered-rings model applies to transaction-oriented database management system (DBMS) ARTE systems, as portrayed in Figure 2. Although more detail is shown in the kernel of the ARTE (KARTE), the case will be made that the features of the KARTE are applicable to the APSE also. In fact, an ARTE can use many of the same facilities, or share the same processor with an APSE (as happens today in logistics systems that run applications software in the processor used to support software-development organizations).
We shall first examine the need for this form of run-time environment.

Problems of Existing Large-Scale Information Systems

There are six categories of problems that exist in current systems; we shall first show how these are solved in a current programming and run-time environment (there are also many problems with existing systems that are only remediable with improved software methodologies, better maintenance, more money, and the like). These are:

1. Hardware—Hardware, as used in large systems is often obsolete before the applications software is placed in operation; in fact this may occur early in the application's life cycle. Hardware is also often not suitable for new, heretofore unthought-of applications of existing software, such as occur when rapid-deployment concepts force portability of previously stationary systems.

2. DBMS—DBMSs are large, complex beasts. The successful ones marketed to Fortune 500 companies seem to have upwards of 2,000,000 source lines, including the various optional (but essential) tools that surround the DBMS. These tools are constantly evolving, and upgrading from one major version to the next is a major undertaking. Furthermore, as the DBMS systems evolve, new generations of systems appear. In the absence of KAPSE-like layers separating applications from underlying implementations, portability is not possible. Existing applications are enormously expensive to upgrade.

3. Tools—DBMSs contain, in their environments, a large quantity of very diverse tools; these vary from initialization, backup, and restoration tools, to online dictionaries and schema-maintenance tools, reporting and configuration-management tools, applications generators, query tools (structured and English-like), and so on. It is not reasonable to switch tools every time new hardware is obtained, or when switching to an improved underlying DBMS.

4. Applications Generators—Such applications generators as the Cullinane Corp ADS-OnLine,1 the IBM IMS-ADF,2 and the Air Force On Line Data System (AFOLDS) DUEL3 are a vital part of today's environment. These products allow transactions to be coded in very high level applications languages, or to be expressed as a set of rules or decisions. The Cullinane product is said by commercial users to provide a sevenfold to tenfold productivity improvement over "old" (manually-coded) programming techniques. The USAF product's users estimate a 60% improvement in productivity. Applications generators are expected to become a vital part of the future of DBMS-based transaction processing systems, even though they are not, strictly speaking, DBMS tools. Thus, their future use will need some attention to the portability of the applications-generator toolset independently of the underlying DBMS or hardware.

5. Transaction-Based Terminal Handlers—The ability to create user-friendly and easily maintainable transaction-processing software depends, to a large extent, on user terminal handling processors that are intimately integrated into the underlying system. At the same time, they must be sufficiently separable to permit migration among significantly differing terminals, networking philosophies, communications facilities, and report generators that supply the user needs. Standish discusses the needs for a bounded set of user interfaces for accessing tools and the possible need for standardization in this area. User interfaces have been categorized as "normal conversational" (prompted command lines), "form filling" (CRT with prompts and fill-in fields), "tree of menus" (hierarchy of levels of choices), and "graphical" (windows and iconics). All these will be needed at future user interfaces.

6. Configuration Management—Configuration-management features are an integral part of today's complex large-scale transaction-based systems. However, they are often treated lightly and incompletely; the result is that logistics-type systems have never had great success in supporting variants and versions of schemas, programs, and transactions. Nearly all major DBMSs and their dictionaries today support identification of versions or variants, and yet almost none of these support the coexistence or automatic maintenance of different versions at different sites, exchanging distributed data, converting formats, ensuring proper library control and testing, and recovering crash and archival data of differing versions.

SOME EXISTING SYSTEMS

The U.S. Air Force relies on two very different large-scale DBMS-oriented logistics systems for aircraft maintenance tracking.

- The Maintenance Management Information Control System (MMICS), presently implemented on the Burroughs Medium System computer family (the Phase I system) handles fighter aircraft (F-15, F-16, and A-10). This system tracks engine parts and operating conditions, and performs calculations to predict the need for preventative maintenance.
- The Automated Maintenance System (AMS), based on the IBM 370 family architecture, handles airlift transporters (C-5, C-141). It has also been considered for handling logistics for the B-1, the Space Shuttle (if the USAF takes responsibility for it), and the MX system.

The MMICS system is huge, encompassing far more than fighter engine maintenance tracking; MMICS provides much of the data required to manage maintenance equipment and personnel resources, worldwide, for aircraft, missile, and communications-electronics-meteorological environments. MMICS includes over 3 million source lines of coding.

The AMS system is also huge, including a large number of
MMICS is an "updated system;' its COBOL programs reflect batch processing punched-card transactions and a "home-grown" database structure (embedded in the transaction programs). MMICS has since evolved to be online, but only by overlaying on the Burroughs Master Control Program (MCP) a USAF "home-grown" transaction-analyzer program (itself over 110,000 source lines) to simulate punched-card inputs and line-printer output on CRTs.

MMICS operates in a base support computer environment, where the complete system is known as Phase II. Not all Phase II programs are COBOL-based with embedded database handling: the Civil Engineering, Accounting and Finance, Medical, Operations, and Transportations applications are all implemented (on the same B3500 hardware) using AFOLDS, which includes data description capabilities, an applications generator, a transaction-oriented terminal handler (and forms builder), and an English-like query language.

Neither Phase II nor AMS is transportable; Phase II is not because its MMICS has a large volume of embedded assembler coding within Cobol programs to handle database access. Phase II's AFOLDS-based functions are also machine dependent. AMS is not transportable because of the use of IMS and 327X terminal formatting facilities.

Table I examines these systems with respect to the problem areas enumerated previously.

**Hardware and Transportability**

Phase II is tied to Burroughs Medium-Sized Systems. Most bases have "ancient" B3500 computers installed. These transistor and discrete component curiosities cannot be replaced, under Government Accounting Office rules, except by competitive bidding. (A few bases have slightly newer B4700s.) Although Burroughs makes modern equivalents, such as the 900 series B3900, the USAF has not been able to convert to them (due to the obvious lack of competition) and is thus stuck with the B3500s.

Phase IV, the upgrade-in-process for Phase II, was directed by public law to be a competitive procurement (the authors do not mean to imply that it is bad, only that STONEMAN concepts are needed to make such future actions reasonable). Two companies are in a "compute-off" conversion, Burroughs and Univac; both are re-implementing the present system on new (different) hardware families, using new operating systems and terminal handlers, and using manufacturer-supplied DBMSs.

Even Phase IV hardware will become obsolete in the near term (owing to the pressures of an advancing semiconductor industry), and at that time the DBMS, operating system, and terminal handling systems will remain untransportable. The Phase IV system, once deployed, will be no more readily transportable (in competitive reprocurement) than Phase II.

AMS, being based on IBM architecture, appears to have a longer life: there is a large industry of instruction-set-compatible processor builders. AMS users can thus develop new software and not worry that near-term hardware upgrades will remove the IMS and 327X terminal architectural footing.

**Independent Systems and Interoperability**

There will always be organizational, spatial, and temporal reasons for independent procurements of systems with similar requirements. For example, the aircraft maintenance software for fighter aircraft and transport aircraft has been handled by separate organizations, and it is therefore not surprising that MMICS and AMS are two different systems. MMICS is over ten years old. If it were not for its lack of transportability, some of the MMICS sites would have been upgraded to more recently procured hardware. However, unless the new system remains interoperable with the old one (in addition to being...
able to reuse its software), there is no way to phase-in new equipment, and there would thus be no way to reduce acquisitions of side-by-side systems such as MMICS and AMS which could otherwise share resources.

ENVIRONMENT CONCEPTS APPLIED TO THE EXAMPLES

Both the Phase II and AMS systems will, at some time in the future, need to be transferred to new-technology support systems. There is only one cost-effective way that this can be accomplished within the constraints of implementing transportability and interoperability. Software costing studies show that a robust modern support environment, providing a toolset which minimizes the complexity of applications programs, minimizes costs. Clearly there are two prerequisites to such a transition: a robust DBMS and a complete set of tools to support its use (e.g., applications generators).

Given that these prerequisites were met on an Ada-supportive system, one would still be faced with the problem that the resultant new systems would be nontransportable (except among families of the Ada-supportive system chosen). For example, if one were to base a reimplementation on the ALS architecture (the U.S. Army Ada Language System effort), the system would be tied to DEC VAX architecture and its current operating systems. Furthermore, the DEC architecture, as it is being used by the Ada products, is not supportive of transaction-based terminal networks (e.g., multidropped externally clustered buffered approaches).

An attempt to build a run-time system for the transported systems is likely to meet with two barriers, the cost and the lack of guarantees of support in future environments.

The only solution is to base the transported system on the use of the STONEMAN concept, where the transaction-processing software itself is an outer layer of programs around a "ring" of DBMS and transaction-supportive tools.

COST/BENEFIT CONSIDERATIONS

The Phase II to Phase IV conversion, now in progress, provides interesting cost data. Each vendor is charging the taxpayer (in round numbers) $50 million for the first increment of software conversions. In addition, the USAF has approximately 500 staff members supporting the three-year effort.

Given that 1500 man-years cost approximately another $50 million, the initial conversion cost is of the order of $150 million—owing to the lack of software transportability. The initial nine applications converted (of several hundred) are said to be the most difficult, and may thus represent as much as half of the total effort.

Moreover, Phase II is only one system, within one service, and the selected hardware, however good it is at the time of delivery, will be hopelessly obsolete by the end of the decade and, under federal rules, unreplaceable except by a new multi-vendor competition.

In an ARTE, only the kernel would need re-interfacing to transport a system; that would be some orders of magnitude less expensive to the taxpayer.

RELATIONSHIP BETWEEN APSE AND RTE

Where would one find an ARTE? Certainly the most obvious possibility is to utilize the structure of an APSE and replace some of the tools in the MAPSE with transaction and database oriented functions, and utilize the same core KAPSE for both APSE and ARTE.

This idea's merit becomes clearer when one looks at the systems given as examples above, and realizes that, in both cases, the identical operating system (the major part of a KAPSE) is used by both the programming-support centers and the operations centers; in fact, operations often shares the same processor with development on a time sharing basis.

It is then worth examining whether a KAPSE can also be the kernel for an RTE.

Differences in KAPSE Databases

The STONEMAN document is a generic statement of the goals of PSEs. Therefore, it is not unexpected that "instances" of environments supposedly designed to meet these goals differ substantially in matters important to the use of a KAPSE for an ARTE kernel. Indeed, the definition is so loose that it is possible to produce conflicting (and definitely nontransportable) software based on different KAPSE implementations.

Key areas are in the database support mechanism. The SofTech Ada Language System implements a hierarchical "database" structure based on the concept of trees of nodes, attributes, stream storage (unformatted), etc. This database structure is well suited for "bulk" storage (of program text, compiled code, and unstructured small files). The Intermetrics Ada Integrated Environment implements a "semirelational" sort of database that uses a mechanism similar to directory trees to locate each stored element (attribute, unstructured file, or indexed file). Both of these environments basically place the responsibility for the structuring of data with the tools and programs that lie outside the KAPSE (e.g., compilers, program library managers, etc.). Two European efforts (the UK and the EEC) place in the KAPSE the structuring mechanism for program support access to code trees, program-library configuration data, and so on.

Other key areas include "options" such as configuration management. The SofTech and Intermetrics effort both place this responsibility as an embedded function within using tools, rather than as a kernelized system service. (An effort to implement configuration management in a transportable and interoperable manner is presently funded to CSC; however, this is being provided as a MAPSE tool, rather than being integrated into the run-time environment.)

A Structured Database Project for the KAPSE

There is one known attempt to place a "structured" database onto a KAPSE environment. The continuing success of this effort could be used to base an argument that KAPSEs, in general, are capable of supporting the form of database
required by transaction-based logistics systems, such as the two discussed above.

The Computer Corporation of America’s Adaplex effort is developing an Ada-compatible DBMS. Their approach is based on an entity and function database structure, with typed data. Adaplex uses a database model said to furnish more capabilities than either the hierarchical, network, or relational database models. Although Adaplex is being implemented to run on a DEC VAX computer, using the SofTech KAPSE and MAPSE toolset, the authors of Adaplex intend to use the VAX VMS operating system calls to access the disk directly (instead of utilizing the SofTech KAPSE for disk access).

Although Adaplex is coded in Ada, it will be non-transportable because of its VMS dependencies. The SofTech KAPSE could probably be redefined to include the appropriate disk-access services. Yet even given that the KAPSE contains the necessary services, the SofTech KAPSE is non-transportable because it is heavily dependent on VMS services and non-Ada internal coding. The KIT and KITIA have efforts to create standardized KAPSE interfaces; fruition of interface standardization would allow the Adaplex DBMS to be transported to other KAPSEs.

Given that (a) the Adaplex can eliminate VMS dependencies by a change to SofTech’s KAPSE services and (b) the KAPSE interface-standardization efforts meet with success, one would postulate that Adaplex could become an instance of a transportable DBMS usable for constructing an ARTE. This example would place the DBMS in the tool layer surrounding the kernel, not in the kernel itself.

Using a Non-Ada DBMS in an APSE

The enormous cost of implementing a DBMS from scratch, along with the necessary “optional” tools (applications generators, transaction terminal handlers, etc) has led to the study of incorporating existing, non-Ada DBMS implementations with Ada environment tools and programs. A proposed solution allows structured queries and updates from Ada packages to map into non-APSE DBMS primitives, by translating the operations and binding the data. Thus, a set of package interfaces “crosses” the APSE domain into “foreign” DBMS facilities, without going through a standard KAPSE interface. Existing DBMS tools, application generators, terminal handlers, and English-like or well-designed query tools of the original environment are not available to the APSE user, except through the underlying operating system. This shortcut solution thus does little to provide for transportability and interoperability at the DBMS application level.

USING AN ARTE DBMS IN THE APSE

Conventional DBMSs do not make a good job of dealing with bulk data in the form of the so-called unformatted file or in the form of libraries of nonhomogeneous data, such as a set of programs that have been partially or fully link-edited as a system ready to be executed. Such data occurs in streams of bits representing machine structures: words, or bytes, or paragraphs, or syllables, or blocks, and so on. The stream or its parts may be directly or randomly accessible, or it may only be serially accessible. Such data are sometimes called unstructured data, and the database said to have no knowledge of the internal form of the data. This may be true in some cases, but it does not capture the essential difference. For example, in Ada, a file is said to be “associated with an unbounded sequence of elements, all of the same type.” It can be argued that the system is required to know the element type, to ensure that all users access it using the same element type.

A suitable treatment of bulk data is essential in the KAPSE, because the entities that are controlled through a programming support environment (PSE) are primarily associated in storage as bulk text (e.g., Ada source and compiled objects). The normal way to deal with bulk data in the past has depended on the usage of that data. If the data were an entire system, a program, or a part of a program, and so on, it was placed in a “library’ that could access the program-unit by its name. The structure of the program-unit was generally simple or nonexistent. If the data were to be accessed by a procedure, they were stored as a relatively conventional set of records in a file or in some similar fashion (indeed, they could be stored as a stream of characters or even as a stream of bits, but the procedures and the supplied access methods were the only way that the data structure was known). One of the special Ada data structures is, of course, the Diana tree. This has a structure that has been standardized, and the fact that the Diana tree has needed to be made a standard is an interesting example of the need for standards within the KAPSE or with the MAPSE levels—in order to allow inter-tool action at that level.

Clearly, if a DBMS is to be provided for use by the PSE, it would be very convenient and useful if the same DBMS were also suitable for applications use. This would mean that there was a DBMS in the KAPSE to allow for the addition of a formatted data concept to the PSE; as a result, the DBMS could be used for other functions (as discussed later) and if the same DBMS were to support such actions as an ad hoc query, then the KARTE would potentially have the same interface DBMS.

Classes of Data Supported

As already discussed, there are at least three classes of data that are important in the Ada environment. These are:

1. Unformatted data.—This is a broad class of data that may have structure, but which have no structure that may be known to any program or procedure except through special communication from a programmer. This class of data could be a report in character form (possibly internally indexed, or part of a word processing system with a retrieval mechanism), or it could be a traditional file (with a well-defined file structure that needs a special access method—such as ISAM—to act as
an indexing device for rapid retrieval), or a "bucket of bits" that might be the results of a transmission or a program. All types within the class of unformatted data have one characteristic in common—they consist of a stream of bits that may have structure, but this structure is unknown outside the suite of procedures that access the data.

2. **Standard formatted data.**—Such data have a predefined format that has been previously defined by a community of users who are on their honor to see that all tool-using devices are used consistently. Typical of these data are the groups of procedures that work on a common data structure. In a run-time environment these may be a suite of personnel programs that provide accounting, payroll, and personnel support services; in Ada the compiler and editor interactions (via Diana trees) are examples of standard formatted data systems. The difficulty with such systems is that they rely on the good will of the users (or the hard heads of auditors) and often are violated, either deliberately or in error.

3. **Fully formatted data.**—These are seen in database-managed systems and in some structured PSEs. The essential characteristic of such data is that the environment is aware of the structure of data as a whole and of any part of them. For example, the PSE being designed in the UK has a built-in structure which assures that the program parts are properly controlled—thus the subprograms of a main program are associated with it; moreover, the particular version of the subprogram that is valid with it. Thus the "structure" of these data is an exact match of the configuration that must be managed. The run-time environment, then, will be able to retrieve the required version of the main program and with it all relevant and correctly versioned copies of its subprograms.

**APSE AND ARTE INTERACTION**

The ARTE and the APSE are not really easy to differentiate, yet an attempt to do so for the programming environment has led to the definition of an entity termed a KAPSE. And it appears that in the time since the STONEMAN document was approved the Ada community has forgotten that the prime reason for a programming environment is to provide programs that can be run—presumably in a run-time environment. The idea of a host-to-target environment, of course, has led to some of this apparent neglect, but most target machines have a need for some environment, and it seems reasonable to assume that future target-machine architecture will benefit from the definition of a standard run-time environment. Naturally, any other run-time environment that interacts with other machines/computing devices will benefit even more if a standard KARTE exists.

Much then has been said of the KAPSE, but little of the KARTE. If, as appears very likely, the Ada language is used to implement logistic and other large-scale nonoperational (large-scale administrative) systems, then some of the requirements of a KARTE that were only marginally necessary in the KAPSE will be more essential. These features include the need for security features; a way to store data structures and define the meaning of the data entities (an information-resource dictionary); a means for storing and retrieving data based on these definitions (a generalized DBMS with good user interfaces for query, table generation, and reporting); methods for recording program structures, their relationships to data and users, and so on (a good software-configuration-management system); and interfaces to the documentation, which can be a part of the combined configuration and information-resource management system.

There are at least three types of security that should be investigated: user checking, procedure validation and initiation, and data sensitive checking.

The DBMS-like features should include the capability of interfacing to a dictionary. Modern dictionaries have many different features, but they are all generally able to capture compiler data to document data and program usage. Some are even able to hold information on the users, some security needs, and some configuration-management controls. The controls between a DBMS and a dictionary, and even the configuration manager, are implemented through an active interface between the dictionary and its users (automated or human). The value of the dictionary is only fully realized if it can operate in this controlling role.

**CONCLUSIONS AND A PROPOSED ARCHITECTURE**

The purpose of this paper was twofold:

1. To show that there was a need to consider the use of the KAPSE as an ARTE.
2. To suggest some of the problems in making the transition from one to the other.

It does seem, however, that there is a possibility of using architectures that have been developed for operating systems, DBMSs, and information resource dictionary/configuration management systems in the past. One possible high-level architecture was given in Figure 2.

This shows that it would be possible to use the general architecture of the STONEMAN KAPSE for a combined programming and run-time support environment, but some additional controls will need to be added to the system, and the configuration management system would need to have control of any access to the libraries.

However, the use of an Ada environment that had realized these features would make future software more easily transportable and allow real "software reusability," thereby reducing the rising costs of software while allowing major systems of the future to be implemented in spite of the expected "gap" in available programmers and systems implementors.

**ACKNOWLEDGMENT**

The authors are grateful to Mr. Leo Meany of the U.S. Air Force Data Systems Design Center, who provided information on AFOLDS.
REFERENCES
