A REVISED STONEMAN FOR DISTRIBUTED ADA* SUPPORT ENVIRONMENTS

Jeremy P. Goodwin**

Department of Computer Science
Virginia Polytechnic Institute
and State University
Blacksburg VA 24061

Report No. CS830010
June 21, 1983

ABSTRACT

This paper extends the conceptual model of the "STONEMAN" document to more completely model the interfaces and protocols that exist in the Ada Programming Support Environment (APSE). A previous extension to the STONEMAN model is reviewed and critiqued, the guidelines for the APSE set forth in STONEMAN are reviewed, and an updated model is proposed. The new model is shown to meet the guidelines set forth in STONEMAN, and to include subsequent ideas as well. The new model is then applied to the problem of user communication with an APSE, and it is shown how the new model extends to include distributed APSEs as well as single host APSEs. The issue of security enforcement, as a necessary subset of dynamic verification, is also included in the new model.

---

* Ada is a registered Trademark of the Ada Joint Program Office of the U.S. Department of Defense.

** This research was supported by the Ada Joint Program Office through the Office of Naval Research Information Sciences Department. The Principal Investigator for the grant was Dr. T.E. Lindquist, and the development work was supervised by Dr. J.A.N. Lee.

I. INTRODUCTION

A fundamental objective of the Department of Defense (DoD) initiative to develop Ada was to increase the portability and maintainability of embedded software [1]*. To achieve this objective, the Ada Joint Program Office (AJPO) is working to ensure that Ada remains as independent of computing systems and applications as possible. The Ada language has been accepted as an American National (ANSI) standard, and has been proposed as an International (ISO) standard. The Ada Validation Organization (AVO) has been set up to enforce and protect the standards for the Language. However, the Ada project has evolved beyond an effort toward a common programming language for embedded software systems. In addition to the Ada language standardization effort, work has been done to define requirements for a common (standardized) Ada Programming Support Environment (APSE), and its kernal computing system interface (KAPSE). These programming support environments are an integral part of the Ada standardization effort because they will provide a standardized development and runtime environment for Ada programs.

A previous report [2] recommended that the Open Systems Interconnection model be accepted as the underlying model of APSEs, and that there be developed a 'Strawman' to extend Ada systems into a networking environment, based on the OSI Reference Model. This report further suggested that the security aspects

---

* Numbers in brackets refer to references at the end of the report.

of the design of APSEs be investigated and that the results of this study be incorporated into the Stoneman requirements [3]. This paper will examine these ideas further, criticizing the model suggested by [2] and proposing an updated and more detailed model for communication between APSE programs that is based on the OSI Reference Model and that takes into account the need to extend Ada systems into a network environment. This model also incorporates a security layer, as recommended by [2].

II. REVIEW OF PREVIOUS MODEL

This section reviews the APSE model suggested by [2]. In that report, the authors note that "The original intent of the OSI Reference Model was not to actually represent an implementation strategy but instead to model those elements of a communications environment which need attention" (pg 8). In other words, the OSI model is not intended to force all implementations to have seven layers, but rather to encourage all implementors to layer their implementations, and clearly specify which functions are being implemented by each layer. Thus an application of the OSI model to a specific implementation could quite conceivably merge several layers into one, and split one of the OSI layers into one or more layers. This argument is given as a justification for the model that the report suggests should underlie all APSEs. This model has three layers, which correspond roughly to the top three layers of the OSI Reference Model. The bottom four layers of the OSI model (the ones

typically implemented in hardware) are not present in the proposed model for APSEs. The top layer (the OSI Application layer) is mapped to the APSE portion of the environment (as distinguished from the KAPSE portion). If the current environment is a minimal Ada environment, (a MAPSE), then the top layer will only include the necessary and sufficient toolset, not user programs or additional tools. Conceptually then, there is no difference between an APSE and a MAPSE. The second layer down (corresponding to the OSI Presentation layer) is renamed the Data Transfer Layer. This layer is intended to act as an interface layer between the APSE and the KAPSE, and to implement the verification and security mechanisms suggested in the report. The middle layer accomplishes all matching of formal and actual parameters, and associated typechecking. The bottom layer is the KAPSE layer, and is mapped onto the Session layer of the OSI model. The report notes that the KAPSE will be implemented as a collection of Ada packages.

This model has the advantage that it clearly delineates what interfaces and protocols exist. The report called attention to the existance of certain "hidden protocols" within the current STONEMAN model, and noted that these presented a difficult verification problem. In the model suggested herein, these hidden protocols are no longer hidden, they are revealed as KAPSE layer to KAPSE layer communication. Furthermore, the new model is better than the STONEMAN model because it takes into account both verification and security. The STONEMAN model was not detailed enough to represent these problems.

III. PROBLEMS WITH PREVIOUS MODEL

The previous model has three basic problems. The first, and most basic, is that it makes the wrong use of the OSI model. Since the OSI Reference Model is a model of a communications environment, and not a programming environment, it is inappropriate to overlay the Ada Programming Support Environment on top of it. The fact that the bottom four layers of the OSI model have been ignored in the APSE model shows that the model produced for the APSE does not quite fit onto the OSI model, and therefore the OSI model should not be used as the directly underlying model. Nonetheless, the principles of the OSI model are very much appropriate and ought be adopted in the design of the APSE environment. Therefore the APSE should be layered, and the internal implementation of one layer should be changeable without necessitating a revision or rewriting of code in the other layers. Furthermore, specific functionality should be assigned to each layer of the implementation, and this functionality should follow the overall design principle of virtualization, where the functionality of each layer is implemented in terms of (and by calls to) the functionality of the layer immediately below.

A second problem with the model suggested by [2] is that it fails to take into account (even to mention) the Data Base model [4]. The APSE model should be well integrated with the data base model, so that the design of the total environment is consistent

and so that the interfaces between the Data Base and the KAPSE are well defined and easy to validate.

The third problem with the previous model, and the problem with the STONEMAN model before it, is that the model is not well developed enough. It does not show where all the verification mechanisms are to be installed. The report states that all interfaces and protocols must be validated in order to validate the environment, and it is this need that inspires the Data Transfer layer, where validation of the APSE to KAPSE interface is to occur. The problem is that this is not the only place where validation of protocols or interfaces needs to occur. All the horizontal protocols between two instances of layers at the same level must be validated. For instance, a compiler could be communicating with an editor. Both these programs would reside in the top layer. They would communicate via a virtual protocol which would be implemented by calls to the Data Transfer layer. The previous model does not provide a mechanism whereby this protocol is verified. Another place where the previous model needs further development is in the distinction between dynamic and static verification. The verification mechanisms suggested in the report are all static, yet the report suggests the creation of a layer where verification occurs. This verification would be dynamic, and the report does not expand upon the mechanisms by which the verification would be accomplished. It should be noted that there are two types of dynamic verification: dynamic verification in the development environment (ie during the time when the program to be verified is being written), and

dynamic verification in the runtime environment (verification that is done during the time when the program to be verified is running). The first type of dynamic verification occurs in the Ada Programming Support Environment on the host machine, and the second type of dynamic verification occurs in the Ada Runtime Environment on the target machine. Both of these sets of verification mechanisms must be investigated.

The report also mentions the need for some security mechanisms, and suggests that they also be implemented in the Data Transfer Layer. However, the report does not expand on the different types of security mechanisms, nor does it distinguish between friendly and non-friendly or security intensive environments in its discussion of the need for security mechanisms. If a program exists in an environment where security is important, and that program is not secure or does not meet the security requirements associated with its environment, then that program is not valid for that environment. Thus we recognize two fundamental principles about security: (1) Different instances of Ada Environments may have different security requirements, and therefore a program that is correct in one instance of the Ada environment may not be correct in another instance of the Ada environment because it is not secure enough, and (2), security is a subset of validation, because a program that is not secure (enough) is not correct. These ideas clearly need much more attention before a final model for Ada environments can be approached. Another shortcoming of the previous report is that it did not provide sufficient motivation for the model. The OSI

model is no doubt a structured model that would provide a framework for the development of the APSE, but it is only one model for software. Concurrent processes, data driven, rule based or relational models might all apply. More rational needs to be put behind the OSI type model before it should be adopted as the final model type.

Most of these issues were not addressed in the previous report because it was preliminary and did not go into the level of detail that would be necessary to deal with all these issues. This paper attempts to investigate all these issues and refine the model in the light of its findings. Thus the process of successive refinement of the model is carried one more step, arriving at a new model more complete than the earlier model. No doubt further iterations will need to take place.

IV. REQUIREMENTS SECTION

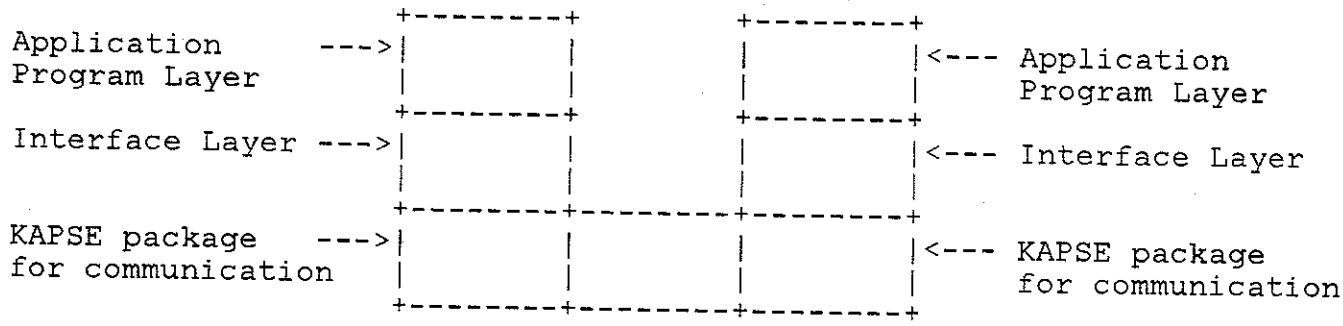
Before updating the previous model the requirements for the APSE should be reiterated, and augmented by the ideas that have been put forward since STONEMAN. The basic STONEMAN philosophy emphasized fourteen general guidelines (see section 3). Five among them are (1) long term software support, (2) host to target system software portability, and (3) an integrated database and toolset, (4) overall simplicity, and (5) uniformity of protocol. Since STONEMAN other principals have been proposed. The user interface should not be direct to the APSE programs or tools, but should be a virtual interface to the KAPSE with minimal JCL

functions such as LOGIN/LOGOUT, CONNECT/RUN and control character processing. Terminal interface drivers should not be part of the KAPSE, but should interface to it, much as the APSE tools do. This model of user communication with the system is similar to one used in many operating systems: user I/O is handled by a special I/O processor (hardware, software or firmware) and buffered into the kernal of the operating system. A second concept that has received attention since STONEMAN is the configuration or version group, made up of shareable objects, each of which has a name and attributes. Each object in the version group has a date as one of its attributes, with later dated objects superceeding earlier ones in subsequent configurations. A third concept, which is related to the second, is that the APSE environment may in practice be distributed, either between the host and target machine (for down-line loading, trace data collection, or emulation) or between several hosts (for resource or information sharing or communication). The problems of managing a distributed system are related to those of managing a configuration with many dated versions of the same objects, so the design of the APSE should encompass both concepts as one goal.

V. DESCRIPTION OF THE UPDATED MODEL

The updated model (see figure 1) is a three layer version of model presented in [2]. The top layer is called the APSE or Program layer since it includes all tools or application

Figure 1



Updated Basic APSE Model

programs. It is analogous to, but not overlayed upon, the Application layer of the OSI Reference model. It is this layer that the user views himself to be at, since the programs he interacts with are at this level: editors, compilers, debuggers, etc. If this layer contains only the minimal necessary tools and not any other tools or applications programs, then it is a MAPSE. Thus a MAPSE is a minimal instance of an APSE.

The middle layer is called the Interface layer. Interface is meant in the sense that it has been used in STONEMAN, and this layer has grown out of and is an expansion of the interface line between the APSE and KAPSE in the STONEMAN model. The parameter passing mechanisms that match the APSE actual parameters to the KAPSE actual parameters and perform dynamic typechecking between them exist in this layer. These mechanisms, formerly represented as the thick line between APSE and KAPSE, have been made into a level due to their complexity. The security mechanisms called for by [2] also exist in this layer and work along with the typechecking mechanisms. The principle is to detect and stop

security exceptions at as early a point as possible. This layer dynamically prevents protected KAPSE packages from even being called. System verification procedures can verify that the security layer properly performs its function, and then use that assertion when verifying the KAPSE facilities. The Interface layer performs all dynamic interface verification, including but not restricted to typechecking and package access control. In addition, any necessary data transformation operations may be performed within this layer. This layer will, like the KAPSE, be implemented as a set of Ada packages, with a correspondence between the names of the KAPSE level packages and the Interface level ones. The KAPSE level packages will only be visible from the Interface layer, and the APSE programs must call a KAPSE facility by calling the appropriate Interface layer package. The Interface layer will perform its verification and conversion functions, and then invoke the KAPSE package with the verified and possibly modified argument list.

The bottom layer is the KAPSE layer. It is a set of Ada packages that implement the kernal function of the Ada programming support environment. The KAPSE is the only layer that will be implemented differently on different host machines, but regardless of implementation its function will be the same over all instances of the APSE. Part of the KAPSE may be implemented in another language, such as the operating system language of the source machine, but as much as possible should be written in Ada itself. No other layer should have any non-Ada code. The functionality of the KAPSE shall be defined in such a

way as to provide a general purpose set of operating system type primitives robust enough to implement the rest of the APSE in, but not defined in such a way as to prejudice the implementation of the KAPSE toward any one particular architecture among those on which the Ada environment must run.

Although three layers have been defined, this does not preclude each layer from being broken down into sub-layers in implementation. This is, in fact, anticipated as being the natural outgrowth of the development of a system whose underlying model is a layered one.

VI. RATIONAL FOR A LAYERED MODEL

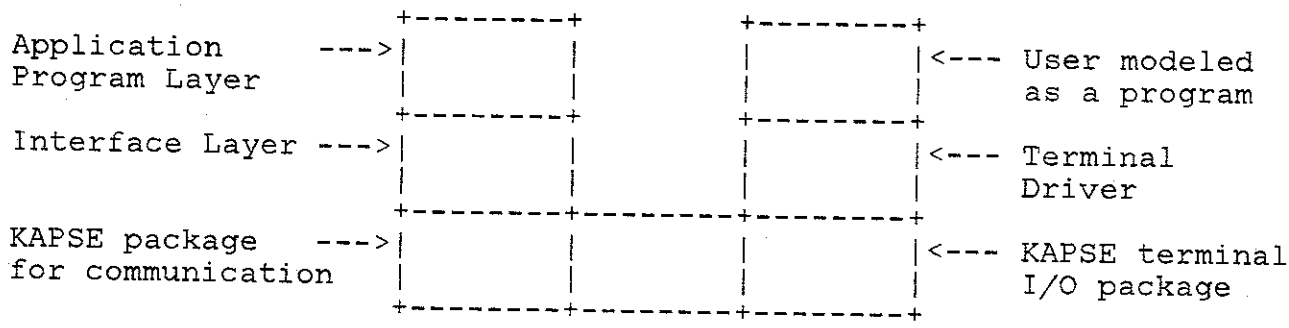
A layered model follows the guidelines set forth by STONEMAN. It aids long term software support by defining and consistently maintaining the functionality of the KAPSE, so that internal modifications to the KAPSE layer will not affect application software, or any other software above the KAPSE layer. Since any program will only reference those programs (or packages) in the layer immediately below its own, and since the functionality (but not necessarily the implementation) of those programs or packages is fixed across all systems, portability for all software above the KAPSE layer is enforced. Since all protocols and interfaces are clearly defined for the entire system, no hidden protocols exist. By rigidly enforcing one means of inter-tool communication through the KAPSE, an integrated toolset is arrived at and uniformity of protocol is

enforced. A layered model is at the same time both simple and complete. For the applications programmer, the APSE is a set of procedures, packages and tasks that are visible to his program. Thus the concepts of the APSE are those of the Ada language itself, and the most straightforward possible. By judiciously grouping the visible packages into a small but well organized set of packages the functionality of the level below can be preorganized for the programmer. Only the necessary portions of the packages need be made visible, and the rest of the underlying layer need not concern the programmer. This model makes the APSE an easy environment to use as a programmer and to maintain as an APSE maintainer, for the same reasons that abstraction and virtualization make any system easier to understand and maintain.

VII. BASIC MODEL APPLIED TO USER COMMUNICATION

The issue of how a user will interface to the APSE is one that should not be overlooked or put off until late in the design phase if it is to be a natural and consistent interface. The layered approach handles the user much as if the user were in fact an application program: that is, there is no difference between communication with the user and communication with another APSE program. All communication goes through the KAPSE, and the KAPSE routes the 'message' back up through the appropriate layers to its destination (see figure 2).

Figure 2



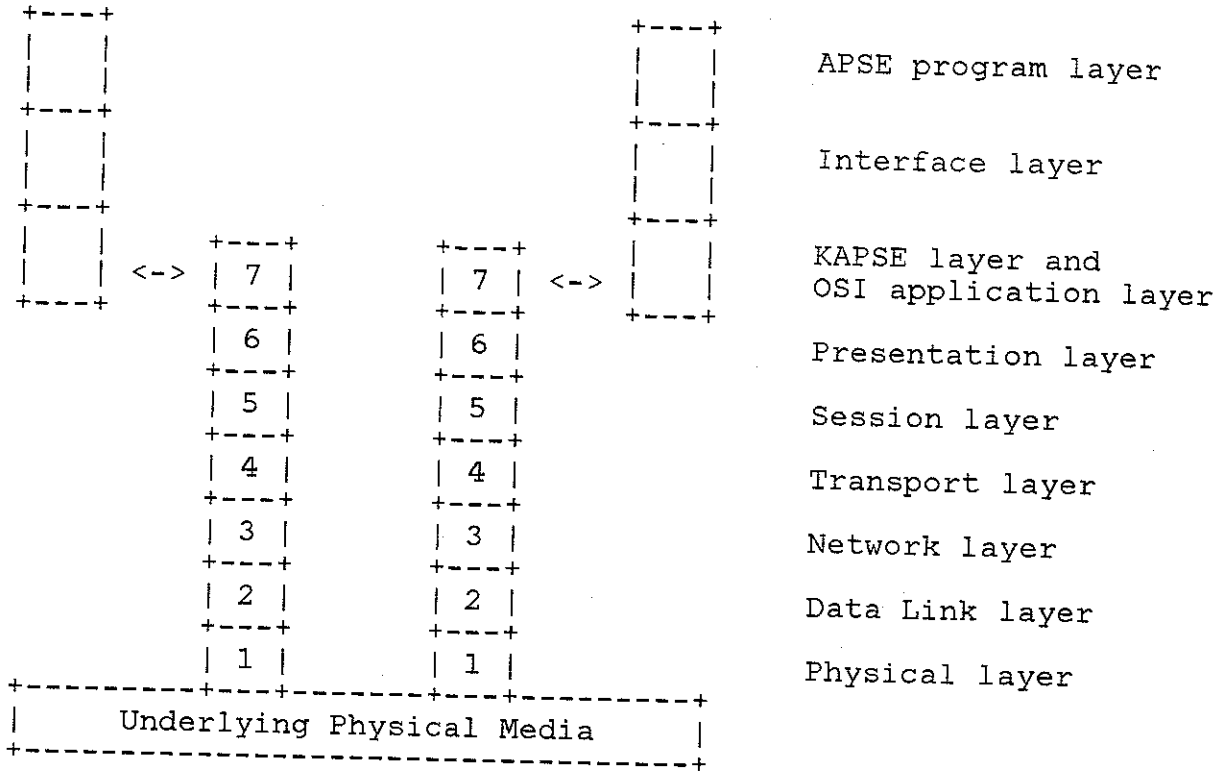
Basic Model as Applied to Communication with Users

VIII. BASIC MODEL FOR DISTRIBUTED APSES

The concept of an APSE as a distributed system or a configured system is also encompassed by the layered model. As in the OSI model itself, the upper layers do not have any knowledge about the physical location of the peer process with which they are communicating. The KAPSE layer to KAPSE layer protocol is the only place where the actual locations of the source and destination programs must be considered. If the source and destination are on the same physical host, then communication may be as modeled above (see figure 1). If not, then the communication is via the network linking the two hosts. It is intended that the networks used to link distributed hosts also be modeled after the OSI reference model. Thus the model is expanded to the version shown in figure 3 in the case of distributed communication. In fact, only a part of the KAPSE layer, the package specifically concerned with communication,

rather than the entire KAPSE layer, need be concerned with

Figure 3



Basic Model Extended to a Distributed APSE Model

whether or not the source and destination are on the same host.

IX. RATIONAL FOR THE INTERFACE LAYER

Since the other two layers have been present in all APSE models thus far, there is no need to motivate their presence in this model. The Interface layer, however, needs further motivation. The first argument in favor of including this new

layer in the model is that this layer was really always present. Previously, the interface between the KAPSE and the APSE was presumed to perform all of the functionality now assigned to the Interface layer. No new functionality has been added, except for the need for security mechanisms as a subset of the verification mechanisms. The case for the inclusion of security mechanisms somewhere in the model has already been made by [2]. The Interface layer is where these mechanisms belong, along with the other dynamic verification mechanisms. The second argument in favor of the new layer is that the interface mechanisms are too complex, and their effects are too far reaching to ignore in the design of the APSE. If the interfaces are to be designed then they must be visible as a part of the model. They should not be allowed to fall into the crack between the APSE and KAPSE. Thirdly, since verification is a necessary step in the life cycle of all APSEs, it is better to provide for it ahead of time. This was the basic thrust of the recommendation of [2], that validation requirements be established and included in APSE requirements specifications. Finally, since security must be considered to completely validate an APSE system, it is better that the security mechanisms also be designed into the model from the beginning, rather than added after the design has been completed. This is more true of security mechanisms than of others because of the subtle nature of security failures and the low level at which most security mechanisms are implemented.

X. CONCLUSIONS

The two part model presented in STONEMAN is no longer descriptive enough to model all the considerations that have been added since STONEMAN. A previous report originated the idea that the OSI model be used as the underlying model of APSEs. It is recommended that a three layered model, patterned after the OSI Reference model, be adopted as the underlying model of APSES.

This report and a previous one have motivated the need for dynamic verification. It is recommended that an investigation of dynamic typechecking and verification techniques be performed, and that the results be incorporated into the design of the middle layer of the proposed model.

Security, as a subset of dynamic verification, is a necessary consideration in any verification system. It is recommended that a security mechanism be designed and incorporated into the middle layer of the proposed model as early as possible to enforce the assertion that no KAPSE facility can be called unless the appropriate security test has been passed.

It may be found that the security mechanisms proposed by the research recommended above are difficult or impossible to implement in Ada as it is now defined. It is recommended that subsequent research be undertaken to investigate ways in which Ada could be extended to include security and verification mechanisms as a part of the language, rather than as a part of the APSE or Ada Runtime Support Environment.

XI. REFERENCES

[1] Carlson, W.E., Druffel, L.E., Fisher, D.A., and Whitaker, W.A., "Introducing Ada", Proc. 1980 ACM Ann. Conf., ACM, New York NY, 1980, 539 pp.

[2] Kafura, D.E., Lee, J.A.N., Lindquist, T.E, and Probert, T., "Validation in Ada Programming Support Environments", technical report, 1982

[3] Buxton, J.N., Requirements for Ada Programming Support Environments, "STONEMAN", U.S. Dept. of Defense, February 1980, pp. 50.

[4] van Griethuysen, J.J., ed., Concepts and Terminology for the Conceptual Schema and the Information Base, ISO TC97/SC5/WG3, ISO TC97 Computers and Information Processing, March 1982.