TAGGED ARCHITECTURES AND PROTECTION: MESSAGE SYSTEMS

by

Edward A. Feustel

Laboratory for Computer Science
and Engineering

October 1, 1975

TECHNICAL REPORT #7514

TAGGED ARCHITECTURES AND PROTECTION: MESSAGE SYSTEMS*

Edward A. Feustel†
Rice University, Houston, Texas

Keywords and Phrases

Protection, computer architecture, messages.

Abstract

Recent work in extensible programming languages and operating systems by Redell, Lampson, Hewitt, and others coupled with developments in distributed computer systems motivates a study in models of the implementation of protection on tagged architectures. An extension of the notions of message and actor as discussed in previous work provides an abstract framework which may lead to the development of a suitable virtual machine. We wish to show that such a system provides a good framework for representation and protection of types, data, and procedures.

Introduction

This note on work in progress presents some preliminary ideas on the semantics of protection for tagged architecture [1]. It proposes a framework for a protection system suitable for use in a distributed computer system. First, it develops the system for application in the general case; then it discusses degenerate examples more often encountered in simple uniprocessor computer systems featuring a centralized file system. It does not deal with detailed questions of implementation but tacitly assumes that system efficiency and reliability within the framework are facilitated by the use of tagged architecture.

The literature of protection systems (as surveyed in [2]) develops several views as to the best methods and mechanisms for protection. We favor the view ([3, 4, 5, 6, 7, 8, 9, 10] for example) that capability architecture is the best solution. We will extend the work of Lampson [11] on messages and Redell [12] on types. We wish to show that an advanced message system provides a good system for representation and protection of types, data, and procedures.

Messages

First, we will develop an adequate notion of message generalizing it as necessary in the ensuing discussion. Let us start with the conventional notion of a message as defined in communication systems. Such a message consists of a header and a body. The header consists of a destination, a sender, a priority, and a time stamp. The header also may contain information about privacy transformations which have been employed on the body [13]. The body is of no concern to the agency transmit-

*To appear in the proceedings of the Fourth Texas Symposium on Computer Systems
†This work supported in part by NSF grants: GJ36471 and DCR72-03609A01

ting the message, the bearer; only the header or portions of the header matter and must be provided.

In the context of a computer system, the body of the message may cause different actions on the part of its receiver, the actor [14]. The body might be data, a command to perform a task, or a method to accomplish a task at a later time. For our purposes a body may be an accretion of N bits of information whose value is itself (immediate mode).* It may be the name of another message (reference mode). It may be a partial closure message [15] (function mode). Or it may be a closure message (command mode).

The header for a computer system is more complicated than for a communication system. In a communication system the only requirement that the bearer places on a message is that it have a destination. While such anonymous messages are useful in this context they pose a threat to secure operation as shown by Lampson. We will insist that the header contain a message giving the identity of the sender (potentially protected from the bearer by a privacy transformation). This signature may be used in authentication [8].

In the hands of the actor, the type of the object and access to the object become crucial. The header must include a specification of the type, possibly in the form of a message. Further, the header may include a privilege message, an access message, an update message, a copy message, and an integrity message. These later messages detail precisely how the body of the main message may be manipulated or used as function or command.

As can be seen from the foregoing, messages may have a complicated structure. We choose the following modified definition to be more useful. A message is a machine readable marker followed by a sequence of messages, which may or may not be encrypted. The marker must denote the presence or absence of the clear destination, the privilege message, the access message, the update message and the integrity message. It must inform the architecture of the relative position of all messages within the accretion of bits; it must indicate the length of the whole message.

Our use of the marker permits efficient representation. An integer number possessed by an actor might consist of a marker, a clear type code, a parity bit, and a body containing a string of binary bits. In this case the marker would indicate to the architecture that the actor was using, the length of the message, the absence of destination, signature, privilege, access, update, and copy messages, and the positions of the type code, the parity bit, and the beginning of the bit string. The type code would identify the string of bits as an integer number of a desired precision.

The use of the message permits the definition and use of complex structures such as files. We observe that in common usage a file contains sequences of records (possibly of varying length). Individually the records consist of procedures and data

*An immediate mode object of zero length is the null message.

which are represented as sequences and nested sequences; i.e., block structured functions and structures. The representation of files as messages is very straightforward.

The Bearer

The function of the bearer of the message is to accept the message without regard to contents, to assure the integrity of the message, and to transmit the message correctly from the actor sending the message to a second actor who receives the message or to store and forward the message. Each bearer may apply a privacy transformation to messages which it transports. Each bearer may group messages for a common receiver, imbedding them in a message which it generates at the source and decomposes at the destination. No bearer may permanently record a message; no bearer may examine any part of a message except the marker and destination, except for purposes of privacy transformation or generation of its own integrity messages.

The Actors

The notion of an actor [14] is specialized for this paper. The actor may be regarded as an entity that may store messages, may apply partial closures to messages, may execute commands, may generate messages, and may send messages. The actor has access to a virtual machine to which it may issue instructions but it does not have direct access to the physical representations which the machine manipulates. For example, the actor may only deal with symbolic names and strings of symbols. It may not generate an address to examine a bit pattern in memory; but it may generate an association between a symbolic name which it has and a symbolic object which it wishes to manipulate. The actor is not a bit pusher (in the same manner as one can program in assembly language) but a symbol slugger (in the same manner as one programs in TRAC* [17] or SNOBOL [18]). Thus the actor's access is limited to those things which it can name, can compute, or to which it can persuade another actor to give it access.

Possession of the name of an object does not imply full access to the message associated with that name. For each message (process) which the actor executes as a command conveys to the actor an identity. The identity message is conveyed to the virtual machine which examines the access/update/copy/type (if present) for each attempted access/update/copy/computation to determine whether the actor may or may not pursue its intentions. If no message is present, the actor may proceed. If the contents of the message is an immediate representation, the virtual machine determines whether the pattern permits the computation. If the contents of the message is a name accessible by the identity of the actor, then the representation associated with the name is retrieved and used as in the previous case. If the contents is a partial closure, the identity and the type of computation are provided to complete the closure and the returned representation is used. In the same manner,

*TRAC is a trademark of the Rockford Research Institute

a closure returns a representation. All of these operations are transparent to the actor. If the named object is not present or if access is denied, a null element is returned so that the actor cannot infer the presence or absence of the object.

Names

The notion of a name is very important since a name may be associated with a representation, a name message, a partial closure message or a closure message. As in PAL [19], a few basic functions for construction, selection, test, reproduction, application, evaluation, definition, sharing, recursion, iteration, arithmetic, and logic are provided; as in EL/1 [20], functions are provided to facilitate dynamic binding, mode conversion, and mode transmission. In addition, special functions for sending and receiving messages, queueing, performing privacy transforms and prioritizing are included. All these functions, together with their names and user representations, i.e., true, false, integers, reals, characters, etc., constitute the basic universe of discourse [19] which all other computations are described. They constitute the available virtual machine "instructions and data".

Utilizing these instructions and data one may build very complicated clusters [21] of data and procedures which satisfy the criteria of generality proposed by Dennis [22] and which lend themselves to the suspicious and unsuspicious "subsystem" alike. Layers of protection may be built on top of layers utilizing levels of privacy transformation, levels of access/update/copy messages, levels of aliasing and encapsulation [12], and levels of authentication checking reminiscent of James Bond movies.

Protection

Using message systems, we have the facilities and flexibility required to provide an arbitrary degree of protection. If the security policy demands an object may be heavily encapsulated with multiple privacy transformations and access and authentication checks; all this at the expense of run-time efficiency. If the security policy permits, the privacy transformations and access and authentication checks may be avoided with an improvement in run-time efficiency. In either case the user of the protected object finds these transformations and checks transparent to itself.

Future Work

Our next task will be twofold. We plan to examine the behavior of the actor during the evaluation of a closure (processing a task). Our purpose will be to determine the minimal set of basic functions required to permit cooperating sequential processes and to investigate costs and methods of implementing them in hardware. Second, we plan to develop the notion of message systems more fully and investigate their potential for multiprocessor, large file applications.

ACKNOWLEDGMENTS

To John Iliffe, S. S. Reddi, L. N. McMahan, J. R. Jump and the members of the Rice Laboratory for Computer Science and Engineering for their valuable insights and discussions.

REFERENCES

1. Feustel, E. A., "On the Advantages of Tagged Architecture", I.E.E.E. Transactions on Computers, C-22, 7, July 1973, pp. 646-656.

2. Scherf, J. A., Computer and Data Security: A Comprehensive Annotated Bibliography, Technical Report 122, Project MAC, Mass. Inst. of Technology, Jan. 1974.

3. Dennis, J. B. and VanHorn, E. C. "Programming Semantics for Multiprogrammed Computations," Comm. A.C.M., 9, 3, Mar. 1966, pp. 143-155.

4. Fabry, R. S., "List Structured Addressing," Ph. D. Thesis, Univ. of Chicago, Mar. 1971.

5. Lampson, B. W., "Dynamic Protection Structures," AFIPS Conf. Proc., 35, FJCC, 1969, pp. 27-38.

6. Morris, J. H. Jr., "Protection in Programming Languages," Comm. A.C.M. 16, 1, Jan. 1973, pp. 15-21.

7. Fabry, R. S., "Capability Based Addressing", Comm. A.C.M., 17, 7, July 1974, pp. 403-412.

8. Morris, J. H. Jr., "Types Are Not Set", Proc. A.C.M. Symposium on Principles of Programming Languages, A.C.M., New York, Oct. 1973.

9. Jones, A. K., Protection in Programmed Systems, Ph. D. Thesis, Carnegie Mellon University, June 1973.

10. Denning, D. E., Secure Information Flow in Computer Systems, Ph. D. Thesis, Purdue Univ., May 1975.

11. Lampson, B. W., "Protection," Proc. Fifth Princeton Symposium on Information Sciences and Systems, Princeton Univ., March 1971, pp. 437-443.

12. Redell, D. D., Naming and Protection on Extendible Operating Systems, Technical Report 140, Project MAC, Mass. Inst. of Technology, Nov. 1974.

13. Turn, R., "Privacy Transformations for Databank Systems," AFIPS National Computer Conference Proceedings, Vol. 42, 1973, pp. 589-601.

14. Hewitt, C., "Planner," Project MAC Progress Report XI, Mass. Inst. of Technology, July 1973 - July 1974, pp. 221-282.

15. Reynolds, J. C., "Gedanken - A Typeless Language Based on the Principle of Completeness and the Reference Concept," Comm. A.C.M., 13, 5, May 1970, pp. 308-319.

16. Morris, J. H. Jr., "Authentication Tags -- The Proper Division of Hardware/Software Responsibility," Private Communication.

17. Mooers, C. N., Deutsch, L. P., "TRAC(T), A Text Handling Language," Proc. A.C.M. National Conference, 20, A.C.M., New York, 1965.

18. Griswold, R. E., Poage, J. F. and I. P. Polonsky, The SNOBOL4 Programming Language, 2nd Edition, Prentice Hall, Inc., England Cliffs, N. J., 1971.

19. Wozencraft, J. M., Evans, J. A., Notes on Programming Linguistics, Department of Electrical Engineering, Mass. Inst. of Technology, Feb. 1971.

20. Wegbreit, B., "The Treatment of Data Types in EL/1," Comm. of A.C.M. 17, 5, May 1974, pp. 251-264.

21. Liskov, B. H., and Zilles, S. N., "Programming with Abstract Data Types," SIGPLAN Notices 9, 4, April 1974, pp. 50-59.

22. Dennis, J. B., Programming Generality, Parallelism, and Computer Architecture, Computation Structures Group Memo 32, Project MAC, Mass. Inst. of Technology, August 1968.

23. Reddi, S. S., Feustel, E. A., "An Approach to Restructurable Computer Systems," Lecture Notes in Computer Science, Vol. 24: Parallel Processing, Springer-Verlag, New York 1975, pp. 229-337.