GENERIC ADA CODE IN THE NASA SPACE STATION COMMAND, CONTROL AND COMMUNICATIONS ENVIRONMENT

D. P. McDougall
T. E. Vollman

Veda Incorporated
Lexington Park, Maryland

1.0 INTRODUCTION

This paper describes the results of efforts to apply powerful Ada constructs to the formatted message handling process. The goal of these efforts has been to extend the state-of-technology in message handling while at the same time producing production-quality, reusable code. The first effort was initiated in September, 1984 and delivered in April, 1985. That product, the Generic Message Handling Facility, met initial goals, has been reused, and is available in the Ada Repository on ARPANET. However, it became apparent during its development that the initial approach to building a message handler template was not optimal. As a result of this initial effort, several alternate approaches were identified, and research is now on-going to identify an improved product.

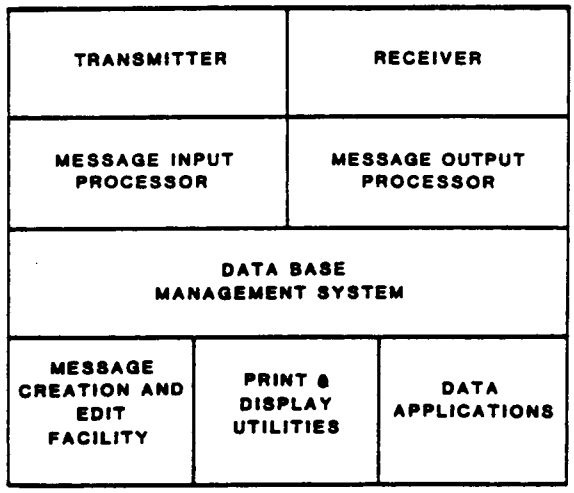
The ultimate goal is to be able to instantly build a message handling system for any message format given a specification of that message format. The problem lies in how to specify the message format, and once that is done, how to use that information to build the message handler. In Section 2 we discuss message handling systems and message types. In Section 3 we describe the "ideal" system. In Section 4 we detail the initial effort, its results and its shortcomings. We then describe the approach now being taken to build a system which will be significantly easier to implement, and once implemented, easier to use. Finally, in Section 5, we offer our conclusion.

2. BACKGROUND

Message handling systems play a major role in command, control, and communications (C3). C3 systems are most often found in military applications, where rapid, accurate dissemination of information is required. Non-military space-related communications systems face many of the same requirements. In this section, we discuss attributes of the message handling systems which support the communications aspect of C3, and we identify the requirements for those systems.

2.1 Message Handling Systems

The typical message handling systems consists of eight components, as depicted in Figure 1. The transmitter and receiver perform the actual communications between this system and some other system with which it is communicating. They handle message blocking, line protocols and other low level functions. They are usually hardware dependent and are typically written in assembler language or in microcode.



MESSAGE HANDLING SYSTEM COMPONENTS
Figure 1

The message input and output processors are the interface between the rest of the system and the transmitter/receiver facility. Usually a message handling system will hold the data in some internal format which makes sense in the context of the applications to be performed upon the data. This format is usually independent of the format in which a message is transmitted or received over any specific I/O line. The message input processor accepts a bit stream input from a line by the receiver, passes it, extract the information and passes it to the Data Base Management System (DBMS). If the system provides for real-time display of incoming messages, the input processor may also pass the data along to a display utility. In a similar manner, when a message is to be transmitted, the message output processor extracts the data from the DBMS, or accepts it from a system operator, and formats a bit string (or character string) to be passed to the transmitter.

The DBMS provides for information storage and retrieval. The data may be stored in message image format, or in some non-message-related format. How the data are stored is typically dependent upon the type of applications being performed upon the data. In systems whose primary function is other than message generation and transmission, the data are not typically stored as message images. In other systems - or subsystems - whose sole or primary function is directly message related rather than data application related, the data are more likely to be stored in message image format. At any rate, when an operator creates a message, he usually wants to see its image prior to transmission; therefore the interface between the DBMS and the Message Creation and Editing facility - the editor - will normally include a utility to extract data from the DBMS and build a message in the specified format.

The editor will provide standard functions such as insert/delete line, cursor movements through the message, and so on. Additional functions to be provided are dependent upon the message format(s), which are discussed below. As we shall see, a critical function is some sort of embedded data validation.

Message handling systems usually provide the capability for visual and hard copy message output, as well as message transmission. In addition to viewing an image of the message they are creating, operators will often want to keep a hard copy of the message after it is sent, both for historical purposes, and for possible future editing.

The final component shown in the figure is not a part of the message handling system per se, but is the reason for data exchange. While there are (sub)systems whose primary purpose is nothing more than message handling (e.o. store-and-forward message drops such as the Communications Line Interface (CLI)), most message handling systems are components of larger systems which perform some applications of the data to non-transmission related problems. The data applications are not treated here; they do, however, impact the format in which the DMBS holds the message data.

Examples of message handling systems include the Force High Level Terminal (FHLT), the Ocean Surveillance Information System (OSIS), the Joint Tactical Information Distribution System (JTIDS), and the World Wide Military Command and Control System (WWMCCS) among many others. These systems employ a number of different message types, or formats.

2.2 Message Types

Examples of message types include RAINFORM (of various subtypes), Unit Reports (UNITREP), Movement Reports (MOVREP), and Joint Interoperability of Tactical Command and Control System (JINTACCS). The message formats have a number of elements in common. First, each type (or subtype) is defined to pass on data concerning a fairly specific event or of a fairly specific nature. For example a RAINFORM Green message provides tasking data to U.S. Naval forces prior to a mission, while those forces use a RAINFORM Purple message to report the results of that mission. For another example, a JINTACCS B704 is an Airbase Status Report while a JINTACCS C100 is a Imagery Interpretation Report.

Given the differing data requirements of these different message types, it would be surprising if they could all be accommodated using the same format. In fact, no such format has yet been found. However, the formats which have evolved over time have a number of similarities.

1) Messages are composed of two pieces, a header which describes the sender and the routing and other information about the message, and the body of the message holds the data content.

2) Both the header and body of the message are composed of line groupings which contain one or more lines in some specific order.

3) Each line is composed of a given sequence of fields (or components) whose appearance or order can vary only within narrow bounds.

4) Each field in a line contains a "molecule" of data which must be given in a predefined format. In fact some fields are composed of subfields (e.g. latitude is composed of degrees, minutes, cardinal point, and in some cases, checksum).

5) There are three types of fields: discrete, numeric, and text:

   a) Discrete fields are fields which must contain one of a (small) finite number of entries - for example a "month" field would have only twelve possible valid entries.

   b) Numeric fields are fields whose entries must evaluate to some numeric value. These fields may have a prescribed format as integer for fixed point. In either case, the number of significant digits (minimum and or maximum) may be specified as may the number of digits on either side of the decimal.

   c) Text fields are freeform fields whose contents may be any string of characters from some predefined character set - usually letters, digits, and some punctuation characters.

Message types differ then in which fields they use (and how each is defined), how those fields are used to define lines, and how those lines are grouped to form line groups. In addition, some message types are fixed format (the fourth field always starts at character 17) while others make use of delimiters to define where one field stops and the next starts. UNITREP is an example of a fixed format message type, while RAINFORM is an example of a "freer form" type.

The ideal message handling system would handle any and all message types with the same (or similar) sets of functions and user interface. If such a uniform system were to be built, the factors listed above define the flexibility requirements for accommodating various message type definitions.

2.3 Message Handling System Functional Requirements

Given that a message creation and editing system for some message type is to be developed, what requirements must it meet? The requirements important for the transmitter/receiver portion of a message handling system are certainly different than those which drive an editor's requirements. it appears that there are three

factors which exert the most influence on an editor's requirements and design: reliability, maintainability, and date validation.

Reliability is important for two reasons. First, communications systems are usually of a critical nature, and their failure can be catastropic. Therefore, message handling systems must work predictably to ensure that the system provides the capability expected during stress periods. Second, the output of one such system is always the input of another. Therefore, the failure of a message handling system to maintain communications or to pass accurate, properly formatted data impacts the ability of other systems to meet their requirements.

Maintainability is important due to the rapidly changing nature of the communications theater. New communications systems are constantly being fielded, and existing systems being upgraded. As this occurs, new message types are being added and existing types updated. For example, one existing message type has increased in size by over 20%, in terms of the number of different line types, over the past six years. As new message types are added and existing types modified, existing message handling systems must be modified to accept these new data.

Data validation is in some sense a component of reliability, but is so critical to the mission of an editor and message handling system that we break it out separately. Newer message handling systems (and some older ones such as FHLT) provide a high degree of input message checking; messages which contain invalid data are either put into an error queue, or discarded. In the former case the valid portions of the message are only available to the system through operator intervention, in the latter case they are not available at all.

2.4 Existing System Deficiencies

The current situtation can be summarized as follows: there are a variety of message formats, each of which is handled on several message handling systems, each of which has its own custom software for each different message type it deals with. This than means that there is in fact not a single RAINFORM message handler, but several, each with its own code, its own set of functions, and its own user interface. Thus, when the RAINFORM message specification is updated, those updates find their way into some systems and not others.

This leads to the following problems:

1) Configuration management is complicated by the various implementations or message handlers for the same message types.

2) Consistency and reliability suffer due to the fact that each message handler implements somewhat different versions of the message standard in questions.

3) Maintenance is difficult and costly since each system is coded in a unique fashion, many in different languages, almost all using different approaches.

Generally speaking, each time we build a new message handler - whether for a new or existing message type - we are gaining nothing from the fact that we have ever built such a thing before. Furthermore, the costs involved in "reinventing the wheel" stay with each system throughout its lifecycle. In the case of C3 systems, the lifecycles are long and therefore the excess cost high.

Significant savings can be realized if we attempt to reduce or eliminate the scope of the problems discussed above through reusing message type definition and message editing and handling technology. This can occur in several ways, ranging from complete reuse of existing code, through partial reuse of code, to reuse of designs and message definitions. In the sections which follow, we describe some initial attempts to explore approaches to reuse of message definitions, designs, and message editor code.

3. The Ideal System

3.1 Message Format Specifications as Ada Constructs With each message format, there exist in some form or another, a format specification. This specification provides detailed information about the message format from the level of a message as an entity on down to the field content level. This information provides guidelines required by applications programs for properly handling formatted messages. Section 2.2 above describes in some detail, the types of information provided by a format specification.

Ada lends itself very nicely to defining such specifications. A field is the lowest level defined by a format specification. As mentioned in Section 2.2, there are three basic types of fields: discrete, numeric, and text. In Ada, discrete fields may be defined as enumeration types. Numeric fields may be defined as either integer, fixed point or floating point type. Text fields may be defined as string. Compound fields may also exist. They are fields which consist of several field components, all of which must be one of the three basic field types. An example of a compound field is a latitude field. In Ada a latitude field may look like:

```
type LATITUDE_FIELD is
  record
    DEGREES        : DEGREES_FIELD;
    MINUTES        : MINUTES_FIELD;
    CARDINAL_POINT : CARDINAL_POINT_FIELD;
    CHECKSUM       : CHECKSUM_FIELD;
  end record;
```

Where the field component types: DEGREES_FIELD, MINUTES_FIELD, CARDINAL_POINT_FIELD, and CHECKSUM_FIELD have previously been defined as either discrete, numeric, or text.

In a formatted message, a line is composed of a given sequence of fields. Using Ada, a line can be represented as a record structure. Each component of the record structure would be a field. For example, a formatted line which reported the contact position of a ship may consist of three fields: contact identifier, latitude of contact, and longitude of contact. In Ada, the contact position line may look like:

```
type CONTACT_POSITION_LINE is
  record
     CONTACT_IDENTIFIER : CONTACT_IDENTIFIER_FIELD;
     CONTACT_LATITUDE   : LATITUDE_FIELD;
     CONTACT_LONGITUDE  : LONGITUDE_FIELD;
  end record;
```

Where the field types: CONTACT_IDENTIFIER_FIELD, LATITUDE_FIELD, and LONGITUDE_FIELD have previously been defined according to the rules for defining field types.

When lines are grouped together in a particular manner, they make up a formatted message. In Ada a formatted message may be represented as a record structure. Suppose a formatted message of a particular type was made up of the following five formatted lines: message identifier line, contact sighting line, contact position line, contact amplification line and a remarks line. The Ada definition of such a message type would be:

```
type FICTIOUS_MESSAGE_FORMAT is
  record
    MESSAGE_IDENTIFIER    : MESSAGE_IDENTIFIER_LINE;
    CONTACT_SIGHTING      : CONTACT_SIGHTING_LINE;
    CONTACT_POSITION      : CONTACT_POSITION_LINE;
    CONTACT_AMPLIFICATION : CONTACT_AMPLIFICATION_LINE;
    REMARKS               : REMARKS_LINE;
  end record;
```

Where the various line types have previously been defined according to the rules for defining formatted lines.

In the "ideal" system, a message format would be defined as an Ada construct similar to that described above. Such a means for defining a message format has many advantages. In particular, the message format specification becomes a compilable unit therefore providing a means of partial validation of the format specification syntactically and semantically. Additionally, the Ada definition of the message format may be used directly in applications Ada programs that require knowledge of the format.

There are a variety of methods for defining message format specifications in Ada, however the record structure described above appears to be the most natural representation of a message format for existing formats. Currently the United States Air Force (USAF) is working with the JINTACCS community to define their message formats as Ada record types.

3.2 A Generic Message Handling System

Though message formats will vary, the requirements for message handling systems, as described in Section 2, tend to remain fairly static. Generating a message handling system for each distinct message format is a costly and time consuming task. A solution, though a non-trivial one, would be to develop in Ada, a generic message handling system. The generic message handling system would essentially be a generic package with its functions and procedures not customized around any specific message format, but rather designed to work with any message format specification that the package is instantiated for. This would imply that the only significant requirement for creating a message handling system for a particular message format would be that the specification for the message format be defined as an Ada record and then the generic would have to be instantiated for the message format. All information about the message format required by the message handling system could than be extracted from the Ada record structure containing the message format. Ideally than, the generic definition would be as follows:

```
generic
--
  type MESSAGE_FORMAT_SPECIFICATION is private;
  -- where the actual parameter here would be a record type
  -- much like that defined in Section 3.1 above
--
package MESSAGE_HANDLING_SYSTEM is
--
```

A person farmiliar with Ada generics or C3 systems would immediately identify the "ideal" system as being highly improbable. However, it is conceivable that a close approximation could be reached. The close approximation would not be as clean cut as the "ideal" but it would have many of the same benefits.

4. Striving for the Ideal System

4.1 GMHF as an Approximation

A first attempt at developing a generic message handling system was completed in April 1985. The project, Generic Message Handling Facility (GMHF), was sponsored by the USAF and the Naval Ocean Systems Center (NOSC). GMHF is not a complete message handling system. It primarily consists of the Message Creation and Editing facility. The feeling being that sufficient amounts would be learned from doing an editor and there was no real requirement to build an entire message handling system for this effort. The purpose behind this effort was three fold. First, a feasability study was to be performed to determine just how close to the "ideal" system could you get using pure Ada features. Secondly, a prototype system was to be developed as a close approximation to the "ideal" system. And thirdly, a final analysis was to be

performed to determine just how cost effective the generic system was to use.

The first question was answered early on. It was apparent that there was no clean cut method for building a generic package around a generic formal parameter which was a message format specification as an Ada record like that defined in Section 3.1. Record types can indeed be passed as generic parameters, however within the generic, little can be done with the record structure since it is private.

In striving for an approximation to the "ideal" system, it became clear that some sacrifices would have to be made. Since a main concern of this effort was to determine cost effectiveness of generics in real world applications, the message format specification as an Ada record was substituted for something less sophisticated. The format specification record was replaced by several generic formal objects and types, and a database of message specification information. Additionally several procedures had to be passed as parameters to the generic. Provided below is the essence of the generic definition for GMHF. Some minor details have been left out for simplification purposes.

```
generic
--
  MAXIMUM_CHARACTERS_PER_LINE : POSITIVE;
  -- constant value telling the generic how many characters
  -- maximum a formatted line may have for the instantiated
  -- message type.
  --
  MAXIMUM_FIELDS_PER_LINE : POSITIVE;
  -- constant value telling generic how many fields maximum a
  -- a formatted line may have for the instantiated message type.
  --
  MESSAGE_FORMAT_FILE_NAME : STRING;
  -- constant value providing the name of the file which contains
  -- the message format specification
  --
  type LIST_OF_FIELD_NAMES is (<>);
  -- an enumerated list of all fields for the message type
  --
  type LIST_OF_LINE_NAMES is (<>);
  -- an enumerated list of all lines for the message type.
  -- the line names are keys into the message format
  -- specification file.
  --
  with procedure GET_FIELD( FIELD_NAME  : in  LIST_OF_FIELD_NAMES;
                            FIELD_VALUE : out STRING ) is   ;
  -- this procedure provides all instantiations of I/O packages
  -- for the field data types of a message type. In addition,
  -- the routine is organized as a large case statement which
  -- calls the appropriate input routine for a given field type
  -- upon request. This has proved to be a long and tedious
```

```
   -- routine to generate.
   --
   with procedure FORMAT_LINE_OF_TEXT(LINE_OF_TEXT: in out STRING)
                                        is NULL_PROCEDURE;
   -- this procedure handles the formatting of a line of text so as
   -- its physical appearence meets the requirements of the
   -- specification.  For example, JINTACCS requires a '/' as a
   -- field delimitter between fields. When a field is left blank
   -- it appears as a '/-/'in the text of the message. This
   -- procedure would be responsible for identifing a field as
   -- being blank and subsequentlly placing a hyphen in the text.
--
package MESSAGE_HANDLING_SYSTEM is
--
```

The new types and objects as formal parameters and the format specification database contain much of the same information as the record construct would have, but with great redundancies and in a less clean, less natural fashion. The end result of all this was a generic message handler which was a successful system but not an optimal one.

With the successful development of a generic message handler, the question of cost effectiveness still remained. To resolve this question, the generic was instantiated for two message types, RAINFORM and UNITREP. The RAINFORM instantiation was completed by one of the developers of the generic software. The UNITREP instantiation was completed by an individual only mildly familiar with the software but very farmiliar with Ada, the idea being that the average instantiator of the generic would know little or nothing about the software itself. The results were very promising. RAINFORM required a fairly significant amount of time for instantiation, about 300 man hours. The majority of this time was spent debugging problems in the generic which were encountered for the first time. The instantiation of the UNITREP message handler took approximately 60 man hours. The time for producing the UNITREP instance was significantly less then the time that would have been required to develop a non generic message handling system unique to the UNITREP message format.

In short, development of GMHF and instantion for RAINFORM and UNITREP message formats yielded one set of positive results. Use of generics in real world applications should prove to be a very cost effective means of software development. At the conclusion of the GMHF effort, the question was raised, " Are there alternative means for developing message handling systems which are better than those imposed by GMHF? "

4.2 Problems with GMHF

To determine better means for developing message handling systems, an attempt was made to identify problems and deficiencies with GMHF. One deficiency was immediately apparent. GMHF required

that the use of message format specifications as records be sacrificed, so that we could develop the system as a generic unit. In place of the record structure, an implementer of the generic was forced to define data types to pass as formal parameters which would normally not have been required. In addition, a small database of message format specifications had to be created by the implementer for use by the generic. These undesirable work-arounds preferably should be avoided in future systems.

A requirement of message handling systems is that they support data input and output (I/O) operations, data validation, etc. The DBMS and Message Creation and Editing facility discussed in Section 2.2 above, clearly have this requirement. I/O operations in this case do not refer to the low level I/O required by the transmitter and receiver, but rather to the I/O routines obtainable by instantiating packages such as TEXT_IO, INTEGER_IO, ENUMERATION_IO, DIRECT_IO, etc.. GMHF supports the I/O requirements, but with one hook. All I/O functions and procedures which are to operate on types defined outside of the generic must be themselves defined outside of the generic and passed into the generic as parameters. This seems like an obvious requirement and it is. Obvious as it may be, it is a tedious, therefore undesirable task instantiating I/O packages for the types and subtypes which comprise the many fields of a message format oftentimes numbering in the hundreds.

To summarize, if the amount of work required by the implementer of the message handling system could be reduced to a minimum, such a system would become a much more powerful, useful tool. Therefore we must solve two problems. First, a way to utilize the record definition of a message format specification must be developed. Secondly, the requirement for the implementer to provide instantiations of all I/O packages for the different field types and subtypes must be eliminated, vastly improving the usability of the system.

Through careful investigation it became clear, there is no clean cut solution. Either you part with the message format specifications as records, or you must part with the idea of a generic message handling system. And in either case, the I/O packages for each of the field types would have to be created or instantiated by the implementer of the system.

4.3 Introducing a Preprocessor to the Problem

Following the conclusive results of GMHF, a new concept was introduced. A preprocessor could be developed which would accept as input the message format specification as a record type, and output as Ada code, a compilable package specification containing all types, instantiations of I/O packages, etc., required to instantiate the generic message handling system. Essentially, this allows us to obtain the desired goal. An implementer is only

required to generate a message format specification as an Ada record and then instantiate the generic. Of course there are some rules to follow when defining the message format specification so as to stay within the bounds of the preprocessor. The development of such a system is currently in progress with an expected completion time frame of September 1986. Portions of the system are being developed under contract to the USAF and NOSC, while the basis of the preprocessor has already been developed by a third party as an internal research and development project.

4.4 Implementation of Such A System

Implementation of such a system can be described as a series of three main steps.

4.4.1 The Message Format Specification

The implementer is first required to generate a package specification containing the record type definition for the message format as demonstrated in Section 3.1 above. Having completed this, the package specification should be compiled to validate it syntactically and semantically.

4.4.2 The Preprocessor

Having successfully compiled the message format package specification, the preprocessor should be activated. The preprocessor will read the message format package specification as an input file and generate an output file which is also a package specification. The output file will contain all types, I/O packages, etc. derivable from the input package specification which are required for instantiation of the generic message handling system.

4.4.3 The Output Package Specification

When the preprocessor is complete, the output package specification should be compiled. The implementers applications program may then "with" the compiled output package specification and in turn, instantiate the generic message handling system. There will be additional generic parameters which the implementer will be required to provide for the instantiation which will not be included as part of the package specification output by the preprocessor.

5. Conclusion

5.1 Status

The preprocessor solution for the "ideal" system is midway through the design phase. Currently a prototype of the message handling system is being developed to determine specific

requirements for the output of the preprocessor. A preliminary version of the preprocessor has been developed, however not with this particular application in mind. An expected completion date for the entire system is September, 1986. The system will be made available in the public domain via ARPANET upon completion.

5.2 Summary

Development of a system such as that proposed by the preprocessor method could in a sense, revolutionize the use of message handling systems in the C3 world. Currently, so much money is poured into the development and maintenance of systems in support of C3. To begin development of code for such systems in Ada is a very large step to improve the reliability, maintainability, and reuseability of such systems. Additionally, the generic message handling system as described in this paper would be a welcome asset to the development of C3 systems. The generic message handling system is portable between hardware, and implementable for most every message format in use today by the DoD.