Utilizing the SNA Alert in the management of multivendor networks

by R. E. Moore

Managing multivendor networks is one of the largest challenges facing vendors and customers in data processing and telecommunications. This paper focuses on one aspect of managing multivendor network environments: problem notification, isolation, and resolution, via Systems Network Architecture's Alert. It describes an extension to the SNA Alert function, termed the generic Alert, that makes it possible for various vendors' products, as well as customer-written applications, to send Alerts of the same type to a single Alert receiver. It also describes IBM's implementation of the Alert receiver for the System/370, the NetView™ program product. Among the facilities that the generic Alert architecture provides to an Alert sender are the following: (1) code points that index short descriptions of Alert conditions, probable causes of these conditions, and recommended operator actions; and (2) vehicles to carry product-unique text. This text can be used for further characterizing an Alert condition or specifying a particular operator action.

In today's environment, multivendor networks are the rule rather than the exception. Such networks are made possible by the existence of agreed-upon rules for communication among products from different manufacturers. These rules may take the form either of international standards, such as X.25 or the emerging ISDN, or of an IBM open architecture standard, such as SNA's LU 6.2. A customer can be confident that two products from different manufacturers which conform to one of these standards will be able to communicate.

Until recently, however, there has been no provision for *management* of a multivendor network. Products from different manufacturers either have provided different, incompatible network management capabilities, or, in some cases, have provided no network management capabilities at all. With the introduction of the new generic Alert structure into SNA's Management Services Architecture, and into IBM's NetView™ and NetView/PC™ program products, a foundation for solving this problem has been provided in the area of problem management. The published generic Alert structure, which IBM has made available to vendors and to customers who write their own application programs, provides a standard mechanism by which every product in an SNA network will be able to provide useful problem notifications to a network operator. Furthermore, with the generic Alert support in the NetView/PC program, IBM has provided an avenue through which problem notifications for non-SNA resources in a network, or even for an entire non-SNA network, can be forwarded to this same network operator.[1]

Once the generic Alert architecture has been implemented by products in a multivendor network, the network operator will be provided with consistent problem management for those products.

An Alert is an unsolicited record sent to a network operator indicating that a problem exists.

Opening the Alert architecture

Alerts have been a part of SNA's Management Services Architecture from its inception.[2] An *Alert* is an unsolicited record sent to a network operator at an Alert receiver by a network component that has detected a problem. In addition to notifying the network operator that a problem exists, the Alert provides the following information:

- The identity of the Alert sender, both as a network entity and as a product
- The identity of the network resource most closely related to the problem
- An indication of the problem's severity, e.g., whether it is a permanent or a temporary failure
- A description of the problem
- A list of probable causes of the problem, ranked according to their probability of occurrence
- A list of recommended actions for the operator to take in response to the problem
- In some cases, additional protocol-unique error data, e.g., data pertaining to a token-ring local-area network (LAN)
- In some cases, additional product-unique error data, e.g., a machine check code
- In some cases, an indication of the time at which the problem was detected

In the past, the majority of this information was stored in a database controlled by the NetView program,[3] in the form of predefined display panels. These panels were grouped within the database according to sending product. The Alert record itself carried two pieces of information that the NetView program used to retrieve the right set of panels from its database of all Alert panels: the identity of the Alert sender, and an indication of which set of its panels the sender was requesting. For example, an Alert from an IBM 3274 might call for the NetView program to display the set of panels known to the NetView program as 3274's set #5; see Figure 1.

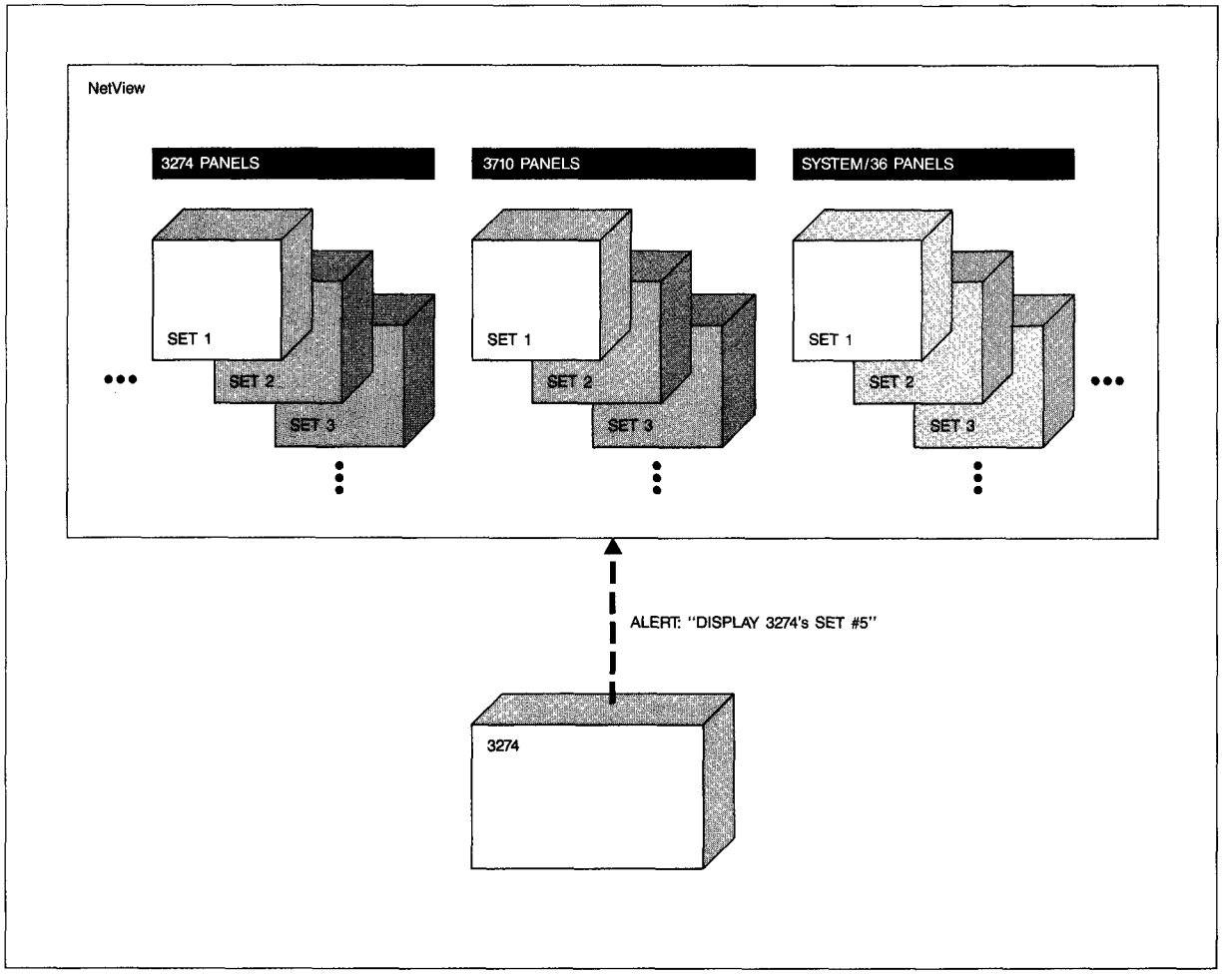
The scheme with stored panels is totally dependent on predefining the contents of the panels. The panels must be incorporated into the NetView program. An Alert sender that did not have panels stored in the NetView program had no means of obtaining a full Alert display.[4] Since panels were defined by NetView product developers in conjunction with other IBM product developers wanting Alert support for their product, manufacturers other than IBM were unable to utilize the Alert function fully.

To circumvent this problem temporarily until the generic Alert architecture could be defined and implemented, in 1986 IBM introduced an enhancement in Release 1 of the NetView program. Sixteen sets of null panels which could be filled in by a customer were included in the NetView program's database of Alert panels. Rather than being indexed by product type, such as 3274, these sets of panels were indexed by the sixteen surrogate values USER0, USER1, ···, USERF. A vendor's product, or a customer-written application, would then be able to create an Alert requesting the NetView program to display set #5 from the group of panels indexed by the surrogate value USER2, as shown in Figure 2.

This was only half the job. The null panels in the NetView program for USER2's Alert #5 still had to be filled in by the customer, so that the correct information would be displayed when the Alert was received. Thus, a vendor utilizing an Alert of this type had to include documentation with the Alert-sending product, instructing the customer how to fill in the null panels. The customer was required to do the customization of the NetView program.

While this enhancement based on the sixteen surrogate values was not a perfect solution, it did make available to the manufacturers for the first time the function inherent in the NetView program's stored Alert panels.

Figure 1 Indexing stored Alert panels



The new generic Alert

With Release 2 of the NetView program in 1987, the shortcomings of the stored-panel approach to Alert presentation were overcome. Figure 3 illustrates the fundamental difference between the earlier stored-panel Alerts and the new generic Alerts. While the sending product is still identified in the Alert for the benefit of the network operator, its identity plays no role in the creation of the basic Alert displays. Rather than being asked to retrieve and display a particular set of predefined Alert panels, the NetView program is told how to *build* a set of panels, using text elements stored within the NetView program itself. An Alert sender is free to request any combination of these text elements. The NetView program simply combines the elements as requested, without regard to the sender's identity. Since the sender's request is transported in the Alert record itself, no prearrangement is necessary; if a sender sends a correctly formatted generic Alert record, the desired information will be presented to the network operator by the NetView program.

In this sense, the Alert architecture resembles that for X.25 or LU 6.2. When both sender and receiver implement the architecture correctly, successful communication between them is assured.

Publication of the architecture. So that manufacturers of SNA and non-SNA products may participate

Figure 2 Indexing stored panels using a surrogate ID

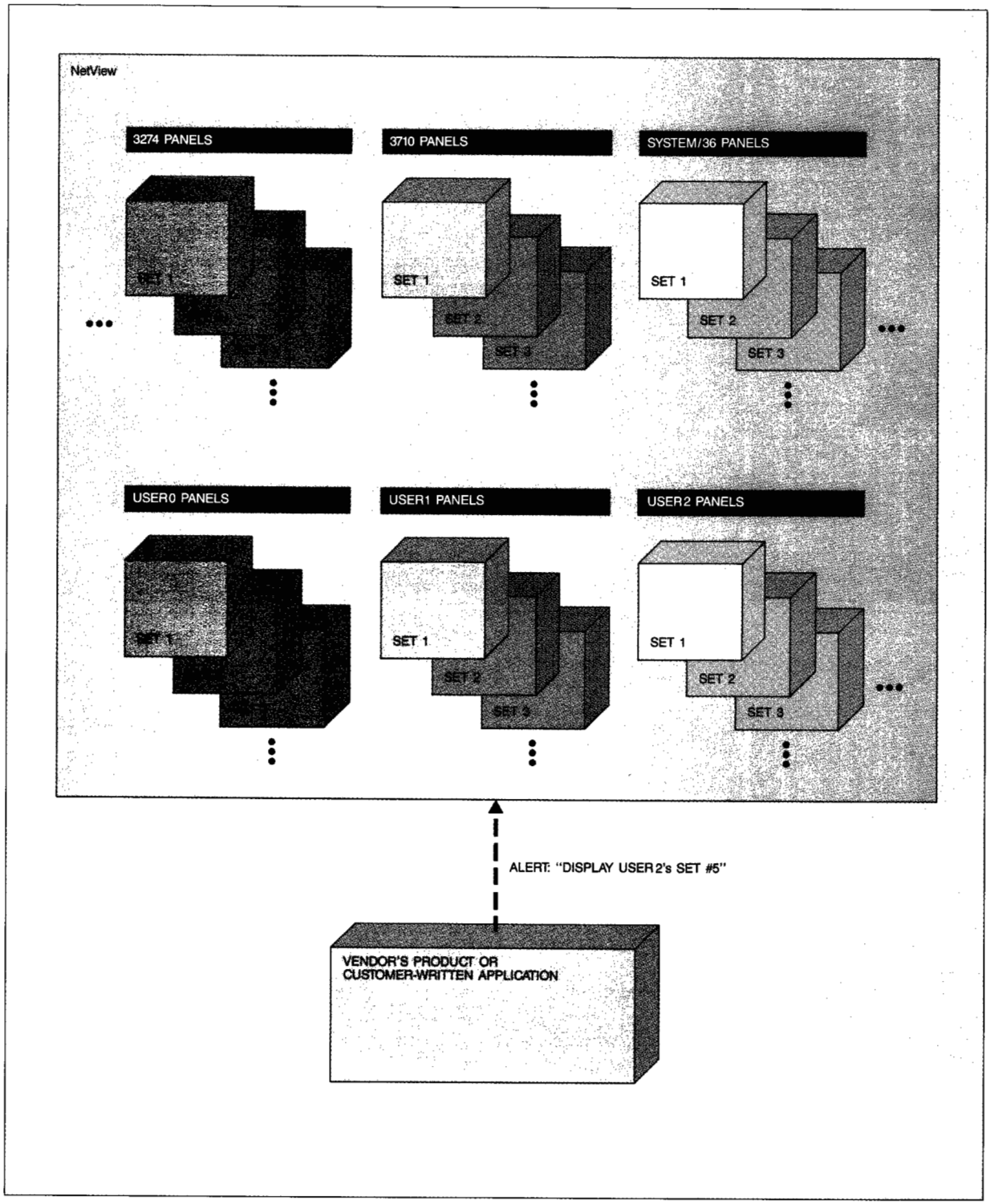
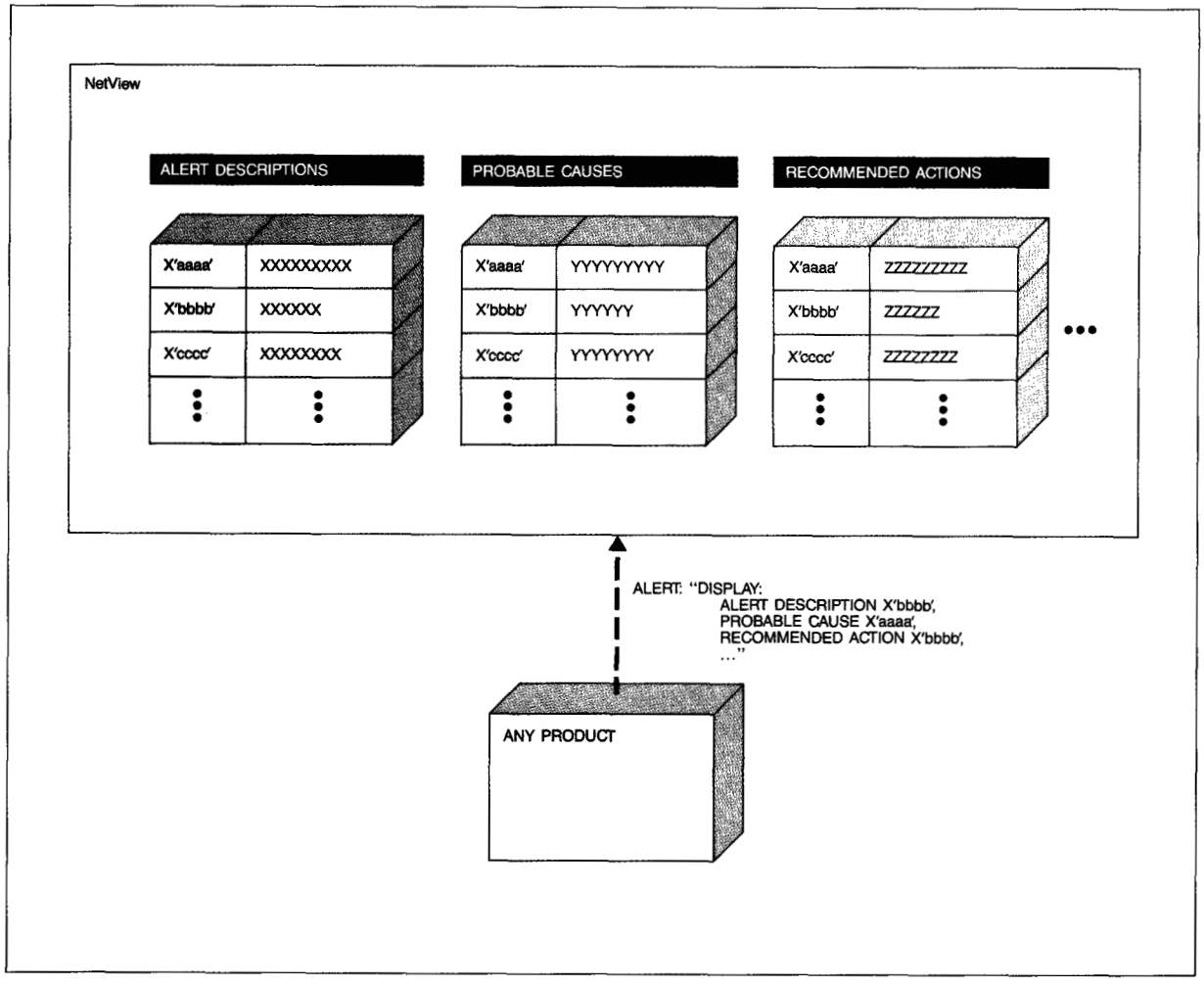


Figure 3 Generic Alert

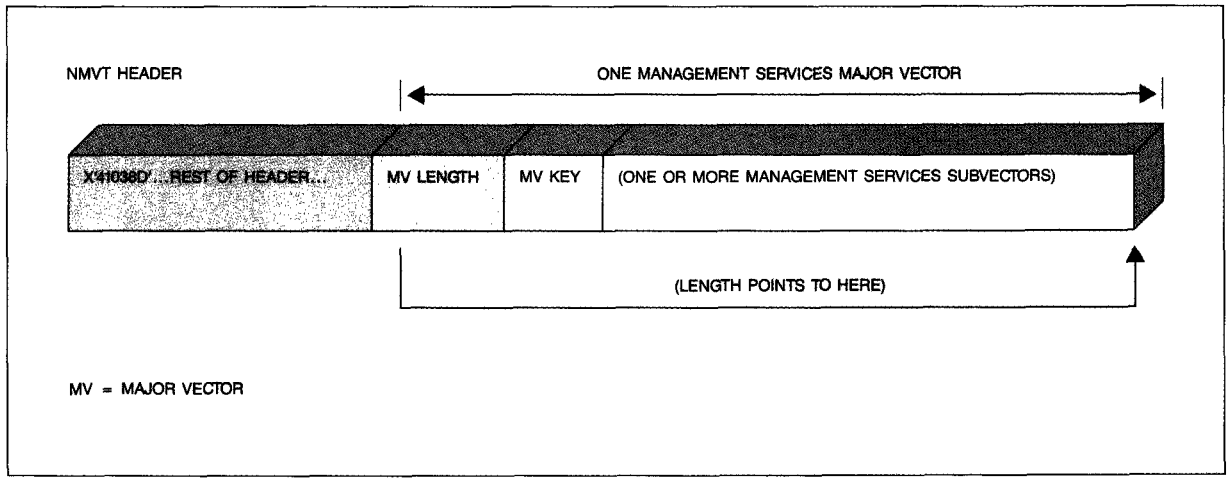


fully in network management, IBM has published the complete architecture for generic Alerts. The actual Alert formats appear in Reference 5, while a discussion of how they are used and the displays they are designed to produce appears in Reference 6. In addition to these publications, IBM has provided classes to assist interested parties in implementing the Alert architecture.

There are three audiences to whom these publications are primarily addressed: vendors who build and market SNA products, those who build and market non-SNA products, and customers with SNA networks who write their own application programs. With the information IBM provides, all of these groups will be able to create SNA components that build and send Alerts via techniques similar to those used by IBM products.[7] A customer will be able to manage problems for an entire network, including IBM products, SNA and non-SNA products from vendors other than IBM, and applications that the customer has written, in a uniform way and with a single product: the NetView program.

Uniformity of Alert presentations. There is another benefit provided by the switch from stored-panel to generic Alerts: Since the generic Alert displays are created dynamically from a common set of stored text elements, more uniform presentations are created. Because stored panels were previously defined

Figure 4 The Network Management Vector Transport (NMVT)



individually for each product that sent an Alert to the NetView program, and used only by that product, there was no reason that the panels for one Alert sender had to look at all like those for another Alert sender. In fact, the NetView program did enforce a certain degree of uniformity in overall presentation style and format on all of its stored panels, but differences in detail and terminology nevertheless crept in. For example, one set of panels might use the term DEVICE CABLE, a second COAXIAL CABLE, and a third TERMINAL CABLE, in all cases referring to exactly the same thing. The cumulative effect of many such small differences was to leave the operator at the NetView program always a little unsure as to whether two Alerts were really reporting the same type of failure.

Since the generic Alert displays are built dynamically from the same stored text elements, they do not contain variations of this type. The text element DEVICE CABLE is defined in the published Alert architecture, and can be indexed by all Alert senders wishing to report a failure on such a cable. Thus the operator always sees the same text. In this way, uniformity of presentation is intrinsic to the generic Alert architecture.

The architecture

In moving from stored-panel to generic Alerts, the potential existed for losing the one outstanding benefit of stored-panel Alerts: their flexibility. A set of panels defined for a single Alert sender could, in principle, say anything whatsoever, so long as it fit on the NetView program's screens.[8] In order to preserve the beneficial aspects of this flexibility, while eliminating unhelpful diversity, it was necessary to create an architecture of sufficient power that Alert senders could cause the NetView program to build displays just as informative as those that would have been defined for a set of stored panels. We now highlight some of the features of the generic Alert architecture that make this possible.

The basic structure of management services data units

Figure 4 shows the format of the SNA Request Unit in which an Alert is transported through the network. The value X'41038D' in the header distinguishes this Request Unit, the Network Management Vector Transport (NMVT), from other SNA Request Units. The NMVT carries other types of management services data besides Alerts; an Alert is identified by the major vector key X'0000'.

Following the fixed-length header is the remainder of the NMVT, a single management services major vector. Since the end of the major vector, which is also the end of the NMVT, is indicated by the major vector length, the length of different NMVTs can vary. For example, one Alert for which a substantial amount of data is available may be 350 bytes in length, while another may contain only 70 or 80 bytes.[9]

Figure 5 Structure of a management services major vector

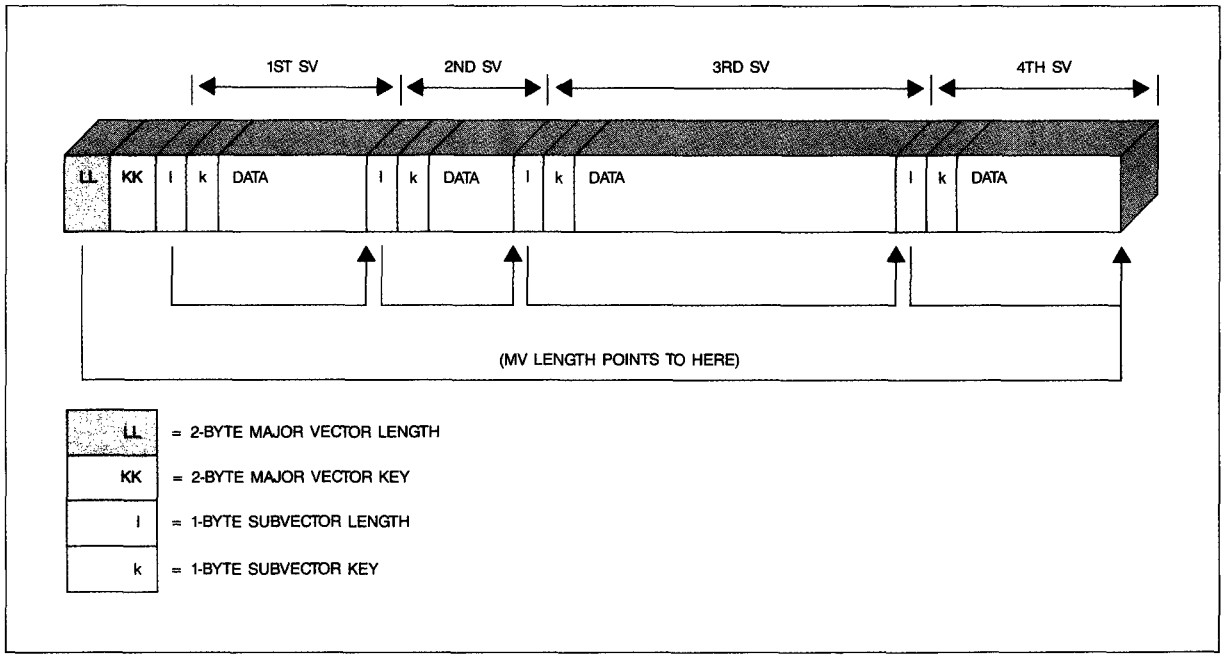


Figure 5 shows the structure of a management services major vector. A major vector is simply an envelope for one or more management services subvectors. Each subvector has a length field indicating where it ends. Thus, just like the NMVT, the major vector and subvectors can be of variable length.

Currently there are 21 subvectors defined in the architecture for inclusion in the Alert major vector. No single Alert contains all of these subvectors; a typical Alert, in fact, contains only about eight. The major vector structure provides for the definition of additional subvectors as they are required. To support additional types of information in an Alert, a receiver such as the NetView program need only add support for an additional subvector. Its general support for parsing the Alert major vector, as well as its support for previously defined subvectors, is unaffected.

An older version of the NetView program will not have any problem with a new subvector, although it will not provide support for it. Since unrecognized subvectors are always ignored by the NetView program, the new subvector will not even be detected by the older version.

In many cases there is one more level of decomposition; some, but not all, management services subvectors are decomposed into management services sub*fields*. As shown in Figure 6, a subfield has exactly the same structure as a subvector; i.e., it has a length, a key, and some data. The only difference between the two is that a subvector is contained immediately within a major vector, while a subfield is contained immediately within a subvector.

References 5 and 6 provide further details on the NMVT and its major vector/subvector/subfield encoding scheme.

Generic Alert code points

The primary Alert displays are created by the NetView program by means of *index code points*. These 1- or 2-byte hexadecimal values index relatively short strings of text stored in tables in the NetView program; when it receives an Alert, the NetView program does a series of table lookups based on the code points contained in the Alert. The text strings retrieved as a result of these lookups are combined to form the display for the Alert.

Figure 6 Decomposition of a subvector into subfields

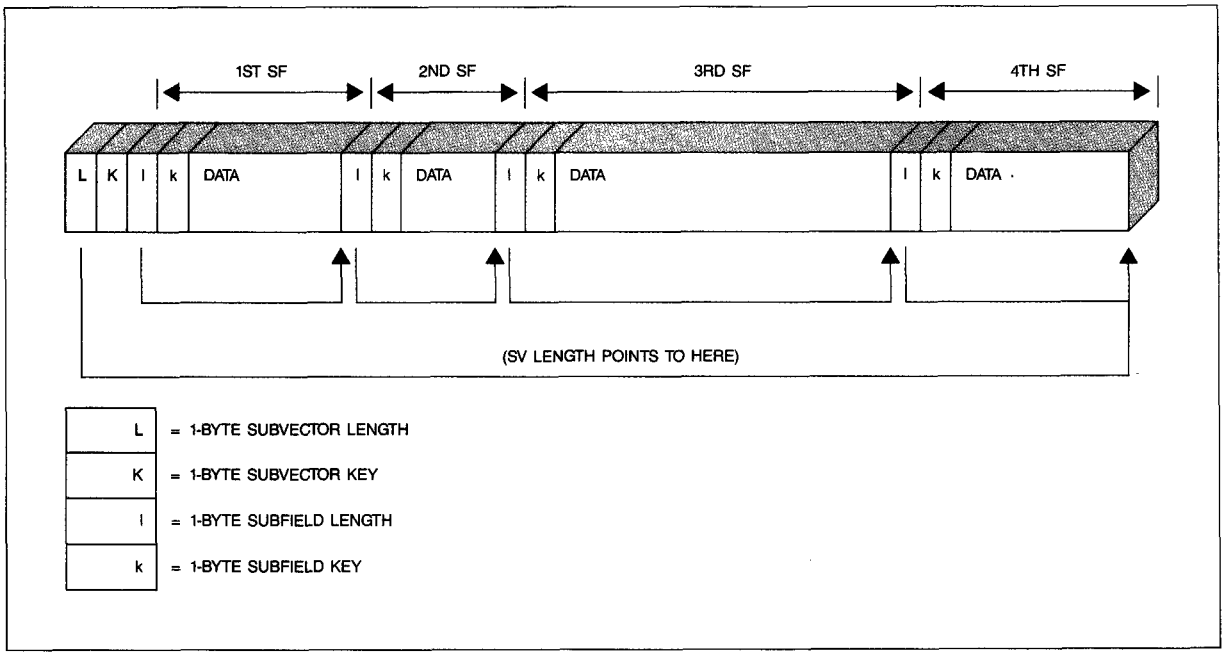


Figure 7 Default and replacement text for Recommended Action code points

CODE POINT	DEFAULT TEXT	REPLACEMENT TEXT
X'0400'	RUN APPROPRIATE TEST	RUN APPROPRIATE TEST
X'0401'	"	RUN CONSOLE TEST
X'0402'	"	RUN CONSOLE LINK TEST
X'0403'	"	RUN MODEM TESTS
X'0500'	RUN APPROPRIATE TRACE	RUN APPROPRIATE TRACE
X'0600'	OBTAIN DUMP	OBTAIN DUMP
X'0601'	"	TRANSFER AND PRINT MOSS DUMP
X'0602'	"	TRANS AND PRINT CONT PROG DUMP
X'0603'	"	TRANS AND PRINT LINE ADAP DUMP
X'0610'	"	DUMP CONTROL PROGRAM
X'0611'	"	DUMP CHANNEL ADAPTER MICROCODE
X'0612'	"	DUMP LINE ADAPTER MICROCODE
X'0613'	"	DUMP MOSS MICROCODE

Figure 7 shows a portion of one of the NetView product's tables. The text strings indicate various recommended actions that can be presented to an operator. When the NetView product receives recommended action code point X'0402', it displays the text RUN CONSOLE LINK TEST.

If the NetView product should receive a code point not contained in its tables, it automatically displays the *default* text indexed by the first byte of the code point; in the case of recommended action code point X'0404', for example, the NetView product would display RUN APPROPRIATE TEST. This technique allows for the introduction of new code points into the architecture. If a newly defined code point is included in an Alert by an Alert sender before the NetView product's tables have been updated to include it, the default text that the NetView product retrieves still provides a meaningful display. Later,

when the NetView product's tables are updated to include the new code point, the more informative replacement text is automatically retrieved.

In addition to the code points and text strings contained in the product itself, the NetView product allows the customer to enter new code points and text strings. A customer who writes an application that sends its own Alerts, for example, might wish to report conditions not covered by any of the text strings initially included in the NetView product. To do this, the customer would (1) select an unused code point, (2) write the application in such a way that it included the code point in the Alert that it sent, and (3) enter the new code point and the desired text into the NetView product. Once all of this had been done, the NetView product would process this Alert in exactly the same way that it processes any other Alert.

Figure 8 indicates how the index code points are carried in the Alert major vector. It shows one of the subvectors defined for this major vector, the Failure Causes (X'96') subvector. In this case the Failure Causes subvector contains two subfields: the Failure Causes (X'01') and Recommended Actions (X'81') subfields. The code points themselves appear within these subfields; in this case there are two Failure Causes code points (X'3451' and X'6210') and one Recommended Action code point (X'0301').

Figure 9 shows the sets of index code points defined by the Alert architecture. The NetView product has a separate table for each set of code points. The nature of a code point, and thus the table into which the NetView program must index, is determined by the structure that contains the code point within the Alert major vector. Thus, a Recommended Action code point appears within the Recommended Action subfield within one of four subvectors: the User Causes, Install Causes, Failure Causes, or Cause Undetermined subvector. A Probable Causes code point, on the other hand, appears within the Probable Causes subvector.

Figure 10 shows one of the NetView program's Alert displays, the Recommended Action screen. The following elements of this display are stored text strings, indexed by the indicated code points:

Figure 8 Example showing how index code points are carried in an Alert major vector

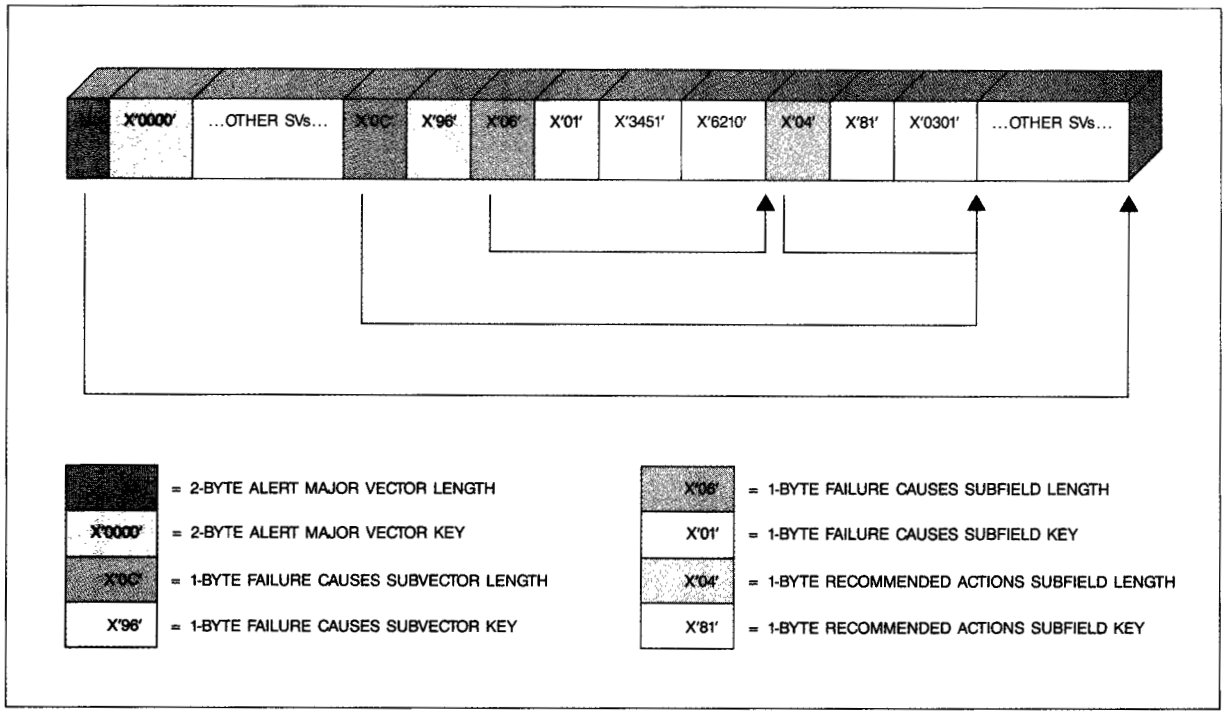


Figure 9 Sets of index code points defined by the Alert architecture

CODE POINT	SIZE	FLOWS IN
ALERT DESCRIPTION	2-BYTE	GENERIC ALERT DATA (X'92') SUBVECTOR
PROBABLE CAUSE	2-BYTE	PROBABLE CAUSES (X'93') SUBVECTOR
USER CAUSE	2-BYTE	CAUSES (X'01') SUBFIELD IN THE USER CAUSES (X'94') SUBVECTOR
INSTALL CAUSE	2-BYTE	CAUSES (X'01') SUBFIELD IN THE INSTALL CAUSES (X'95') SUBVECTOR
FAILURE CAUSE	2-BYTE	CAUSES (X'01') SUBFIELD IN THE FAILURE CAUSES (X'96') SUBVECTOR
RECOMMENDED ACTION	2-BYTE	RECOMMENDED ACTION (X'81') SUBFIELD IN THE USER CAUSES (X'94'), INSTALL CAUSES (X'95'), FAILURE CAUSES (X'96'), AND CAUSE UNDETERMINED (X'97') SUBVECTORS
DATA ID	1-BYTE	DETAILED DATA (X'82') SUBFIELD IN THE X'94'-X'97' AND DETAILED DATA (X'98') SUBVECTORS
RESOURCE TYPE	1-BYTE	HIERARCHY NAME LIST (X'10') SUBFIELD IN THE HIERARCHY/RESOURCE LIST (X'05') SUBVECTOR

COMC, LINE, CTRL, and TERM: Resource-type code points
DEVICE POWER OFF: User cause code point
TERMINAL MULTIPLEXER POWER OFF: User cause code point
DEVICE CABLE NOT CONNECTED: User cause code point
CHECK POWER: Recommended action code point
CHECK CABLES AND THEIR CONNECTIONS: Recommended action code point
NONE: Implicit—the NetView product displays this because the Alert contains no Install Cause subvector
DISPLAY: Failure cause code point
DEVICE CABLE: Failure cause code point
CONTACT APPROPRIATE SERVICE REPRESENTATIVE: Recommended action code point
REPORT THE FOLLOWING: Recommended action code point
ERROR CODE: Data ID code point

The remaining elements of the display, i.e., the resource names PU123, etc., the sending product identification TTTT, and the variable data 21F, are not stored text strings retrieved via code points. These elements will be discussed later.

One useful property of index code points is that they inherently provide support for different national languages. Since a code point itself is in no language at all, nothing special needs to be done when an Alert crosses a national-language boundary. The tables in the receiving NetView program contain text appropriate for the country where it is located, so Alerts from anywhere in the world will automatically produce displays in the correct language: the language spoken at the *receiving* site.[10]

While the 2-byte size of most of the index code points defined in the Alert architecture provides for up to 65536[11] text strings, the question may still arise as to what happens when these numbers are exhausted. The subvector/subfield encoding method used in the NMVT provides a very straightforward answer. New ranges of code points can be introduced at any time simply by adding new subvectors or subfields to the architecture.

Detail qualifiers. While the technique of building Alert displays using stored text strings indexed by code points is quite flexible, it is not by itself sufficient. A second mechanism is required for displaying *variable* data to an operator. If, for example, an Alert reports a failed attempt to set up a switched telephone connection, two key pieces of information are the calling telephone number and the telephone number that was called. Obviously the NetView program cannot store all possible telephone numbers as text strings to be indexed by code points. Instead, the telephone numbers themselves must be carried within the Alert major vector.

Figure 10 Example of a NetView display: the Recommended Action screen

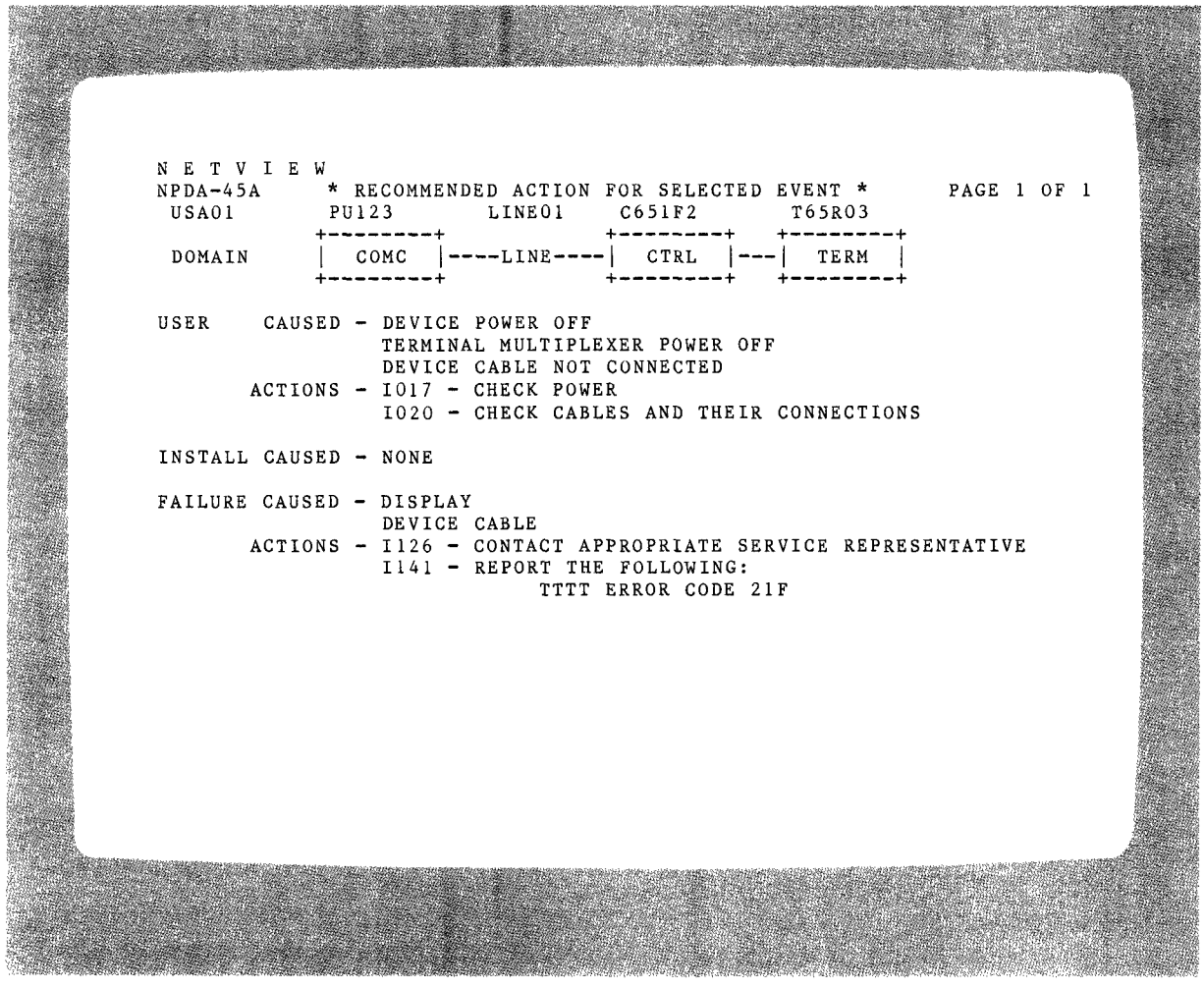


Figure 11 shows the structure defined by the Alert architecture for the transport of variable data such as telephone numbers. [Instances of variable data such as this are referred to as *detail qualifiers* in the architecture; the structure that transports a detail qualifier is the Detailed Data (X'82') subfield.] There are four elements present in every Detailed Data subfield:

- *The Product ID Code:* This code serves as an index to product identification data that are carried elsewhere in the Alert major vector. With this code, a product sending an Alert can specify that it, or another product, should be explicitly identified in conjunction with a particular piece of variable data.
- *The Data ID:* This 1-byte index code point indexes a text string identifying the *type* of variable data contained in the subfield, e.g., CALLING TELEPHONE NUMBER.
- *The Data Encoding:* This code instructs NetView how to display the variable data. For example, the same variable data X'F0F0' will be displayed as "F0F0" if the data encoding code is X'00' (hexadecimal), but as "00" if the data encoding is X'11' (EBCDIC).

Figure 11 The Detailed Data (X'82') subfield

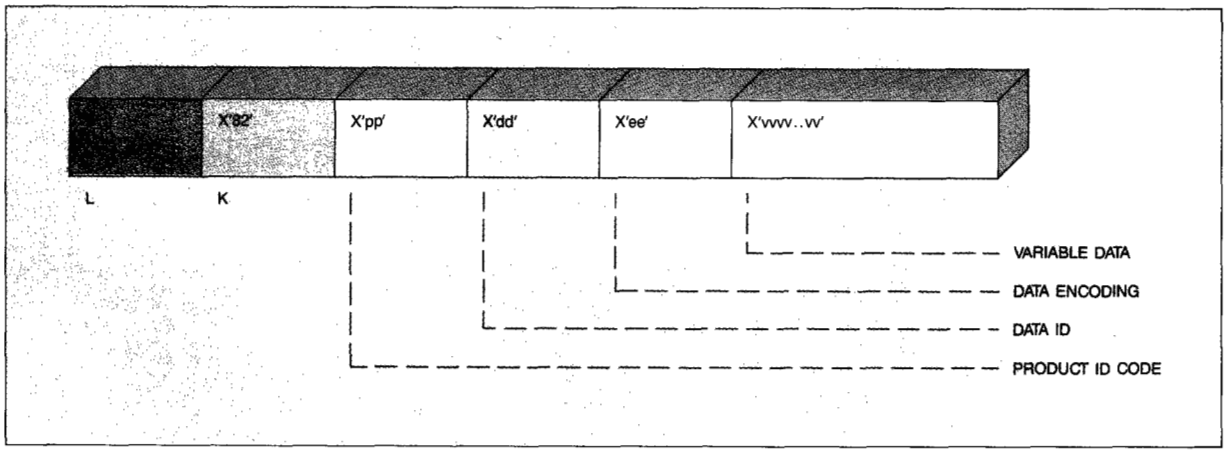
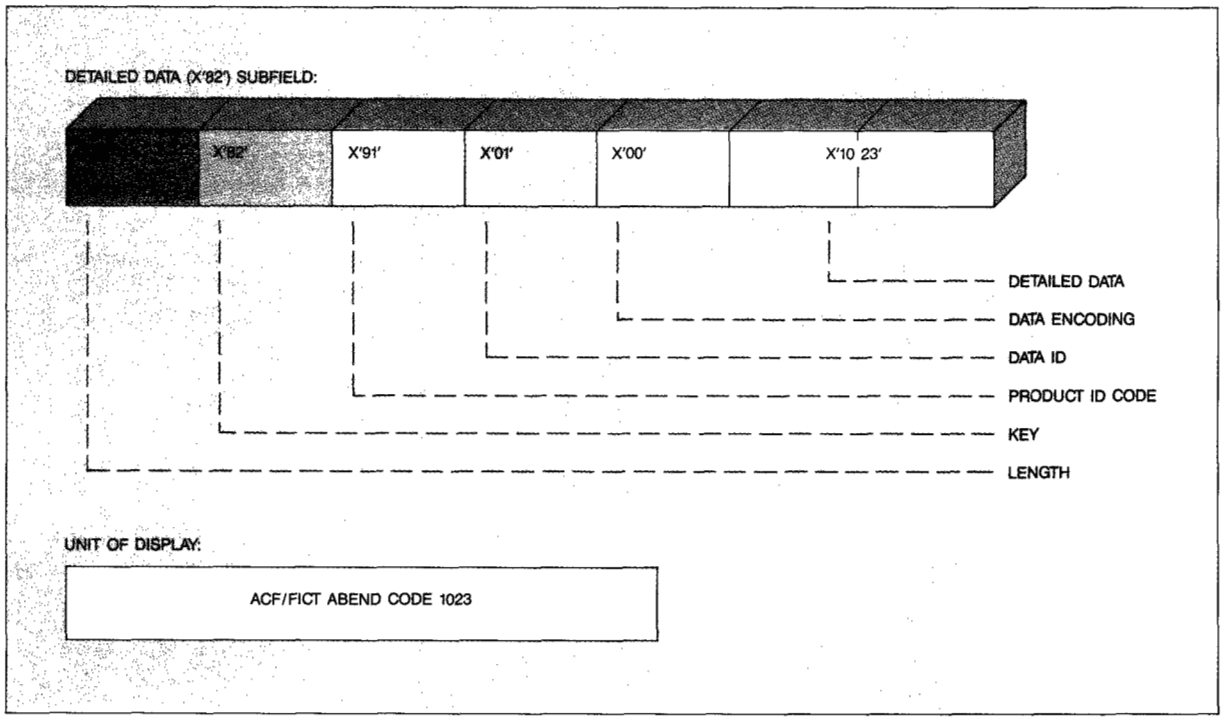


Figure 12 Example of an X'82' subfield and the resulting display



- *The Variable Data:* Up to 44 bytes of variable data. In order to be compatible with the national-language support provided by the Alert code points, the variable data are required to be truly language-independent: telephone numbers, machine error codes, port addresses, etc.

Figure 13 Examples of code points with associated detail qualifiers

CODE POINT	TEXT
X'F0A3'	FAILURE OCCURRED ON (sf82 qualifier)
X'20A3'	NO RESPONSE FROM THE X.21 NETWORK—(sf82 qualifier) EXPIRED
X'12C0'	RETRY AFTER (sf82 qualifier) (sf82 qualifier)
X'32D1'	LOCAL DCE COMMUNICATIONS INTERFACE (sf82 qualifier) (sf82 qualifier) (sf82 qualifier)

Figure 12 illustrates how the Detailed Data subfield might be used to create a unit of display. It assumes that the fictitious IBM software product ACF/FICT has sent an Alert in which it reports one of its own abend codes. The text "ACF/FICT" is carried in a Product Set ID subvector identifying the Alert sender; the value X'91' instructs the NetView program to retrieve this text.[12] The text "ABEND CODE" is retrieved from a table of Data IDs by the NetView program, via the code point X'01'. The text "1023" is displayed by the NetView program because it was instructed to treat the variable data as hexadecimal rather than EBCDIC.

Each instance of the Detailed Data subfield creates a single unit of display at the NetView program, of the type illustrated in Figure 12. The location of this unit of display in the overall set of displays created by the NetView program for an Alert is determined by the location of the Detailed Data subfield in the Alert record. The most interesting case is that in which the unit of display appears on the NetView program's Recommended Action screen, as "TTTT ERROR CODE 21F" did in Figure 10. When a detail qualifier appears on the Recommended Action screen, it is always associated with a particular cause or recommended action. In Figure 10, for example, TTTT ERROR CODE 21F is associated with the recommended action REPORT THE FOLLOWING. Up to three detail qualifiers may be associated with each cause or action; Figure 13 illustrates the convention used in the architecture for indicating where, within the text string, the detail qualifier's unit of display is to be inserted by the NetView program.

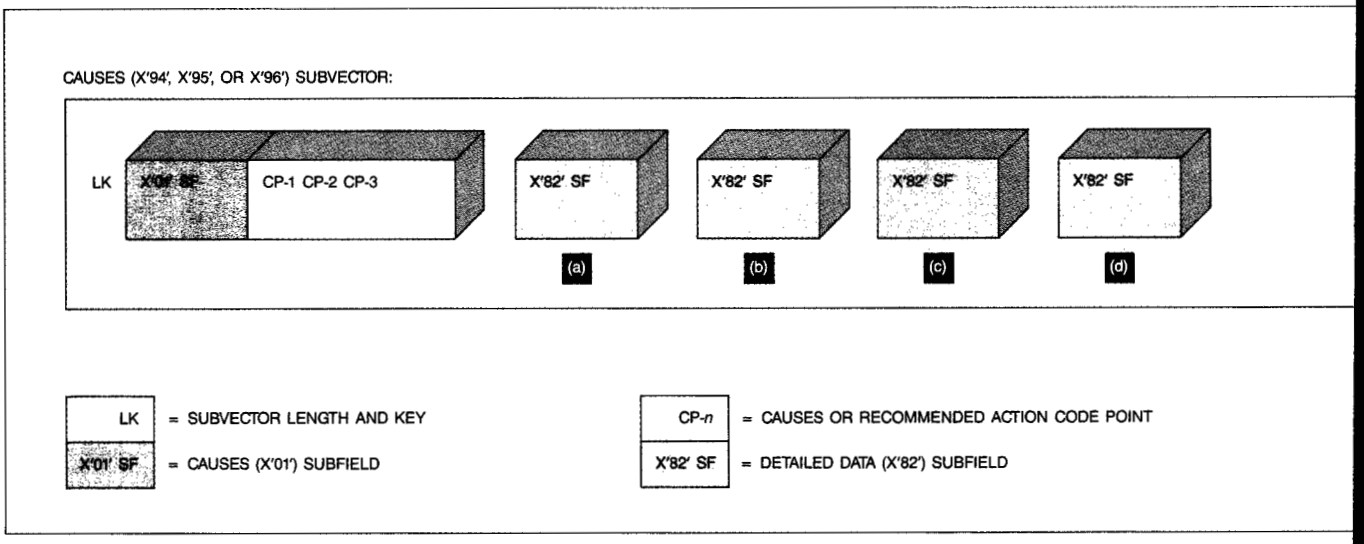
Figure 14 illustrates the common structure shared by the User Cause, Install Cause, and Failure Cause subvectors in the Alert major vector. The first subfield in each of these subvectors is always the Causes (X'01') subfield, containing all of the code points for indexing text elements identifying causes. In Figure 10 there would be three code points contained in this subfield within the User Cause subvector, and two code points in it within the Failure Cause subvector. Next come any Detailed Data (X'82') and/or Product Set ID Index (X'83') subfields associated with the causes code points. There would be none of these in the Alert for Figure 10.

The Product Set ID Index (X'83') subfield will not be discussed in detail here. It provides the same function as that provided by the Product ID Code field within the Detailed Data (X'82') subfield: instructing the NetView program to retrieve and display a product identification. It is used when a product identification is desired, not in conjunction with a piece of variable data, but just by itself. For example, the Product Set ID Index subfield is used with the recommended action text CALL THE APPROPRIATE SERVICE REPRESENTATIVE FOR XXX; the NetView program inserts the appropriate product identification in place of the X's.

Third, there is a Recommended Action (X'81') subfield containing one or more recommended action code points. Finally, there is again a set of X'82' and/or X'83' subfields, providing any detail qualifiers associated with the recommended action code points contained in the X'81' subfield. In the Alert for Figure 10, there is one X'82' subfield at this location in the Failure Causes subvector, providing the detail qualifier TTTT ERROR CODE 21F associated with the recommended action REPORT THE FOLLOWING.

The more general case shown in Figure 14 contains multiple Detailed Data subfields following both the causes and the recommended actions. The question arises of how the NetView program knows which

Figure 14 Structure of User Cause, Install Cause, and Failure Cause subvector



detail qualifiers should be associated with which code points. The problem is complicated by the fact that one or more of the code points may be unknown to the NetView program, since the NetView program is prepared to accept unknown code points and display default text for them. Given, then, the four detail qualifiers (a)–(d) in Figure 14, how does the NetView program know which ones (if any) belong with cause CP-1, which with cause CP-2, and which with cause CP-3?

The answer lies in the code points themselves. As Figure 15 indicates, the third hexadecimal digit of each user cause, install cause, failure cause, or recommended action code point indicates how many qualifiers belong with the code point. If, for example, CP-1 were X'10A0', the NetView program would know that qualifier (a), and only qualifier (a), belonged with it, even if X'10A0' were not in its table. Similarly, if CP-2 were X'2121' and CP-3 were X'11D0', the NetView program would know to associate qualifiers (b)–(d) with CP-3.

In the case of an *unknown* code point with associated qualifiers, the NetView program associates the units of display for the qualifiers with the default text that it displays for the code point.

Obviously it is up to the Alert sender to guarantee that exactly the right number of qualifiers are provided for the causes or recommended action code points included.

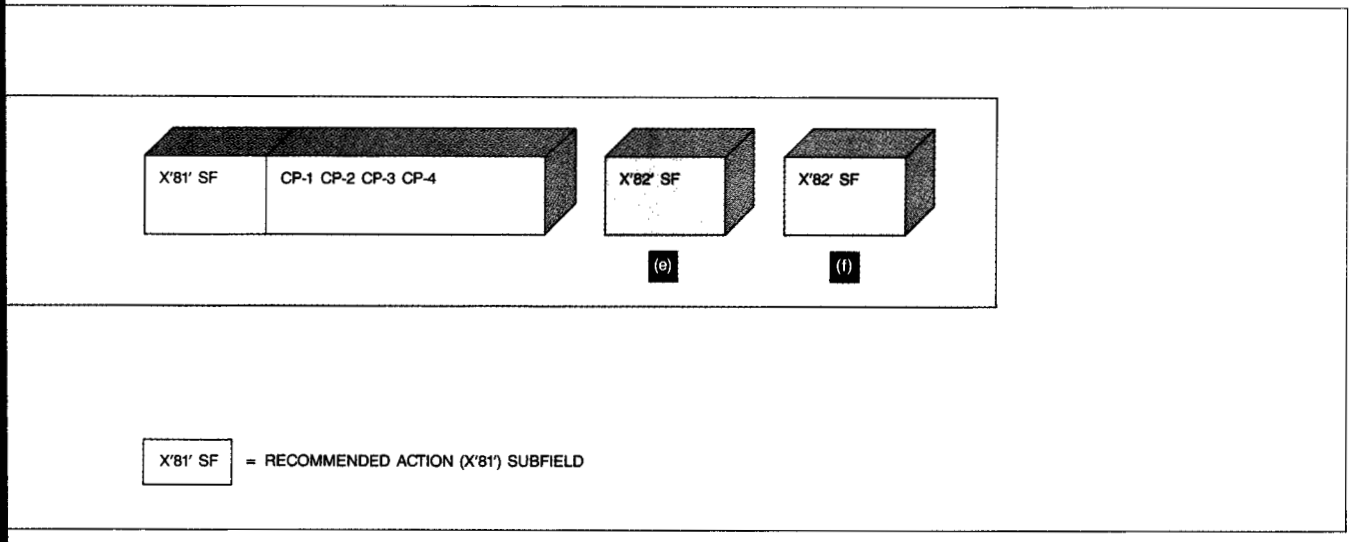
Alert hierarchies

The NetView program's Alert displays have always included hierarchy diagrams, such as that illustrated across the top of Figure 10. For each resource identified in the hierarchy, both an 8-character resource name and a 4-character resource type are displayed. The hierarchy is constructed on the basis of information provided to the NetView program by the Alert sender and by the Virtual Telecommunications Access Method (VTAM). Basically, VTAM is responsible for providing hierarchy information down through the Alert sender, while the Alert sender must provide information on any resources below itself.

Alert hierarchies play a number of roles in the NetView program:

- They provide the operator with a picture of where a failing resource is located in the network, and what path through the network must be taken to reach it from the NetView program.
- They allow the operator to select Alerts from specified senders for special processing, e.g., to have them logged but not displayed immediately by the NetView program.
- They form the basis for retrieval of Alerts from the NetView program's Alert database.

The generic Alert architecture does not introduce any significant changes in the transport or processing of Alert hierarchy information.



Protocol-unique subvectors

The architecture provides a very general mechanism allowing Alert senders to have the NetView program create Alert displays. In certain cases, however, more specialized encodings are needed. A good example of this is the specialized subvectors defined for the various link-level protocols.

Figure 16 shows the types of data carried in the SDLC Link Station Data (X'8C') subvector. While it would be possible to define Data IDs for each of these types of data, and then pass the data themselves in Detailed Data (X'82') subfields, there are several reasons why it is preferable to define a separate subvector for these data.

- Since many of the pieces of data are interrelated, it makes sense to enforce a grouping of them, both in the Alert major vector and on the NetView displays.
- This sort of data might be included in other management services records besides the Alert, e.g., in a major vector reporting details concerning a link connection. Putting this type of data into its own subvector facilitates its inclusion in different major vectors.
- This sort of data lends itself to automated analysis by various types of software. Putting the data into a separate subvector, rather than passing them as a series of detail qualifiers that could appear within any of a number of subvectors, makes it much easier for an automated routine to find the data within the Alert major vector.

Figure 15 Scheme for associating detail qualifiers with code points

CODE POINTS	QUALIFIERS
X'xx0x'-X'xx9x'	NO X'82' OR X'83' SUBFIELDS
X'xxAx'-X'xxBx'	ONE X'82' SUBFIELD
X'xxCx'	TWO X'82' SUBFIELDS
X'xxDx'	THREE X'82' SUBFIELDS
X'xxEx'	ONE X'83' SUBFIELD
X'xxFx'	RESERVED FOR FUTURE USE

Currently the Alert architecture contains three subvectors for transporting protocol-unique data: the SDLC Link Station Data (X'8C') subvector described in Figure 16, the LAN Link Connection Subsystem Data (X'51') subvector, for transporting data on token-ring, CSMA/CD, and bridged local-area networks, and the Link Connection Subsystem Configuration Data (X'52') subvector, for transporting

Figure 16 Subfields in the SDLC Link Station Data (X'8C') subvector

SUBFIELD	DESCRIPTION
X'01'	CURRENT N(S)/N(R) COUNTS
X'02'	OUTSTANDING FRAME COUNT
X'03'	LAST SDLC CONTROL FIELD RECEIVED
X'04'	LAST SDLC CONTROL FIELD SENT
X'05'	SEQUENCE NUMBER MODULUS FOR THE LINK STATION
X'06'	LINK STATION STATE (LOCAL OR REMOTE LINK STATION BUSY)
X'07'	NUMBER OF TIMES THE LLC REPLY TIMER (T1) HAS EXPIRED
X'08'	LAST RECEIVED N(R) COUNT

data on various types of link connections. Additional protocol-unique subvectors may be defined later if they are required.

Conclusion

With its introduction of the generic Alert architecture in Release 2 of the NetView program, IBM has provided a foundation upon which customers will be able to base the management of their increasingly common multivendor networks. This paper has described several key features of this architecture. It has also discussed how equipment and software manufactured by IBM, by other vendors, and by the customer can, by using the architecture, all receive equivalent support from the NetView program.

Cited references and notes

1. For a full description of the NetView/PC program, see M. Ahmadi, J. H. Chou, and G. Gafka, "NetView/PC," *IBM Systems Journal* **27**, No. 1, 32–44 (1988, this issue).
2. Robert E. Moore, "Problem Detection, Isolation, and Notification in Systems Network Architecture," *IEEE Infocom'86 Proceedings*, Miami, April 8–10, 1986, pp. 377–381.
3. In 1986 IBM released the NetView program, which contained a number of previously separate network management applications. One of these applications was Network Problem Determination Application (NPDA), a product that received Alerts from the network and presented them to the network operator. References in this paper to functions provided by the NetView program in many cases identify functions that were initially provided by NPDA.
4. Provision was made, in the Alert architecture and in the NetView program, for a partial display based on some variable data transported in the Alert, but this display was clearly inferior to the ordinary stored-panel displays.
5. *SNA Formats*, GA27-3136, IBM Corporation; available through IBM branch offices.
6. *SNA Format and Protocol Reference Manual: Management Services*, SC30-3346, IBM Corporation; available through IBM branch offices.
7. In the case of non-SNA products, the SNA component is an application program that runs on the NetView/PC program. This application program communicates with the non-SNA product itself to gather the necessary data on an error; then it formats these data as a generic Alert and makes use of the capability provided by the NetView/PC program to forward the Alert to the NetView program.
8. The negative side of this flexibility, excessive diversity, was discussed earlier. It is important to recognize, though, the benefits that it also provided.
9. The maximum length allowed for an NMVT is 512 bytes.
10. The same point obviously applies to stored panels as well: Since the index to a set of panels is not in any language either, it can index panels in different languages at different instances of the NetView program. The difference is one of practicality: Translation of generic Alert text strings is a manageable undertaking, whereas translation of literally thousands of sets of stored panels is not.
11. Actually, the number is not quite this large. First, the default/replacement structure reduces the number of available code points: Unless there are exactly 255 replacements under each default code point, some numbers will go unused. Second, there is a meaning assigned to the third hexadecimal digit of each code point that potentially causes other numbers to go unused as well. This use of the third digit of the code points is explained in the "Detail Qualifiers" section of the text.
12. The Product Set ID is not discussed in this paper. For details, see References 5 and 6.

Robert E. Moore *IBM Communication Products Division, P.O. Box 12195, Research Triangle Park, North Carolina 27709.*

Dr. Moore joined IBM in 1983. He has been in Network Management Architecture since that time, working primarily on the development of the generic Alert architecture. Currently he is working in the area of Performance and Accounting Management. Dr. Moore received a B.A. in mathematics and philosophy from Rice University in 1971, an M.A. and a Ph.D. in philosophy from Duke University in 1974 and 1977, respectively, and an M.S. in computer science from the University of Houston in 1983.

Reprint Order No. G321-5308.