A Message Handling System for B-ISDN User-Network Interface Signaling Test Software

B-ISDN user-network interface signaling has many different protocol variants and each of them has tens of different types of messages. The message handling system provides a powerful tool for the developer to easily support these variants and messages in the HP Broadband Series Test System (BSTS).

by Satoshi Naganawa and Richard Z. Zuo

Over the past several years, as the "information superhighway" has gained the attention of many countries and communication network service providers, the focus of the Broadband Integrated Services Digital Network (B-ISDN) industry has shifted from providing transmission capabilities via Asynchronous Transfer Mode (ATM) to providing a variety of B-ISDN network services such as private and public B-ISDN, switched virtual connections, and LAN emulation over ATM. Signaling plays a key role in realizing these services. The key functions of signaling are:

- Call setup and disconnection
- Negotiating and agreeing on quality of service (QoS)
- Providing accounting and charging information
- Resolving internetwork and intranetwork routing
- Facilitating basic services and supplementary services
- Identifying the occurrence and cause of any network or service failure.

Signaling consists of user-network interface (UNI) signaling, network-node interface (NNI) signaling for public networks, and private network-node interface (PNNI) signaling for private networks. The UNI is the boundary between a private or public network and the user equipment. User equipment can be a telephone, a computer, or video conference equipment as illustrated in Fig. 1.

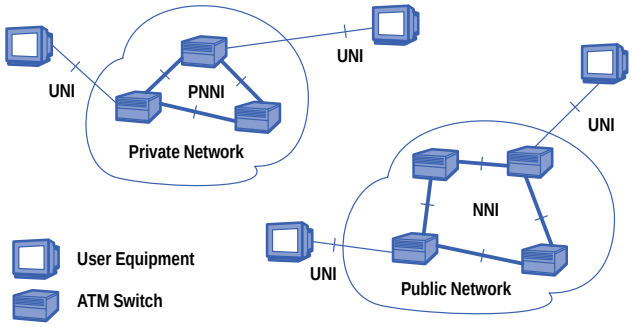


Fig. 1. *User-network interface (UNI), network-node interface (NNI), and private network-node interface (PNNI) signaling.*

The Broadband ISDN UNI signaling test software, HP E4214A, is a new addition to the HP Broadband Series Test System (BSTS). It is designed for R&D engineers and network engineers to develop, troubleshoot, and verify the features and functionality of UNI signaling. Fundamental features of the UNI signaling test software are protocol message decoding and encoding. With decoding, the user can monitor the message transaction between the user equipment and an ATM switch. With encoding, the user can execute a condition test such as constructing a correct or incorrect message and sending it to a switch to analyze the response.

UNI Signaling Standards

As ATM emerges as the mainstream technology in the telecommunication network market, the standardization of UNI signaling has been progressing at a very rapid pace during the last several years. Two major standardization organizations —the International Telecommunications Union Telecommunications Standardization Sector (ITU-T) and the ATM Forum—created their UNI signaling standards in an interlocked way. In May 1993, ITU-T published its first draft recommendation Q.93B[1] to support point-to-point ATM switched virtual connections. The ATM Forum adapted Q.93B to support point-to-multipoint connection control, variable-bit-rate connections, and the ability to signal quality of service (QoS) on a per-call or per-connection basis. These enhancements were published in the ATM Forum UNI 3.0 specification[2] in September 1993. Later, the ITU-T updated Q.93B as Q.2931[3] with some important changes. Following those changes the ATM Forum also updated UNI 3.0 into UNI 3.1[4] in September 1994. In the meantime, the ITU-T defined point-to-multipoint switched virtual connection features and published its standard as Q.2971[5] in June 1995. About half a year later, the ATM Forum published UNI 4.0[6] to incorporate the leaf-initiated join capability and the available bit rate (ABR) service in Q.2931 and Q.2971. In addition to the protocols promoted by the ITU-T and the ATM Forum, some countries and network service providers also created their own UNI signaling protocol variants. From this brief history, it can be understood why there are many UNI signaling protocol variants. Undoubtedly, more variants will come soon to support more new features.

In addition to having many different UNI signaling protocol variants, each variant has many different types of messages. A message is a protocol data unit (PDU) used in a signaling protocol. Each message can contain multiple information elements, called IEs. Some IEs must be contained in a particular message and are referred to as mandatory IEs. Some IEs are optional for a particular message, and therefore will not always appear in that message type. For example, the UNI 3.0 standard[2] has 16 different types of messages and 21 different types of IEs. SETUP, CALL PROCEEDING, CONNECT, and CONNECT ACKNOWLEDGE are the messages for call establishment, as shown in Fig. 2. The other categories of messages are designed for call clearing, point-to-multipoint communication, and other purposes.

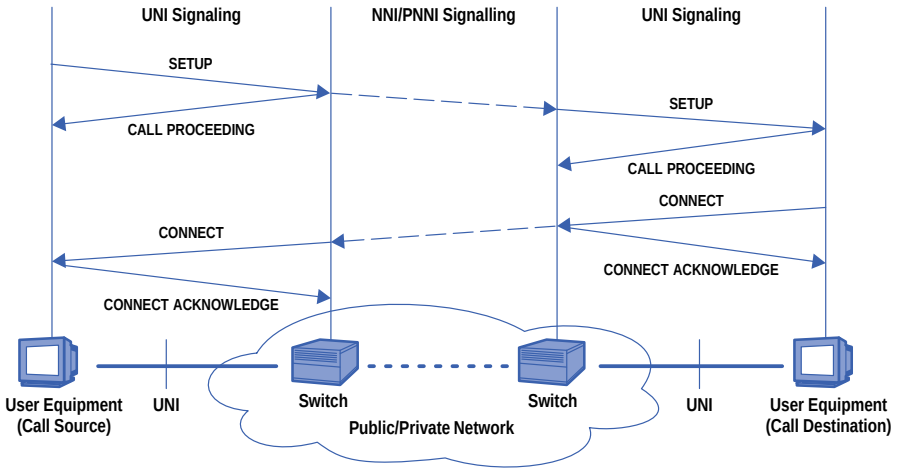


Fig. 2. *Setting up a connection between two user equipments via signaling.*[7]

The situation described above created a challenge for HP's BSTS UNI signaling test software developers. The software needed to support many protocol variants, with many different types of messages and IEs in each variant. Simultaneously, the R&D cycle time was much reduced. We realized that hard-coding each message and IE for each protocol variant in the decoding and encoding engines would make software maintenance a nightmare, would require the programmer to change the decoding and encoding engines whenever adding a new protocol variant, and would be of little help to us in developing NNI and PNNI signaling test software in the future.

Message Handling System

The *message handling system* was developed to solve this problem. The message handling system:

- Separates the structure description of messages and IEs from the remainder of the software
- Specifies the structure descriptions using a protocol-specification-like language in separate script files
- Compiles the script files to generate an object file
- Reads the object file into a database and allows the other parts of the BSTS to access those messages and IEs through library functions.

Fig. 3 depicts the architecture of the message handling system.

The message handling system describes a signaling protocol in one or more text files by means of a script language called the Message Description Language or MDL. A compiler converts the MDL script files into an object file by using the lex and yacc utilities provided by the UNIX® operating system. Like most compilers, the MDL compiler consists of a parser and a

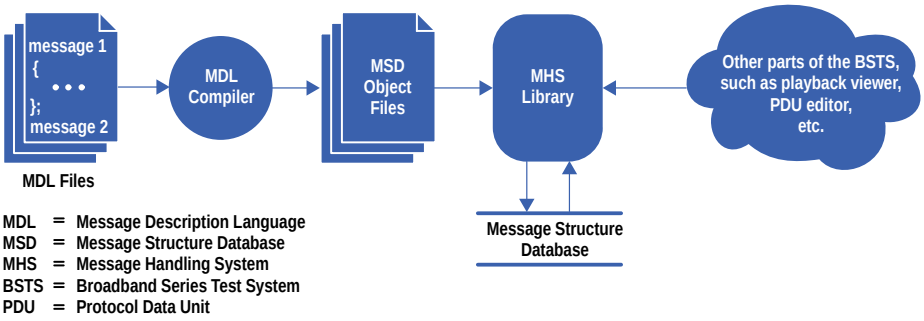
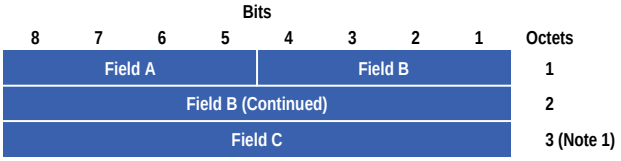


Fig. 3. *Message handling system overview.*

code generator. The parser reads an MDL text file, checks the syntax, and then puts the information in an internal data structure. The code generator generates an object file from the internal data structure.

With ease of use as a top priority, we designed the MDL as a simple protocol-specification-like language that allows a developer to easily translate a UNI signaling protocol specification into a script file. The MDL is also generic enough to handle different types of messages and IEs used in different UNI signaling variants. In the future, such as when using the message handling system for NNI and PNNI signaling, only slight changes are needed in the MDL and its parser.

Fig. 4 shows what the MDL looks like. This is a very simple and artificial example, but one that demonstrates some of the most important common characteristics of all types of messages. This example will also be used later to explain several interesting points in the message handling system structure database. The octet-field map with notations shown in Fig. 4a to describe a message structure is a widely used style in the signaling specifications and is easily translated into an MDL script as shown in Fig. 4b.



(a)

```
message example
{
    name “simpl_example”;

    octet 1 required {
        field Field_A[4]          Field_A_Prm;
        field Field_B[4].[12..9] Field_B_Prm;
    }
    octet 2 required {
        field Field_B[8].[8..1]  Field_B_Prm;
    }

    octet 3 Note_1 {
        field Field_C[8]          Field_C_Prm;}

    rule Note_1 {
        if (decode(Field_A) == 0) {
              return required;
        } else {
              return absent;
        }
    }
}
```

(b)

Fig. 4. *A very simple example of a Message Description Language (MDL) script. (a) The specification. (b) The MDL script.*

The message structure database object file shown in Fig. 3 is designed as a binary disk file version of the message structure database. The disk file format lets us easily separate the MDL compiler from the message handling system library so that a compiler developer doesn't need specific knowledge about the other parts of the test software such as the decoding and encoding engines. Unlike the MDL files, in which the description for a single protocol can be separated into several source files if more convenient, all the information for a single protocol is put into a single message structure database object file to make the run-time loading easier.

The message handling system library in Fig. 3 provides application programming interfaces (APIs) for the message handling system. The other parts of the BSTS can access the protocol-specific information stored in the message structure database through the library functions. Functions are available to:

- Initialize the message handling system library (this includes loading a message structure database object file to generate a message structure database)
- Decode a signaling PDU
- Construct a default signaling PDU
- Encode a signaling PDU.

An example of using the message handling system library is displaying captured PDUs. After a signaling PDU is captured, a message handling system library function is called to extract the values for all fields by using the protocol knowledge stored in the message structure database. Another message handling system library function is called to convert those values into a proper text format. A playback viewer in the other parts of the BSTS can display the formatted decoded result on the screen.

Another example of using the message handling system library is construction of a signaling PDU by the user. A PDU editor in the other parts of the BSTS calls a message handling system library function to create a PDU, which uses the most frequently used values as its default values, and shows the PDU on the screen. After the user makes any modifications to the PDU, the PDU editor calls another library function to put each field value in proper bit and byte order in an octet stream, as described by the protocol specification, to construct a new signaling PDU. During the interaction, the appearance of the user interface can change with the user's input. These dynamic changes are also controlled by message handling system library functions described later.

Message Structure Database

The message structure database shown in Fig. 3 is generated in the application memory space after a message structure database object file is loaded by the message handling system library. The message handling system library can load several different message structure database object files to generate several different message structure databases within the same application. This allows the user to use several different protocols simultaneously.

The message structure database is a set of data structures that define the structure of a PDU and give information about how to decode, encode, and construct a PDU. The message structure database is the core of the message handling system, and it is generic enough to support not only UNI signaling but also NNI and PNNI signaling. The key points in successfully building the message structure database are to correctly analyze the common requirements to express different types of messages and IEs, and to design proper data structures to meet these requirements. Some important requirements are:

- Mapping an octet format of a PDU into a field format or vice versa
- Handling field dependencies
- Handling predefined values for parameters
- Automatic inclusion of mandatory IEs and optional IEs selected by the user
- Automatic calculation of message length.

In the remainder of this section, the mapping and handling field dependencies are used as examples to show in detail what these general requirements entail and how the message structure database meets them.

The bit stream in a PDU can be interpreted into either an octet format or a field format. A single octet can contain part of a field or one or more fields, and in turn, a single field can span part of an octet or one or more octets as shown in Fig. 4a. The octet format is usually used in general computing, such as computing the length of a PDU. The field format is more significant when the user is only concerned with the value of a field but not with the field's bit position. In most cases, such as when decoding or encoding a PDU, both formats are important. Implementation requires three tables—the octet, field, and mapping tables—for each message in the message structure database. Fig. 5 shows how to map the octet format into the

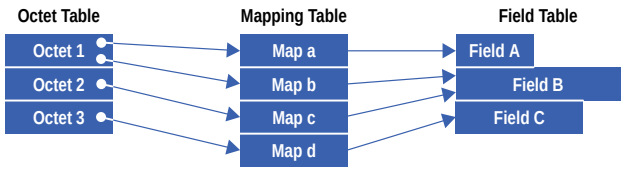
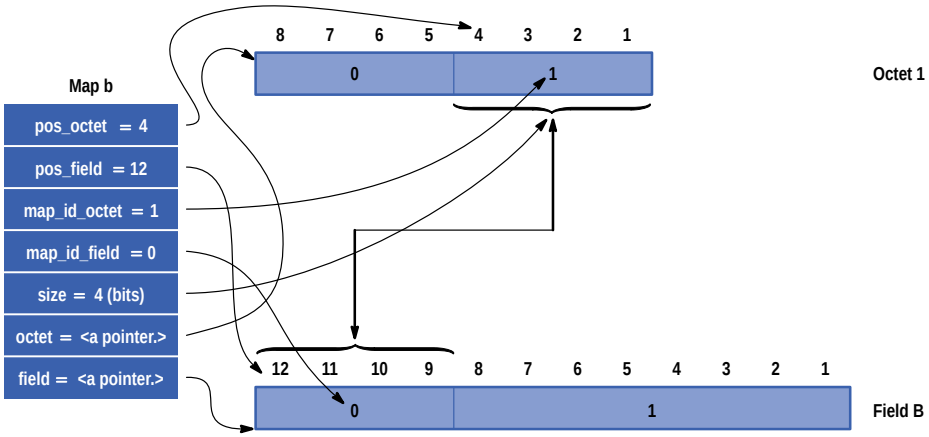


Fig. 5. *Mapping the octet format into the field format.*

field format using these three tables for the example displayed in Fig. 4. Each element in the mapping table is defined by the data structure mhs_msd_map_r shown in Fig. 6a. The definitions of the variables in mhs_msd_map_r are explained in Fig. 6b by showing how map b in Fig. 5 maps the second part of octet 1 with the first part of field B.

```
typedef struct
{
     unsigned int        pos_octet;
     unsigned int        pos_field;
     unsigned int        map_id_octet;
     unsigned int        map_id_field;
     unsigned int        size;
     struct octet_struct *octet;
     struct field_struct *field;
} mhs_msd_map_r;
```

(a)



(b)

Fig. 6. *The mapping data structure and an example. (a) The data structure for a map element. (b) Detail of map b in Fig. 5.*

In a signaling message, modifying the value of a field can change the definitions or values of other fields. This is called a field dependency. Usually the effect of the field dependency needs to be interpreted in real time. For the example shown in Fig. 4, Note 1 indicates that field C is present when the value of field A is equal to 0, and is absent otherwise. Therefore, on the user interface of a PDU editor, the *sensitivity* of field C should be changed with the value of field A in real time as indicated by Fig. 7.

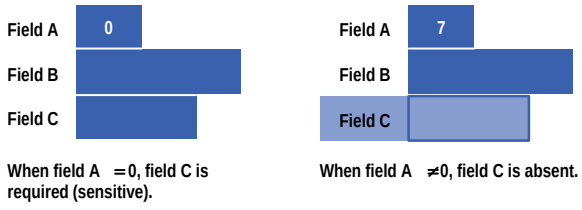


Fig. 7. *An example of field dependency.*

The message handling system internal interpreter was developed to handle field dependencies. The internal interpreter includes two parts: an internal interpreter table for each message in the message structure database to store the field dependency descriptions, and an internal interpreter engine, running a simple and small stack-oriented language, to interpret these descriptions in real time. Fig. 8 shows how the internal interpreter works in this example.

The MDL compiler converts the field dependency description from the MDL script shown in Fig. 8a into an internal interpreter table indicated by Fig. 8b. In this case, the internal interpreter table consists of three blocks of internal interpreter codes. Block[0] indicates how to evaluate the expression in the if(expression) statement. Block[1] will be executed if the expression is true and block[2] will be executed otherwise. Each block contains a sequence of internal interpreter code. Block[0] contains multiple lines of internal interpreter code, while block[1] and block[2] each contain only a single line. A line of internal interpreter code is defined by the structure mhs_mii_code_r, which contains a command and its

```
rule Note_1 {
    if (decode(Field_A) == 0) {
          return required;
    } else {
          return absent;
    }
}
```

(a)

```
typedef struct
{
      commands_e       command;
      unsigned int     value1;
      unsigned int     value2;
} mhs_mii_code_r;
```

```
typedef enum
{
      required = 0,
      optional = 1,
      absent   = 2
} inclusion_type_e;
```

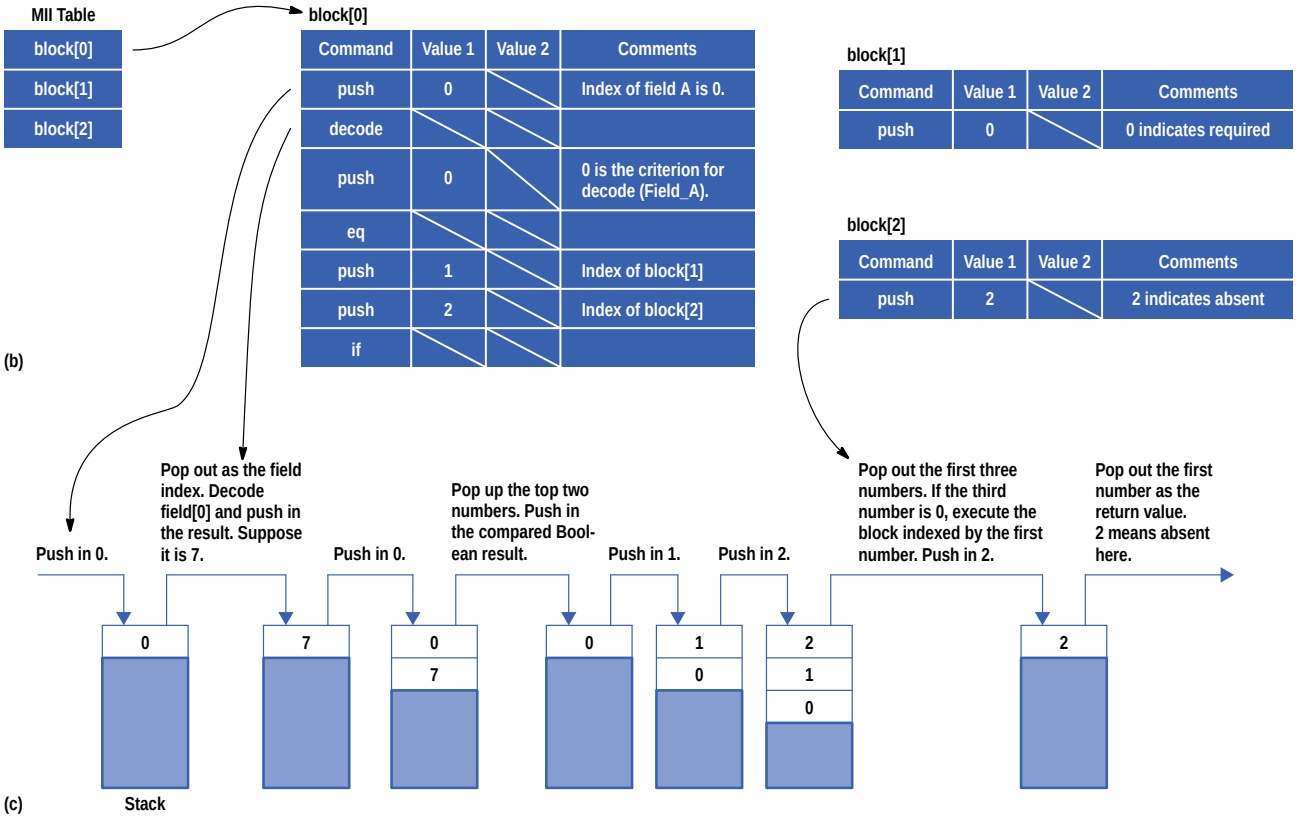


Fig. 8. *An example for the message handling system internal interpreter. (a) The MDL script. (b) The internal interpreter data structure in the structure database. (c) How the internal interpreter engine works.*

corresponding values if applicable. For example, the command push has only a single applicable value located in value 1. The comment columns in each block give the explanation of the values used in each line.

When the user changes the value of field A from 0 to 7, the internal interpreter engine is called to execute the codes started from block[0] to evaluate the expression (decode(Field_A) == 0) as shown in Fig. 8c. The expression is not true in this case, since the value of field A is 7, and therefore block[2] is executed. The absent value, which is defined as 2 in the data structure inclusion_type_e, is returned as the last result of rule Note 1 (Fig. 4a). This makes the PDU editor appropriately change field C from sensitive (required) to insensitive (absent) to meet the real-time interpreting requirement indicated by Fig. 7.

Results

Using the message handling system to add a new UNI signaling protocol variant, the only work that a developer needs to do is to create a script file. In the script files the developer can easily describe the messages in the variant by using the protocol-specification-like language. The message handling system hooks the script file into the other parts of the BSTS automatically, uses the protocol-related knowledge to decode and encode PDUs, and produces the desired results.

In this section, the organization of the general message and one of its instances, the SETUP message, are used as examples to demonstrate the MDL script and the results provided by the BSTS. The SETUP message is one of the most important types of messages used in UNI signaling since it carries all the information for requesting a connection between two user equipments. Fig. 9 shows parts of the protocol specification from UNI 3.0.[2] Fig. 10 is the corresponding MDL script.

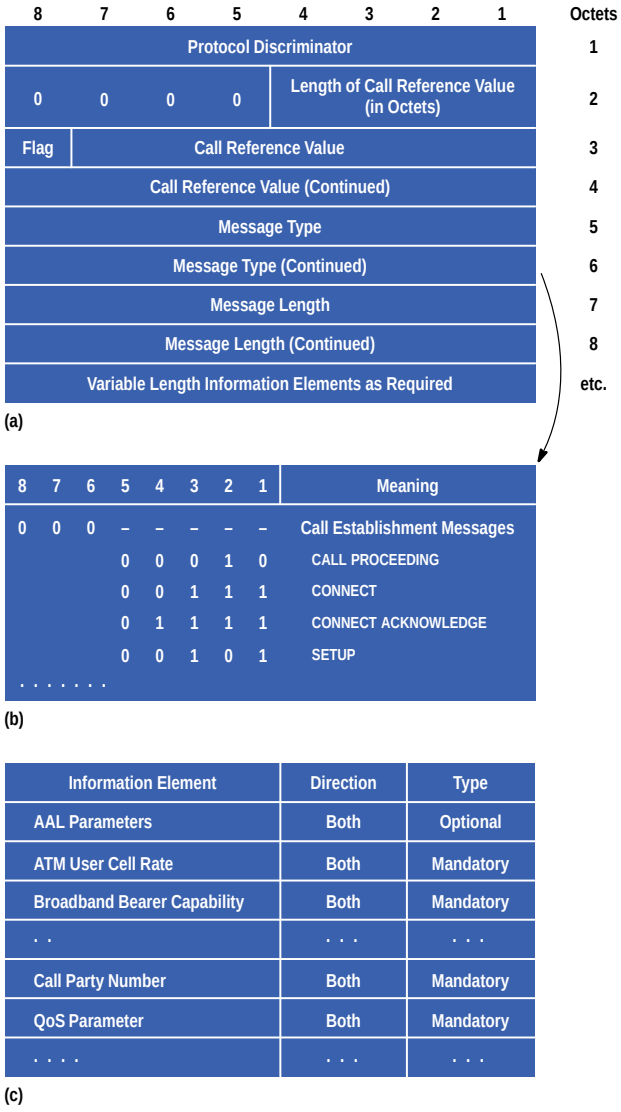


Fig. 9. *General message organization and SETUP message specification.[2] (a) general message organization. (b) Predefined values for message types. (c) IE inclusion of the SETUP message.*

Fig. 9a is the simplified definition of the general message organization. When the value of octet 6 is equal to 5 (binary value 00000101), the message type is defined as SETUP as illustrated in Fig. 9b. Fig. 9c depicts the IE inclusion for the SETUP message. The "Direction" column indicates whether an IE is used in the user-to-network direction, the network-to-user direction, or both.

Fig. 10 shows two segments of the MDL script with C-style comments. Fig. 10a corresponds to the specification of the general message organization shown in Fig. 9a and Fig. 10b is the translation for Fig. 9b and Fig. 9c.

```
generalmessage Message
{
    octet 1
    ....
    octet 6 required {
    /* this octet is required by the
    specification */
        field MessageType[8] MessageType = 5;
        /* MessageType is the variable name.
        [8] indicates the width is 8 bits
        and value 5 (SETUP) is the default
         value */
    }
    ....
    octet 9
    ....
}
```

(a)

```
message SETUP
{
  id   0b00000101;
  /* binary identifier for SETUP message */

  AALParam       both optional;
  /* both directions are optional */
  ATMCellRate    both mandatory;
  /* both directions are mandatory */
  BBearerCap     both mandatory;
  ....
  CalledNum      both mandatory;
  QosParam       both mandatory;
  ....
}
```

(b)

Fig. 10. *MDL scripts. (a) Part of the MDL script for the general message organization. (b) Part of the MDL script for the* SETUP *message.*

Fig. 11 shows the user interface of a UNI signaling PDU editor provided by the BSTS after it hooks into the message handling system. As defined by the specification shown in Fig. 9a and translated by the MDL script shown in Fig. 10a, a message includes the following major parts:

- Protocol discriminator
- Call reference
- Message type
- Message length
- Information elements, as required.

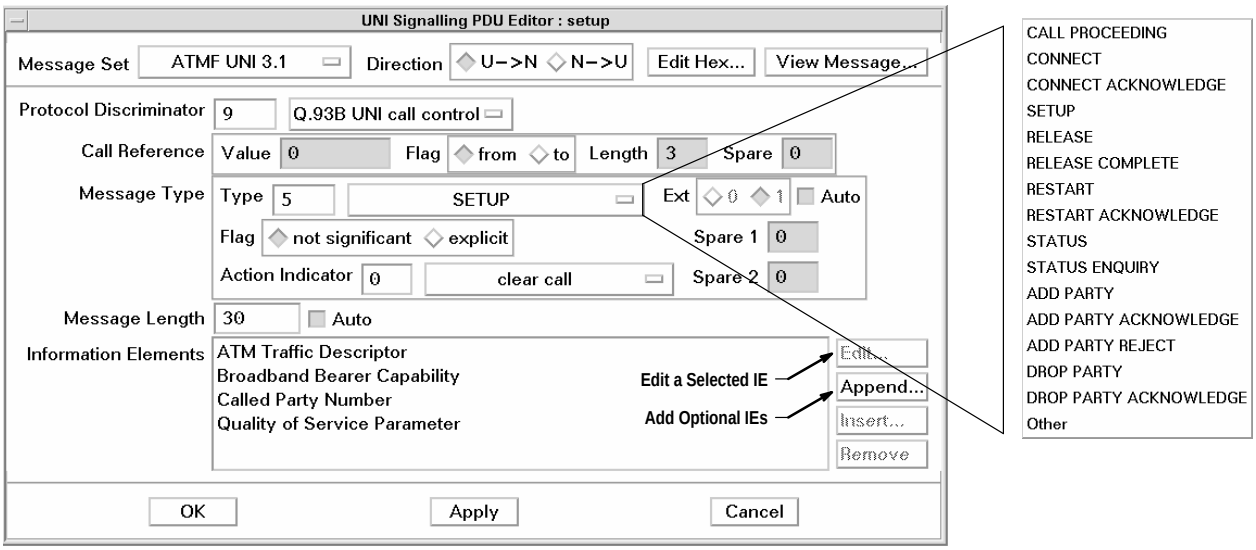


Fig. 11. *Using the PDU editor to construct a* SETUP *message.*

From the option menu in the Message Type item, the user can select any type message to edit. For the selected type, the Type text field automatically displays the corresponding digital value and the Information Elements list shows all the mandatory IEs. In this example, the message has been interpreted as a SETUP message (value 5) and four mandatory IEs have been listed according to the MDL script in Fig. 10b. The user can also edit an IE by clicking the Edit button and add optional IEs by clicking the Append button.

A PDU is constructed by clicking the OK or Apply button. All the field values specified by the user are put in proper bit and byte positions according to the description of the MDL script.

Using the BSTS playback viewer, Fig. 12 shows the decoding result of the constructed message in this example. Fig. 13 illustrates the hexadecimal format of the message. In Fig. 12, the window contents are in two parts: the message header, which includes the first nine octets in the general message organization, and part of the first contained IE, the ATM user cell rate. Notice that the value of octet 6 in the message header is 00000101 in binary, which is interpreted as SETUP according to the MDL script shown in Fig. 10b. With this decoding result, the user can easily monitor the transaction between the user equipment and an ATM switch.

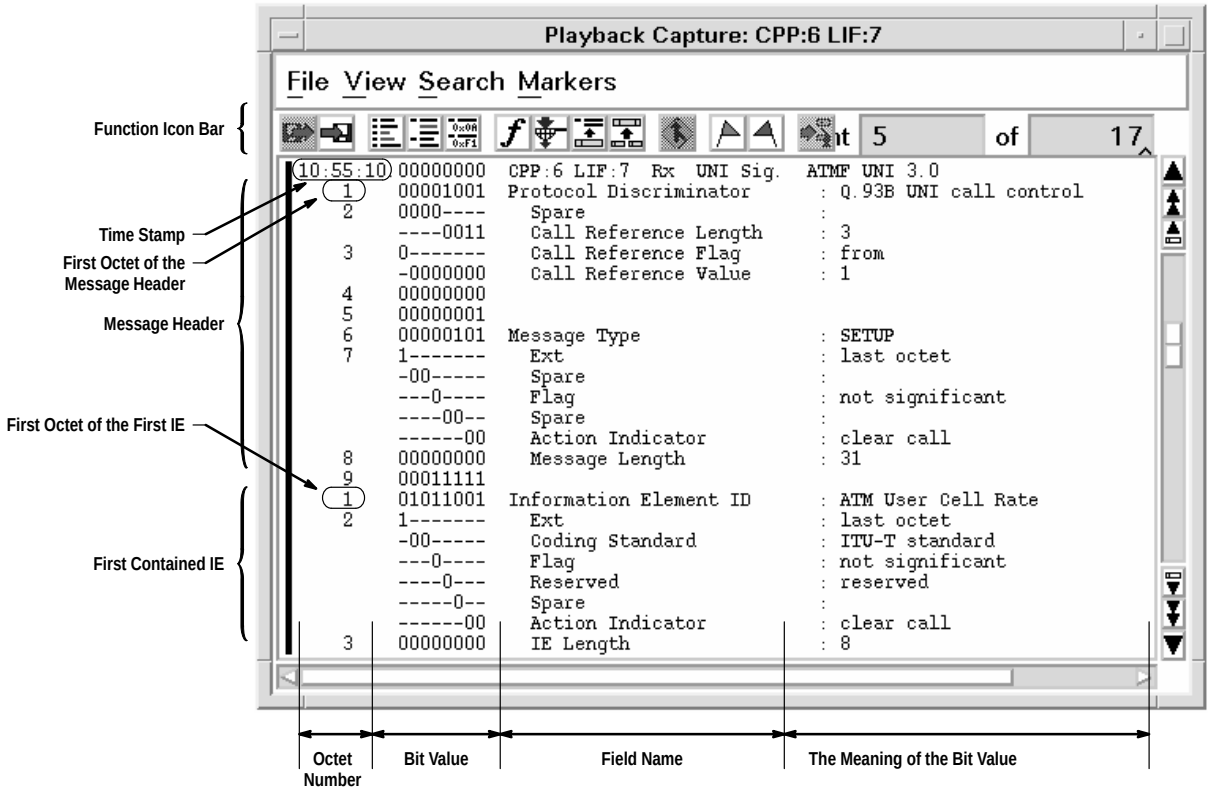


Fig. 12. *Decoding result displayed on the playback viewer.*

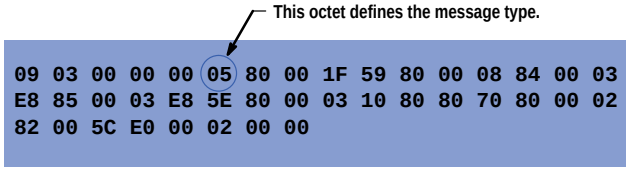


Fig. 13. *The hexadecimal format of a message.*

Summary

The message handling system provides a powerful tool for developers to handle many UNI signaling protocol variants and also many different types of messages and IEs used in each variant. The message handling system significantly improves the maintainability, extensibility, and flexibility of the UNI signaling test software. Because there are a lot of similarities between UNI signaling protocols and NNI/PNNI signaling protocols, it would be possible to reuse the message handling system were we to add NNI and PNNI signaling test software to the BSTS in the future. The message handling system enables HP to keep its competitive position in the signaling test software field, in which old protocols are constantly updated and new protocols are constantly emerging.

Acknowledgments

We would like to thank Mitsuo Matsumoto, who is one of the key developers of the message handling system, and Patrick Siu, who wrote the draft for the introduction and reviewed other parts of this paper. Thanks also to Janet Lomas and Nick Malcolm, who reviewed this paper and gave helpful suggestions for improvement.

References

1. *B-ISDN User Network Interface Layer 3 Specification for Basic Call/Bearer Control*, ITU-T Recommendation Q.93B, May 1993.
2. *User-Network Interface Specification Version 3.0*, ATM Forum, September 1993.
3. *B-ISDN DSS-2 User Network Interface Layer 3 Specification for Basic Call/Connection Control*, ITU-T Recommendation Q.2931, formally approved by ITU in September 1994.
4. *User-Network Interface Specification Version 3.1*, ATM Forum, September 1994.
5. *B-ISDN DSS 2 User Network Interface Layer 3 Specification for Point-to-Multipoint Call/Connection Control*, ITU-T Recommendation Q.2971, June 1995.
6. *User-Network Interface Specification Version 4.0*, ATM Forum, February 1996.
7. A. Scott, *Implementing ATM Signaling: Avoiding the Interoperability Pitfalls*, HP BSTS Solution Note 5963-7514E.

▶ Go to Next Article

▶ Go to Journal Home Page