Architecture of a User Interface Module for Structured Internet Messages

MORIOKA Tomohiko[1]

Japan Advanced Institute of Science and Technology, Hokuriku
1-1 Asahidai, Tatsunokuchi, Nomi, Ishikawa, 923-12, Japan

Abstract. MIME introduces a structured Internet message format. It can represent various data formats and various coded character sets. MIME thus represents an important part of Internet message processing. Conventional message user agents (MUAs) which process Internet messages have different characteristics. The quality of the user interface depends on the user's preference or habits. In this regard, it may be preferable to allow the user to select a good combination of the MUA and a separate MIME module. This paper describes the architecture of a general MIME processing module to provide its features for various conventional MUAs. In the design of such a module, two complementary models are proposed: "representation layer model" which handles complex structured data and the "acting situation model" which processes various kinds of data under various environments.

1 Introduction

MIME (Multipurpose Internet Mail Extensions) is a proposed standard to provide an extension of the traditional Internet message format defined by STD 11[2]. STD 11 messages can represent only US-ASCII[1] plain text. MIME messages, however, can represent various formats specified by a "media-type"[3] for various kinds of information. MIME messages can also represent various coded character sets specified by a "MIME charset"[4] for various languages. Rather than forcing a single unified language coding system or media format, MIME enables the coexistence of different formats. In this way, MIME is an important protocol for Internet message exchange.

There are many existing message user agents (MUAs) built on many user interface models. This is reasonable because the user is free to select the one he is most comfortable with. A number of these MUAs have built in MIME support which the user may or may not be comfortable with. It seems logical to extend the same modular approach used with MUAs to the MIME support. One approach in designing MUAs is to consider the MIME module separately.

MIME offers a "multipart"[4] feature for messages. Structured messages form a hierachical tree in which each node of the tree may be called an "entity"[3]. Each entity has an attribute known as "media-type" for representing its data type. Each message may have multiple entities. Although conventional MUAs such as mail readers, netnews readers, etc, provide message navigational features, they do not typically provide any functionality for navigating structured messages containing multiple entities.

In this study, a preview mechanism based on the notion of a "representation layer model" is proposed to overcome this deficiency. The "representation layer" separates the message contents into "raw message" and "message-preview". "Raw message" is the original information specified by STD 11 and MIME, whereas "message-preview" carries information about the position in the display. The message processing modules described in this paper are based on such a representation system.

A conventional MUA cannot process the entities itself, the MIME module has to be added to process these entities according to their media-type using the appropriate program. There is usually a simple one-to-one mapping between media-type and program. Entity processing is not dependent on media-type alone and a simple mapping may not be adequate. The "acting situation model" is thus proposed as a solution. Instead of a simple mapping between media-type and the appropriate program handler, a generalised MIME processing module is designed.

The proposed system has been implemented on GNU Emacs and its variants, such as "tm"[5] and "SEMI"[6].

2 Display for message contents

Conventional Internet messages are limited to unstructured text that does not require complex message processing. Conventional MUAs thus ignore any internal structure and simply manage messages. The proposed MIME processing module must provide adequate features to process and manage a hierarchy of entities and at the same time it has to offer an interface between a message and its structure. In the present approach, SEMI and tm employ the *"preview"* function for supporting these functions. Entity related information is embedded to allow the user to navigate entities in *preview*. Tm and SEMI manage the mapping between a message and its *preview*. The interface modules between the MIME processing module and an MUA replace the display of a message by its *preview*. *Preview* functions as a message display for MUAs and therefore, MIME processing features are added to the message features of the original MUAs.

The design of the *preview* mechanism is based on the "representation layer model". This model combines layers and allows represention and manipulation of structured data. It organizes a complex system.

2.1 Representation layer model

In the "representation layer model", structured data is represented by several "representation layers". A "representation layer" stores structured data and provides manipulations of this data in a manner determined by its format.

Structured data consists of a number of units. In MIME, an entity is a unit of message content. A "representation unit" is represented by different formats in different kinds of “representation layers”. However, they are only different shadows for the same object. The term shadow is used because it may be an incomplete copy of the original. The shadows are attributes of the object and the object is represented by its shadows.

A “representation unit” is a link between “representation layers”. “Representation unit” indicates a unit of structured data. A structured data is represented by a unit in a layer. It is called a “shadow”. “Representation unit” is a relation between its shadows in “representation layers”.

If the user manipulates a shadow in a layer and its operation requires data in another layer, processing is propagated to the other layer by a “representation unit”. Each layer has its own manipulating features for shadows and some of them are propagated to other layers.

Thus “representation units” link “representation layers” and organize them into a system. Such systems are called “representation systems”.

While it may be possible to represent complex data by one structure, it may be too complicated (Fig. 1). If it is separated into several “representation layers” by format, each representation layers can be simple (Fig. 2). The latter is much simpler to implement and maintain.

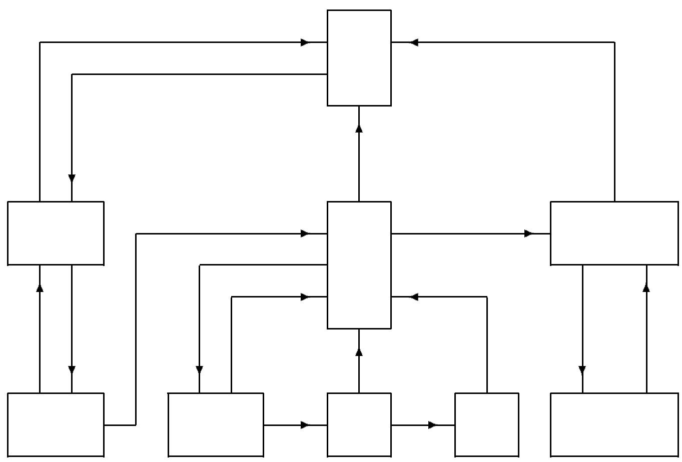

Fig. 1. Example of single structured representation

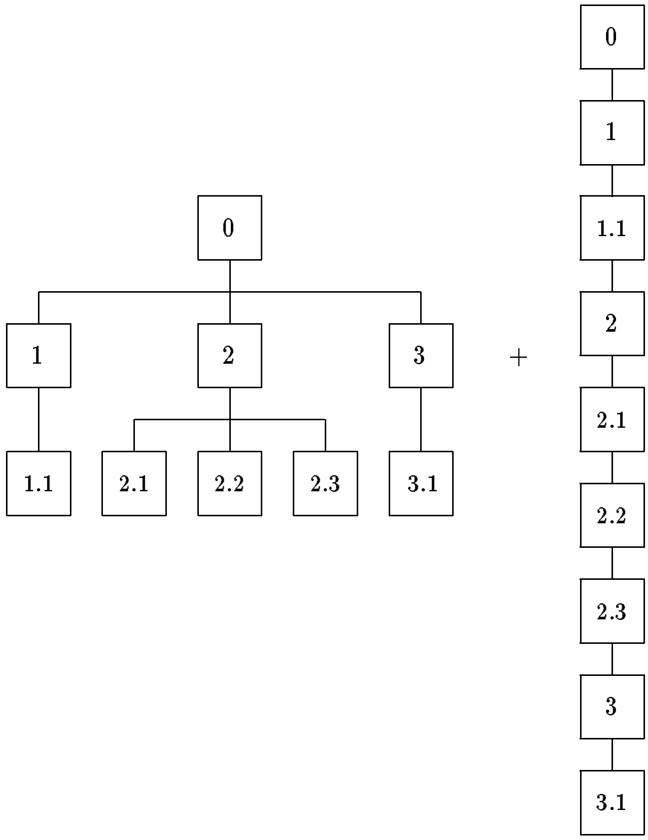

Fig. 2. Example of separated representation layers

2.2 Representation system for Internet message contents

In tm or SEMI, the "representation system" for message content consists of two "representation layers". One is the original content of received message, called the "raw-layer". The other is displayed to the user and is called the "previewed-layer".

Conventional MUAs do not distinguish between the "raw-layer" and the "previewed-layer" (Fig. 3). Because messages as specified by STD 11 do not have structured contents, conventional MUAs can display the original contents. Although they might do some processing of the message [1], the transformations

[1] Most conventional MUAs perform header filtering.

involved do not require that the original data be saved. MIME has various kinds of media-types and MIME charsets and it permits a mixture of different kinds of each. Therefore, the MUA must display the message contents after parsing is done. Some media-types, like sound or other specialized data cannot be displayed. In this case, the MUA should process them when the user would like it processed after the message content is displayed. Thus, message content managed by conventional MUAs should be separated to the "raw-layer" and "preview-layer" (Fig. 4).

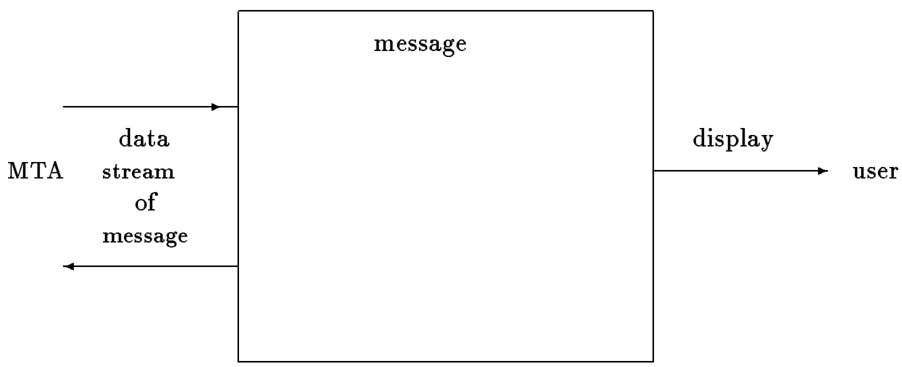

Fig. 3. Message model in non-MIME MUA

The message management of conventional MUAs are classified as two types: "raw-layer" and "preview-layer". The former is the original data. It is usually only used internally. The latter is displayed to user. The interface module between a general MIME module and a conventional MUA must distinguish between them and provide hooks into them.

Construction for both "representation layers" is done by the following steps. First, the "raw-layer" is made from the original message content as parsed by message syntax. After parsing is done, a hierarchy of entities is generated. The "raw-layer" is the representation of the parsing result.

The "preview-layer" is generated from the "raw-layer". Each shadow corresponding entity is flattened and sorted for the display model. The preview generator calls the "preview-filter" for each shadow element. The "Preview-filter" displays and returns the shadow to the "preview-layer". Shadows in the "raw-layer" and "preview-layer" generate the "representation-unit". In tm or SEMI, this is implemented with cell sharing. The shadow in the "preview-layer" has a

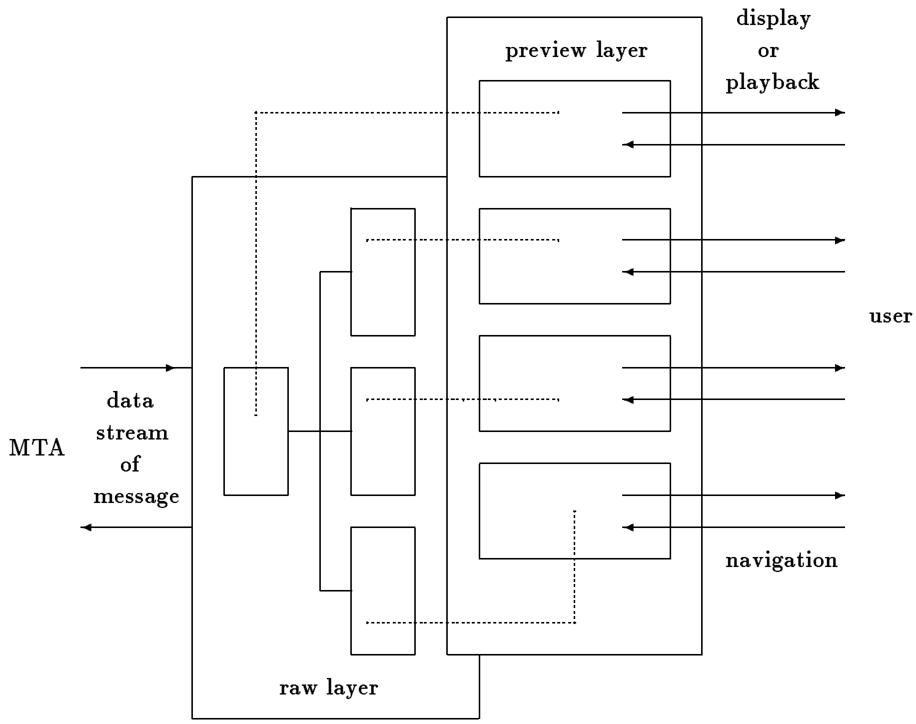

Fig. 4. Message model in MIME processing module

pointer to the "raw-layer". This implementation is able to bypass the high cost of searching in the "preview-layer". A lot of navigations are performed by the user and executed in the "preview-layer", so it is a good method.

Tm and SEMI have multiple "preview-layers" and one "raw-layer". A "preview-layer" may be a hierarchy of "preview-layers". For example, MIME has a "message/rfc822" media-type. Its content can be a MIME message and it's usually used to forward a message. Therefore MIME processing module should allow navigation for message or display as a message. Recursive previewing is available in this case, and several "preview-layers" for one "raw-layer" is good way.

3 Entity Processing

Each entity of MIME message content has a "media-type"[3] to specify the format of its content. It is used for processing by general rule. Of course, conventional MUAs do not use media-types and do not have entity processing features based on the MIME specification. A general MIME module should provide this feature.

MIME message contents have information of their formats specified by MIME syntax. Each entity has syntactical information specified by its header. For example, `Content-Type`[3] fields specify media-type and some of them may have other values of their attributes. It is possible to automatically play entities if the processing method is defined by MIME rules. For example, mailcap[7] specifies mapping rules between media-types and corresponding programs. Notice that entities are not specified only by media-types but also with other parameters. A simple mapping between media-type and program is not sufficient.

Even if sufficient mapping rules are specified, the processing of entities are controlled by message sender and the MIME module. For example, the user can not control display of WWW pages when viewing them. The MIME module should support controllability about what to display and what kind of processing should be done for entities. System customization of the mapping rule is available, but since the user is also allowed to customize it the system mapping rule and the user's mapping rule should be able to be merged.

MIME processing should be controlled by user operations. It is inconvenient for the user to have to specify low level details of processing, so we use some operation types that correspond to abstracted user operations.

The MIME module should harmonize with the MUA's provided MIME features. The behavior of the MIME module should also be compatible with each MUA's typical mode of operation. In other words, the behavior of the MIME module should be changed for different MUAs.

In the present study, the notion of "acting situation model" is proposed to overcome the above mentioned problems. In the "acting situation model", processing methods for entities are mappings with an "acting situation" instead of simple mappings between media-types and corresponding programs.

3.1 Acting situation model

An "Acting situation" consists of the following elements:

1. information of entity specified by header (ex. `Content-Type`[3])
2. conditions related to program execution
3. operation type
4. information related to MUA type

An "acting situation" is generated by the operation on entity. In the case of tm and SEMI, an acting situation is represented by an association list:

$$((t_1.v_1)(t_2.v_2)...(t_n.v_n))$$

An acting situation consists of attributes with type t_i and value v_i.

3.2 Information of entity specified by header

It is generated from information of header of each entity. An example of the representation in SEMI is:

```
((media-type . text)(media-subtype . plain)
 ("charset" . "iso-2022-jp")
 (encoding . 7bit)(disposition-type . inline))
```

Media-types (primary type and subtype) are specified by the symbols 'media-type' and 'media-subtype'. Content-Transfer-Encoding is specified by the symbol 'encoding'. Disposition-type is specified by the symbol 'disposition-type'. The parameters of the Content-Type and Content-Disposition fields are specified as strings.

3.3 conditions related to program execution

They are a list of acting situation types. Each acting situation type is a subset of acting situations. It specifies expecting acting situation to run its method. Representation of SEMI is list of association lists.

```
(((media-type . video)(method "xanim" nil "+Ae" -c 'name)
  (mode . play)(encoding binary 8bit 7bit))
 ((media-type . audio)
  (method "tm-au" nil 'file 'type 'encoding 'mode 'name)
  (mode . play)(encoding binary 8bit 7bit))
 ((media-type . image)(method "display" nil 'name)
  (mode play print)(encoding binary 8bit 7bit))
 ((media-type . message)(media-subtype . external-body)
  ("access-type" . "anon-ftp")
  (method . mime-display-message/external-ftp))
 ((media-type . message)(media-subtype . rfc822)
  (method . mime-display-message/rfc822)(mode . play))
 ((media-type . message)(media-subtype . partial)
  (method . mime-combine-message/partials-automatically)
  (major-mode . gnus-original-article-mode)
  (summary-buffer-exp . gnus-summary-buffer))
 ((media-type . message)(media-subtype . partial)
  (method . mime-combine-message/partials))
 ((method "metamail"
          'with-header "-m" "tm" "-x" "-d" "-z" "-e" 'file)
  (mode . play))
 ((method . mime-extract-file)(mode . extract)))
```

Each elements are as same as acting situations, like following:

```
((media-type . message)(media-subtype . partial)
 (method . mime-combine-message/partials-automatically)
 (major-mode . gnus-original-article-mode)
 (summary-buffer-exp . gnus-summary-buffer))
```

3.4 Operation type

tm and SEMI provide three different user operations: 'play', 'file extraction' and 'print'.

3.5 Information related to MUA type

This indicates the kind of running MUA. GNU Emacs and its variants have a variable 'major-mode' to indicate kinds of applications. tm and SEMI use it.

3.6 Composition of acting situation

During an operation on entity, tm or SEMI generates the draft for acting situation with the help of (1) [**information of entity specified by its header**], (3) [**operation type**] and (4) [**information related to MUA type**]. They are unique and therefore they are simply appended. Finally the appropriate condition is retrieved from (2) [**conditions related to program execution**] and it is compared with the draft of acting situation.

Comparing method between two acting situations S_1 and S_2 succeeds when every attributes of S_1 and S_2 are compared successfully. If type of an attribute a_i of S_1 is found in S_2 (a_j), and if values of a_i and a_j are same, then the attributes comparing about type of a_i and a_j succeeds and returns a_i. If type of an attribute a_k of S_1 is not found in S_2, then the attributes comparing about type of a_k succeeds and returns a_k. Similarly, if type of an attribute a_l of S_2 is not found in S_1, then the attributes comparing about type of a_l succeeds and returns a_l. Otherwise attribute comparing fails. If every attributes comparing succeed, comparing between S_1 and S_2 succeeds and return acting situation $S3$ consists of every results of attribute comparing. For example, comparing between $((a.1)(b.2))$ and $((a.1)(c.3))$ succeeds and returns $((a.1)(b.2)(c.3))$, but comparing between $((a.1)(b.2))$ and $((a.3)(c.3))$ fails.

For example, if the draft for action situation is:

```
((media-type . message)(media-subtype . partial)
 ("number" . "2")("total" . "3")
 ("id" . "foo@tsukuyomi.jaist.ac.jp")
 (mode . play)(major-mode . gnus-original-article-mode))
```

and

```
((media-type . message)(media-subtype . partial)
 (method . mime-combine-message/partials-automatically)
 (major-mode . gnus-original-article-mode)
 (summary-buffer-exp . gnus-summary-buffer))
```

is matched with the list of conditions (2), then

```
((media-type . message)(media-subtype . partial)
 ("number" . "2")("total" . "3")
 ("id" . "foo@tsukuyomi.jaist.ac.jp")
 (mode . play)(major-mode . gnus-original-article-mode)
 (method . mime-combine-message/partials-automatically)
 (summary-buffer-exp . gnus-summary-buffer))
```

will be generated as the acting situation. Next, the program specified by the 'method' attribute is executed. The remaining attributes are passed to the program.

4 Conclusion

The proposed MIME processing module employs *preview* to provide an adequate mapping mechanism between a message and its hierarchy of entities. It is based on the proposed "representation layer model". This model is available to represent and navigate complicated structured data like MIME message contents.

The functions of the proposed MIME processing module should be consistent with the original MUA functions. The acting situation model allows the user to specify conditions along with various factors. In the proposed method the acting situation is generated dynamically, hence it is flexible. However, the present implementation of the model might generate the wrong acting situation. Therefore, the subject needs further investigation.

References

1. Coded Character Set – 7-Bit American Standard Code for Information Interchange. ANSI X3.4 (1986)
2. Crocker, D.: Standard for the format of ARPA Internet text messages. RFC 822 (1982)
3. Freed, N. and Borenstein, N.: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045 (1996)
4. Freed, N. and Borenstein, N.: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. RFC 2046 (1996)
5. ftp://ftp.jaist.ac.jp/pub/GNU/elisp/mime/
6. ftp://ftp.jaist.ac.jp/pub/GNU/elisp/mime/alpha/
7. Borenstein, N.: A User Agent Configuration Mechanism For Multimedia Mail Format Information. RFC 1524 (1993)

This article was processed using the LaTeX macro package with LLNCS style