MULTIMEDIA MAIL: AN IMPLEMENTATION PROVIDING A SCENARIO SERVICE

By

TAHAR AGASTANI

Bachelor of Electrical Engineering

University of Indonesia

Jakarta, Indonesia

1988

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the degree of
MASTER OF SCIENCE
December, 1995

MULTIMEDIA MAIL: AN IMPLEMENTATION
PROVIDING A SCENARIO SERVICE

Thesis Approved:

Thesis Adviser

Dean of the Graduate College

ACKNOWLEDGMENTS

Praise be to God. Because of His Mercy I was finally able to finish this thesis.

I wish to express my sincere thank to my graduate adviser Dr. K. M. George for the guidance, help and time he has given me for the completion of my thesis work. Without his constant support, supervision and ideas, this thesis would have been impossible. I also sincerely thank Dr. J. P. Chandler and Dr. Mitchell L. Neilsen for serving on my graduate committee and providing me with some feedback for improving my thesis.

I would like to acknowledge the financial support I have received from the Indonesian Goverment during my graduate study. This financial support is administered by the Minister for Research and Technology/the Agency for the Assessment and Application of Technology in the framework of the implementation of the Science and Technology for Industrial Development Program.

My respectful and very special thanks to my late father Syambas Hanafi and my mother Mrs. Rolinah Syambas for their love, encouragement, support and confidence on me. And, last but not certainly least, I thank all other members of my family for their love and support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Multimedia systems suggest a wide variety of potential applications. One of the important applications already in use is multimedia electronic mail (multimedia mail). This distributed multimedia application extends the widely accepted electronic mail (e-mail) to include various types of data such as text, still image, graphic, audio, and video.

The most services provided by an electronic mail, even a multimedia mail is "passive" in the sense that data are unidirectional [Bor94]. Mail messages that contain text, image, sound, or video are simply displayed to the user. The less commonly used has been an "active" mail. In the active mail, a message contains a program to be executed when the recipient reads the message. The program, for example can be created to display message in a certain way. Conceptually, the general model for the introduction of computational power in e-mail systems called *Enable Mail* has been proposed in [RB94].

The active mail can be useful to add or improve services in multimedia mail. The scenario service [KGH94], is actually the example of the active mail implementation to improve messaging service. This service provides a scenario in the form of code (program) included in a message to be interpreted when the recipient reads the message.

Kervella *et al.* [KGH94] proposed the scenario service in the multimedia mail system based on X.400 standard which is a CCITT standard for message handling system. The service allows the sender to coordinate the message body parts (the parts of message body) it sends and to adapt the restitution of the message depending on the media the receiver workstation can restore. This is the main element necessary for the good development of the multimedia mail.

We propose a new scheme for scenario service to be implemented in the messaging system that support MIME (Multipurpose Internet Mail Extension) [BF93]. MIME is the format standard for multimedia mail that widely used in the internet community. The Rand Message Handling System (MH) is used as a user agent for this implementation.

We develop a user interface that is capable of composing multimedia message, creating the scenario for that message, sending and presenting the message.

The remainder of this thesis is organized into the following chapters:

- Chapter 2: A Discussion on previous work related to multimedia mail system is presented.
- Chapter 3: The proposed scheme is described and the interface is presented.
- Chapter 4: The implementation is discussed.
- Chapter 5: The results obtained are presented and explained.
- Chapter 6: A summary of the thesis and suggestions for future work are presented.

CHAPTER 2

LITERATURE REVIEW

2.1 Message Structure

The primary goal of the Electronic mail system is to convey messages from one user to another. A message is a single piece of electronic information. It consists of an envelope and a content. The envelope is all delivery information associated with the message. The content is an information object that usually has two components: control information called the *headers*, and data information called the *body*. The message body may consist of one or more message body parts. A diagram that illustrates the message structure is given in Figure 2.1.

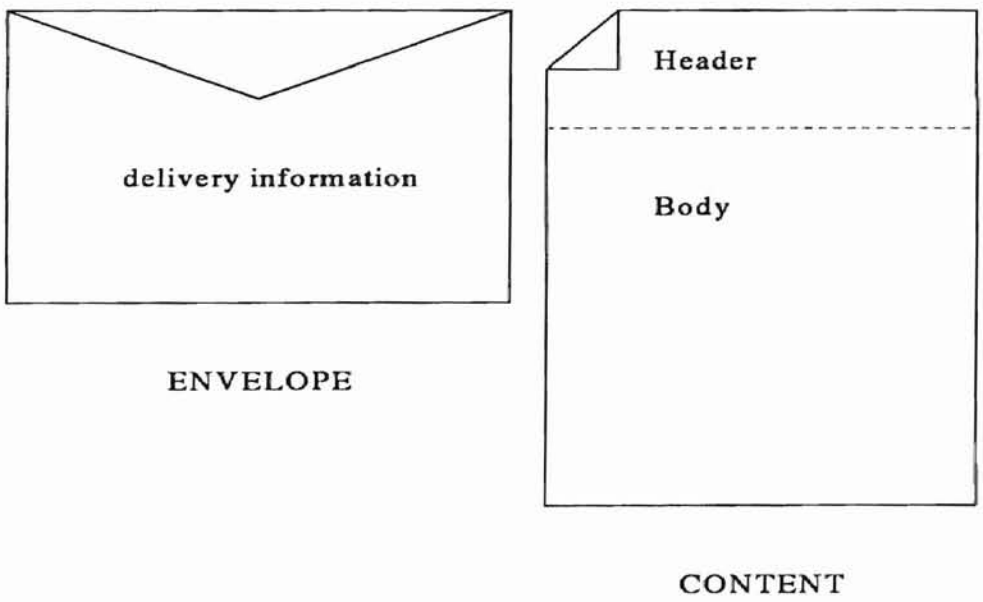

Figure 2.1 The Message Structure

2.2 Message Handling Model

Generally, a model for message handling system is organized into three components [FBS94, RB94]:

- The User Agent (UA)
- The Message Transport Agent (MTA) and
- The Message Transport System (MTS)

Figure 2.2 shows the model.

The User Agent (UA) is responsible for interfacing directly with the end user. A UA may be implemented as a program that can create, send, receive, save, and retrieve messages. The Message Transfer Agent (MTA) provides the routing and relaying of electronic mail. Messages sent from originating UA may be stored temporarily in many intervening MTAs before delivering to recipient UA. A collection of the MTAs is called a Message Transfer System (MTS). MTS is the backbone of the communication systems of the message handling system and it is distributed in nature.

The interaction between the UA and the MTA depends on their physical connection. If both are implemented as processes on the same computer system, then submission and delivery is performed by direct local interaction between the two. An MTA in co-operation with other MTAs transports messages through MTS. The MTA handles service requests from two sources: submission request by a UA, and message received from remote MTA.

As shown in Figure 2.2, the interactions among these three components involve three general protocols :

- A *messaging* protocol used between two UAs.

- A *relaying* protocol used between two MTAs and
- *Submission/delivery* protocol used between a MTA and a UA.

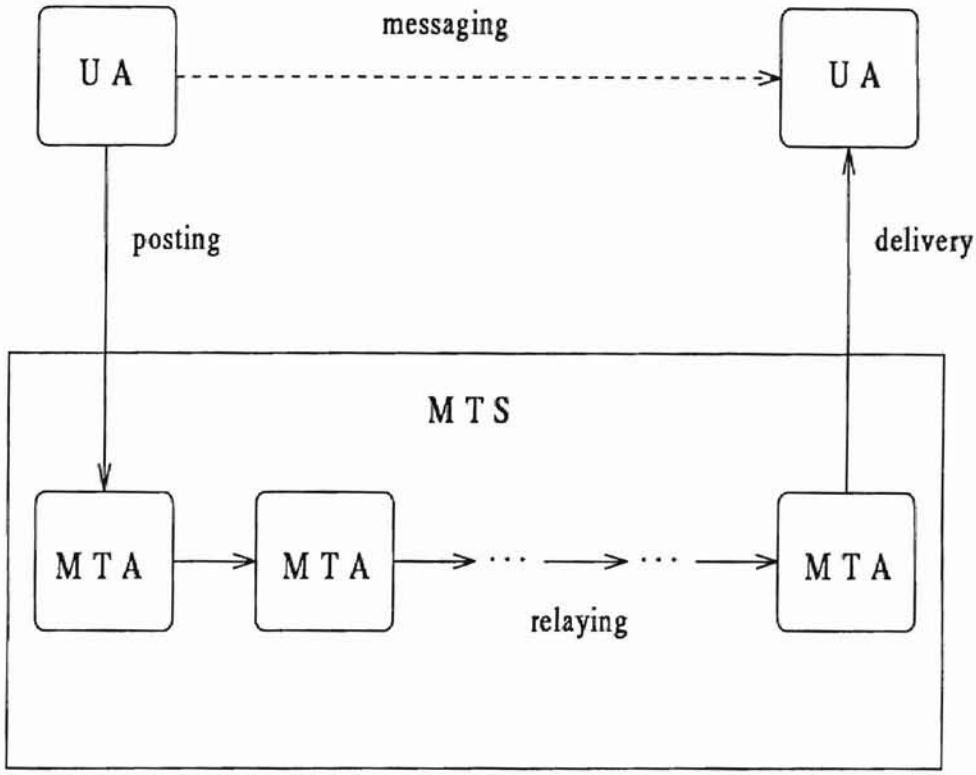

Figure 2.2 The Message Handling Model

In the Internet community, the messaging protocol is defined by RFC 822 [Cro82] and MIME. Both are the standards for Internet message format. MIME is an extension from the RFC 822. The relaying protocol is defined by SMTP (Simple Mail Transfer Protocol) [Pos82]. It is built on the top of the TCP/IP[1] link. SMTP usually operates directly between the source and destination machines, so intermediate machines do not get involved (except for gateways) [Lew95]. Protocols for submission and delivery are considered a local matter, even though it is common to use SMTP

[1] TCP/IP : Transmission Control Protocol/Internet Protocol

for submission and DMSP[2], IMAP[3] versions (IMAP2 and IMAP4), or POP[4] versions (POP2 and POP3) for retrieving messages after delivery processing has occurred.

2.3 Multimedia Mail

2.3.1 Standards

There are two main standards that are widely used for multimedia mail. The first standard is the CCITT X.400, the international standard for mail transport. X.400 recommendation describes a functional model for a Message Handling System (MHS) and its associated services and protocols [FBS94]. The second one is MIME (Multipurpose Internet Mail Extensions) [BF93], a new standard-track Internet format defined by an Internet Engineering Task Force Working Group. It offers a simple standardized way to represent and encode a wide variety of media types, including textual data in non-ASCII character sets, for transmission via internet mail [Bor92]. MIME is an extensions for internet message format RFC 822. Some of the other standards used on different networks are EARN[5], BITNET[6], and UUCP standard.

2.3.2 Characteristics and Requirements

To create a multimedia mail service, there are several characteristics and requirements [KGH94] to be fulfilled. They include :

- A user friendly interface allowing the user to create and restore the multimedia messages it sends and receives.

---

[2] DMSP : Distributed Mail System Protocol

[3] IMAP : Interactive Mail Access Protocol

[4] POP : Post Office Protocol

[5] EARN : European Academic and Research Network

[6] BITNET : Because It's Time Network

- Fast and reliable devices and high storage capacity.

- Reasonable delay in conveying the message to the receiver.

- Required equipment such as high definition screen, audio interface, video interface, and specialized softwares.

- Functionalities requirements including conception, transfer, and restitution. Conception includes creating the content of the message, manipulating the information and creating the parts, and composing the message. Transfer needs to be reliable and be within reasonable delay. The restitution includes reception, preparation, and presentation.

2.4 Active Mail

In the attempt to increase the power and utility of electronic mail systems, a model called *Enabled Mail* has been proposed [RB94]. In general, this model augments the e-mail system by introducing computational power at several key points in the e-mail process [RB94, Bor94]. The key points include three different phases. The first is *delivery time* which occurs immediately before the message crosses the delivery slot. The second is *receipt time* which occurs immediately after the message crosses the delivery slot. And the third is *activation time* which occurs whenever the recipient processes the message. An active mail is enabled mail which occurs in the third phase. In the active mail, a message contains a program to be executed when the recipient reads the message. This certainly will increase the utility and capability of e-mail.

However, the use of active mail is faced with some constraints including security, interface portability, and standardization [Bor94]. Among these problems, the security is the most critical constraint for active mail [BR93]. The use of an arbitrary powerful programming language is not acceptable. It is possible for malicious users to

send e-mail messages that delete the recipient's files or steal confidential information or cause any number of other kinds of mischief. Solution for this problem, according to [Bor94] is simple: avoid any features of the language that can be used to do harm, and then provide a safe subset of the removed functionality. Borenstein and Rose [BR93] proposed a new language based on the Tcl language called Safe-Tcl to be used on active mail, and in general on enabled mail.

2.5 Scenario Service

In most existing UAs, the message body parts (parts of the message body) are presented all together or sequentially. Rather than presenting in such a simple way, Kervella *et al.* [KGH94] proposed an added service called scenario service to provide a better development of multimedia mail. The user can express a scenario which is the notion of timing and ordering of the different body parts. The scenario also provides the substitute body part to adapt the restitution of the message depending on the media the recipient workstation can restore.

Basically, their proposal is the implementation of an active mail for a special purpose (scenario service). Kervella *et al.* also defined a basic language used to create a scenario. The language is actually used as a special purpose programming tool for active mail. It can be used in a user friendly way. We can use a menu that will generate code in this language. The basic scenario language consist of two statement sets: substitution and synchronization sets. The syntax and its explanation is presented in Figure 2.3 and Figure 2.4.

So far, they have proposed integration of the scenario service to the X.400-based multimedia mail architecture. However, they have not implemented the whole ideas

of the scenario service.

Statements	Action
IFNOT<bodypart name1> THEN<bodypart name2>	presents <bodypart name2> in case the UA cannot present <bodypart name1>
IFNOT<bodypart name1> THEN SUBSTITUTE	presents a substitute text in case the UA cannot present <bodypart name1>
<statement>&<statement>	combines statements that are atomic

Figure 2.3 Substitution in the language

Statements	Action
<bodypart name>	presents the <bodypart name>
WAIT_END<statement>	waits up to the end of <statement>
WAIT(<value>,<statement>)	waits the number of seconds indicated in <value> before presenting the <statement>
SAME_START (<statement>,<statement>)	indicates that the <statements> have to begin exactly at the same time
SAME_END (<statement>,<statement>)	indicates that the <statements> have to finish exactly at the same time
SAME_LIMITS (<statement>,<statement>)	indicates that the <statements> have to begin and finish exactly at the same time

Figure 2.4 Synchronization in the language

CHAPTER 3

SCENARIO SERVICE IN MULTIMEDIA MAIL

In this thesis, a new simple scheme is proposed to implement the scenario service for Internet multimedia mail. The scheme is designed based on the flexibility and mechanism for extension offered by MIME standard and scenario service. Scenario service will simply be integrated to the user agent that support MIME. It will be an addition to the user interface.

3.1 Multimedia Mail with MIME

As described in Section 2.2 , the model for a message handling system consists of three components : UA, MTA, and MTS. Such a model is also valid for multimedia mail system. The difference between the ordinary electronic mail and the multimedia mail is in the message format (message protocol) used between two user agents. Message format for multimedia mail must define body parts for various media types.

In the Internet community, the format standard commonly used for multimedia mail is MIME. In order to process a MIME message, a UA may either be facilitated by separated MIME program or has its own MIME functions. MIME program is an implementation of MIME that supports the UA. We will describe MIME in detail below.

As an extension of the internet mail, MIME has improved the limitations of RFC 822 which is a standard format of textual mail messages on the Internet. RFC 822 [Cro82] specifies a syntax for text messages that are sent among computer users. This standard was intended for use in the ARPA Internet. However it has currently been adopted, wholly or partially, well beyond the confines of the Internet. RFC 822 does not specify a format for non-text messages, such as multimedia messages. With this limitations, a new standard, MIME is developed by specifying a format including not

only text but also non-text messages. The text is not limited to the US-ASCII, but it could be the enriched text. MIME standardizes additional fields for mail message headers that describe new types of content and organization for messages [Gra93]. It defines the following new header fields:

- The MIME-Version header field, which uses a version number to declare that a message conforms to the MIME standard.

- The Content-Type header field, which can be used to specify the type and subtype of data in the body of message. MIME defines the following seven kinds of content-type values.

  - The Content-Type value Text, which can be used to represent textual information.
  - The Content-Type value Image, for transmitting still image data.
  - The Content-Type value Audio, for transmitting audio data.
  - The Content-Type value Video, for transmitting video data.
  - The Content-Type value Application, for transmitting application data or binary data.
  - The Content-Type value Multipart, which can be used to combine several body parts into a single message.
  - The Content-Type value Message, for encapsulating an RFC 822 format message.

- The Content-Transfer-Encoding header field, that specifies how the data is encoded to allow it to pass through mail transports.

- Two header fields that can be used to further identify and describe the data in a message body: the Content-ID and Content-Description header fields.

The seven mail content-types are enumerated as the valid ones. The mechanism for extension is to define new subtypes of the established content-types. In general, implementors are required to register new subtypes with the Internet Assigned Numbers Authority (IANA) to avoid name conflicts [Bor92]. However, the use of new subtypes privately using "x-" as the beginning of the letters, may be used without registration. It may be defined between cooperating mail composing and reading programs. When a mail reader finds mail with an unknown content-type value, it will generally treat it as equivalent to application/octet-stream. This subtype is used to indicate that a body contains arbitrary binary data [BF93].

The MIME standard as we described defines all the media types, but it does not define the synchronization (timing and ordering) between these media.

3.2 Integration of Scenario Service to the UA

The scenario service proposed by Kervella *et al.* [KGH94] allows the sender to coordinate the message body parts it sends and to adapt the restitution of the message depending on the media the recipient workstation can restore. So far, they have proposed the scenario to be integrated to the X.400-based multimedia mail system.

We propose the scenario service to be integrated to the UA supporting MIME standard. This would certainly add the missing element in the multimedia mail system i.e. synchronization between media types, particularly in MIME standard. For this purpose we define a MIME extension for scenario language. As we described in Section 3.1, MIME defines the content type called "application". This content type is used for most other kinds of data that do not fit into any of the other categories. Those data could be mail-based information servers and mail-based application languages. The scenario language is defined as the subtype of "application" content-type. The MIME extension for this language is labeled as a MIME body part by the use of the "application/x-scenario".

Since the code will be a part of a single message, the message must be a multipart. The multipart is a content-type value that is used to combine several body parts into a single message. This multipart will have at least two subparts. One of them will be of type application/x-scenario. The other subparts will be of arbitrary types except of type multipart and message.

The proposed scheme that enables the integration of the scenario service is illustrated in figure 3.1.

3.2.1 Reducing The Statements

In the basic scenario language proposed by Kervella *et al.*, there are two sets of statements, one for processing of body part substitution and one for the synchronization of the body parts. However, in our implementation we simplify those statements by reducing into five statements. We also optionally provide a *geometry* indicated by a geometry string */geom* as part of the statement. The string is concatenated to the "bodypart name" delimited by a slash (/). This option is used to resize and/or to position the window in which the media is displayed.

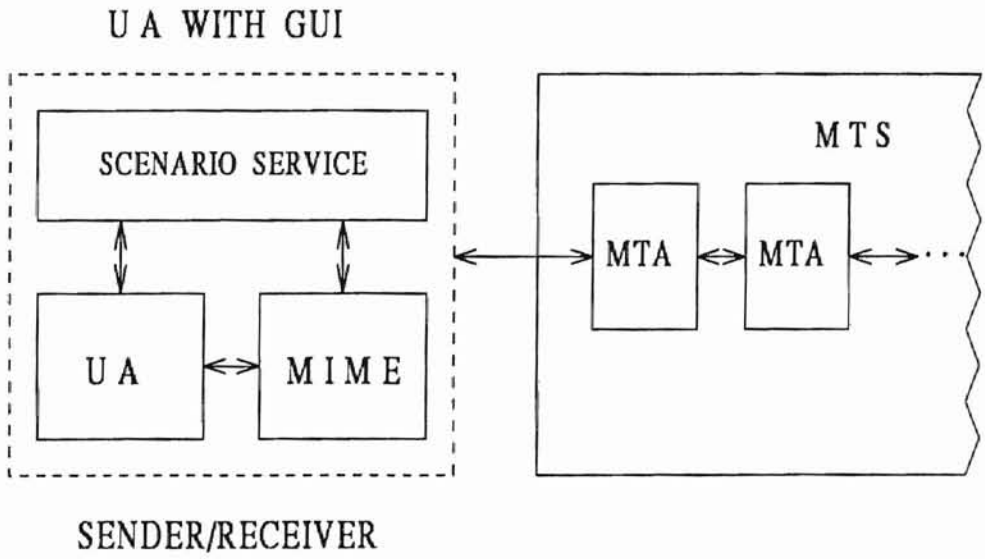

Figure 3.1 The proposed scheme

From the first set of statements, we use two statements:

- IFNOT < *bodypart1*[/*geom*] > THEN < *bodypart2*[/*geom*] > : presents < *bodypart2*[/*geom*] > in case the UA cannot present < *bodypart1*[/*geom*] >

- IFNOT < *bodypart*[/*geom*] > THEN SUBSTITUTION : presents a substitute text in case the UA cannot present < *bodypart*[/*geom*] >

These statements are used to adapt the restitution of the message depending on the media the receiver workstation can restore.

The second set that consists of six statements can be reduced into three statements. We reduce the statements based on the basic components of the synchronization (timing and ordering) i.e. the waiting time, sequential, and parallel ordering. The reducing of these statements will simplify the way to validate the scenario code. It will also avoid the need to modify presentation program (e.g. picture viewer and video player). For example, to interpret the statements such as SAME_END(< *statement* >,< *statement* >) and SAME_LIMITS (< *statement* >, < *statement* >) as shown in figure 2.3, is not easy without modifying presentation programs. We cannot force two or more programs to run the different statements to finish at the same time or to begin and finish at the same time. Even, we probably need to unify these programs as in presenting synchronized video and audio.

These three statements are given below :

- < *bodypart*[/*geom*] > : presents the < *bodypart*[/*geom*] >.

- WAIT(< *value* >,< *bodypart*[/*geom*] >) : waits the number of seconds indicated in < *value* > before presenting < *bodypart*[/*geom*] >.
  If < *bodypart*[/*geom*] > is NULL, nothing presented.

- PAR{< *bodypart*[/*geom*] >|<WAIT(< *value* >,< *bodypart*[/*geom*] >)>;...} : indicates that < *bodyparts*[/*geoms*] > are presented in parallel maybe with the waiting time for certain bodypart name.

With the assumption that the time i.e. the waiting time and the time of presenting a bodypart has been given or determined when creating the scenario, the other statements can be represented by these three statements. However, it must be noted that the statements might not exactly be able to represent the statements proposed by Kervella *et al.*. The timing on presenting a body part frequently depends on the programs, devices, and the size of the body part.

A simple example of the basic scenario code could be as follow:

```
IFNOT video1 THEN photo1/512x512-10+10;
IFNOT voice1 THEN text/+300+300;
PAR{
    video1/+10-30;
    voice1;
    WAIT(10,video2/+400+450);
    photo2/512x420;
}
WAIT(5,NULL);
photo3;
```

From the code, we can see that the user has prepared the body parts he/she wants to send (e.g. video1, photo1, voice1). These body parts will be presented to the user in receiving end based on this code. If the user cannot receive the video, he/she will see a photo1 with the size 512x512 and the position -10+10 on the screen. If he/she cannot receive the voice, he/she will read a text. Otherwise, the video1 and the voice1 will be presented in two ways, first sequentially, then in parallel. In the

latter, he/she will also see a photo2 and then video2 after 10 seconds. After finishing the parallel presentation, 5 seconds later he/she will see a photo3.

3.2.2 Scenario Code

By using the basic language, the scenario can be created in a user friendly way. This will be implemented by using a graphical user interface (GUI) in X-terminal. For executing the code, we need an interpreter. We have developed it using C language.

The scenario code will be sent as a part of the message in the content type value application/x-scenario. As other parts, the code will be encoded before being sent. It is specified by the Content-Transfer-Encoding header field. In the receiving end, the body parts are still in their encoded form. When the message is presented each body part will be decoded to the original form. Usually, the original form will be stored in temporary file before presented. In the case of scenario-based presentation, all body parts including the code will be stored in temporary files. Then, the code will be interpreted to present the body parts which have been in files.

3.2.3 Security Issues

Since the scenario service uses the scheme of an active mail, the security issues must be addressed. In Section 2.4, we briefly described how to overcome this problem, namely by avoiding any features of the language that can be used to do harm.

The basic scenario language is very simple. There is no command that can be used to do harm. Therefore, it is considered "execution safe".

CHAPTER 4

IMPLEMENTATION

4.1 Hardware and Software

In this Implementation we use as workstation a X-terminal connected to a Sequent Symmetry S81. This workstation does not have audio and color video facilities but it is still feasible for our implementation. In the future, if the workstations has been installed with more complete multimedia facilities, it would be no more changes to the scenario service implementation. We only need a modification to the program in order to handle added facilities. We actually, can prepare the program to anticipate these changes.

The Sequent operating system (Dynix/ptx) can provide facilities for concurrent processes that are intensely needed by scenario service. For capturing sound, image, or video clip, we can use PCs that have the capturing facilities. Then, we upload the files to Sequent to be sent in e-mail message. As another alternative, we can use Linux operating system for our implementation.

Since we use UNIX operating system as a platform, we create graphical user interface (GUI) using Tcl/Tk [Ous94] and C language. GUI will allow the user to create and restore the multimedia message in a user friendly manner.

4.2 The MH User Agent

In this implementation, we use the mailing system that have been installed in the Sequent. The user agent available is MH, the Rand Message Handling System. As a user agent, MH is not responsible for delivering mail. It interacts with a message transfer system (MTS) at two interfaces (see the message handling model in Figure 2.1). First, it sends mail by placing it through a posting slot to the MTS. Second,

it receives mail by retrieving it through a delivery slot from the MTS. The other tasks which MH addresses are: the composition (including sending the messages), the reading, and the organization of messages.

The MH supports four different MTS interfaces [RSS93]: SendMail, the standard mailer for 4.2 BSD systems; MMDF and MMDF-II, the Multi-channel Memo Distribution Facility; SMTP, the ARFA Internet Simple Mail Transfer Protocol; and a stand-alone delivery system.

4.2.1 MH Features

MH is different from other user agents. Rather than creating a large subsystem that appears as a single command, it is a collection of separate commands which are run as separate programs [UCI93, RSS93, RS86]. The UNIX shell is the user's interface to MH, so that the MH can use the full power of the shell. In addition, in MH, each message is a UNIX file, and collections of the messages (called folders) are grouped into UNIX directories. In contrast, most other message systems store messages in a complicated data structure within a monolithic file. Using these approaches, MH can make extensive use of other UNIX software.

Basically, there are three main aspects of MH [UCI93]: the way the messages are stored, the user's profile (which directs how certain actions of the message handler take place), and the commands for dealing with messages.

4.2.2 Multimedia Mail with MH

MH can support MIME standard to provide multimedia mail capabilities by patching it to a separate MIME program (i.e. Metamail). However, the new version of MH contains MIME extensions. Most of the functions related to MH's multimedia are implemented in one program called *mhn* [Swe93]. Mhn performs four functions: message composition, message display, message scanning, and message decomposition.

MH does not provide a visual front end for creating MIME messages. We need to type some simple *directives* into the message draft. Each directive "#" indicates a message part and identifies what type of information is to be carried in that part. When mhn performs multimedia message composition, it is used as a simple editor. Mhn can be invoked by a *whatnow* program. When it is invoked, each directive in the message draft causes data from a file or from a command to be inserted. If a file is not specified with the directive, then a command is used to generate the data.

There are four kinds of mhn directives: *type* directives, which name the type and subtype of the content; *external-type* directives, which also name the type and subtype of the content but for external access purpose; the *forw* directives, which are used to forward a digest of messages; and, the *begin* directive, which is used to create a multipart content.

The kind of mhn directive that will be used for the implementation of scenario service is the *type* directives. The following is the syntax of the *type* directive:

```
directive ::=    "#" type "/" subtype
                     0*(";" attribute "=" value)
                     [ "(" comment ")" ]
                     [ "<" id ">" ]
                     [ "[" description "]" ]
                     [ filename ]
                     EOL
```

Type names and attributes used correspond to those used in the MIME specification. The name of file containing the contents in native (decoded) format may optionally be specified by the user. If the filename starts with the | character, then this gives a command whose output is captured accordingly. If a filename is not given, mhn will look for information in the user's profile to determine how the different contents

should be composed. The user may include a brief description of the content. The user also may define the ID, although by default, mhn will generate a unique Content ID. Following is a brief example of what a user's component file might look like:

```
To:
cc:
Subject:
--------
#image/gif [Planet Earth] \
    /home/tahar/MM/lib/earth.gif
#video/mpg [San Fransisco] \
    /home/tahar/MM/lib/frisco.mpg
#application/Postscript [Thesis] \
    /home/tahar/Thesis/thesis.ps
```

In addition to directives, plaintext can be present. Plaintext is gathered, until a directive is found or the draft is exhausted. The syntax for plaintext is as follows:

```
plaintext ::=   [ "Content-Description:"
                      description EOL EOL ]
                  1*line
                [ "#" EOL ]
              | "#<" type "/" subtype
                  0*(";" attribute "=" value)
                  [ "(" comment ")" ]
                  [ "[" description "]" ]
                  EOL
                  1*line
                [ "#" EOL ]
```

```
line      ::=   "##" text EOL
                -- interpreted as "#"text EOL
              | text EOL
```

By default plaintext is captured as a text/plain content. However, we can override this by starting the plaintex with #< followed by a content type specification.

To perform other functions that manipulate MIME messages, the mhn program can be invoked directly from the shell using some kinds of options. Three main options are:

- The -show option displays the contents (the message parts);
- The -list option enumerates the contents; and
- The -store option stores the contents into files.

4.3 Implementation

This section describes the implementation of our scheme to add the new scenario service to the multimedia mail provided by MH. The scenario service is integrated to the MH by creating a graphical user interface using Tcl/Tk on X-terminal. The interface handles three tasks: composition, restitution, and organization of messages in a user friendly way.

As noted earlier, MH does not provide a visual front end for creating MIME messages. The interface designed performs this function for MH. Basically, the interface in some parts substitutes the functions of mhn's simple editor, particularly for typing the directives. The user does not need to type the directives to a message draft directly. The component file will be created when the user visually composes the message parts.

In the implementation, we only need the type directive and/or plaintext for composing a multimedia message. So, the body of the draft will be formatted with the following syntax :

```
body    ::=  1*(content | EOL)
content ::=  directive | plaintext
```

The directive is the type directive as described in Section 4.2.2. The plaintext is used by default. The body contains one or more contents (message parts). A content consists of either a type directive or plaintex. The filename will be specified when the user decides to attach the file as a part of message.

In this scheme, a scenario will be written after the contents (which will be present based on the scenario) have been chosen and become the parts of the message. Interactively, by using graphical interface, the user can create scenario, while he/she can also visually order the parts presented on the screen. The user can select and determine what parts will be presented sequentially or concurrently, when this will occur, and where the parts will be positioned on the screen and in what size. In order to adapt the restitution of the message depending on the media, a receiver can use the conditional statement IFNOT-THEN.

After creating a scenario, the code will be stored in a temporary file. This file has an extension name "sce". Before "attaching" the file name to the user's component file, the scenario can be evaluated by executing it. For this purpose a code interpreter will be invoked to run the code. If it is not appropriate the user can recreate a new scenario. A type directive which identifies the file name must be defined. Mhn provides a mechanism for extensions which allows to define a new subtype in the directive.

As described in Section 3.2.2, the basic scheme proposed is by defining a new content-type labeled application/x-scenario for scenario language. In the implemen-

tation, this new content-type will be a type directive with the "type" and "subtype" are respectively "application" and "x-scenario". An example of the user's component file containing type directive application/x-scenario may look as follow:

```
To:
cc:
Subject:
--------
#image/gif [Planet Earth] \
    /home/tahar/MM/lib/earth.gif
#video/mpg [San Fransisco] \
    /home/tahar/MM/lib/frisco.mpg
#video/mpg [New York] \
    /home/tahar/MM/lib/newyork.mpg
#image/jpg [Planet Jupiter] \
    /home/tahar/MM/lib/jupiter.jpg
#application/x-scenario [Scenario Code] \
    /home/tahar/Research/code.sce
```

After the composition has been performed, the next step is to send the message or just store the message draft in a folder. The simple idea to send message using interface is by invoking a new xterm. This is carried out simultaneously with invoking MH's program for sending message, on the same command line. The MH's program used is "comp" with the option -form whose argument is the name of component file or draft. By invoking comp program, we actually open a simple editor (prompter). However, for our purpose, we only use it to type destination address, carbon copy (cc), subject, and probably to write the initial text. The main purpose is to send message based on the draft that has been created, to the destination address.

Another task handled by the user interface is the restitution of messages. The main element of the restitution is message presentation. This is performed in two ways: the ordinary and the scenario-based presentations. In the Section 4.2.2 we described some mhn functions to manipulate multimedia messages. These functions are used by both kinds of presentation. To check whether or not a message contains scenario code, an mhn can be invoked to search for the subtype labeled x-scenario.

The important element in performing scenario-based presentation is the code interpreter. This program written in C language, performs three tasks:

- to check validity of the code;
- to check whether or not the presentation facilities (e.g. picture viewer or video player) are available in workstation. This is carried out by checking the user's profile (mh or mhn profile) and
- to interpret the code to be executed.

The parts of the message will be stored in the temporary files to be read by the program. Presentation of each part based on the code is carried out by creating a child process, in which a presentation program will be executed. If the presentation program is not available, the program will present a message window. This can occur when interpreting the conditional statement. The interpreter is also used when evaluating the scenario just created.

Finally, the user interface also performs the organization of the messages. MH programs perform these tasks, while the interface provide to the user interactively.

CHAPTER 5

RESULTS

The main window of our graphical user interface for the multimedia mail system is shown in Figure 5.1. This interface gives the user an easy way to interact with messaging functions. The principal functions such as composing, sending, and reading messages can be performed easily using the interface. The interface is user friendly as required in a multimedia mail system.

Figure 5.2 shows other windows giving composition functions. Visually, the user can select the files to be "attached" into a message body. In fact, the user was creating a component file (message draft) containing some type directives. All the selected files, the component form, and the body parts can be checked before ending this operation as shown in Figure 5.3. How the form of a component file is given is shown in Figure 5.4. The interface provides a windowing visual front end to the user in manipulating multimedia messages.

Other windows shown in Figures 5.5, 5.6, 5.7, and 5.8 are interfaces given to create a scenario code. All parts of the message were presented to help the user to create a scenario for that message easily. Code is generated interactively through the window interface. After creating a scenario, the code can be shown as given in Figure 5.9.

Figure 5.10 shows how the message draft (the user's component file) is sent. A program called xterm will be invoked to display prompter (the simple MH editor) in which the destination address will be typed.

Finally, Figure 5.11 displays an example of scenario-based presentation. The message presentation is performed on window by clicking Run button. It can be repeated as many times as one wishes to view.

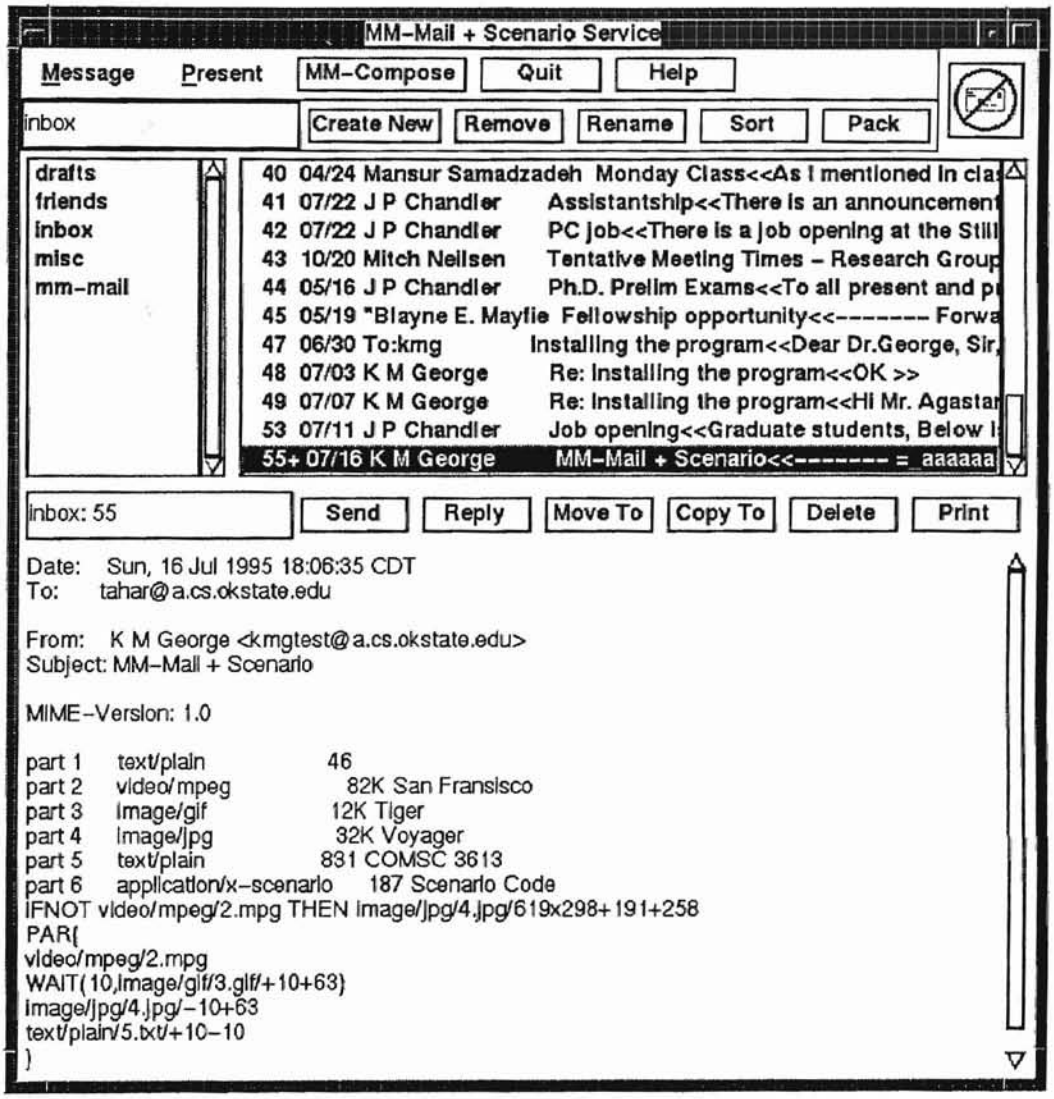

Figure 5.1 Main window

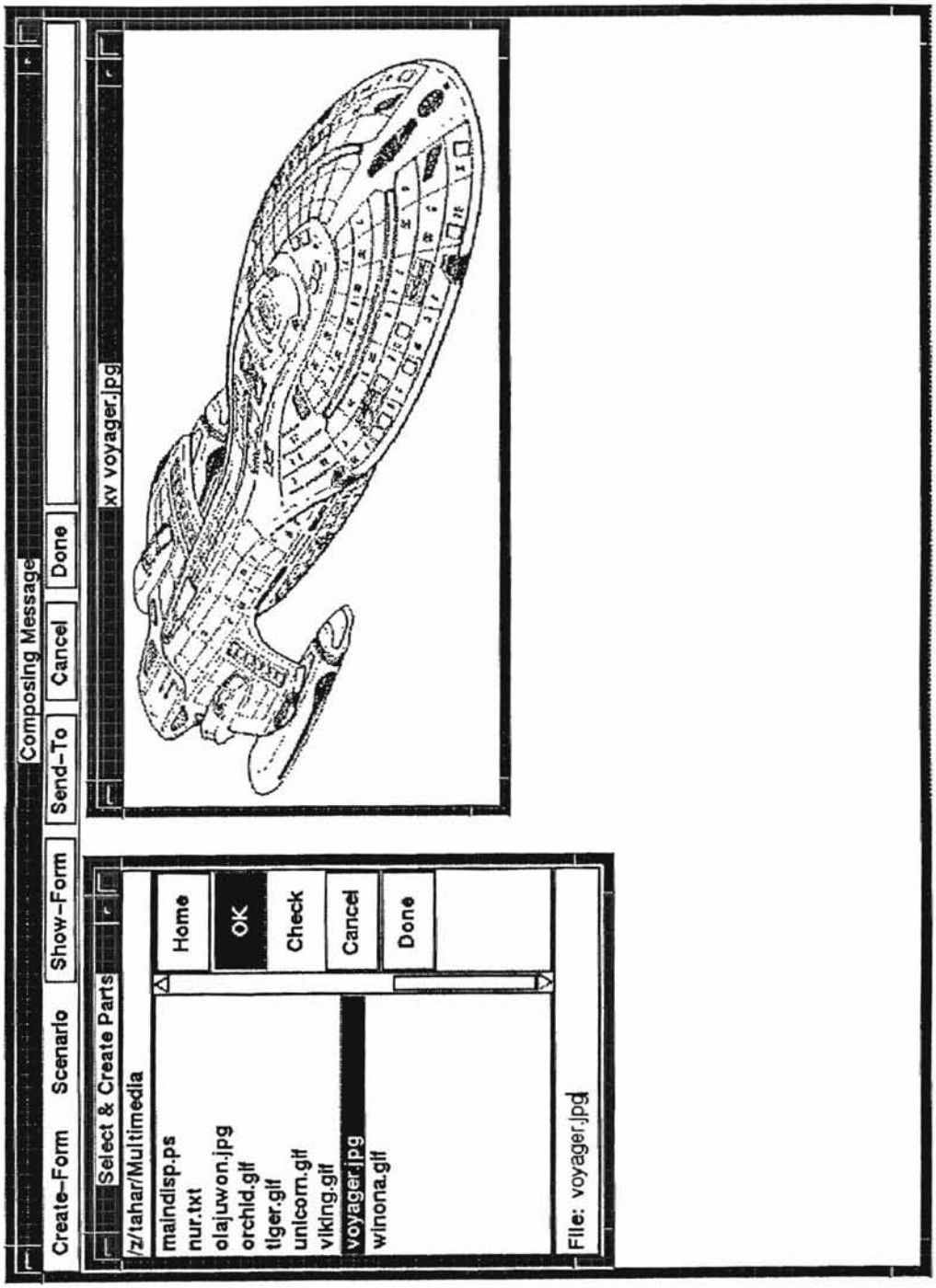

Figure 5.2 File selection

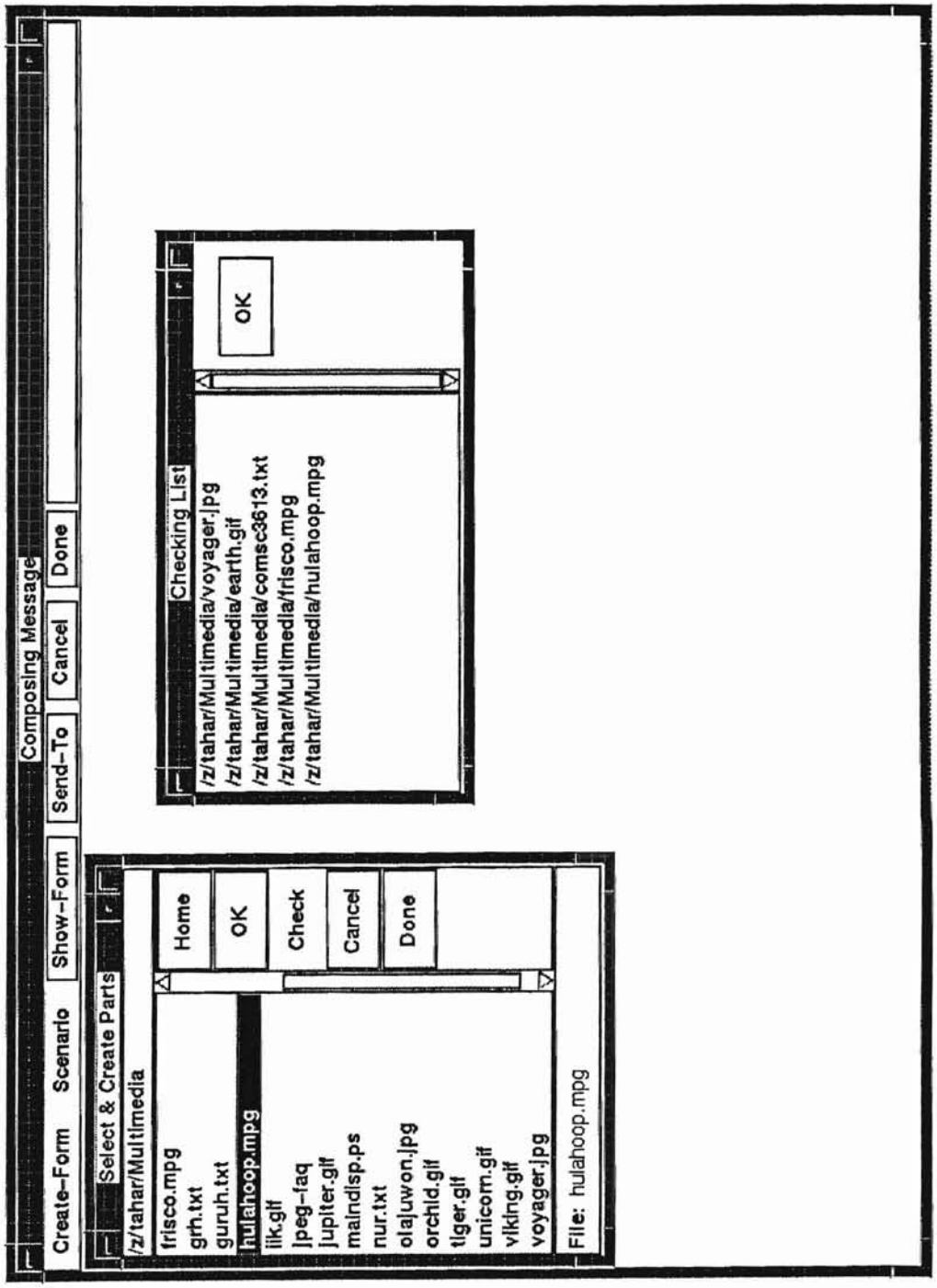

Figure 5.3 Checking the selected files

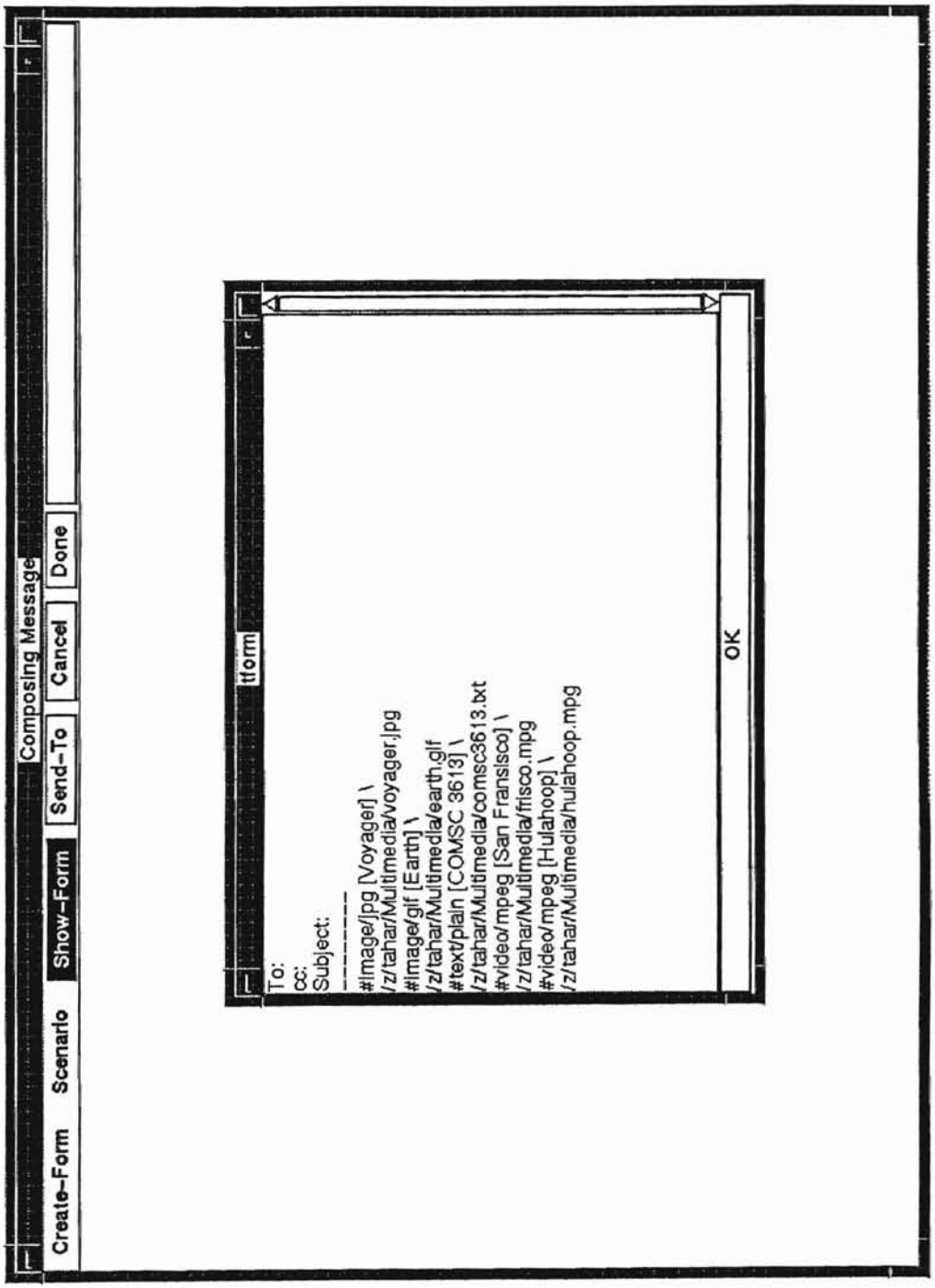

Figure 5.4 Showing the component form

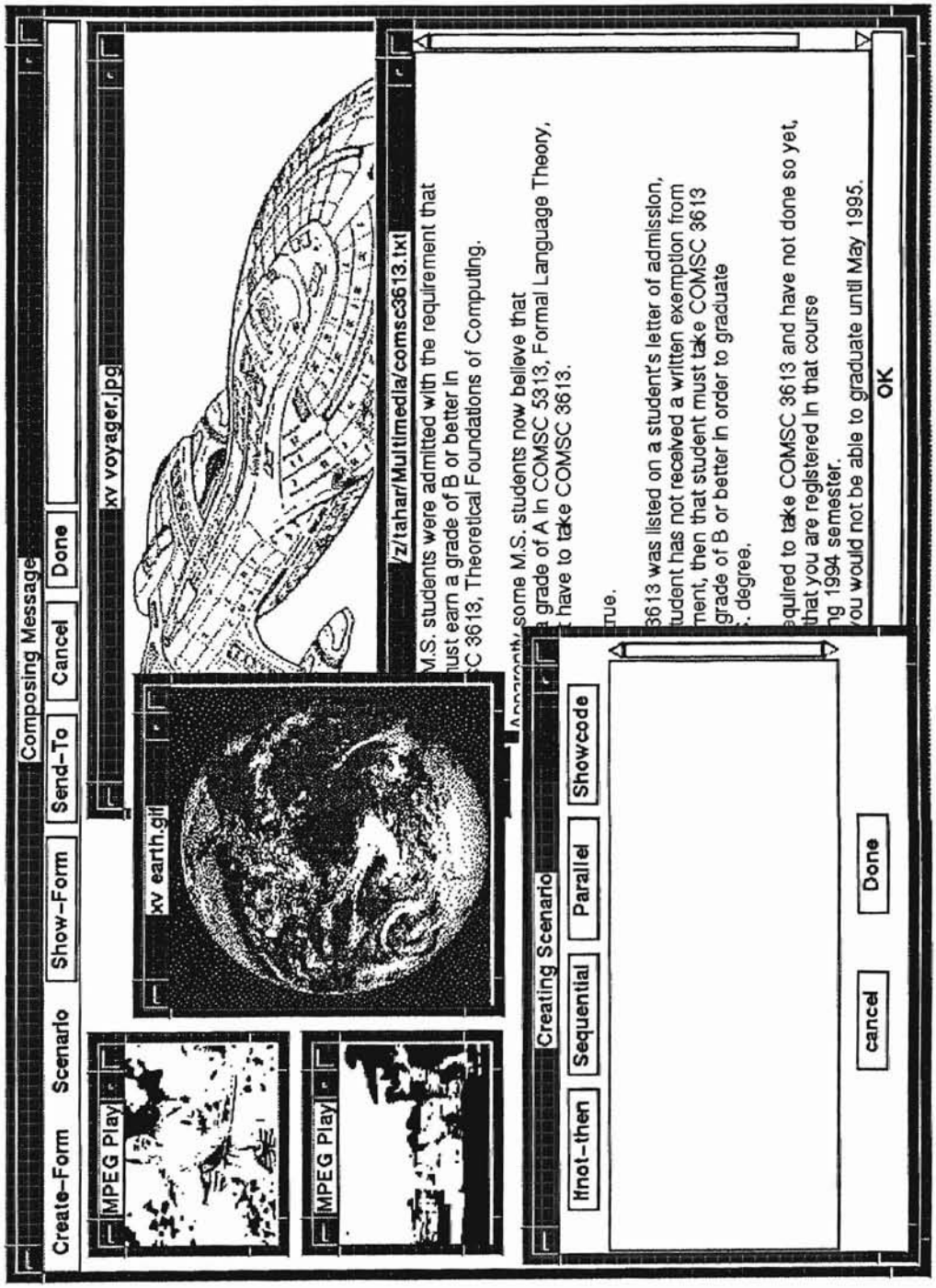

Figure 5.5 Scenario window

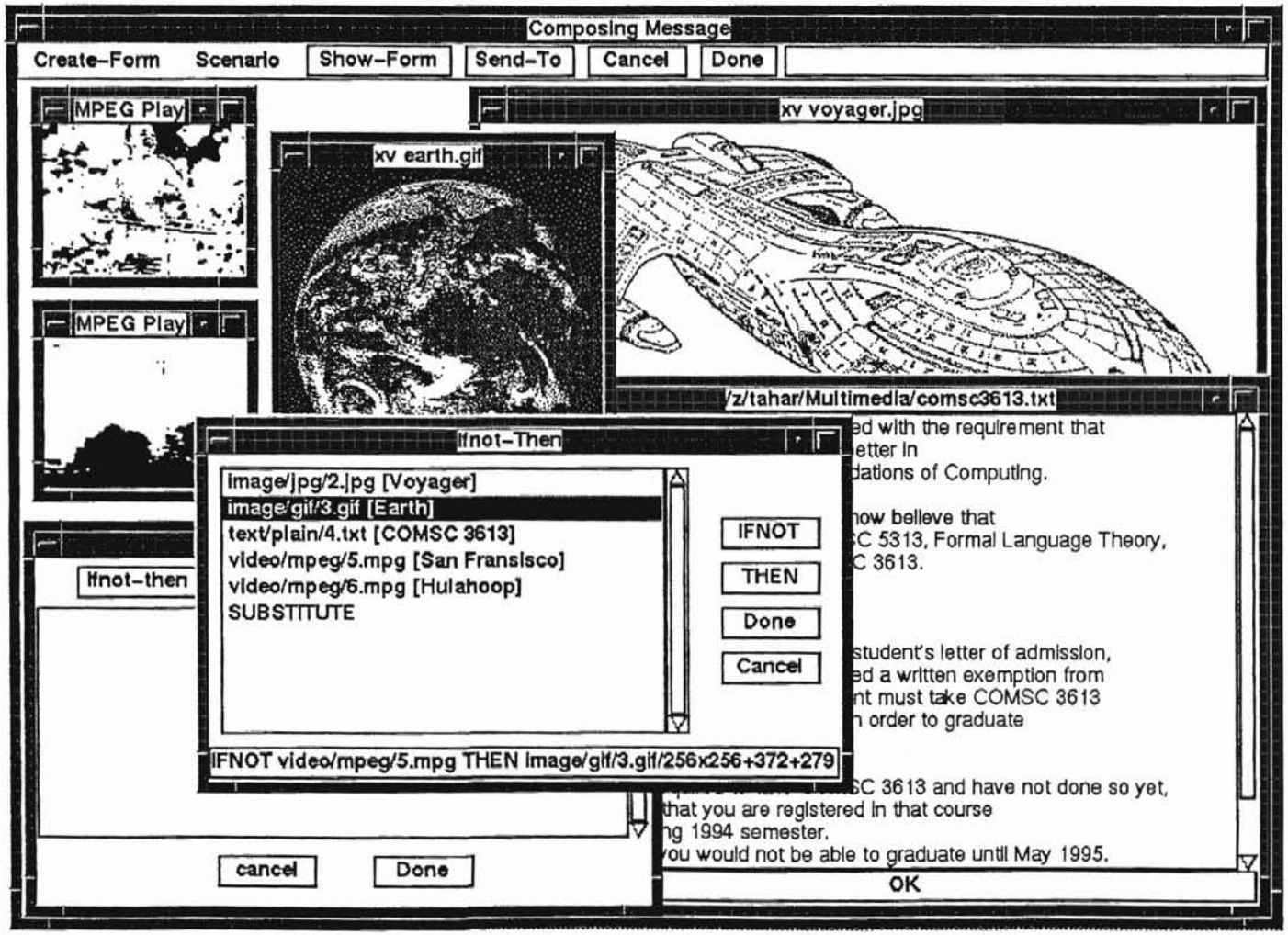

Figure 5.6 Ifnot-Then window

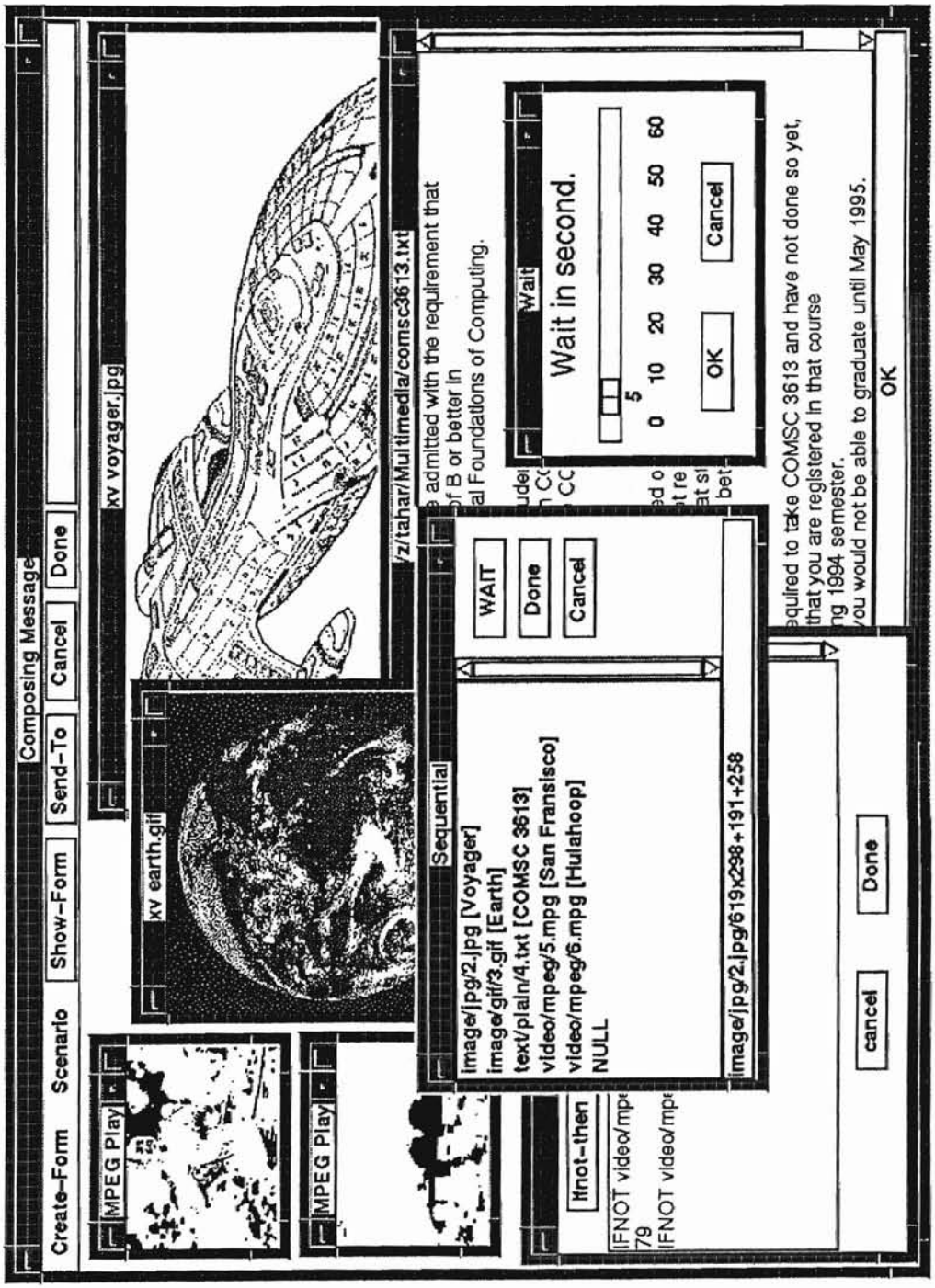

Figure 5.7 Sequential and Wait windows

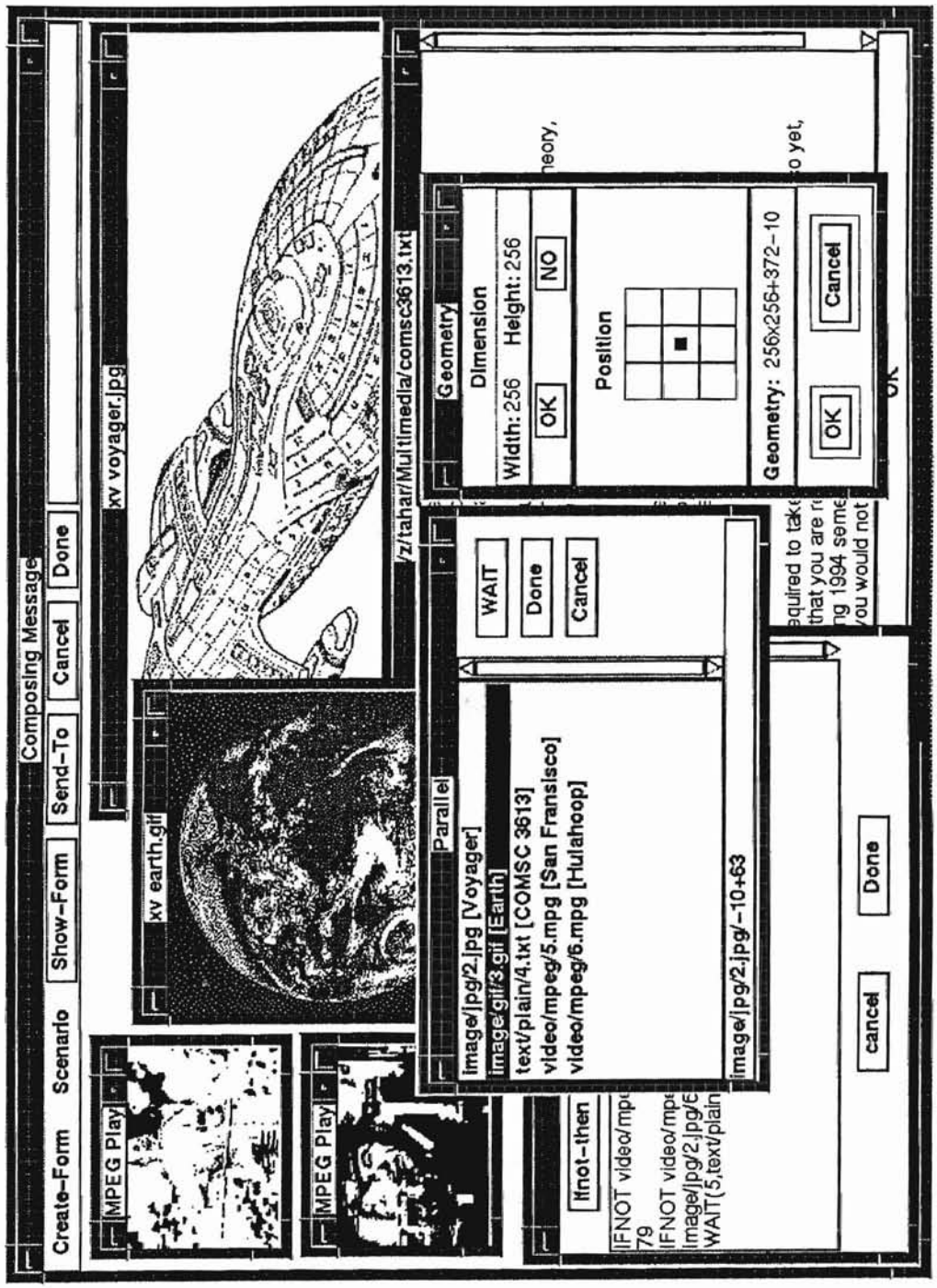

Figure 5.8 Parallel and Geometry windows

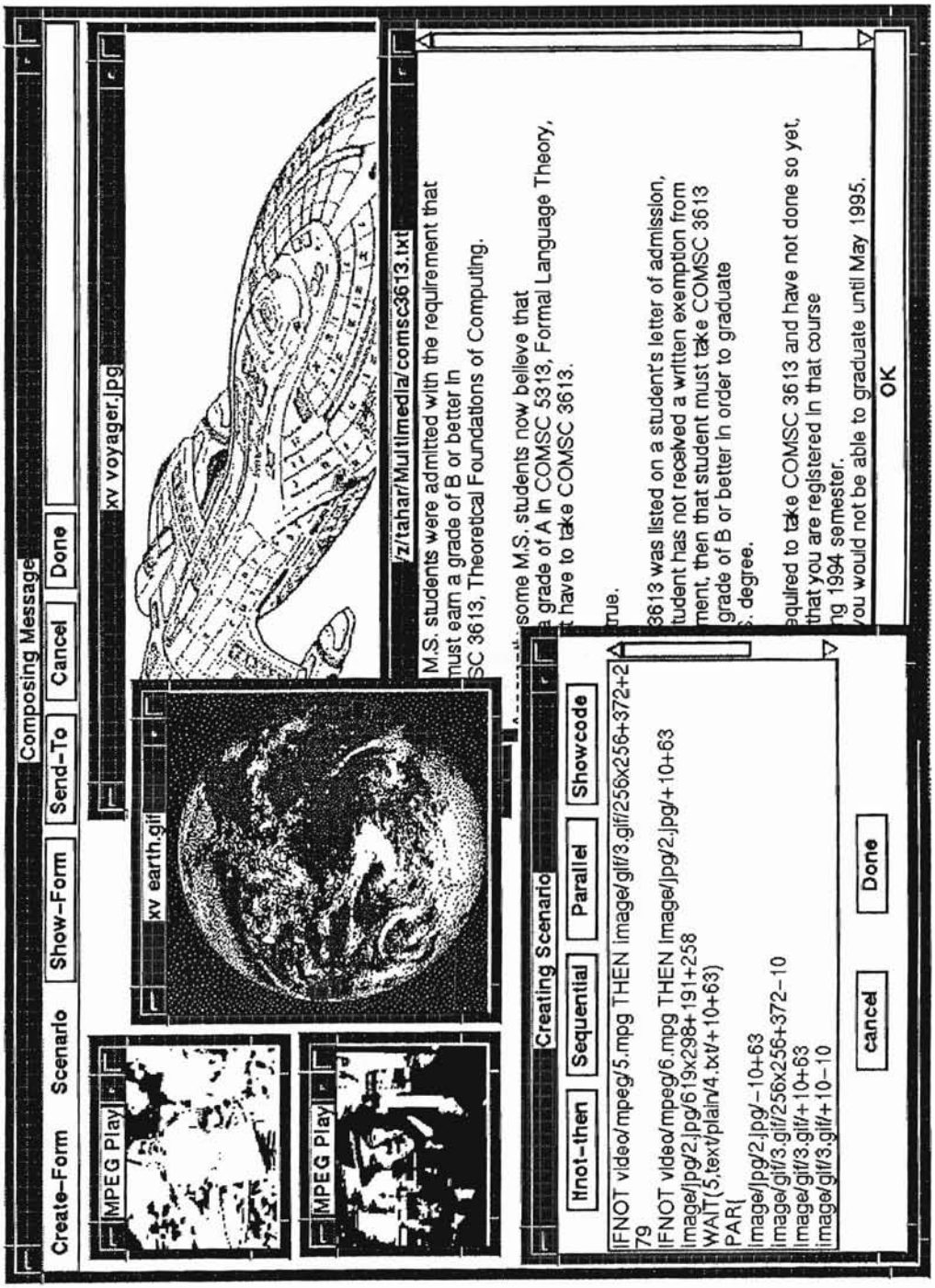

Figure 5.9 Showing the code

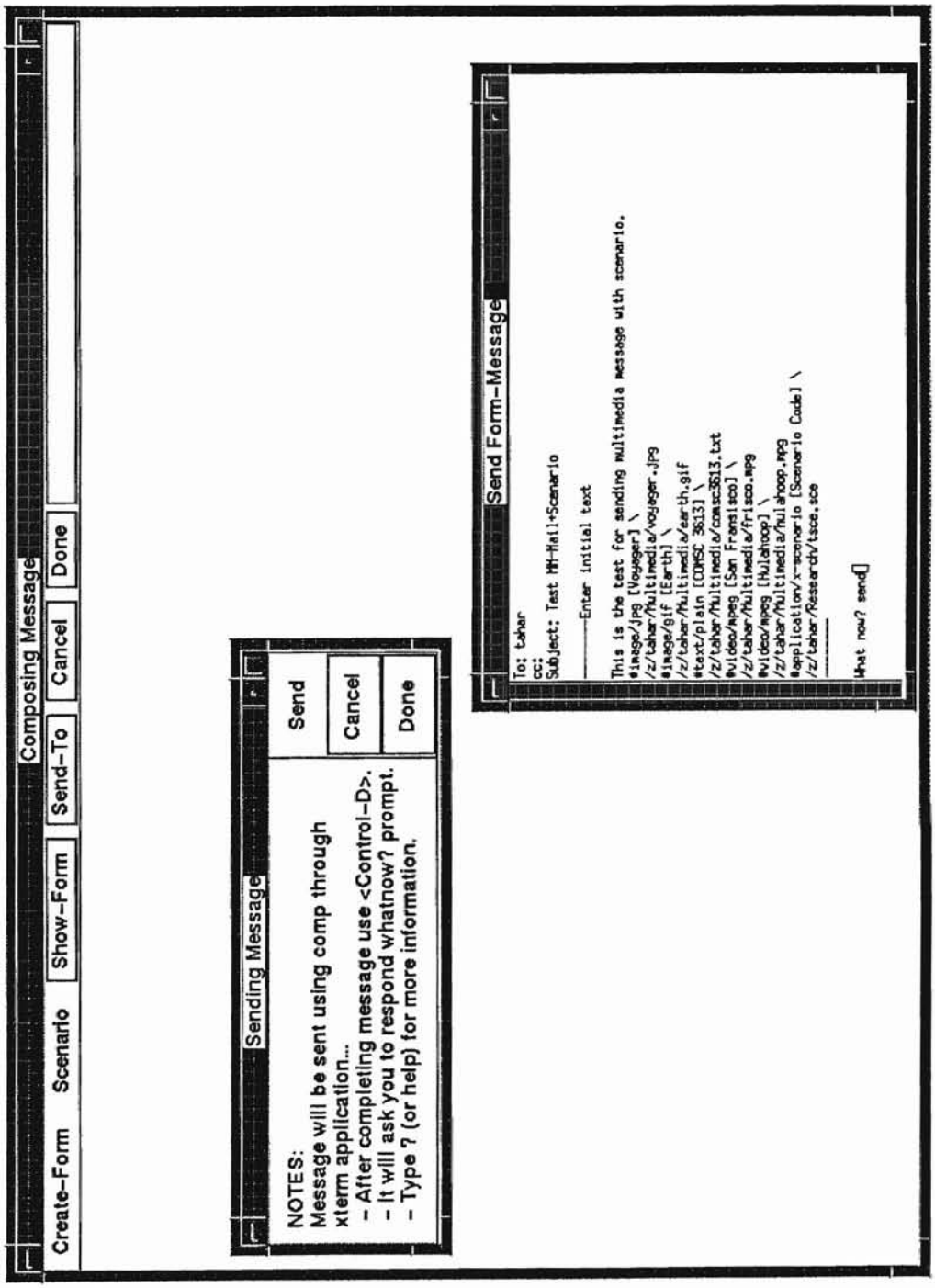

Figure 5.10 Sending the message draft

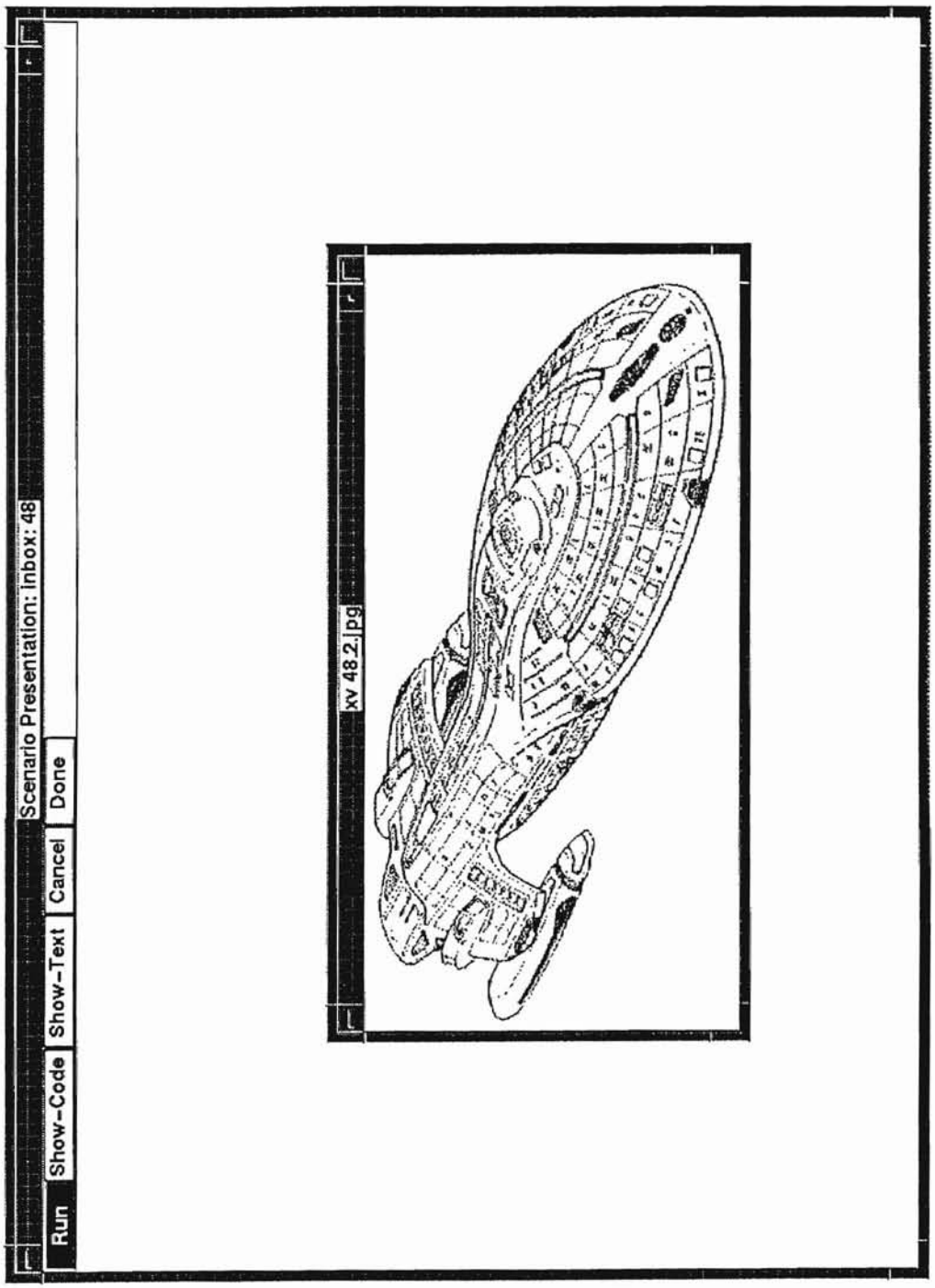

Figure 5.11 Scenario Presentation window

CHAPTER 6

CONCLUSION

6.1 Summary

An implementation providing a scenario service to a multimedia mail system and a GUI are presented in this thesis. This implementation provides a better development to multimedia mail. This is achieved by using a simple scheme based on extension mechanisms of MIME message format. The implementation uses the Rand Message Handling System MH as a user agent which supports MIME. The interface developed allows the user to create and restore the multimedia messages it sends and receives as required in a multimedia mail service. The scenario service improves the capability of the user agent in presenting a multimedia message.

6.2 Future Work

The implementation performs good features and fulfills requirements for a good multimedia messaging system. However, there are some other features and another scheme that could be developed for the future works. These features and scheme are listed below:

- Adding geometry option to each presentation program so that the display position can be defined by the user.
- Modifying presentation program to be able to terminate in the time defined by the user, particularly for picture viewer.
- Displaying the media (parts of message) on the same screen.
- Develop a better interface for sending multimedia message.

- Develop a new scheme, which could interpret scenario to manipulate the body parts without storing them in temporary files.

BIBLIOGRAPHY

[BF93] Nathaniel S. Borenstein and Ned Freed. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanism for Specifying and Describing the Format of Internet Message Bodies. RFC 1521, Bellcore, Innosoft, September, 1993.

[Bor92] Nathaniel S. Borenstein. Internet Multimedia Mail with MIME: Emerging Standards for Interoperability. A Copy of a paper from The ULPAA'92 Conference in Vancouver, May, 1992.

[Bor94] Nathaniel S. Borenstein. EMail With a Mind of Its Own: The Safe-Tcl Language for Enabled Mail. A Copy of a paper from The ULPAA'94 Conference in Vancouver, May, 1994.

[BR93] Nathaniel Borenstein and Marshall T. Rose. MIME Extensions for Mail-Enabled Applications: application/Safe-Tcl and multipart/enabled-mail, Working Draft, November, 1993.

[Cro82] David H. Crocker. Standard for The Format of ARPA Internet Text Messages, STD 11, RFC 822, UDEL, August 13, 1982.

[FBS94] Sualeh Fatehi, Shoeb Bhinderwala, and Khrisnan G S. An Overview of X.400 and Related Recommendations. Paper from ftp site, June, 1994.

[Gra93] Mark Grand. MIME Overview. Paper from ftp site, October, 1993.

[KGH94] B Kervella, V Gay, and E Horlait. Integration of a Scenario Service in a Multimedia Messaging System. In *Proceedings of the IASTED/ISMM: Dis-*

tributed Multimedia Systems and Applications, Honolulu, Hawaii, August 1994.

[Lew95] Christ Lewis. Unix Email Software: A Survey. FAQ from newsgroup comp.mail.misc, January 26, 1995.

[Ous94] J. Ousterhout. *Tcl and The Tk Toolkit.* Addison-Wesley, Reading Massachusetts, 1994.

[Pos82] J.B. Postel. Simple Mail Transfer Protocol, RFC 821, USC/Information Sciences Institute, August, 1982.

[RB94] Marshall T. Rose and Nathaniel Borenstein. A Model for Enabled Mail (EM), Working Draft, July 19, 1994.

[RS86] Marshall T. Rose and Jerry N. Sweet. A Rand MH Message Handling System: Tutorial, May 21, 1986.

[RSS93] Marshall T. Rose, Einar A. Stefferud, and Jerry N. Sweet. A Multifarious User Agent, Working Draft, July 19, 1993.

[Swe93] Jerry N. Sweet. A Multimedia E-Mail Tutorial with MH, March 17, 1993.

[UCI93] UCI. The Rand MH Message Handling System: User's Manual, November 30, 1993.

APPENDIX A

USING THE SYSTEM

A.1 System Overview

The main program is the Tcl/Tk script for performing the user interface. This script consists of a number of procedures which are interpreted by an interpreter. Another Tcl/Tk script has special functions for creating scenario code.

The interpreter includes two programs: interpreter used when presenting message (showsce.c) and interpreter for testing the scenario code when composing message (testsce.c). Both are written in C language. Another program written in C is the program for presenting concurrently all parts of message when composing message. This program also executes program for creating scenario code which is written in Tcl/Tk script. Such program is named showpar.c. These programs need the data from MH profile (.mh_profile) to check all presentation facilities available in workstation.

To execute the program, MH.6.8 (or versions that support MIME) and Tcl/Tk must have been installed in the system.

A.2 Main Window

The main window or the MM-Mail+Scenario Service window is displayed when the program (tksce) starts. This window consists of two menus, two lists, one text window and some buttons. We will describe them briefly.

- The **Message** menu, displays operations which include compose new message, send, reply, delete, and print the message(s).

- The **Present** menu, provides selection. A message may be presented ordinarily or by using scenario-based presentation.

- The **Folder list**, the list on the left shows all current folders. Double clicking the mouse button-1 on one, opens the folder and displays the contents on the message list.

- The **Message list**, the list on the right shows the contents of a folders. When the tksce starts, initially it shows the contents of inbox (a special folder in which the message(s) are stored when incorporated). Double clicking the mouse button-1 on one, shows the message on the text window or pop up the scenario presentation window.

- The **text window**, shows the header and the body of message.

The main window consists of 14 buttons. The name of the button and its functions is presented in Figure A.1.

A.3 MM-Compose Window

The MM-compose window shows the working area where we compose a multimedia message and create scenario for it.

- The **Create-Form** menu, gives an operation to create a user's component form. We can select the type of the form either "send" or "reply". The directive of the form is of the "type" directive. By using a browser, we select a file to be displayed (presented) and then attach it to the form.

- The **Scenario** menu, performs some operations to create, show, test, and attach the scenario code. We will describe the Scenario menu later in detail.

- The **Show-Form** button, shows the contents of the component form.

- The **Send-To** button lets us send the composed message by creating xterm application.

- The **Cancel** button, cancels this session.

- The **Done** button, ends this session.

Buttons	Functions
MM-Compose	To bring up the MM-Compose window.
Quit	To quit from the program.
Incorporate (bitmap)	Disable if there is no mail. Flash as soon as mail arrives and be enabled. Single click on this button will incorporate the incoming mail into inbox.
Create New	To create a new folder.
Remove	To remove a folder
Rename	To rename an existing folder.
Sort	To sort the contents of a folder by date.
Pack	To pack the contents of a folder
Send	To send a message using an Xterm application.
Reply	To reply a selected message using an Xterm appl.
Move-To	To move one or more selected message(s) from one folder to another folder.
Copy-To	To copy one or more selected message(s) from one folder to another folder.
Delete	To delete one or more selected message(s).
Print	To print one or more selected message(s).

Figure A.1 The buttons in main window

A.3.1 The Scenario Menu

To create a scenario for a message, click on create menu to display a scenario window and all the parts of the composed message. So, while creating the scenario, we can see

the parts of the message. The window shows six buttons and one text window. **Show-Code** button shows the code that has been created on the text window. **Cancel** button cancels all activities in this session and **Done** button ends this session.

The other three buttons: **Ifnot-Then, Sequential,** and **Parallel** perform the main role in creating the code. Single clicking on each button will display a window. Through this window, we create scenario code interactively. Each window shows the list of message body parts being composed. We can select one by double clicking mouse button-1. Every selection on the list will be followed by showing geometry window. Each statement just created will be displayed. We can cancel or accept the code using cancel or done button respectively. We will describe these windows (plus geometry and wait windows) below.

1. The Ifnot-Then Window

   To create one statement:

   Click on IFNOT button followed by selection of the part, then click THEN button followed also by selection on the list.

2. The Sequential Window

   To create statements can be performed by two ways:

   (a) Select the part on the list.

   (b) Click WAIT button to determine the number of seconds, then select the part.

3. The Parallel Window

   To create statements follow the steps as in the Sequential window. The result will be the code of parallel statements.

4. The Geometry Window

```
The window shows dimension entries (width and height), the default
positions, and geometry (dimension and position) entry.
We can type the geometry string directly or get it from dimension
entries and/or default positions.

5. The Wait Window
To select a value, hold the mouse button-1 on scale and slide
the mouse. Release the button on the selected value.
```

After creating the code, from the scenario menu we can examine the code. We can also evaluate (test) the code to see how it works. If it is appropriate, we can attach the code to the component form. If it is not we can recreate the new scenario.

A.4 Scenario Presentation Window

The Scenario Presentation Window will be displayed when we view a message containing scenario code using scenario-based presentation. To show the message, we click mouse button-1 double on the selected message. The window shows five buttons which have functions as follows (click on the button to activate operation):

- The **Run** button let us to run scenario-based presentation.
- The **Show-Code** button shows the scenario code.
- The **Show-Text** button shows the initial text of the message if it exists.
- The **Cancel** button cancels the session.
- The **Done** button ends the session.

APPENDIX B

THE MAIN SCRIPT

The main script of the program and the Makefile are given in this section.

```
#
# Makefile for tksce
#
# Change these three variables to point to the directories where wish can
# be found, and where you want to install the tksce library and executable.
# To complete the configuration, you will also need to edit the file
# tksce.rc, and set the commands to use and the bindir for MH.
#

WISH = /contrib/bin/wish
LIBDIR = /z/tahar/contrib/lib
BINDIR = /z/tahar/contrib/bin

#
# Do not change anything below this, Tksce might not work correctly
# if you do.
#

TKSCE_LIB = $(LIBDIR)/tksce
TKSCE_BITMAP = $(TKSCE_LIB)/BM

TCLSRC = compose.tcl message.tcl present.tcl scenario.tcl \
         misc.tcl config.tcl main.tcl

PROG = showsce testsce showpar showcode showtxt msgW

TKSCE_RC = dot_tksce dot_mh tksce.rc help

# Rules

all: clean tksce scen

install: all instlib index instbitmaps
@echo Installing Tksce executable in $(BINDIR)
@cp tksce $(BINDIR)
@chmod 755 $(BINDIR)/tksce
@echo Installing Scen program in $(TKSCE_LIB)
@cp scen $(TKSCE_LIB)
@rm scen
@chmod 755 $(TKSCE_LIB)/scen

instlib:
@rm -rf $(TKSCE_LIB)
@mkdir $(TKSCE_LIB)
```

```
@chmod 755 $(TKSCE_LIB)
@echo Installing library in $(TKSCE_LIB)
@for i in $(TCLSRC) $(TKSCE_RC); \
do \
  cp ./lib/$$i $(TKSCE_LIB); \
  chmod 644 $(TKSCE_LIB)/$$i; \
done
@for i in $(PROG); \
do \
  cp ./lib/$$i $(TKSCE_LIB); \
  chmod 755 $(TKSCE_LIB)/$$i; \
done

instbitmaps:
@echo Installing bitmaps in $(TKSCE_BITMAP)
@mkdir $(TKSCE_BITMAP)
@chmod 755 $(TKSCE_BITMAP)
@cp -r ./lib/BM/* $(TKSCE_BITMAP)

tksce:
@echo Creating Tksce
@echo \#!$(WISH) -f >tksce
@echo set tksce_lib $(TKSCE_LIB) >>tksce
@echo lappend auto_path \$$tksce_lib >>tksce
@echo \# >>tksce
@echo \# Main program MM-Mail + Scenario Service >>tksce
@echo \# >>tksce
@echo InitTksce >>tksce
@echo MainWin >>tksce
@echo Loop >>tksce
@echo "" >>tksce
@chmod 755 tksce

scen:
@echo Creating Scen
@echo \#!$(WISH) -f >scen
@echo set tksce_lib $(TKSCE_LIB) >>scen
@echo lappend auto_path \$$tksce_lib >>scen
@echo \# >>scen
@echo \# Program for Creating Scenario >>scen
@echo \# >>scen
@echo InitScen >>scen
@echo ScenWin >>scen
@echo "" >>scen
@chmod 755 scen
```

```
index: $(TKSCELIB)
@echo Creating tclIndex in $(TKSCE_LIB)
@echo "auto_mkindex $(TKSCE_LIB) *.tcl; exit" | $(WISH)

test:
@echo Creating test
@echo \#!$(WISH) -f >tksce.test
@echo set tksce_lib [pwd]/lib >>tksce.test
@echo lappend auto_path \$$tksce_lib >>tksce.test
@echo InitTksce >>tksce.test
@echo MainWin >>tksce.test
@echo Loop >>tksce.test
@chmod 755 tksce.test
@echo Running tksce test
@./tksce.test
@rm tksce.test

clean:
@rm -f tksce

uninstall: clean
@rm -rf $(TKSCE_LIB) $(BINDIR)/tksce

#--------------------------------------------------------------------#
#!/contrib/bin/wish -f
set tksce_lib /z/tahar/contrib/lib/tksce
lappend auto_path $tksce_lib
#
# Main program MM-Mail + Scenario Service
#
InitTksce
MainWin
Loop

# All global variables used in program

# Bitmaps
   set Bitmap(nomail) "$tksce_lib/BM/noletters.xbm"
   set Bitmap(mail) "$tksce_lib/BM/letters.xbm"
# MH environment
   set MH(libdir) "/local/lib/mh"
   set MH(bindir) "/local/bin"
   set MH(Path) "Mail"
   set MH(Inbox) "inbox"
   set MH(Draft-Folder) "drafts"
   set MH(MailDrop) $env(MAIL)
```

```
# Fonts
   set Font(helvb) {-*-helvetica-bold-r-*-*-*-120-*-*-*-*-*-*}
   set Font(helvm) {-*-helvetica-medium-r-*-*-*-120-*-*-*-*-*-*}
   set Font(helvmo) {-*-helvetica-medium-o-*-*-*-120-*-*-*-*-*-*}
   set Font(norm) {-Adobe-helvetica-medium-r-normal--*-180*}
# Preference
   set Pref(DeIcon)     1;        # deiconify when mail
   set Pref(Delay)     30;        # seconds to wait between mail checks
   set Pref(OpenInbox)  1;        # open inbox on incorporate
   set Pref(LastMsg)    1;        # Last Message
   set Pref(BackupMsg)  1;        # Backup Message
   set Pref(MsgFont)   $Font(helvm);
# Commands used in OS
   set OScmd(cat) cat
   set OScmd(mkdir) mkdir
   set OScmd(rmdir) "rm -r"
   set OScmd(cp) cp
   set OScmd(mv) mv
   set OScmd(print) "lp -d"
# Where the exec files stored
   set Prog(showpar) "$tksce_lib/showpar"
   set Prog(showsce) "$tksce_lib/showsce"
   set Prog(showtxt) "$tksce_lib/showtxt"
   set Prog(testsce) "$tksce_lib/testsce"
   set Prog(msgW)    "$tksce_lib/msgW"
   set Prog(scen)    "$tksce_lib/scen"
# Where help text found
   set Help(main)    "$tksce_lib/help"

# The configuration and initialization....

# Procedure to initialize Tksce
# Read in the global variables. Check for a .tksce and a .mh_profile
# if they're not there, create them. Then read in the .tksce, .mh_profile
# and build the folder.
# --- adapted from TKMH
proc InitTksce {} {
   global tksce_lib env FolList OScmd

   uplevel #0 source $tksce_lib/tksce.rc
   if {![file isfile $env(HOME)/.mh_profile]} {
    if [Confirm "No .mh_profile!\nCan't run without one\n\nCreate it?\n"] {
       eval exec $OScmd(cp) $tksce_lib/dot_mh $env(HOME)/.mh_profile
       MesgWin "Created $env(HOME)/.mh_profile\
                \nYou need to configure it by editing .mh_profile \
                in your home directory before using it!.\n"
```

```
    } else {
       MesgWin "Abort now....\
                \n\nIf you want to run tksce again, \
                you need to create a .mh_profile in your home directory.\n"
       exit
    }
   }
   ReadConfTksce
   ReadConfMH
   CreateFolList
}

# ReadConfTksce {}
# Read in the .tksce configuration file in the users home
# --- adapted from TKMH
proc ReadConfTksce {} {
    global Pref Font Bitmap OScmd env tksce_lib

    if [file isfile [RF .tksce]] {
        source [RF .tksce]
    }
}

# ReadConfMH {}
# Read the MH configuration file: .mh_profile in the users home
# --- adapted from TKMH
proc ReadConfMH {} {
    global MH env OScmd tksce_lib

    if {![file isfile $env(HOME)/.mh_profile]} {
        MesgWin "I'm in trouble!\
                        \nNo $env(HOME)/.mh_profile...\
                        \n\nI have to create a new one\n"
        eval exec $OScmd(cp) $tksce_lib/dot_mh $env/(HOME)/.mh_profile
    }

    set FHandle [open $env(HOME)/.mh_profile r]
    while {[gets $FHandle Line] != -1} {
      regexp {^(.*):[   ]*(.*)[  ]*$} $Line Line Field Value
      switch -- $Field {
        Path            {set MH(Path) $Value}
        Aliasfile       {set MH(AliasFile) $Value}
        Draft-Folder    {set MH(Draft-Folder) [string trimleft $Value "+"]}
        Inbox           {set MH(Inbox) [string trimleft $Value "+"]}
        MailDrop        {set MH(MailDrop) $Value}
        Sigfile         {set MH(SigFile) $Value}
```

```
    }
  }
  close $FHandle

  # Check if Maildir exists, if not, Create it
  if {![file isdir [RF $MH(Path)]]} {
      if [file isfile [RF $MH(Path)]] {
          MesgWin "$MH(Path) is not a directory"
          exit
      }
      if [Confirm "$MH(Path) does not exist. Create it?"] {
          eval exec $OScmd(mkdir) [RF $MH(Path)]
          eval exec $OScmd(mkdir) [RF $MH(Path)/$MH(Inbox)]
          eval exec $OScmd(mkdir) [RF $MH(Path)/$MH(Draft-Folder)]
      } else {
          MesgWin "Cannot run without a Mailing dir:\
               \nChange your settings and try again."
      }
  }
  # If it still doesn't exist, exit
  if {![file isdir [RF $MH(Path)]]} {
      MsgWindow "Exiting...."
      exit
  }
  # Check if inbox and drafts exist, and create them
  if {![file isdir [RF $MH(Path)/$MH(Inbox)]]} {
      if {![file isfile [RF $MH(Path)/$MH(Inbox)]]} {
          eval exec $OScmd(mkdir) [RF $MH(Path)/$MH(Inbox)]
      } else {
          eval exec $OScmd(mv) [RF $MH(Path)/$MH(Inbox)] \
                               [RF $MH(Path)/$MH(Inbox)].file
          eval exec $OScmd(mkdir) [RF $MH(Path)/$MH(Inbox)]
      }
  }
  if {![file isdir [RF $MH(Path)/$MH(Draft-Folder)]]} {
      if {![file isfile [RF $MH(Path)/$MH(Draft-Folder)]]} {
          eval exec $OScmd(mkdir) [RF $MH(Path)/$MH(Draft-Folder)]
      } else {
          eval exec $OScmd(mv) [RF $MH(Path)/$MH(Draft-Folder)] \
                               [RF $MH(Path)/$MH(Draft-Folder)].file
          eval exec $OScmd(mkdir) [RF $MH(Path)/$MH(Draft-Folder)]
      }
  }
}
```

```
# Main Window
#
proc MainWin {} {
   global Bitmap MH env Pref Font OScmd Prog
   global MDSize MDOldSize FolList Fol Msg
   global Scefl Txtfl Initxt
   set Fol {}
   set Msg {}

   wm title . "MM-Mail + Scenario Service"
   wm iconname . "MMScenario"
   wm geometry . +10+10

   frame .top
   frame .top.mn
   frame .top.mn.m1 -relief raised -borderwidth 1
   frame .top.mn.m2 -relief raised -borderwidth 1
   frame .pane
   frame .pane.pane1 -borderwidth 5
   frame .pane.pane2 -borderwidth 5
   frame .mid -borderwidth 5
   frame .bot -borderwidth 5

   menubutton .top.mn.m1.mesg -text "Message" -height 1 -width 9 \
    -menu .top.mn.m1.mesg.m -underline 0
   menu .top.mn.m1.mesg.m
   .top.mn.m1.mesg.m add command -label "New" -command \
   "Compose" -underline 0
   .top.mn.m1.mesg.m add command -label "Send" -command \
   "Msg_Send" -underline 0
   .top.mn.m1.mesg.m add command -label "Reply" -command \
   "Msg_Reply" -underline 0
   .top.mn.m1.mesg.m add command -label "Delete" -command \
   "Msg_Delete" -underline 0
   .top.mn.m1.mesg.m add command -label "Print" -command \
   "Msg_Print" -underline 0

   menubutton .top.mn.m1.present -text "Present" -height 1 -width 9 \
   -menu .top.mn.m1.present.m -underline 0
   menu .top.mn.m1.present.m
   .top.mn.m1.present.m add radio -label "Ordinary" \
   -variable Pres -value ord
   .top.mn.m1.present.m add radio -label "Scenario" \
   -variable Pres -value sce
   .top.mn.m1.present.m invoke 0
```

```
button .top.mn.m1.comp -text "MM-Compose" -height 1 -width 13 \
-command "Compose"

button .top.mn.m1.quit -text "Quit" -height 1 -width 9 -command \
{Rm_Temp; destroy .; exit}
button .top.mn.m1.help -text "Help" -height 1 -width 9 -command \
"Help"
button .top.inc -bitmap @$Bitmap(nomail) -command {IncMail}

button .top.mn.m2.fn -text "Create New" -height 1 -width 10 -command \
"CreateFol"
button .top.mn.m2.fr -text "Remove" -height 1 -width 8 -command \
"RemoveFol"
button .top.mn.m2.pa -text "Pack" -height 1 -width 8 -command \
"PackFol"
button .top.mn.m2.so -text "Sort" -height 1 -width 8 -command \
"SortFol"
button .top.mn.m2.re -text "Rename" -height 1 -width 8 -command \
"RenameFol"

label .top.mn.m2.l -text {} -relief raised \
-font $Font(helvm) -anchor w

scrollbar .pane.pane1.scroll -relief sunken -command \
".pane.pane1.list yview"
listbox .pane.pane1.list -yscroll ".pane.pane1.scroll set" \
-relief sunken -geometry 12x11 -setgrid 1

scrollbar .pane.pane2.scroll -relief sunken -command \
".pane.pane2.list yview"
listbox .pane.pane2.list -yscroll ".pane.pane2.scroll set" \
-relief sunken -geometry 60x11 -setgrid 1

label .mid.l -text {} -relief raised \
-font $Font(helvm) -anchor w

button .mid.snd -text "Send" -height 1 -width 8 -command "Msg_Send"
button .mid.rep -text "Reply" -height 1 -width 8 -command "Msg_Reply"
button .mid.del -text "Delete" -height 1 -width 8 -command "Msg_Delete"
button .mid.mov -text "Move To" -height 1 -width 8 -command "Msg_Move"
button .mid.cop -text "Copy To" -height 1 -width 8 -command "Msg_Copy"
button .mid.prt -text "Print" -height 1 -width 8 -command "Msg_Print"

text .bot.t -yscroll ".bot.s set" -wrap word -font $Pref(MsgFont) \
 -height 21
scrollbar .bot.s -command ".bot.t yview"
```

```
pack append .top.mn.m1 \
  .top.mn.m1.mesg {left padx 10 pady 10} \
  .top.mn.m1.present {left padx 10} \
  .top.mn.m1.comp {left padx 10} \
  .top.mn.m1.quit {left padx 10} \
  .top.mn.m1.help {left padx 10}

pack .top.mn.m2.l -side left -expand yes -fill both
pack append .top.mn.m2 \
  .top.mn.m2.pa {right padx 10 pady 10} \
  .top.mn.m2.so {right padx 10} \
  .top.mn.m2.re {right padx 10} \
  .top.mn.m2.fr {right padx 10} \
  .top.mn.m2.fn {right padx 10}

pack append .top.mn \
  .top.mn.m1 {top expand fill} \
  .top.mn.m2 {top fillx}

pack append .top \
  .top.mn {left expand fillx} \
  .top.inc {left padx 15}

pack append .pane.pane1 \
  .pane.pane1.list {left expand fill} \
  .pane.pane1.scroll {left filly}

pack append .pane.pane2 \
  .pane.pane2.list {left expand fill} \
  .pane.pane2.scroll {left filly}

pack append .pane \
  .pane.pane1 {left expand fill} \
  .pane.pane2 {left expand fill}

pack .mid.l -side left -expand yes -fill both
pack append .mid \
  .mid.prt {right padx 10 pady 10} \
  .mid.del {right padx 10} \
  .mid.cop {right padx 10} \
  .mid.mov {right padx 10} \
  .mid.rep {right padx 10} \
  .mid.snd {right padx 10}

pack append .bot \
```

```
      .bot.t {left expand fill} \
      .bot.s {left filly}

   pack append . \
      .top {top expand fill} \
      .pane {top expand fill} \
      .mid {top expand fill} \
      .bot {top expand fill}

   bind .pane.pane1.list <Double-Button-1> {
     if {[set Sel [GetListboxSel .pane.pane1.list]] != {}} {
FlashSel .pane.pane1.list 2; OpenFol $Sel;
.top.mn.m2.1 configure -text $Sel; set Fol $Sel}}

   bind .pane.pane2.list <Double-Button-1> {
     if {[set Msg [GetListboxSel .pane.pane2.list]] != {}} {
FlashSel .pane.pane2.list 2;
set Msg [GetMsgNum $Fol];
.mid.1 configure -text "$Fol: $Msg";
if {$Pres == "sce"} {
   set exist [Find_Sce $Fol $Msg]
   if {$exist > 0} {
      Present_Sce
   } else {
      MesgWin "No Scenario Code!\n Use Ordinary Presentation"
      Remove_Tmp
   }
} elseif {$Pres == "ord"} {
   Msg_Show $Fol $Msg
}}}

   update
   DisplayFols
   tk_listboxSingleSelect .pane.pane1.list
   if $Pref(OpenInbox) {
      set Fol $MH(Inbox)
      OpenFol $Fol 1
      .top.mn.m2.1 configure -text $Fol
   }
   .top.inc configure -state disabled

# Set up for keyboard-based menu traversal

   bind . <Any-FocusIn> {
      if {("%d" == "NotifyVirtual") && ("%m" == "NotifyNormal")} {
 focus .top.mn.m1
```

```
        }
    }

    tk_menuBar .top.mn.m1 .top.mn.m1.mesg .top.mn.m1.present \
    .top.mn.m1.comp

    set MDSize 0
    set MDOldSize 0
}

# Procedure to remove temporary files created when composing message

proc Rm_Temp {} {
    if [file isfile tflof] {exec rm tflof}
    if [file isfile tpart] {exec rm tpart}
    if [file isfile tform] {exec rm tform}
    if [file isfile trform] {exec rm trform}
    if [file isfile tsce.sce] {exec rm tsce.sce}
}

# Procedure to show help text

proc Help {} {
    global Help Font
    if [file isfile help] {
       ShowText help "+10+10" $Font(helvb) "70"
    } elseif [file isfile $Help(main)] {
       ShowText $Help(main) "+10+10" $Font(helvb) "70"
    }
}

# Procedure for Composing Message.

proc Compose {{w .comp}} {
    global env Font Prog Lof LoForm LoPart Form direct
    catch {destroy $w}
    toplevel $w
    wm geometry $w +0+0
    wm title $w "Composing Message"
    wm iconname $w "Compose"
    wm geometry $w 1010x760

    set Lof {}
    set LoForm {}
    set LoPart {}
```

```
frame $w.menu -relief raised -borderwidth 1
pack $w.menu -side top -fill x

label $w.menu.l -text {} -relief raised \
-font $Font(helvm) -anchor w

menubutton $w.menu.cf -text "Create-Form" -width 12 -menu $w.menu.cf.m
menu $w.menu.cf.m
$w.menu.cf.m add cascade -label {Component Form} \
-menu $w.menu.cf.m.form
$w.menu.cf.m add cascade -label {Directives} -menu $w.menu.cf.m.direct
$w.menu.cf.m add command -label {Create} -command "Create $env(HOME)"

menu $w.menu.cf.m.form
$w.menu.cf.m.form add radio -label "Send-Form" \
-variable Form -value send
$w.menu.cf.m.form add radio -label "Reply-Form" \
-variable Form -value reply
$w.menu.cf.m.form invoke 0

menu $w.menu.cf.m.direct
$w.menu.cf.m.direct add radio -label "type" \
-variable direct -value type
$w.menu.cf.m.direct add radio -label "extern" \
-variable direct -value extern
$w.menu.cf.m.direct invoke 0

menubutton $w.menu.scen -text "Scenario" -width 9 -menu $w.menu.scen.m
menu $w.menu.scen.m
$w.menu.scen.m add command -label "Create" -command \
"Create_Scen"
$w.menu.scen.m add command -label "Show Code" -command \
"Show_ScenCode"
$w.menu.scen.m add command -label "Test Code" -command \
"Test_Code"
$w.menu.scen.m add command -label "Attach Code" -command \
"Sce_Attach"

button $w.menu.sf -text "Show-Form" -height 1 -width 12 -command \
"Show_Form"
button $w.menu.send -text "Send-To" -width 9 -command "Msg_Send"
button $w.menu.cancel -text "Cancel" -width 8 -command \
"Reset_List; Rm_Temp; destroy $w"
button $w.menu.done -text "Done" -width 6 -command \
"Reset_List; destroy $w"
```

```
    pack append $w.menu \
      $w.menu.cf {left padx 10 pady 2} \
      $w.menu.scen {left padx 10}\
      $w.menu.sf {left padx 10}\
      $w.menu.send {left padx 10}\
      $w.menu.cancel {left padx 10}\
      $w.menu.done {left padx 10}
    pack $w.menu.l -side left -expand yes -fill x
}

# Procedure to select file(s) and create the message parts

proc Create {{Dir .}} {
    global Font env Pno OldDir
    set OldDir [pwd]
    cd $Dir
    set Pno 2;  # The first part no. for scenario code is 2
    set w .create
    catch {destroy .create}
    toplevel $w
    wm geometry $w +10+63
    wm title $w "Select & Create Parts"

    label $w.l -text [pwd] -relief raised \
      -font $Font(helvb) -anchor w

    frame $w.c
    frame $w.c.b -relief raised -borderwidth 2
    button $w.c.b.home -text Home -height 2 -width 8 -command \
      "cd $env(HOME); $w.l configure -text \[pwd\]; Create_Fill $w.c.dir"
    button $w.c.b.ok -text OK -height 2 -width 8 -command \
     "set f \[$w.e.e get\]
      if {\[file isdirectory \[string trimright \$f {/}\]\] && \$f != {}} {
cd \$f; set Dir \$f; Create_Fill $w.c.dir;
$w.e.e delete 0 end; $w.l configure -text \[pwd\]\
      } else {
Exec_File \[pwd\]/\$f
        grab $w
        tkwait window $w
      }"
    button $w.c.b.cancel -text Cancel -height 2 -width 8 -command \
      "Reset_List; cd $OldDir; destroy $w"
    button $w.c.b.done -text Done -height 2 -width 8 -command \
      "Make_File; Reset_List; cd $OldDir; destroy $w"
    menubutton $w.c.b.check -text "Check" -height 2 -width 8 \
    -menu $w.c.b.check.m
```

```
set lfl 1; set lfo 2; set lpa 3
menu $w.c.b.check.m
$w.c.b.check.m add command -label {Selected Files} \
-command "Check_List $lfl; grab $w; tkwait window $w"
$w.c.b.check.m add command -label {Components} \
-command "Check_List $lfo; grab $w; tkwait window $w"
$w.c.b.check.m add command -label {Message Parts} \
-command "Check_List $lpa; grab $w; tkwait window $w"

listbox $w.c.dir -relief raised -yscroll "$w.c.s set" \
-font $Font(helvb) -geometry 20x15
scrollbar $w.c.s -relief flat -command "$w.c.dir yview"

frame $w.e -relief raised -borderwidth 2
label $w.e.l -text "File:" -font $Font(helvb)
entry $w.e.e -font $Font(helvm) -width 28

pack append $w.c.b \
  $w.c.b.home {top pady 1} \
  $w.c.b.ok {top pady 1} \
  $w.c.b.check {top pady 1} \
  $w.c.b.cancel {top pady 1} \
  $w.c.b.done {top pady 1}
pack append $w.c \
  $w.c.dir {left expand fill} \
  $w.c.s {left filly} \
  $w.c.b {left filly}
pack append $w.e \
  $w.e.l {left} \
  $w.e.e {left padx 10 pady 10}
pack append $w \
  $w.l {top fillx} \
  $w.c {top expand fillx} \
  $w.e {top fillx}

bind $w.e.e <Key-Return> "$w.c.b.ok invoke"
bind $w.c.dir <1> "
  %W select from \[%W nearest %y\]
  $w.e.e delete 0 end
  $w.e.e insert end \[selection get\]"
bind $w.c.dir <Double-1> "$w.c.b.ok invoke"

tk_listboxSingleSelect $w.c.dir
focus $w.e.e
Create_Fill $w.c.dir
```

```
    grab $w
    tkwait window $w
    cd $OldDir
}

proc Create_Fill {list {spec {*}}} {
    $list delete 0 end
    $list insert end ".."
    foreach i [lsort [eval glob -nocomplain $spec]] {
if {$i == "." || $i == ".."} continue
if [file isdirectory $i] {
    $list insert end "$i/"
} else {
    $list insert end $i
}
    }
}

# Procedure to Execute selected file

proc Exec_File {flname} {
    global Prog arg Eln OldDir Font

    #set file [concat $OldDir/.sce_profile]
    set file [concat ~/.mh_profile]
    set fl [open $file r]
    #create list of profile
    while {[gets $fl line] >= 0} {
       set x [lsearch [split $line :- ] show]
       if {$x > -1} {
  set eln [split $line ]
  lappend Eln $eln
       }
    }
    close $fl

    # extract name of file and its extention
    # name = file name; endx = extention
    set spname1 [split $flname /]
    set len1 [llength $spname1]
    set name [lindex $spname1 [expr $len1-1]]
    set spname2 [split $name .]
    set len2 [llength $spname2]
    if {$len2 > 1} {set endx [lindex $spname2 [expr $len2-1]]
    } else {set endx {}}
    # check device from list of profile
```

```
    set ok [Check_Pro $endx]
    if {$ok == 1} {
       if {$endx == "txt" || $endx == "sce"} {
          puts [eval exec $arg $flname]
          Attach_Form $flname
       } else {
          eval exec $arg $flname
          Attach_Form $flname
       }
    } elseif {$endx == "" || $endx == "txt"} {
       ShowText $flname "-10-10" $Font(helvm)
       Attach_Form $flname
    } else {
       set mesg "No player for $endx"
       eval exec $Prog(msgW) -geometry -10+63 $mesg }
}

# Procedure for checking profile (from mh_profile)

proc Check_Pro {end} {
  global result arg Eln
  set result 0
  foreach ele $Eln {
    set arg0 [lindex $ele 0]
    set spty [split $arg0 :-]
    set type [lindex $spty 2]
    if { $type == "application/x" } { set type [lindex $spty 3] }
    switch $type {
       "image" {if {$end == "gif" || $end == "jpg"} {
 set arg [string trim [lrange $ele 1 end] %p%f]
 set result 1
 }
     }
       "video" {if {$end == "mpg" || $end == "mpeg"} {
 set arg [string trim [lrange $ele 1 end] %p%f]
 set result 1
 }
     }
       "application/Postscript" {if {$end == "ps"} {
 set arg [string trim [lrange $ele 1 end] %p%f]
 set result 1
 }
     }
       "text" {if {$end == "txt"} {
 set arg [string trim [lrange $ele 1 end] %p%f]
 set result 1
```

```
 }
     }
       "scenario" {if {$end == "sce"} {
 set arg [string trim [lrange $ele 1 end] %p%f]
 set result 1
 }
     }
    }
  }
  return $result
}

proc Attach_Form {flname} {
    global Lof Font Pno
    set w .attf
    catch {destroy $w}
    toplevel $w
    wm geometry $w +380+350
    wm title $w "Attachment"
    wm geometry $w 250x150

    label $w.l -text "Attach this file?" -relief raised \
     -font $Font(helvb) -anchor center

    frame $w.c -relief raised -borderwidth 2

    button $w.c.ok -text OK -height 2 -width 8 -command \
    "lappend Lof $flname; Make_FormPart $flname; \
     destroy $w"
    button $w.c.no -text NO -height 2 -width 8 -command \
    "destroy $w"

    pack $w.c.ok $w.c.no -side left -padx 5 -pady 5 -expand 1
    pack $w.l $w.c -side top -expand yes -fill both

    grab $w
    tkwait window $w
}

# Procedure to make component form and parts of message body

proc Make_FormPart {flname} {
    global direct Pno
    if {$direct == "type"} {
       set Desc [In_TypDirective $flname]
       Make_TypLFoPa $flname $Pno $Desc
```

```
        incr Pno
    } elseif {$direct == "extern"} {
        MesgWin "extern is NOT implemented yet!"
    }
}

# Procedure of Inputting Type directive

proc In_TypDirective {fname {w .des}} {
   global descr Font direct
   catch {destroy $w}
   toplevel $w
   wm geometry $w +380+550
   wm title $w "Directive"
   wm iconname $w "Dir"

   set spname1 [split $fname /]
   set len1 [llength $spname1]
   set name [lindex $spname1 [expr $len1-1]]

   frame $w.top -relief raised -border 1
   label $w.top.mesg -text "Description for $name?" \
    -font $Font(helvb) -width 40 -anchor center

   frame $w.ent -relief raised -borderwidth 2
   label $w.ent.l -text "Description: " -font $Font(helvb)
   entry $w.ent.e -font $Font(helvm) -width 28

   frame $w.bot -relief raised -border 1
   frame $w.bot.mid -relief sunken -border 1
   button $w.bot.mid.ok -text OK -width 4 -command \
   "set descr \[$w.ent.e get\]; destroy $w"

   pack $w.top $w.ent $w.bot -side top -fill both
   pack $w.top.mesg -side left
   pack append $w.ent \
     $w.ent.l {left} \
     $w.ent.e {left padx 10 pady 10}

   pack $w.bot.mid -expand yes -padx 10 -pady 10
   pack $w.bot.mid.ok -expand yes -padx 6 -pady 6

   bind $w.ent.e <Key-Return> "$w.bot.mid.ok invoke"

   focus $w.ent.e
   grab $w
```

```
    tkwait window $w
    return $descr
}

# Procedure of making the list of Form and Part of the type directive

proc Make_TypLFoPa {fname x desc} {
    global LoForm LoPart
    set spname1 [split $fname /]
    set len1 [llength $spname1]
    set name [lindex $spname1 [expr $len1-1]]
    set spname2 [split $name .]
    set len2 [llength $spname2]
    if {$len2 > 1} {set endx [lindex $spname2 [expr $len2-1]]
    } else {set endx {}}
    switch $endx {
        "gif" { lappend LoForm [concat "#image/gif" \[$desc\] \\]
        lappend LoForm $fname
        lappend LoPart [concat "image/gif/$x.gif" \[$desc\]]
      }
        "jpg" { lappend LoForm [concat "#image/jpg" \[$desc\] \\]
        lappend LoForm $fname
        lappend LoPart [concat "image/jpg/$x.jpg" \[$desc\]]
      }
        "mpg" { lappend LoForm [concat "#video/mpeg" \[$desc\] \\]
        lappend LoForm $fname
        lappend LoPart [concat "video/mpeg/$x.mpg" \[$desc\]]
      }
        "sce" { lappend LoForm \
                [concat "#application/x-scenario" \[$desc\] \\]
        lappend LoForm $fname
        lappend LoPart \
                [concat "application/x-scenario/$x.sce" \[$desc\]]
      }
        "ps"  { lappend LoForm \
                [concat "#application/Postscript" \[$desc\] \\]
        lappend LoForm $fname
        lappend LoPart \
                [concat "application/Postscript/$x.ps" \[$desc\]]
      }
        "txt" { lappend LoForm [concat "#text/plain" \[$desc\] \\]
        lappend LoForm $fname
        lappend LoPart [concat "text/plain/$x.txt" \[$desc\]]
      }
        ""    { lappend LoForm \
                [concat "#application/octet-stream" \[$desc\] \\]
```

```
        lappend LoForm $fname
        lappend LoPart \
                [concat "application/octet-stream/$x.ocs" \[$desc\]]
      }
    }
}

# Check all selected files

proc Check_List {ltype {w .r}} {
    global Lof LoForm LoPart
    catch {destroy $w}
    toplevel $w
    wm geometry $w +380+120
    wm title $w "Checking List"
    wm iconname $w "CheckList"
    wm minsize $w 1 1

    frame $w.c
    frame $w.c.b -borderwidth 10
    button $w.c.b.ok -text OK -height 2 -width 8 -command \
    "destroy $w"

    scrollbar $w.c.scroll -relief sunken -command "$w.c.list yview"
    listbox $w.c.list -yscroll "$w.c.scroll set" -relief sunken \
    -geometry 35x10 -setgrid 1

    pack $w.c.b.ok -side top -pady 10

    pack append $w.c \
      $w.c.list {left expand fill}\
      $w.c.scroll {left filly}\
      $w.c.b {left filly}

    pack append $w \
      $w.c {top expand fillx}

    switch $ltype {
       "1" { if {$Lof == {}} {
$w.c.list insert 0 "No list of files!"
     } else {
foreach part [lrange $Lof 0 end] {
   $w.c.list insert end $part
       }
     }
   }
```

```
       "2" { if {$LoForm == {}} {
$w.c.list insert 0 "No Component form!"
     } else {
foreach part [lrange $LoForm 0 end] {
   $w.c.list insert end $part
}
     }
   }
       "3" { if {$LoPart == {}} {
$w.c.list insert 0 "No list of parts!"
     } else {
foreach part [lrange $LoPart 0 end] {
   $w.c.list insert end $part
}
     }
   }
    }
    grab $w
    tkwait window $w
}

# Procedure to make all temporary files

proc Make_File {} {
    global OldDir MH
    cd $OldDir;      # Change directory to the Previous dir

    # Making a file of the List of selection files
    global Lof
    set fd [open tflof w]
    foreach part [lrange $Lof 0 end] {
       puts $fd $part
    }
    close $fd

    # Making a file of the List of Body Parts
    global LoPart
    set fd [open tpart w]
    foreach part [lrange $LoPart 0 end] {
       puts $fd $part
    }
    close $fd

    # Making a file of the list of component form
    global LoForm Form
    if {$Form == "send"} {
```

```
        #if [file isfile tform] {
           set fd [open tform w]
           set to "To:"
           set cc "cc:"
           set subj "Subject:"
           set sprt "--------"
           puts $fd $to
           puts $fd $cc
           puts $fd $subj
           puts $fd $sprt
           foreach part [lrange $LoForm 0 end] {
              puts $fd $part
           }
           close $fd
        #} else { MesgWin "No Form!" }
     } elseif {$Form == "reply"} {
        if [file isfile $MH(libdir)/replcomps] {
           #if [file isfile trform] {
              eval exec cp $MH(libdir)/replcomps trform ; #copy file
              set fd [open trform a]
              foreach part [lrange $LoForm 0 end] {
                 puts $fd $part
              }
              close $fd
           #} else { MesgWin "No Form!"}
        } else { MesgWin "No Reply Component!" }
     }
}

# Procedure to show the Code

proc Show_ScenCode {} {
     global Prog Font
     if [file isfile tsce.sce] {
         ShowText tsce.sce "+220+210" $Font(helvm)
      } else {MesgWin "No Scenario Code!"}
}

# Procedure to create scenario

proc Create_Scen {} {
     global Prog
     if [file isfile tflof] {
         eval exec $Prog(showpar) tflof $Prog(scen)
     } else {MesgWin "No List of files!"}
}
```

```
# Procedure to test the scenario code

proc Test_Code {} {
    global Prog
    set w .comp
    if {([file isfile tsce.sce]) && ([file isfile tflof])} {
       eval exec $Prog(testsce) tsce.sce tflof $Prog(msgW)
    } else {MesgWin "No Code or List of files!"}
}

# Procedure to show the component form

proc Show_Form {} {
    global Prog Form Font
    if {$Form == "send"} {
       if [file isfile tform] {
          ShowText tform "+220+210" $Font(helvm)
       } else {MesgWin "No component file!"}
    } elseif {$Form == "reply"} {
       if [file isfile trform] {
          ShowText trform "+220+210" $Font(helvm)
       } else {MesgWin "No component file!"}
    }
}

# Procedure to reset and empty the lists

proc Reset_List {} {
    global Lof LoForm LoPart
    set Lof {}
    set LoForm {}
    set LoPart {}
    set Pno 2
}


# Procedure Msg_Send {}

proc Msg_Send {{w .p3}} {
    global Font
    catch {destroy $w}
    toplevel $w
    wm geometry $w +20+180
    wm title $w "Sending Message"
    wm iconname $w "send"
```

```
    set a Form; set b New; set c Draft
    frame $w.menu -relief raised -borderwidth 1
    message $w.ms -font $Font(helvb) -relief raised \
    -width 500 -borderwidth 1 -text " NOTES:
 Message will be sent using comp through
 xterm application...
  - After completing message use <Control-D>.
  - It will ask you to respond whatnow? prompt.
  - Type ? (or help) for more information."

    menubutton $w.menu.send -text "Send" -height 2 -width 8 \
    -menu $w.menu.send.m
    menu $w.menu.send.m
    $w.menu.send.m add command -label "Form" -command "XCompSend $a"
    $w.menu.send.m add command -label "New" -command "XCompSend $b"
    $w.menu.send.m add command -label "Draft" -command "XCompSend $c"
    button $w.menu.cancel -text "Cancel" -height 2 -width 8 -command\
    "destroy $w"
    button $w.menu.done -text "Done" -height 2 -width 8 -command\
    "destroy $w"

    pack append $w.menu \
       $w.menu.send {top padx 2 pady 2}\
       $w.menu.cancel {top padx 2}\
       $w.menu.done {top padx 2}
    pack append $w \
       $w.ms {left expand fill}\
       $w.menu {left fillx}
    grab $w
    tkwait window $w
}

# Procedure to send message through xterm session

proc XCompSend {s} {
    set geom -50-40
    set title "Send $s-Message"
    if {$s == "Form"} {
       if [file isfile tform] {
 exec xterm -T $title -geometry $geom -e comp -form tform
       } else { MesgWin "No Form!" }
    } elseif {$s == "New"} {
       exec xterm -T $title -geometry $geom -e comp;#make the new one
    } elseif {$s == "Draft"} {
       MesgWin "Not implemented yet!"
```

```
#       exec xterm -T $title -geometry $geom -e comp -use $MsgDraft;
    }
}

# Procedure to reply to a message

proc Msg_Reply {{w .repl}} {
    global Font Fol Msg
    catch {destroy $w}
    toplevel $w
    wm geometry $w +20+180
    wm title $w "Reply Message"
    wm iconname $w "reply"
    set a Form; set b Nform; set c Draft

    #get message from the list
    set fol $Fol
    set Msg [GetMsgNum $fol]

    frame $w.menu -relief raised -borderwidth 1
    message $w.ms -font $Font(helvb) -relief raised \
    -width 500 -borderwidth 1 -text " NOTES:
 A reply to a message will be sent using
 xterm application...
  - After completing message use <Control-D>.
  - It will ask you to respond whatnow? prompt.
  - Type ? (or help) for more information."

    menubutton $w.menu.rep -text "Reply" -height 2 -width 8 \
    -menu $w.menu.rep.m
    menu $w.menu.rep.m
    $w.menu.rep.m add command -label "Form" -command "XReply $a"
    $w.menu.rep.m add command -label "Without-Form" -command "XReply $b"
    $w.menu.rep.m add command -label "With-Draft" -command "XReply $c"

    button $w.menu.cancel -text "Cancel" -height 2 -width 8 -command\
    "destroy $w"
    button $w.menu.done -text "Done" -height 2 -width 8 -command\
    "destroy $w"

    pack append $w.menu \
       $w.menu.rep {top padx 2 pady 2}\
       $w.menu.cancel {top padx 2}\
       $w.menu.done {top padx 2}
    pack append $w \
       $w.ms {left expand fill}\
```

```
        $w.menu {left fillx}
    grab $w
    tkwait window $w
}

# Procedure to send message through xterm session

proc XReply {s} {
  global MH Fol Msg
  set geom -50-40
  set title "Reply $s-Message"
  if {$s == "Form"} {
     if {($Fol != {}) && ($Msg != {})} {
if [file isfile trform] {
  set rform [concat [pwd]/trform]
  exec xterm -T $title -geometry $geom -e repl +$Fol $Msg -form $rform;
} else { MesgWin "No Form!" }
     } else {
MesgWin "No message will be replied!"
     }
  } elseif {$s == "Nform"} {
     if {($Fol != {}) && ($Msg != {})} {
exec xterm -T $title -geometry $geom -e repl +$Fol $Msg; #without form
     } else {
MesgWin "No message will be replied!"
     }
  } elseif {$s == "Draft"} {
     MesgWin "Not implemented yet!"
  }
}

# Procedure Msg_Show {Fol Msg}

proc Msg_Show {Fol Msg} {
    global MH Pref

    set OldFont $Pref(MsgFont)
    set w .bot
    if [winfo exists $w] {
       $w.t configure -state normal
       $w.t delete 0.0 end
       $w.t insert end [exec $MH(bindir)/show +$Fol $Msg -nopause]
       return
    }
    $w.t configure -state disabled
    focus $w.t
```

```
    tkwait window $w
    set Pref(MsgFont) $OldFont
}

# Msg_Move {}
# Move $Msg from $FromFol to $ToFol, append as last
# --- Adapted from TKMH
proc Msg_Move {} {
    global MH OScmd Fol

    set w1 .bot
    set w2 .mid
if {[set ToFol [Ask {Move To Folder?}]] != {}} {
 if [conv_listbox_seq $Fol movemsgs] {
    # read in the sequences
    set movemsgs [conv_seq $Fol movemsgs]
    set allmsgs [conv_seq $Fol allmsgs]
    set first [lsearch $allmsgs [lindex $movemsgs 0]]
    set last [lsearch $allmsgs [lindex $movemsgs \
              [expr [llength $movemsgs] -1]]]
    set firstMsg [expr [FindLastFile $ToFol] + 1]

    set NewMsg $firstMsg
    foreach Msg $movemsgs {
eval exec $OScmd(mv) [RF $MH(Path)/$Fol/$Msg] \
   [RF $MH(Path)/$ToFol/$NewMsg]
incr NewMsg
    }

    del_from_seq $Fol allmsgs $first-$last
    del_seq $Fol movemsgs
    Cache:Write [RF $MH(Path)/$Fol/.xmhcache] \
      [lreplace [Cache:Read [RF $MH(Path)/$Fol/.xmhcache]] $first $last]

    append_seq $ToFol allmsgs $firstMsg-last
    Cache:Append [RF $MH(Path)/$ToFol/.xmhcache] \
      [split [exec $MH(bindir)/scan +$ToFol $firstMsg-last] "\n"]

    ScanFol $ToFol
    ScanFol $Fol
    $w1.t delete 0.0 end
    $w2.l configure -text {}
  }
 }
}
```

```
# Msg_Copy {}
# Copy $Msg from $Fol to $ToFol, append as last, use seq copymsg
# --- Adapted from TKMH
proc Msg_Copy {} {
    global MH OScmd Fol
    set w1 .bot
    set w2 .mid
    if {[set ToFol [Ask {Copy To Folder?}]] != {}} {
       if [conv_listbox_seq $Fol copymsgs] {
  # read in the sequences
  set copymsgs [conv_seq $Fol copymsgs]
  set firstMsg [expr [FindLastFile $ToFol] + 1]

  #copy the messages
  set NewMsg $firstMsg
  foreach Msg $copymsgs {
     eval exec $OScmd(cp) [RF $MH(Path)/$Fol/$Msg] \
   [RF $MH(Path)/$ToFol/$NewMsg]
     incr NewMsg
  }

  # adapt the ToFol seq, cache and display
  del_seq $Fol copymsgs
  append_seq $ToFol allmsgs $firstMsg-last
  Cache:Append [RF $MH(Path)/$ToFol/.xmhcache] \
  [split [exec $MH(bindir)/scan +$ToFol $firstMsg-last] "\n"]
  set Fol $ToFol
  ScanFol $Fol
  $w1.t delete 0.0 end
  $w2.l configure -text {}
      }
    }
}

# Msg_Delete {}
# delete a message, or a range of messages, reading the sequence
# delmsgs from the .mh_sequences file
# --- Adapted from TKMH
proc Msg_Delete {} {
    global MH OScmd Pref Fol

  set w1 .bot
  set w2 .mid
  if [conv_listbox_seq $Fol delmsgs] {
    # read in the sequences
    set delmsgs [conv_seq $Fol delmsgs]
```

```
    set allmsgs [conv_seq $Fol allmsgs]
    set first [lsearch $allmsgs [lindex $delmsgs 0]]
    set last [lsearch $allmsgs [lindex $delmsgs \
              [expr [llength $delmsgs] -1]]]

    #delete the files
    eval exec $MH(bindir)/rmm +$Fol $delmsgs
    if {!$Pref(BackupMsg)} {
foreach Msg $delmsgs {
    if {![file isfile [RF $MH(Path)/$Fol/,$Msg]]} break
    eval exec $OScmd(rm) [RF $MH(Path)/$Fol/,$Msg]
}
    }

    #adapt the sequences and the message cache
    del_from_seq $Fol allmsgs $first-$last
    del_seq $Fol delmsgs
    Cache:Write [RF $MH(Path)/$Fol/.xmhcache] \
      [lreplace [Cache:Read [RF $MH(Path)/$Fol/.xmhcache]] $first $last]

    # adapt the display if there is one
    ScanFol $Fol
    $w1.t delete 0.0 end
    $w2.l configure -text {}
  }
}

# Procedure to print message(s)
# --- Adapted from TKMH
proc Msg_Print {} {
    global OScmd MH Fol Msg

  set Msgs [GetMsgNum $Fol 10]
  if {$Msgs != {}} {
    if {[set printer [Ask {Printer used: lp [-d?]}]] != {}} {
      set OScmd(print) [concat $OScmd(print)$printer]
      if {![regsub "%f" $OScmd(print) {[RF $MH(Path)/$Fol/$Msg]} \
         PrintCmd]} {
 set PrintCmd "$OScmd(print) {[RF $MH(Path)/$Fol/$Msg]}"
      }
      foreach Msg $Msgs {
 eval exec $PrintCmd
      }
    }
  }
}
```

```
# Procedure to present message based on scenario code

proc Present_Sce {{w .p1}} {
    global Scefl Initxt Fol Msg
    catch {destroy $w}
    toplevel $w
    wm geometry $w +0+0
    wm title $w "Scenario Presentation: $Fol: $Msg"
    wm iconname $w "ScePresent"
    wm geometry $w 1000x760

    frame $w.menu -relief raised -borderwidth 1
    pack $w.menu -side top -fill x

    button $w.menu.run -text Run -height 1 -width 7 -command \
    "Exec_Sce $w"
    button $w.menu.sce -text Show-Code -height 1 -width 11 -command \
    "Show_Sce"
    button $w.menu.txt -text Show-Text -height 1 -width 11 -command \
    "Show_Text"
    button $w.menu.cancel -text Cancel -height 1 -width 7 -command \
    "Remove_Tmp; destroy $w"
    button $w.menu.done -text Done -height 1 -width 7 -command \
    "Remove_Tmp; destroy $w"

    pack $w.menu.run $w.menu.sce $w.menu.txt $w.menu.cancel\
    $w.menu.done -side left
}

# Procedure to find scenario code in the message

proc Find_Sce {Fol Msg} {
    global MH xsce Scefl Initxt Txtfl
    set Initxt 0
    set delay 200
    set Txtfl {}
    exec mhn -store +$Fol $Msg 2> part
    set fl [open part r]
    while {[gets $fl line] >= 0} {
       set spline [split $line]
       set ln [llength $spline]
       set endx [lindex $spline [expr $ln-1]]
       set y [lsearch [split $endx /.] txt]
       if {$y > 0} {
 set eln [llength [split $endx /]]
```

```
 set Txtfl [lindex [split $endx /] [expr $eln-1]]
 set txt $Txtfl
 set elnn [llength [split $txt .]]
 set prt [string trimright $txt .txt]
 set prt [string trimleft $prt $Msg]
 set prt [string trimleft $prt .]
 set z [lindex [split $txt .] [expr $elnn-2]]
 if {$z == 1} {
    set Initxt 1
    set w .bot
    $w.t configure -state normal
    $w.t delete 0.0 end
    $w.t insert end \
       [eval exec $MH(bindir)/mhn +$Fol $Msg -show -part $prt -nopause]
 }
    after $delay
       }
       set xsce [lsearch [split $endx /.] sce]
       if {$xsce > 0} {
 set eln [llength [split $endx /]]
 set Scefl [lindex [split $endx /] [expr $eln-1]]
       }
    }
    close $fl
    return $xsce
}

# Execute Scenario code

proc Exec_Sce {w} {
    global Prog Scefl Initxt
    if [file isfile $Scefl] {
       eval exec $Prog(showsce) $Scefl $Initxt $Prog(msgW)
    } else {MesgWin "No Scenario Code!"}
}

# Procedure to show scenario code

proc Show_Sce {} {
    global Prog Scefl Font
    if [file isfile $Scefl] {
       ShowText $Scefl "+220+210" $Font(helvm)
    } else {MesgWin "No Scenario Code!"}
}

# Procedure to show scenario code
```

```
proc Show_Text {} {
    global Prog Txtfl Font
    if {$Txtfl != {} && ([file isfile $Txtfl])} {
       ShowText $Txtfl "+220+210" $Font(helvm)
    } else {
       MesgWin "No Initial Text in the message!"
    }
}

# Remove temporary files

proc Remove_Tmp {} {
   set fl [open part r]
   while {[gets $fl line] >= 0} {
      set spline [split $line]
      set ln [llength $spline]
      set endx [lindex $spline [expr $ln-1]]
      eval exec rm $endx
   }
   close $fl
   eval exec rm part
}

# Procedure to show text on a window

proc ShowText {flname geom font {wd 60} {w .wtxt}} {
    catch {destroy $w}
    toplevel $w
    wm title $w "$flname"
    wm iconname $w "$flname"
    wm geometry $w $geom

    set OScmd(cat) cat
    button $w.ok -text OK -command "destroy $w"
    text $w.t -relief raised -bd 2 -yscrollcommand "$w.s set" \
     -setgrid 1 -width $wd -height 20 -font $font
    scrollbar $w.s -relief flat -command "$w.t yview"
    pack $w.ok -side bottom -fill x
    pack $w.s -side right -fill y
    pack $w.t -expand yes -fill both
    $w.t insert end [exec $OScmd(cat) $flname]

    $w.t mark set insert 0.0
    bind $w <Any-Enter> "focus $w.t"
    grab $w
```

```
    tkwait window $w
}

# Procedure to show any message

proc MesgWin {mesg {w .wmsg}} {
    global Font
    catch {destroy $w}
    toplevel $w
    wm title $w "Messages"
    wm iconname $w "messages"

    frame $w.top -relief raised -border 1
    frame $w.bot -relief raised -border 1

    message $w.top.name -font $Font(norm) -text "$mesg" \
    -width 300 -justify left

    frame $w.bot.mid -relief sunken -border 1
    button $w.bot.mid.ok -text OK -command "destroy $w" -width 4

    pack $w.top $w.bot -side top -fill both -expand yes
    pack $w.top.name -side left
    pack $w.bot.mid -expand yes -padx 10 -pady 10
    pack $w.bot.mid.ok -expand yes -padx 6 -pady 6
    grab $w
    tkwait window $w
}

#-------------------------------------------------------------------#
#  The Procedures below were adapted from the TKMH (a Tk GUI for MH)  #
#-------------------------------------------------------------------#

# Procedure IncMail {}

proc IncMail {} {
    global MH Bitmap Pref Fol
    if {[catch {exec $MH(bindir)/inc +$MH(Inbox)}] == 1} {
       MesgWin "Something wrong with the Maildrop file!\
\nMaybe it's locked because you were just receiving mail.\
\n\n Try again.\n"
       return
    }
    .top.inc configure -bitmap @$Bitmap(nomail) -state disabled
    wm iconbitmap . @$Bitmap(nomail)
    if $Pref(OpenInbox) {
```

```
        set Fol $MH(Inbox)
        OpenFol $Fol 1
        .top.mn.m2.1 configure -text $Fol
    }
}

# Procedure Loop {}

proc Loop {} {
    global Pref
    CheckForMail
    if {![winfo exists .]} {exit}
    after [expr $Pref(Delay) * 1000] Loop
}

# CheckForMail {}

proc CheckForMail {} {
    global MH Bitmap Pref
    global MDSize MDOldSize

    if {![file isfile $MH(MailDrop)]} {
        return
    }
    set MDOldSize $MDSize
    set MDSize [file size $MH(MailDrop)]

    if {$MDSize > $MDOldSize} {
        # There is new mail
        if $Pref(DeIcon) {wm deiconify .; raise .}
        .top.inc configure -bitmap @$Bitmap(mail) -state active
        wm iconbitmap . @$Bitmap(mail)
        .top.inc flash; .top.inc flash
    }
    if {$MDSize < $MDOldSize && $MDSize <= 1} {
        # The maildrop file must be emptied by something else
        .top.inc configure -bitmap @$Bitmap(nomail) -state disabled
        wm iconbitmap . @$Bitmap(nomail)
    }
}

# CreateFolList {{cache 1} {cachefile .folders}}
# Fill the global var FolList with the foldernames. cache controls
# wether the cache should be read or not

proc CreateFolList {{cache 1} {cachefile .folders}} {
```

```
    global MH FolList

    if {$cache && [file isfile [RF $MH(Path)/$cachefile]]} {
set FolList [Cache:Read [RF $MH(Path)/$cachefile]]
    } else {
set FolList {}
dofolders [RF $MH(Path)] FolList
Cache:Write [RF $MH(Path)/$cachefile] $FolList
    }
}

# RF {FileName {Prefix $env(HOME)}}
# Prepend a file or directory name with the Prefix if the name is not
# absolute (doesn't begin with '/'). Return this name.

proc RF "FileName {Prefix $env(HOME)}" {
    if {[string first "/" $FileName] != 0} {
set FileName $Prefix/$FileName
    }
    return $FileName
}

# Cache:Read {cachefile}

proc Cache:Read cachefile {
    if {![file exists $cachefile]} {
MesgWin "No such cache file: $cachefile.\n"; return {}
    }
    set List {}
    set Fhandle [open $cachefile r]
    while {[gets $Fhandle Element] != -1} {
lappend List $Element
    }
    close $Fhandle
    return $List
}

# Cache:Write {cachefile List}
# read or write a cache file line by line and translate each line to
# a list element or vv. Cache:Read returns the List.

proc Cache:Write {cachefile List} {
    set Fhandle [open $cachefile w]
    foreach Element $List {puts $Fhandle $Element}
    close $Fhandle
}
```

```
proc Cache:Append {cachefile List} {
    set Fhandle [open $cachefile a+]
    foreach Element $List {puts $Fhandle $Element}
    close $Fhandle
}

# FlashSel {w {num 1} {delay 50}}
# Flashes the current selection in $w $num times. delay gives the
# delay between flashes.

proc FlashSel {w {num 1} {delay 50}} {
    set bgopt [$w configure -selectbackground]
    set fgopt [$w configure -selectforeground]
    set oldbg [lindex $bgopt [expr [llength $bgopt] - 1]]
    set oldfg [lindex $fgopt [expr [llength $fgopt] - 1]]
    for {set i 0} {$i < $num} {incr i} {
      $w configure -selectbackground $oldfg
      $w configure -selectforeground $oldbg; update
      after $delay
      $w configure -selectbackground $oldbg
      $w configure -selectforeground $oldfg; update
    }
}

# GetListboxSel {Listbox maxnum}
# Get the current selection list from the named listbox. Return a list
# with a maximum number of maxnum members if there is a selection,
# return {} if none. defaults to one entry return
# TODO: maybe more efficient coding....

proc GetListboxSel {Listbox {maxnum 1}} {
    global Result

    if {[set Sels [$Listbox curselection]] == ""} {
return {}
    }
    set Result {}
    foreach Sel $Sels {
lappend Result [$Listbox get $Sel]
    }
    if {[llength $Result] <= $maxnum} {
return $Result
    }
    return [lrange $Result 0 [expr $maxnum - 1]]
}
```

```
# GetMsgNum {Fol maxnum}
# Get the message number from the selected message in the Folder
# list and return it. maxnum is the maximum number of messages,
# defaults to 1.

proc GetMsgNum {Fol {maxnum 1}} {
    set Result {}
    if [conv_listbox_seq $Fol selmsgs] {
set msgs [conv_seq $Fol selmsgs]
del_seq $Fol selmsgs
return [lrange $msgs 0 [expr $maxnum - 1]]
    } else {return {}}
}

# FindLastFile {Fol}
# Find the last file in a folder, and return it.

proc FindLastFile {Fol} {
    global MH

    set cwd [pwd]
    cd [RF $MH(Path)/$Fol]
    set List [lsort -integer \
[glob -nocomplain {[0-9]} {[0-9][0-9]} {[0-9][0-9][0-9]} \
  {[0-9][0-9][0-9][0-9]} {[0-9][0-9][0-9][0-9][0-9]}]]
    set LastMsg [lindex $List [expr [llength $List] - 1]]
    if {$LastMsg == ""} {
set LastMsg {0}
    }
    cd $cwd
    return $LastMsg
}

# dofolders { dir lstname {path {}}}
# approximate substitute for the MH folders command, thanks to
# Mark Moraes (moraes@deshaw.com) --from TKMH

proc dofolders {dir lstname {path {}}} {
    upvar 1 $lstname lst
    set f [open "|/bin/ls $dir" r]
    while {[gets $f line] >= 0} {
if {[regexp {^,?[0-9]+$} $line]} {
    continue
}
catch {file stat $dir/$line st}
```

```
if {$st(type) == "directory"} {
    lappend lst "$path$line"
    if {$st(nlink) > 2} {
dofolders $dir/$line lst $path$line/
    }
}
    }
    close $f
}

# DisplayFols {}

proc DisplayFols {} {
    global FolList
    set w .pane.pane1
    set List $FolList
    set curYview [lindex [$w.scroll get] 2]
    $w.list delete 0 end
    foreach Fol $List {
       $w.list insert end $Fol
    }
    $w.list yview $curYview
}

# OpenFol {Fol force}
# Create a window in which the headers from messages in $Fol can
# be displayed, use ScanFol to read the messages. if force is 1,
# a folder rescan is forced

proc OpenFol {Fol {force 0}} {
    global FolTree Aliases
    global Font Pref

    set w .pane.pane2
    set NumMsg [ScanFol $Fol]
    if $Pref(LastMsg) {
      set height [lindex [split [lindex [$w.list configure -geometry]\
                  4] "x"] 1]
      $w.list yview [expr $NumMsg - $height]
    }
    return
}

# ScanFol {Fol}
# Fill the Folder window with the message headers, using the MH scan
# command.
```

```
proc ScanFol {Fol {force 0}} {
    global MH
    set w .pane.pane2
    if {$force || ![file isfile [RF $MH(Path)/$Fol/.xmhcache]]
       || [file mtime [RF $MH(Path)/$Fol]] > \
  [file mtime [RF $MH(Path)/$Fol/.xmhcache]]} {
 if [catch {exec $MH(bindir)/scan +$Fol} Msgs] {
    Cache:Write [RF $MH(Path)/$Fol/.xmhcache] {}
    del_seq $Fol allmsgs
    if [winfo exists $w] {$w.list delete 0 end}
    return 0
}
set Msgs [split $Msgs "\n"]
set Msgnums [exec $MH(bindir)/pick]
Cache:Write [RF $MH(Path)/$Fol/.xmhcache] $Msgs
write_seq $Fol allmsgs
    } else {
set Msgs [Cache:Read [RF $MH(Path)/$Fol/.xmhcache]]
write_seq $Fol allmsgs
    }

    set curYview [lindex [$w.scroll get] 2]
    $w.list delete 0 end
    foreach Msg $Msgs {$w.list insert end $Msg}
    $w.list yview $curYview
    return [llength Msgs]
}

# CreateFol {}
# Create the Folder with some checks. returns List of folders it created

proc CreateFol {} {
    global MH OScmd FolList

  if {[set NewFol [Ask {Folder to Create?}]] != {}} {
    if {[string first { } $NewFol] != -1} {
MesgWin "No spaces allowed in foldernames"
return
    }
    set dirs {}
    set dummy {}
    foreach name [split $NewFol "/"] {
if {$dummy == {}} {set dummy $name} \
else {set dummy "$dummy/$name"}
lappend dirs $dummy
```

```
    }
    foreach dir $dirs {
if [file isfile [RF $MH(Path)/$dir]] {
    MesgWin "$dir already exists as a file"
    return
}
if [file isdirectory [RF $MH(Path)/$dir]] {continue}
eval exec $OScmd(mkdir) [RF $MH(Path)/$dir]
lappend FolList $dir
    }
    set FolList [lsort $FolList]
    Cache:Write [RF $MH(Path)/.folders] $FolList
    DisplayFols
  }
}

# PackFol {Fol}
# Renumber the files in the folder so that they will go from
# 1-nummessages, also delete files starting with a , which are
# backup files from deleted messages.
# TODO: implement it
proc PackFol {} {
    global MH Fol
    catch {exec $MH(bindir)/folder -pack +$Fol}
    ScanFol $Fol
}

# RemoveFol {}
# Remove the folder, ask for confirmation
proc RemoveFol {} {
    global MH OScmd FolList Fol

    set w .pane.pane2
    if {$Fol == $MH(Inbox) || $Fol == $MH(Draft-Folder)} {
MesgWin "You can't delete your drafts or inbox folders!\n"
return 0
    }
    if [Confirm "Remove $Fol?"] {
eval exec $OScmd(rmdir) [RF $MH(Path)/$Fol]
set i [lsearch -exact $FolList "$Fol"]
set FolList [lreplace $FolList $i $i]
while {[set i [lsearch -regexp $FolList "^$Fol/"]] != -1} {
  set FolList [lreplace $FolList $i $i]
}
Cache:Write [RF $MH(Path)/.folders] $FolList
DisplayFols
```

```
$w.list delete 0 end
return 1
    }
    return 0
}

# SortFol {Fol} then Scan folder.

proc SortFol {} {
    global MH Fol
    catch {exec $MH(bindir)/sortm +$Fol -noverbose}
    ScanFol $Fol
}

proc RenameFol {} {
    global MH OScmd FolList Fol

    set w .top.mn.m2
    set NewName [Change Folder $Fol "Rename to" {} 25]
    if {$NewName == {}} {return 0}
    if {[lsearch -exact $FolList "$NewName"] != -1} {
if {![Confirm "$NewName already exists!
    \n\nMove this folder to $NewName?\n"]} {
   return 0
}
    }
    eval exec $OScmd(mv) [RF $MH(Path)/$Fol] [RF $MH(Path)/$NewName]
    set i [lsearch -exact $FolList "$Fol"]
    set FolList [lreplace $FolList $i $i]
    while {[set i [lsearch -regexp $FolList "^$Fol/"]] != -1} {
      regsub "^$Fol\(.*\)$" [lindex $FolList $i] "$NewName\\1" REGSUB
      set FolList [lreplace $FolList $i $i $REGSUB]
    }
    lappend FolList $NewName
    set FolList [lsort $FolList]
    Cache:Write [RF $MH(Path)/.folders] $FolList
    DisplayFols
    set Fol $NewName
    OpenFol $Fol
    $w.l configure -text $Fol
    return 1
}

# read_seq {Fol seqname}
# returns the sequence $seqname list from $Fol
```

```
proc read_seq {Fol seqname} {
    global MH
    if {[catch {open [RF $MH(Path)/$Fol/.mh_sequences] r} in] == 0} {
while {[gets $in Line] != -1} {
    if [regexp "^$seqname:" $Line] {
set seq [lrange $Line 1 end]
close $in
return $seq
    }
}
close $in
    }
    return 0
}

# write_seq {Fol seqname {List {}}}
# writes the messages from list to the sequence. if list is empty, it
# will write all messages. returns 0 on success, else a number

proc write_seq {Fol seqname {List {}}} {
    global MH
    catch {eval exec $MH(bindir)/pick +$Fol -sequence $seqname $List}
}

# del_seq {Fol seqname}
# remove the sequence seqname

proc del_seq {Fol seqname} {
    global MH
    catch {exec $MH(bindir)/mark -delete all -sequence $seqname}
}

# append_seq {Fol seqname seq}

proc append_seq {Fol seqname seq} {
    global MH
    catch {exec $MH(bindir)/mark -add $seq -sequence $seqname}
}

# del_from_seq {Fol seqname seq}

proc del_from_seq {Fol seqname seq} {
    global MH
    catch {exec $MH(bindir)/mark -delete $seq -sequence $seqname}
}
```

```
# conv_seq {Fol seqname}
# output a list of Message numbers, corresponding to the entries
# in $seqname. returns nothing if seq doesn't exist...

proc conv_seq {Fol seqname} {
    global MH
    if [catch {eval exec $MH(bindir)/pick +$Fol \
       [read_seq $Fol $seqname]} Msgnums] {
return {}
    } else {
return [split $Msgnums "\n"]
    }
}

# conv_listbox_seq {Fol seqname}
# convert the listbox selection in the folder window belonging to
# $Fol to a sequence in .mh_sequences in $Fol. return 0 if no
# selection

proc conv_listbox_seq {Fol seqname} {
    if {[set indices [.pane.pane2.list curselection]] == {}} {return 0}
    set first [lindex $indices 0]
    set last [lindex $indices [expr [llength $indices] - 1]]
    write_seq $Fol $seqname [lrange [conv_seq $Fol allmsgs] $first $last]
    return 1
}

# seq_index {Fol seqname}
# return the indices of the messages in $seqname for further use in
# listboxes and lists.

proc seq_index {Fol seqname} {
    set msgs [conv_seq $Fol $seqname]
    set all [conv_seq $Fol allmsgs]
    set indices {}
    foreach msg $msgs {
if {[set i [lsearch -exact $all $msg]] != -1} {lappend indices $i}
    }
    return $indices
}

# Ask {Question}

proc Ask {{Question "What?"}} {
    global Result
    global Font
```

```
    toplevel .ask
    wm title .ask Question
    wm geometry .ask +15-15
    frame .ask.e -relief raised -borderwidth 2
    message .ask.e.msg -font $Font(helvb) -text "  $Question" \
    -anchor w -width 300
    entry .ask.e.e -width 40
    frame .ask.b
    button .ask.b.ok -text "OK" -command \
      {set Result [.ask.e.e get]; destroy .ask}
    button .ask.b.cancel -text "Cancel" -command {set Result {};\
      destroy .ask}

    pack append .ask.e \
      .ask.e.msg {top  fillx pady 10} \
      .ask.e.e {top pady 20 padx 30}
    pack append .ask.b \
      .ask.b.ok {top expand fill} \
      .ask.b.cancel {top expand fill}
    pack append .ask \
      .ask.e {left filly} \
      .ask.b {left filly}

    focus .ask.e.e
    bind .ask.e.e <Return> {.ask.b.ok invoke}
    bind .ask.e.e <Space> {}
    grab .ask
    tkwait window .ask
    return $Result
}

# Confirm {Msg}

proc Confirm {{Msg "Sure about that?"}} {
    global Result
    global Font

    toplevel .confirm
    wm title .confirm Confirm
    wm geometry .confirm +15-15
    frame .confirm.m -relief raised -borderwidth 2
    message .confirm.m.m -font $Font(helvb) -text $Msg -width 200 -anchor w
    frame .confirm.b
    button .confirm.b.yes -text "Yes" -width 8 \
      -command {set Result 1; destroy .confirm}
```

```
    button .confirm.b.no -text "No" -width 8 \
      -command {set Result 0; destroy .confirm}

    pack append .confirm.m \
      .confirm.m.m {top padx 20 pady 20}
    pack append .confirm.b \
      .confirm.b.yes {top expand fill} \
      .confirm.b.no {top expand fill}
    pack append .confirm \
      .confirm.m {left expand fill} \
      .confirm.b {left expand fill}

    focus .confirm
    bind .confirm <Return> {.confirm.b.ok invoke}
    grab .confirm
    tkwait window .confirm
    return $Result
}

# Change {Operation Label Entry Default length}
# Popup a window that with four 6 widgets. Three labels, Operation,
# Label, and Entry, describing what to do, to what, what to enter.
# one entry field, contents are returned after ok or <return>
# two buttons, ok and cancel. length is the length of the entry field
#
proc Change {Operation Label Entry Default {length 40}} {
    global Result
    global Font

    toplevel .ea
    wm title .ea "$Operation"
    wm geometry .ea +15-15
    frame .ea.b
    button .ea.b.cancel -text "Cancel" \
      -command "set Result {}; destroy .ea"
    button .ea.b.ok -text "OK" \
      -command {set Result [.ea.e.f2.e1 get]; destroy .ea}
    frame .ea.e -relief raised -borderwidth 2
    frame .ea.e.f1
    label .ea.e.f1.l1 -text "$Operation:" -anchor e -font $Font(helvb)
    label .ea.e.f1.l2 -text "$Entry:" -anchor e -font $Font(helvb)
    frame .ea.e.f2
    label .ea.e.f2.l1 -text $Label -font $Font(helvb)
    entry .ea.e.f2.e1 -width $length -font $Font(helvm)

    pack append .ea.e.f1 \
```

```
    .ea.e.f1.l1 {top fill} \
    .ea.e.f1.l2 {top fillx pady 10}
  pack append .ea.e.f2 \
    .ea.e.f2.l1 {top fill} \
    .ea.e.f2.e1 {top fillx pady 10}
  pack append .ea.e \
    .ea.e.f1 {left padx 10 pady 20} \
    .ea.e.f2 {left padx 20 pady 20}
  pack append .ea.b \
    .ea.b.ok {top expand fill} \
    .ea.b.cancel {top expand fill}
  pack append .ea \
    .ea.e {left expand fill} \
    .ea.b {left expand fill}

  bind .ea.e.f2.e1 <Return> {.ea.b.ok invoke}
  .ea.e.f2.e1 insert end $Default
  grab .ea
  focus .ea.e.f2.e1
  tkwait window .ea
  return $Result
}
```

APPENDIX C

TCL/TK SCRIPT FOR CREATING SCENARIO CODE

```
#!/contrib/bin/wish -f
set tksce_lib /z/tahar/contrib/lib/tksce
lappend auto_path $tksce_lib
#
# Program for Creating Scenario
#
InitScen
ScenWin

#
# Initialization
#
proc InitScen {} {
    global tksce_lib

    uplevel #0 source $tksce_lib/tksce.rc
    ReadConfTksce
}
#
# The Main Window For Creating Scenario Code
#
proc ScenWin {} {
    global wait par clp Font Dxs Dys Xsr Ysr
    global Lop Losce pos
    set wait "WAIT"
    set par "PAR{"
    set clp "}"
    set xs 1000;  # size of screen 1000x760
    set ys 760
    set Dxs 10;   # right, left, and bottom margin
    set Dys 63;   # top margin
    set Xsr [expr $xs - 2*$Dxs];  # the working area size
    set Ysr [expr $ys - $Dys - $Dxs]
    set pos(nw) +10+63
    set pos(n) +300+63
    set pos(ne) -10+63
    set pos(e) -10+300
    set pos(c) +300+300
    set pos(se) +10-10
    set pos(s) +300-10
    set pos(sw) -10-10
    set pos(w) +10+300
    wm geometry . +10-10
    wm geometry . 350x300
    wm title . "Creating Scenario"
    wm iconname . "CreateSce"
```

```
    Reset_Lsce
    Make_ListPart

    frame .b1 -borderwidth 1
    frame .tcode -borderwidth 1
    frame .b2 -borderwidth 10

    button .b1.ifthen -text "Ifnot-then" -height 1 -width 10 -command \
    "Sce_Ifthen"
    button .b1.seq -text "Sequential" -height 1 -width 10 \
    -command "Sce_Seq"
    button .b1.par -text "Parallel" -height 1 -width 10 -command "Sce_Par"
    button .b1.show -text "Showcode" -height 1 -width 10 -command \
    "Sce_Showcode"

    text .tcode.txt -relief raised -bd 2 -yscrollcommand ".tcode.scr set" \
      -setgrid true -width 52 -height 10 -font $Font(helvm)
    scrollbar .tcode.scr -relief flat -command ".tcode.txt yview"

    button .b2.cancel -text cancel -height 1 -width 8 -command \
    "Reset_Lsce; destroy ."
    button .b2.done -text Done -height 1 -width 8 -command \
    "Make_FileSce; destroy ."

    pack append .b1 \
      .b1.ifthen {left pady 10 padx 10}\
      .b1.seq {left padx 10} \
      .b1.par {left padx 10} \
      .b1.show {left padx 10}

    pack append .tcode \
      .tcode.txt {left expand fill} \
      .tcode.scr {left filly}

    pack append .b2 \
      .b2.cancel {left pady 5 padx 50}\
      .b2.done {left padx 40}

    pack .b1 -side top -expand yes
    pack .tcode -side top -expand yes
    pack .b2 -side bottom -expand yes
}

# Procedure to reset (empty) the lists
```

```
proc Reset_Lsce {} {
    global Lop Losce
    set Lop {}
    set Losce {}
}

# Procedure to make list of bodyparts from the file

proc Make_ListPart {} {
    global Lop
    if [file isfile tpart] {
       set fd [open tpart r]
       while {[gets $fd line] >= 0} {
  lappend Lop $line
       }
       close $fd
    } else {MesgWin "No Bodyparts File!"}
}

# Procedure to create code of the statement IFNOT-THEN

proc Sce_Ifthen {{w .int}} {
    global Lop Losce Font ifthen Tmplist
    set ifnot "IFNOT"; set then "THEN"
    Res_List;  # set to empty
    catch {destroy $w}
    toplevel $w
    wm geometry $w +150+330
    wm title $w "Ifnot-Then"
    wm iconname $w "Ifnot-Then"
    wm minsize $w 1 1

    frame $w.c
    frame $w.c.b -borderwidth 10
    frame $w.c.c -borderwidth 10

    label $w.d -text {} -relief raised \
     -font $Font(helvb) -anchor w

    button $w.c.b.if -text IFNOT -height 1 -width 8 -command \
    "set ifthen $ifnot; set ok 0; grab $w.c.c"
    button $w.c.b.then -text THEN -height 1 -width 8 -command \
    "Cat_Then; set ok 1; grab $w.c.c"
    button $w.c.b.ok -text Done -height 1 -width 8 -command \
    "Cat_List; Res_List; destroy $w"
    button $w.c.b.cancel -text Cancel -height 1 -width 8 -command \
```

```
    "Res_List; destroy $w"

    scrollbar $w.c.c.scroll -relief sunken -command "$w.c.c.list yview"
    listbox $w.c.c.list -yscroll "$w.c.c.scroll set" -relief sunken \
    -geometry 35x10 -setgrid 1

    pack append $w.c.b \
      $w.c.b.if {top pady 10 padx 10}\
      $w.c.b.then {top pady 10}\
      $w.c.b.ok {top pady 10}\
      $w.c.b.cancel {top pady 10}

    pack append $w.c.c \
      $w.c.c.list {left expand fill}\
      $w.c.c.scroll {left filly}

    pack append $w.c \
      $w.c.c {left expand fill} \
      $w.c.b {left}

    pack $w.c -side top -expand yes -fill x
    pack $w.d -side bottom -fill x

    foreach part [lrange $Lop 0 end] {
       $w.c.c.list insert end $part
    }
    set subs "SUBSTITUTE"
    $w.c.c.list insert end $subs

    bind $w.c.c.list <1> "
      %W select from \[%W nearest %y\] "
    bind $w.c.c.list <Double-1> {foreach partnm [selection get] \
       { FlashSel .int.c.c.list 2
set pname [lindex $partnm 0]
if {$pname != "SUBSTITUTE"} {
    if {[set Geo [Sce_Geom]] != {}} {
       set pname [concat $pname/$Geo]
       set ifthen [concat $ifthen $pname]
    } else {set ifthen [concat $ifthen $pname] }
} else {set ifthen [concat $ifthen $pname] }
.int.d configure -text $ifthen
grab release .int.c.c
if {$ok == 1} {
    lappend Tmplist $ifthen
    set ifthen {}
    grab release .int.c.b.then
```

```
 } elseif {$ok == 0} {
    grab .int.c.b.then
 }}}

    grab $w
    tkwait window $w
}

# Procedure concatenation THEN

proc Cat_Then {} {
    global ifthen
    set ifthen [concat $ifthen "THEN"]
}

# Procedure to create the code of the Sequential statements

proc Sce_Seq {{w .seq}} {
    global Lop Losce wait waitval Font Tmplist
    set waitval 0
    Res_List;
    catch {destroy $w}
    toplevel $w
    wm geometry $w +150+330
    wm title $w "Sequential"
    wm iconname $w "Sequential"
    wm minsize $w 1 1

    frame $w.c
    frame $w.c.b -borderwidth 10

    button $w.c.b.wait -text WAIT -height 1 -width 8 -command \
    "Sce_Wait"
    button $w.c.b.ok -text Done -height 1 -width 8 -command \
    "Cat_List; Res_List; destroy $w"
    button $w.c.b.cancel -text Cancel -height 1 -width 8 -command \
    "Res_List; destroy $w"

    scrollbar $w.c.scroll -relief sunken -command "$w.c.list yview"
    listbox $w.c.list -yscroll "$w.c.scroll set" -relief sunken \
    -geometry 35x10 -setgrid 1

    label $w.d -text {} -relief raised \
     -font $Font(helvb) -anchor w

    pack append $w.c.b \
```

```
      $w.c.b.wait {top pady 10 padx 10}\
      $w.c.b.ok {top pady 10}\
      $w.c.b.cancel {top pady 10}

    pack append $w.c \
      $w.c.list {left expand fill}\
      $w.c.scroll {left filly}\
      $w.c.b {left filly}

    pack $w.c -side top -expand yes -fill x
    pack $w.d -side bottom -fill x

    foreach part [lrange $Lop 0 end] {
       $w.c.list insert end $part
    }
    set null NULL
    $w.c.list insert end $null

    bind $w.c.list <1> "
      %W select from \[%W nearest %y\] "
    bind $w.c.list <Double-1> {foreach partnm [selection get] \
      { FlashSel .seq.c.list 2
set pname [lindex $partnm 0]
if {$pname != "NULL"} {
   if {[set Geo [Sce_Geom]] != {}} {
     set pname [concat $pname/$Geo]
   }
}
if {$waitval == 0} {
  lappend Tmplist $pname
  .seq.d configure -text $pname
} else {
  set await [concat $wait\($waitval,$pname\)]
  lappend Tmplist $await
  .seq.d configure -text $await
  set waitval 0
}
       }}

    grab $w
    tkwait window $w
}

# Procedure to create the code of the parallel statement

proc Sce_Par {{w .par}} {
```

```
global Lop Losce par clp wait waitval Font Tmplist
set waitval 0
Res_List;
lappend Tmplist $par
catch {destroy $w}
toplevel $w
wm geometry $w +150+330
wm title $w "Parallel"
wm iconname $w "Parallel"
wm minsize $w 1 1

frame $w.c
frame $w.c.b -borderwidth 10

button $w.c.b.wait -text WAIT -height 1 -width 8 -command "Sce_Wait"
button $w.c.b.ok -text Done -height 1 -width 8 -command \
"Cat_List; Res_List; lappend Losce $clp; destroy $w"
button $w.c.b.cancel -text Cancel -height 1 -width 8 -command \
"Res_List; destroy $w"

scrollbar $w.c.scroll -relief sunken -command "$w.c.list yview"
listbox $w.c.list -yscroll "$w.c.scroll set" -relief sunken \
-geometry 35x10 -setgrid 1

label $w.d -text {} -relief raised \
-font $Font(helvb) -anchor w

pack append $w.c.b \
  $w.c.b.wait {top pady 10 padx 10}\
  $w.c.b.ok {top pady 10}\
  $w.c.b.cancel {top pady 10}

pack append $w.c \
  $w.c.list {left expand fill}\
  $w.c.scroll {left filly}\
  $w.c.b {left filly}

pack $w.c -side top -expand yes -fill x
pack $w.d -side bottom -fill x

foreach part [lrange $Lop 0 end] {
   $w.c.list insert end $part
}

bind $w.c.list <1> "
  %W select from \[%W nearest %y\] "
```

```
    bind $w.c.list <Double-1> {foreach partnm [selection get] \
       { FlashSel .par.c.list 2
 set pname [lindex $partnm 0]; \
 if {[set Geo [Sce_Geom]] != {}} {
    set pname [concat $pname/$Geo]
 }
 if {$waitval == 0} {
    lappend Tmplist $pname
    .par.d configure -text $pname
 } else {
    set await [concat $wait\($waitval,$pname\)]
    lappend Tmplist $await
    .par.d configure -text $await
    set waitval 0
 }
       }}

    grab $w
    tkwait window $w
}

# Procedure to get the value (in seconds) of WAIT statement

proc Sce_Wait {} {
    global waitval Font
    set w .scale2
    catch {destroy $w}
    toplevel $w
    wm geometry $w +650+400
    wm title $w "Wait"
    wm iconname $w "Wait"

    label $w.msg -font $Font(norm) \
    -text "Wait in second." -anchor center
    frame $w.b -borderwidth 10
    frame $w.s -borderwidth 10

    button $w.b.ok -text OK -height 1 -width 8 -command \
    "Settime $w; destroy $w"
    button $w.b.cancel -text Cancel -height 1 -width 8 -command \
    "destroy $w"

    scale $w.s.scale -orient horizontal -length 270 -from 0 -to 60 \
    -tickinterval 10 -bg Bisque1\
    -bg Bisque1
    pack $w.s.scale -side bottom -expand yes -anchor nw
```

```
    $w.s.scale set 0

    pack append $w.b \
       $w.b.ok {left pady 10 padx 50}\
       $w.b.cancel {left padx 30}

    pack append $w \
       $w.msg {top fillx}\
       $w.s   {top fillx}\
       $w.b   {top expand fillx}

    grab $w
    tkwait window $w
}

# Assign the value to variable waitval

proc Settime {w} {
   global waitval
   set waitval [$w.s.scale get]
}

# Procedure to get Geometry string

proc Sce_Geom {{w .geo}} {
    global geoval pos Font
    global width height Dxs Dys Xsr Ysr
    catch {destroy $w}
    toplevel $w
    wm geometry $w +400+300
    wm title $w "Geometry"
    wm iconname $w "Geometry"
    set c $w.c
    set width 0; set height 0;

    frame $w.e -relief raised -borderwidth 2
    frame $w.e.d -relief raised -borderwidth 1
    frame $w.e.b -relief raised -borderwidth 1
    label $w.e.l -text "Dimension" -font $Font(helvb)
    label $w.e.d.l1 -text "Width:" -font $Font(helvb)
    entry $w.e.d.e1 -font $Font(helvm) -width 5
    label $w.e.d.l2 -text "Height:" -font $Font(helvb)
    entry $w.e.d.e2 -font $Font(helvm) -width 5
    button $w.e.b.ok -text OK -width 4 -command \
    "set width \[$w.e.d.e1 get\]
```

```
   set height \[$w.e.d.e2 get\]
   Dimension $w
   Make_Squares $c"

  button $w.e.b.no -text NO -width 4 -command \
  "set pos(n) +300+63; set pos(e) -10+300
   set pos(s) +300-10; set pos(w) +10+300
   set pos(c) +300+300
   Make_Squares $c"

  canvas $c -relief raised -width 220 -height 140

  frame $w.ent -relief raised -borderwidth 2
  label $w.ent.l -text "Geometry:" -font $Font(helvb)
  entry $w.ent.e -font $Font(helvm) -width 15

  frame $w.bot -relief raised -border 1
  frame $w.bot.mid -relief sunken -border 1
  button $w.bot.mid.ok -text OK -width 4 -command \
  "set geoval \[$w.ent.e get\];
   set pos(n) +300+63; set pos(e) -10+300
   set pos(s) +300-10; set pos(w) +10+300
   set pos(c) +300+300
   destroy $w"

  frame $w.bot.r -relief sunken -border 1
  button $w.bot.r.cancel -text Cancel -width 8 -command \
  "set geoval {};
   set pos(n) +300+63; set pos(e) -10+300
   set pos(s) +300-10; set pos(w) +10+300
   set pos(c) +300+300
   destroy $w"

pack $w.e $w.c $w.ent $w.bot -side top -expand yes -fill both
pack append $w.e \
   $w.e.l {top} \
   $w.e.d {top fillx} \
   $w.e.b {top fillx}

pack append $w.e.b \
   $w.e.b.ok {left expand padx 10 pady 5}\
   $w.e.b.no {left expand padx 10 pady 5}

pack append $w.e.d \
   $w.e.d.l1 {left} \
   $w.e.d.e1 {left} \
```

```
      $w.e.d.l2 {left} \
      $w.e.d.e2 {left}

   pack append $w.ent \
      $w.ent.l {left} \
      $w.ent.e {left padx 10 pady 10}

   pack append $w.bot \
      $w.bot.mid {left} \
      $w.bot.r {left padx 10 pady 10}

   pack $w.bot.mid -expand yes -padx 10 -pady 10
   pack $w.bot.mid.ok -expand yes -padx 6 -pady 6

   pack $w.bot.r -expand yes -padx 10 -pady 10
   pack $w.bot.r.cancel -expand yes -padx 6 -pady 6

   bind $w.ent.e <Key-Return> "$w.bot.mid.ok invoke"

   Make_Squares $c;

   focus $w.e.d.e1
   focus $w.e.d.e2
   focus $w.ent.e
   grab $w
   tkwait window $w
   return $geoval
}

# Making 9 squares as a visualization of the default positions

proc Make_Squares {c} {
   global pos
   set x 70
   set y 40
   set color LightSkyBlue1
   Geo_pos $c $x $y $pos(nw) $color
   Geo_pos $c [expr $x+30] [expr $y] $pos(n) $color
   Geo_pos $c [expr $x+60] [expr $y] $pos(ne) $color
   Geo_pos $c [expr $x] [expr $y+30] $pos(w) $color
   Geo_pos $c [expr $x+30] [expr $y+30] $pos(c) $color
   Geo_pos $c [expr $x+60] [expr $y+30] $pos(e) $color
   Geo_pos $c [expr $x] [expr $y+60] $pos(se) $color
   Geo_pos $c [expr $x+30] [expr $y+60] $pos(s) $color
   Geo_pos $c [expr $x+60] [expr $y+60] $pos(sw) $color
   set item [$c create rect [expr $x+40] [expr $y+40] \
```

```
    [expr $x+50] [expr $y+50] -outline black -fill red]

   $c create text [expr $x+40] [expr $y-7] -text {Position} -anchor s \
    -fill brown
   $c bind config <Enter> "geoEnter $c"
   $c bind config <Leave> "$c itemconf current -fill \$geoConfigFill"
   set geoConfigFill {}
}

# Calculate the dimension to determine the suitable position

proc Dimension {w} {
   global pos width height Xsr Ysr Dxs Dys
   set subhxw [expr round(0.5*($Xsr-$width)) + $Dxs]
   set subhyh [expr round(0.5*($Ysr-$height)) + $Dys]
   set pos(n) [concat +$subhxw+$Dys]
   set pos(s) [concat +$subhxw-$Dxs]
   set pos(w) [concat +$Dxs+$subhyh]
   set pos(e) [concat -$Dxs+$subhyh]
   set pos(c) [concat +$subhxw+$subhyh]
   set wh [concat $width\x$height]; $w.ent.e delete 0 end
   $w.ent.e insert end $wh
}

# Getting the geometry string to be assigned to geoval

proc Geo_pos {w x y geoval color} {
    set item [$w create rect [expr $x] [expr $y] [expr $x+30] \
    [expr $y+30] -outline black -fill $color -width 1]
    $w bind $item <1> "
       .geo.ent.e insert end $geoval"
    $w addtag config withtag $item
}

proc geoEnter {w} {
    global geoConfigFill
    set geoConfigFill [lindex [$w itemconfig current -fill] 4]
    $w itemconfig current -fill black
}

# Procedure append 2 lists

proc Cat_List {} {
    global Losce Tmplist
    foreach part [lrange $Tmplist 0 end] {
       lappend Losce $part
```

```
    }
}

# Reset temporary list

proc Res_List {} {
    global Tmplist
    set Tmplist {}
}

# Procedure to show scenario code

proc Sce_Showcode {} {
    global Losce
    set w .tcode
    if {$Losce != {}} {
       $w.txt delete 0.0 end
       foreach line [lrange $Losce 0 end] {
  $w.txt insert end $line
  $w.txt insert end "\n"
       }
    } else {
       set line "No code!"
       $w.txt delete 0.0 end
       $w.txt insert end $line
    }
}

# Procedure to make temporary file of scenario code

proc Make_FileSce {} {
   global Losce
   set fd [open tsce.sce w]
   foreach line [lrange $Losce 0 end] {
      puts $fd $line
   }
   close $fd
}

# Procedure to attach the code to the component form

proc Sce_Attach {} {
    global Font
    set w .form
    toplevel $w
    wm geometry $w +360+350
```

```
    wm title $w "SceCode Attachment"
    wm iconname $w "attach"

    set formfl 0
    frame $w.top -relief raised -border 1
    frame $w.bot -relief raised -border 1

    message $w.top.name -font $Font(norm) -text "Attach code to Form?" \
    -width 300 -justify left
    button $w.bot.ok -text OK -height 2 -width 8 -command \
     "AttachOk; destroy $w"
    button $w.bot.cancel -text Cancel -height 2 -width 8 -command \
    "destroy $w"

    pack $w.top $w.bot -side top -fill both -expand yes
    pack $w.top.name -side left
    pack $w.bot.ok -side left -pady 10 -padx 30
    pack $w.bot.cancel -side left -pady 10 -padx 10

    tkwait visibility $w
    grab -global $w
    tkwait window $w
}

# Ask for Attachment wether or not it is OK.

proc AttachOk {} {
    global Form
    if {$Form == "send"} {
       if [file isfile tform] {
          set formfl [open tform a]
       } else {MesgWin "No Component File!"; return}
    } elseif {$Form == "reply"} {
       if [file isfile trform] {
          set formfl [open trform a]
       } else {MesgWin "No Component File!"; return}
    }
    if [file isfile tsce.sce] {
       set first "#application/x-scenario \[Scenario Code] \\"
       puts $formfl $first
       set second [concat [pwd]/tsce.sce]
       puts $formfl $second
       close $formfl
    } else {MesgWin "No Scenario Code!"; close $formfl}
}
```

APPENDIX D

CODE INTERPRETER

The scenario code interpreter and the Makefile are given in this section.

```
#
# Makefile for compiling interpreter and other programs.
#
all: scenario

scenario:  showsce testsce showpar

showsce.o testsce.o showpar.o: sce.h

showsce: showsce.o prolist.o checkpar.o
cc -o $@ showsce.o prolist.o checkpar.o

testsce: testsce.o prolist.o checkpar.o
cc -o $@ testsce.o prolist.o checkpar.o

showpar: showpar.o prolist.o
cc -o $@ showpar.o prolist.o

clean:
rm *.o \
showsce testsce showpar \

/*-----------------------------------------------------------------*
 *  Header for all programs.
 *  File name: sce.h
 *-----------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <ctype.h>
#include <stdlib.h>
#include <sys/types.h>

#define  MAX_L  300  /* max line */
#define  MAX_ARG 10  /* max number of arguments */
#define  MAX_P   10  /* max number of parallel parts */
#define  MAX_F   10  /* max number of files */

/* data structure for listing of parallelized bodyparts */

struct parallel{
   int wait_t;                  /* waiting time */
   char part[50];               /* bodypart name presented */
   struct parallel *next;       /* link to the next */
```

```
};

typedef struct parallel PARA;
typedef PARA            *LINK;

/* data structure for listing of available devices/players */

struct profile{
   char type[25];               /* content type */
   char subtype[25];            /* content subtype */
   char dir[70];                /* location of player */
   char player[15];             /* player name */
   char arg[MAX_ARG][35];       /* arguments if any */
   int  narg;                   /* number of arguments */
   struct profile *next;        /* link to the next */
};

typedef struct profile  PROF;
typedef PROF            *FLINK;

/* data structure for listing of parallelized bodyparts */

struct par{
   char part[70];          /* bodypart name presented */
   struct par *next;       /* link to the next */
};

typedef struct par PAR;
typedef PAR            *PLINK;

/*-------------------------------------------------------------------*
 *  This Program interprets scenario codes.
 *  File name: showsce.c
 *-------------------------------------------------------------------*/
#include "sce.h"

void par_list();
void pro_list();
void Par();
int Seq();
int check_device();
void ShowMsg();
FILE *fpr, *fpp;          /* file discriptors */
char StoDir[50];          /* storage directory */
char Msg[7];              /* message number */
int  Txt1;                /* text initialization in message */
```

```
char MsgDir[50];          /* directory of the message window */

main(argc, argv)
int argc;
char *argv[];
{
   char fnr[45], targv1[20];
   char s[MAX_L];                /* string var */
   char *ptr;
   char part[50], atime[4];
   char *getenv(), profile[35];
   char mesg[80];
   int  time, n_part;
   LINK head_par=NULL;           /* head of linked list */
   FLINK head_pro=NULL;
   int ok;

   if(argc!=4){
      printf("Usage: command <scenario code> <textpos> <msgW dir>\n");
      exit(1);
   }

   /* open file "$HOME/.mh_profile" */

   if ((ptr = getenv("HOME")) == (char *) 0){
       printf("HOME is not defined\n");
       exit(0);
   }else
       sprintf(profile,"%s/.mh_profile",ptr);
   if((fpp = fopen(profile, "r")) == NULL){
      printf("\nFile opening error --- program terminated.\n");
      exit(1);
   }

   /* Creating the list of available devices/players */

   strcpy(StoDir,"");
   while(fgets(s,MAX_L,fpp)!=NULL){
      if(!strncmp(s,"mhn-show",8)) pro_list(&head_pro, s);
      else if(!strncmp(s,"mhn-storage",11)){
         ptr = strtok(s, " :");
         ptr = strtok(ptr+strlen(ptr)+1," \n");
         strcpy(StoDir,ptr);
      }
   }
   fclose(fpp);
```

```
Txt1 = atoi(argv[2]);
strcpy(MsgDir,argv[3]);
strcpy(targv1, argv[1]);
ptr = strtok(targv1,".");
strcpy(Msg, ptr);
if(strcmp(StoDir,"")){
   sprintf(fnr,"%s/%s",StoDir,argv[1]);
}else strcpy(fnr,argv[1]);
if((fpr = fopen(fnr, "r")) == NULL){
   printf("\nFile opening error --- program terminated.\n");
   exit(1);
}

/* Interpreting and Executing the scenario code */

while(fgets(s,MAX_L,fpr)!=NULL){
   if(!strncmp(s,"IFNOT",5)){
      ptr = strtok(s, " ");
      ptr = strtok(ptr+strlen(ptr)+1," ");
      strcpy(part, ptr);
      if(!Seq(head_pro,part)){
         ptr = strtok(ptr+strlen(ptr)+1," ");
         ptr = strtok(ptr+strlen(ptr)+1," ;\n");
         if(!strcmp(ptr,"SUBSTITUTE")){
            sprintf(mesg,"No Player for: %s", part);
            ShowMsg(mesg);
         }else{
            strcpy(part, ptr);
            if(!Seq(head_pro,part)){
               sprintf(mesg,"No Player for: %s", part);
               ShowMsg(mesg);
            }
         }
      }
   }else if(!strncmp(s,"WAIT",4)){
      ptr = strtok(s," (");
      ptr = strtok(ptr+strlen(ptr)+1," ,);");
      strcpy(atime, ptr);
      time = atoi(atime);
      ptr = strtok(ptr+strlen(ptr)+1," ,);");
      strcpy(part, ptr);
      sleep(time);                    /* waiting time */
      if(strcmp(part,"NULL")){
         if(!Seq(head_pro,part)){
            sprintf(mesg,"No Player for: %s", part);
```

```
                ShowMsg(mesg);
            }
        }
    }else if(!strncmp(s,"PAR",3)){
        n_part = 0;
        while(strcmp(ptr,"}\n")){
            fgets(s,MAX_L,fpr);
            ptr = strtok(s, " (;)");
            if(!strcmp(ptr,"WAIT")){
                ptr = strtok(ptr+strlen(ptr)+1," ,);");
                strcpy(atime, ptr);
                time = atoi(atime);
                ptr = strtok(ptr+strlen(ptr)+1," ,);");
                strcpy(part, ptr);
                n_part++;
                par_list(&head_par, time, part);
            }else if(strcmp(ptr,"}\n")){
                strcpy(part, ptr);
                n_part++;
                par_list(&head_par, 0, part);
            }
        }
        Par(&head_par, head_pro, n_part);
    }else{
        ptr = strtok(s, "\n");
        strcpy(part, ptr);
        if(!Seq(head_pro,part)){
            sprintf(mesg,"No Player for: %s", part);
            ShowMsg(mesg);
        }
    }
  }
  fclose(fpr);
}
/*
 * Procedure to interpret and execute sequential statement
 */
int Seq(head_pro, part)
FLINK head_pro;
char *part;
{
  char *ptr, *ptr1, temp[15], name[50];
  char *args[MAX_ARG];
  char *argd[MAX_ARG];
  char dir[70], ply[15];
  int childpid, pid;
```

```
int status;
int  i, j, npart;
int  nargd;

if(check_device(head_pro, part, dir, ply, &nargd, argd)){
   ptr = strtok(part, "/");
   ptr = strtok(ptr+strlen(ptr)+1,"/");
   ptr = strtok(ptr+strlen(ptr)+1," /;\n");
   strcpy(temp, ptr);
   ptr1 = strtok(temp, ". ");
   if(!Txt1){
      npart = atoi(ptr1) - 1;
      ptr1 = strtok(ptr1+strlen(ptr1)+1," ");
      if(strcmp(StoDir,"")){
         sprintf(name,"%s/%s.%d.%s",StoDir, Msg, npart, ptr1);
      }else sprintf(name,"%s.%d.%s",Msg, npart, ptr1);
   }else{
      if(strcmp(StoDir,"")){
         sprintf(name,"%s/%s.%s",StoDir, Msg, ptr);
      }else sprintf(name,"%s.%s",Msg, ptr);
   }
   ptr = strtok(ptr+strlen(ptr)+1," ;\n");
   j = 0;
   args[j] = calloc(strlen(ply)+1, sizeof(char));
   strcpy(args[j],ply);
   j++;
   if(nargd > 0){
      for(i=0;i<=nargd-1;i++){
         if(!strcmp(argd[i],"-geometry")){
            if(ptr){
               args[j] = calloc(strlen(argd[i])+1, sizeof(char));
               strcpy(args[j],argd[i]); j++;
               args[j] = calloc(strlen(ptr)+1, sizeof(char));
               strcpy(args[j],ptr); j++;
            }
         }else{
            args[j] = calloc(strlen(argd[i])+1, sizeof(char));
            strcpy(args[j],argd[i]);
            j++;
         }
      }
   }
   args[j] = calloc(strlen(name)+1, sizeof(char));
   strcpy(args[j], name); j++;  /* file name */
   for(i=j;i<=MAX_ARG;i++) args[i] = NULL;
   childpid = fork();
```

```
      /*if(childpid < 1)
         perror("can't fork!");*/
      if(childpid==0){
         if(!strcmp(dir,ply))
            execvp(dir,args);
         else
            execv(dir,args);
         exit(1);
      }
      pid = wait(&status);
      return 1;
   }else return 0;
}
/*
 * Procedure to interpret and execute Parallel statement
 */
void Par(head_par, head_pro, n_part)
LINK *head_par;
FLINK head_pro;
int  n_part;
{
   LINK tmp;
   char dir[70], ply[15];
   char *adir[MAX_P], *aply[MAX_P];
   char *args[MAX_P][MAX_ARG];
   char *argd[MAX_ARG];
   char *ptr, *ptr1, name[50], temp[15];
   char mesg[80];
   int childpid, pid;
   int status;
   int  wt[MAX_P];
   int  nargd, npart, i, j, k;

   tmp = *head_par;
   i = 0;
   while(tmp!=NULL){
      if(tmp->wait_t > 0){
         wt[i] = tmp->wait_t;
      }else wt[i] = 0;
      if(check_device(head_pro, tmp->part, dir, ply, &nargd, argd)){
         adir[i] = calloc(strlen(dir)+1, sizeof(char));
         strcpy(adir[i], dir);
         aply[i] = calloc(strlen(ply)+1, sizeof(char));
         strcpy(aply[i], ply);
         ptr = strtok(tmp->part, "/");
         ptr = strtok(ptr+strlen(ptr)+1,"/");
```

```
    ptr = strtok(ptr+strlen(ptr)+1," /;\n");
    strcpy(temp, ptr);
    ptr1 = strtok(temp, ". ");
    if(!Txt1){
       npart = atoi(ptr1) - 1;
       ptr1 = strtok(ptr1+strlen(ptr1)+1," ");

       if(strcmp(StoDir,"")){
          sprintf(name,"%s/%s.%d.%s",StoDir, Msg, npart, ptr1);
       }else sprintf(name,"%s.%d.%s",Msg, npart, ptr1);
    }else{
       if(strcmp(StoDir,"")){
          sprintf(name,"%s/%s.%s",StoDir, Msg, ptr);
       }else sprintf(name,"%s.%s",Msg, ptr);
    }
    ptr = strtok(ptr+strlen(ptr)+1," ;\n");
    j = 0;
    args[i][j] = calloc(strlen(ply)+1, sizeof(char));
    strcpy(args[i][j],ply);
    j++;
    if(nargd > 0){
       for(k=0;k<=nargd-1;k++){
          if(!strcmp(argd[k],"-geometry")){
             if(ptr){
                args[i][j] = calloc(strlen(argd[k])+1, sizeof(char));
                strcpy(args[i][j],argd[k]); j++;
                args[i][j] = calloc(strlen(ptr)+1, sizeof(char));
                strcpy(args[i][j],ptr); j++;
             }
          }else{
             args[i][j] = calloc(strlen(argd[k])+1, sizeof(char));
             strcpy(args[i][j],argd[k]);
             j++;
          }
       }
    }
    args[i][j] = calloc(strlen(name)+1, sizeof(char));
    strcpy(args[i][j], name); j++;
    for(k=j;k<=MAX_ARG;k++) args[i][k] = NULL;
 }else{
    adir[i] = calloc(strlen(MsgDir)+1, sizeof(char));
    strcpy(adir[i], MsgDir);
    strcpy(ply,"msgW");
    aply[i] = calloc(strlen(ply)+1, sizeof(char));
    strcpy(aply[i], ply);
    j = 0;
```

```
            args[i][j] = calloc(strlen(ply)+1, sizeof(char));
            strcpy(args[i][j],ply);
            j++;
            sprintf(mesg,"No player for: %s",tmp->part);
            args[i][j] = calloc(strlen(mesg)+1, sizeof(char));
            strcpy(args[i][j], mesg); j++;
            for(k=j;k<=MAX_ARG;k++) args[i][k] = NULL;
         }
         tmp = tmp->next; i++;
      }
      *head_par = tmp;
      for(i=0;i<=n_part-1;i++){
         childpid = fork();
         if(childpid < 0) perror("can't fork");
         if(childpid==0){
            if(i==i){
               sleep(wt[i]);
               if(!strcmp(adir[i],aply[i]))
                  execvp(adir[i],args[i]);
               else
                  execv(adir[i],args[i]);
            }
            exit(1);
         }
      }
      for(i=0;i<=n_part-1;i++){
         pid = wait(&status);
      }
   }
   /*
    * Procedure for showing an error message (Substitution)
    */
   void ShowMsg(mesg)
   char mesg[];
   {
      int childpid, pid, status;
      childpid = fork();
      /*if(childpid < 1)
         perror("can't fork!");*/
      if(childpid==0){
         execl(MsgDir,"msgW",mesg,0);
         exit(1);
      }
      pid = wait(&status);
   }
```

```
/*-------------------------------------------------------------------*
 *  This Program interprets scenario codes for testing purpose.
 *  File name: testsce.c
 *-------------------------------------------------------------------*/
#include "sce.h"

FILE *fpr, *fpp, *fpi;          /* file discriptors */
void par_list();
void pro_list();
void Par();
int Seq();
int check_device();
void ShowMsg();
char File[MAX_F][50];           /* array of files */
char MsgDir[50];

main(argc, argv)
int argc;
char *argv[];
{
   char s[MAX_L];               /* string var */
   char *ptr;
   char part[50], atime[4];
   char *getenv(), profile[35];
   char mesg[80];
   int  time, n_part;
   LINK head_par=NULL;          /* head of linked list */
   FLINK head_pro=NULL;
   int ok, ix;

   if(argc!=4){
      printf("Usage: command <scenario code> <infile> <msgW dir>\n");
      exit(1);
   }
   strcpy(MsgDir,argv[3]);

   /* open file "$HOME/.mh_profile" */

   if ((ptr = getenv("HOME")) == (char *) 0){
       printf("HOME is not defined\n");
       exit(0);
   }else
       sprintf(profile,"%s/.mh_profile",ptr);
   if((fpp = fopen(profile, "r")) == NULL){
      printf("\nFile opening error --- program terminated.\n");
      exit(1);
```

```
}

/* Creating the list of available devices/players */

while(fgets(s,MAX_L,fpp)!=NULL){
   if(!strncmp(s,"mhn-show",8)) pro_list(&head_pro, s);
}
fclose(fpp);

/* open file of the list of files (with their paths) */

if((fpi = fopen(argv[2], "r")) == NULL){
   printf("\nFile opening error --- program terminated.\n");
   exit(1);
}

/* Creating the list of files */

ix = 0;
while(fgets(s,MAX_L,fpi)!=NULL){
   ptr = strtok(s,"\n");
   strcpy(File[ix], ptr);
   ix++;
}
fclose(fpi);

if((fpr = fopen(argv[1], "r")) == NULL){
   printf("\nFile opening error --- program terminated.\n");
   exit(1);
}

/* Interpreting and Executing the scenario code */

while(fgets(s,MAX_L,fpr)!=NULL){
   if(!strncmp(s,"IFNOT",5)){
      ptr = strtok(s, " ");
      ptr = strtok(ptr+strlen(ptr)+1," ");
      strcpy(part, ptr);
      if(!Seq(head_pro,part)){
         ptr = strtok(ptr+strlen(ptr)+1," ");
         ptr = strtok(ptr+strlen(ptr)+1," ;\n");
         if(!strcmp(ptr,"SUBSTITUTE")){
            sprintf(mesg,"No Player for: %s",part);
            ShowMsg(mesg);
         } else {
            strcpy(part, ptr);
```

```
            if(!Seq(head_pro,part)){
               sprintf(mesg,"No Player for: %s",part);
               ShowMsg(mesg);
            }
         }
      }
   }else if(!strncmp(s,"WAIT",4)){
      ptr = strtok(s," (");
      ptr = strtok(ptr+strlen(ptr)+1," ,);");
      strcpy(atime, ptr);
      time = atoi(atime);
      ptr = strtok(ptr+strlen(ptr)+1," ,);");
      strcpy(part, ptr);
      sleep(time);                    /* waiting time */
      if(strcmp(part,"NULL")){
         if(!Seq(head_pro,part)){
            sprintf(mesg,"No Player for: %s",part);
            ShowMsg(mesg);
         }
      }
   }else if(!strncmp(s,"PAR",3)){
      n_part = 0;
      while(strcmp(ptr,"}\n")){
         fgets(s,MAX_L,fpr);
         ptr = strtok(s, " (;)");
         if(!strcmp(ptr,"WAIT")){
            ptr = strtok(ptr+strlen(ptr)+1," ,);");
            strcpy(atime, ptr);
            time = atoi(atime);
            ptr = strtok(ptr+strlen(ptr)+1," ,);");
            strcpy(part, ptr);
            n_part++;
            par_list(&head_par, time, part);
         }else if(strcmp(ptr,"}\n")){
            strcpy(part, ptr);
            n_part++;
            par_list(&head_par, 0, part);
         }
      }
      Par(&head_par, head_pro, n_part);
   }else{
      ptr = strtok(s, "\n");
      strcpy(part, ptr);
      if(!Seq(head_pro,part)){
         sprintf(mesg,"No Player for: %s",part);
         ShowMsg(mesg);
```

```
        }
      }
   }
   fclose(fpr);
}
/*
 * Procedure to interpret and execute Sequential statement
 */
int Seq(head_pro, part)
FLINK head_pro;
char *part;
{
   char *ptr, *ptr1, temp[15], name[50];
   char *args[MAX_ARG];
   char *argd[MAX_ARG];
   char dir[70], ply[15];
   int childpid, pid;
   int status;
   int  i, j, npart;
   int  nargd;

   if(check_device(head_pro, part, dir, ply, &nargd, argd)){
      ptr = strtok(part, "/");
      ptr = strtok(ptr+strlen(ptr)+1,"/");
      ptr = strtok(ptr+strlen(ptr)+1," /;\n");
      strcpy(temp, ptr);
      ptr1 = strtok(temp, ". ");
      npart = atoi(ptr1);
      strcpy(name, File[npart-2]);   /* 2 is the first file position */
      ptr = strtok(ptr+strlen(ptr)+1," ;\n");
      j = 0;
      args[j] = calloc(strlen(ply)+1, sizeof(char));
      strcpy(args[j],ply);
      j++;
      if(nargd > 0){
         for(i=0;i<=nargd-1;i++){
            if(!strcmp(argd[i],"-geometry")){
               if(ptr){
                  args[j] = calloc(strlen(argd[i])+1, sizeof(char));
                  strcpy(args[j],argd[i]); j++;
                  args[j] = calloc(strlen(ptr)+1, sizeof(char));
                  strcpy(args[j],ptr); j++;
               }
            }else{
               args[j] = calloc(strlen(argd[i])+1, sizeof(char));
               strcpy(args[j],argd[i]);
```

```
                j++;
            }
        }
    }
    args[j] = calloc(strlen(name)+1, sizeof(char));
    strcpy(args[j], name); j++;  /* file name */
    for(i=j;i<=MAX_ARG;i++) args[i] = NULL;
    childpid = fork();
    /*if(childpid < 1)
        perror("can't fork!");*/
    if(childpid==0){
        if(!strcmp(dir,ply))
            execvp(dir,args);
        else
            execv(dir,args);
        exit(1);
    }
    pid = wait(&status);
    return 1;
}else return 0;
}
/*
 * Procedure to interpret and execute Parallel statement
 */
void Par(head_par, head_pro, n_part)
LINK *head_par;
FLINK head_pro;
int  n_part;
{
    LINK tmp;
    char dir[70], ply[15];
    char *adir[MAX_P], *aply[MAX_P];
    char *args[MAX_P][MAX_ARG];
    char *argd[MAX_ARG];
    char *ptr, *ptr1, name[50], temp[15];
    char mesg[80];
    int childpid, pid;
    int status;
    int  wt[MAX_P];
    int  nargd, npart, i, j, k;

    tmp = *head_par;
    i = 0;
    while(tmp!=NULL){
        if(tmp->wait_t > 0){
            wt[i] = tmp->wait_t;
```

```
}else wt[i] = 0;
if(check_device(head_pro, tmp->part, dir, ply, &nargd, argd)){
   adir[i] = calloc(strlen(dir)+1, sizeof(char));
   strcpy(adir[i], dir);
   aply[i] = calloc(strlen(ply)+1, sizeof(char));
   strcpy(aply[i], ply);
   ptr = strtok(tmp->part, "/");
   ptr = strtok(ptr+strlen(ptr)+1,"/");
   ptr = strtok(ptr+strlen(ptr)+1," /;\n");
   strcpy(temp, ptr);
   ptr1 = strtok(temp, ". ");
   npart = atoi(ptr1);          /* In scenario code : */
   strcpy(name, File[npart-2]); /* 2 is the first file position */
   ptr = strtok(ptr+strlen(ptr)+1," ;\n");
   j = 0;
   args[i][j] = calloc(strlen(ply)+1, sizeof(char));
   strcpy(args[i][j],ply);
   j++;
   if(nargd > 0){
      for(k=0;k<=nargd-1;k++){
         if(!strcmp(argd[k],"-geometry")){
            if(ptr){
               args[i][j] = calloc(strlen(argd[k])+1, sizeof(char));
               strcpy(args[i][j],argd[k]); j++;
               args[i][j] = calloc(strlen(ptr)+1, sizeof(char));
               strcpy(args[i][j],ptr); j++;
            }
         }else{
            args[i][j] = calloc(strlen(argd[k])+1, sizeof(char));
            strcpy(args[i][j],argd[k]);
            j++;
         }
      }
   }
   args[i][j] = calloc(strlen(name)+1, sizeof(char));
   strcpy(args[i][j], name); j++;
   for(k=j;k<=MAX_ARG;k++) args[i][k] = NULL;
}else{
   adir[i] = calloc(strlen(MsgDir)+1, sizeof(char));
   strcpy(adir[i], MsgDir);
   strcpy(ply,"msgW");
   aply[i] = calloc(strlen(ply)+1, sizeof(char));
   strcpy(aply[i], ply);
   j = 0;
   args[i][j] = calloc(strlen(ply)+1, sizeof(char));
   strcpy(args[i][j],ply);
```

```
            j++;
            sprintf(mesg,"No player for: %s",tmp->part);
            args[i][j] = calloc(strlen(mesg)+1, sizeof(char));
            strcpy(args[i][j], mesg); j++;
            for(k=j;k<=MAX_ARG;k++) args[i][k] = NULL;
         }
         tmp = tmp->next; i++;
      }
      *head_par = tmp;
      for(i=0;i<=n_part-1;i++){
         childpid = fork();
         if(childpid < 0) perror("can't fork");
         if(childpid==0){
            if(i==i){
               sleep(wt[i]);
               if(!strcmp(adir[i],aply[i]))
                  execvp(adir[i],args[i]);
               else
                  execv(adir[i],args[i]);
            }
            exit(1);
         }
      }
      for(i=0;i<=n_part-1;i++){
         pid = wait(&status);
      }
}
/*
 * Procedure for showing an error message (Substitution)
 */
void ShowMsg(mesg)
char mesg[];
{
   int childpid, pid, status;
   childpid = fork();
   /*if(childpid < 1)
      perror("can't fork!");*/
   if(childpid==0){
      execl(MsgDir,"msgW",mesg,0);
      exit(1);
   }
   pid = wait(&status);
}
```

```
/*-------------------------------------------------------------------*
 *  This Program presents all media types concurently.
 *  File name: showpar.c
 *-------------------------------------------------------------------*/
#include "sce.h"

FILE *fpr, *fpp;          /* file discriptors */
void par_list();
void Par();
void pro_list();
int  check_device();
char ScenDir[60];

main(argc, argv)
int argc;
char *argv[];
{
   char fnr[45], targv1[20];
   char s[MAX_L];                /* string var */
   char *ptr;
   char part[70];
   char *getenv(), profile[35];
   int  n_part;
   PLINK head_par=NULL;          /* head of linked list */
   FLINK head_pro=NULL;

   if(argc!=3){
      printf("Usage: showpar <infile> <scen dir>\n");
      exit(1);
   }
   strcpy(ScenDir,argv[2]);

   /* open file ".mh_profile" */

   if ((ptr = getenv("HOME")) == (char *) 0){
      printf("HOME is not defined.\n");
      exit(0);
   }else
     sprintf(profile,"%s/.mh_profile",ptr);
   if((fpp = fopen(profile, "r")) == NULL){
      printf("\nFile opening error --- program terminated.\n");
      exit(1);
   }

   /* Creating the list of available devices/players */
```

```
   while(fgets(s,MAX_L,fpp)!=NULL){
      if(!strncmp(s,"mhn-show",8)) pro_list(&head_pro, s);
   }
   fclose(fpp);

   if((fpr = fopen(argv[1], "r")) == NULL){
      printf("\nFile opening error --- program terminated.\n");
      exit(1);
   }

   /* Presenting the file in parallel */

   n_part = 0;
   while(fgets(s,MAX_L,fpr)!=NULL){
      ptr=strtok(s,"\n");
      strcpy(part, ptr);
      n_part++;
      par_list(&head_par, part);
   }
   Par(&head_par, head_pro, n_part);
   fclose(fpr);
}
/*
 * Function for checking available devices/players
 */
int check_device(head_pro, part, dir, ply, nargd, argd)
FLINK head_pro;
char part[], dir[], ply[], *argd[];
int *nargd;
{
   char *ptr;
   char type[25], subtype[25];
   char file[20], ext[5];
   FLINK tmp;
   int i;

   ptr = strrchr(part,'/');
   if(ptr){
      ptr = strtok(ptr,"/");
      strcpy(file,ptr);
   }
   ptr = strtok(file,".");
   ptr = strtok(ptr+strlen(ptr)+1," ");
   strcpy(ext, ptr);
   if(!strcmp(ext,"gif")){
      strcpy(type,"image"); strcpy(subtype,"gif");
```

```
    }else if(!strcmp(ext,"jpg")){
       strcpy(type,"image"); strcpy(subtype,"jpg");
    }else if(!strcmp(ext,"mpg")){
       strcpy(type,"video"); strcpy(subtype,"mpeg");
    }else if(!strcmp(ext,"ps")){
       strcpy(type,"application"); strcpy(subtype,"Postscript");
    }else if(!strcmp(ext,"txt")){
       strcpy(type,"text"); strcpy(subtype,"plain"); }
    tmp = head_pro;
    while(tmp!=NULL){
       if(!strcmp(subtype, tmp->subtype)){
          strcpy(dir, tmp->dir);
          strcpy(ply, tmp->player);
          *nargd = 0;
          if(tmp->narg > 0){
             *nargd = tmp->narg;
             for(i=0;i<=tmp->narg-1;i++){
                argd[i] = calloc(strlen(tmp->arg[i])+1, sizeof(char));
                strcpy(argd[i], tmp->arg[i]);
             }
          }
          return 1;
       }else if(!strcmp(tmp->subtype,"")){
        if(!strcmp(type, tmp->type)){
          strcpy(dir, tmp->dir);
          strcpy(ply, tmp->player);
          *nargd = 0;
          if(tmp->narg > 0){
             *nargd = tmp->narg;
             for(i=0;i<=tmp->narg-1;i++){
                argd[i] = calloc(strlen(tmp->arg[i])+1, sizeof(char));
                strcpy(argd[i], tmp->arg[i]);
             }
          }
          return 1;
        }
       }
       tmp = tmp->next;
    }
    return 0;
}
/*
 * Creating the list of parallel parts
 */
void par_list(head_par, part)
PLINK *head_par;
```

```
char part[];
{
   PLINK tmp;
   tmp = *head_par;

   if(tmp==NULL){
      tmp = malloc(sizeof(PAR));
      strcpy(tmp->part, part);
      tmp->next = NULL;
      *head_par = tmp;
      return;
   }
   while(tmp->next != NULL) tmp = tmp->next;
   tmp->next = malloc(sizeof(PAR));
   tmp = tmp->next;
   strcpy(tmp->part, part);
   tmp->next = NULL;
}
/*
 * Execute parts concurently
 */
void Par(head_par, head_pro, n_part)
PLINK *head_par;
FLINK head_pro;
int  n_part;
{
   PLINK tmp;
   char dir[70], ply[15];
   char *adir[MAX_P], *aply[MAX_P];
   char *args[MAX_P][MAX_ARG];
   char *argd[MAX_ARG];
   int childpid, pid;
   int status;
   int  nargd, i, j, k;

   tmp = *head_par;
   i = 0;
   while(tmp!=NULL){
      if(check_device(head_pro, tmp->part, dir, ply, &nargd, argd)){
         adir[i] = calloc(strlen(dir)+1, sizeof(char));
         strcpy(adir[i], dir);
         aply[i] = calloc(strlen(ply)+1, sizeof(char));
         strcpy(aply[i], ply);
         j = 0;
         args[i][j] = calloc(strlen(ply)+1, sizeof(char));
         strcpy(args[i][j],ply);
```

```
            j++;
            if(nargd > 0){
               for(k=0;k<=nargd-1;k++){
                  if(!strcmp(argd[k],"-geometry")){
                     k++;
                  }else{
                     args[i][j] = calloc(strlen(argd[k])+1, sizeof(char));
                     strcpy(args[i][j],argd[k]);
                     j++;
                  }
               }
            }
            args[i][j] = calloc(strlen(tmp->part)+1, sizeof(char));
            strcpy(args[i][j], tmp->part); j++;
            for(k=j;k<=MAX_ARG;k++) args[i][k] = NULL;
         }
         tmp = tmp->next; i++;
      }
      strcpy(dir,ScenDir);  /* directory of scen program = program */
      adir[i] = calloc(strlen(dir)+1, sizeof(char));
      strcpy(adir[i],dir);
      strcpy(ply,"scen");  /* program name for scenario window */
      aply[i] = calloc(strlen(ply)+1, sizeof(char));
      strcpy(aply[i],ply);
      j = 0;
      args[i][j] = calloc(strlen(ply)+1, sizeof(char));
      strcpy(args[i][j],ply); j++;
      for(k=j;k<=MAX_ARG;k++) args[i][k] = NULL;

      *head_par = tmp;
      for(i=0;i<=n_part;i++){   /* part including browse */
         childpid = fork();
         if(childpid < 0) perror("can't fork");
         if(childpid==0){
            if(i==i){
               if(!strcmp(adir[i],aply[i]))
                  execvp(adir[i],args[i]);
               else
                  execv(adir[i],args[i]);
            }
            exit(1);
         }
      }
      for(i=0;i<=n_part;i++){ pid = wait(&status); }
}
```

```
/*-----------------------------------------------------------------*
 *  Function for listing mhn-show profile
 *  File name: prolist.c
 *-----------------------------------------------------------------*/
#include "sce.h"

void pro_list(head_pro, s)
FLINK *head_pro;
char *s;
{
   FLINK tmp;
   char *ptr, *ptr1;
   char tmps[MAX_L];
   char comp[MAX_ARG+1][70];
   char type[20], subtype[20];
   char dir[70];
   char ply[15];
   int  i, j, k;

   strcpy(tmps, s);
   ptr = strtok(tmps, " -");
   ptr = strtok(ptr+strlen(ptr)+1, "-");
   i = 0;
   while((ptr=strtok(NULL, " :\n"))!=NULL){
      strcpy(comp[++i],ptr);
   }
   ptr = strchr(comp[2], '/');
   if(ptr) strcpy(dir, ptr);
   ptr = strrchr(comp[2], '/');
   if(ptr){
      ptr = strtok(ptr,"/");
      strcpy(ply, ptr);
   }

   if(strchr(comp[1],'/')){
      ptr = strtok(comp[1],"/");
      strcpy(type, ptr);
      ptr = strtok(ptr+strlen(ptr)+1, " ");
      strcpy(subtype, ptr);
   } else {
      strcpy(type, comp[1]);
      strcpy(subtype, "");
   }
   tmp = *head_pro;
   if(tmp==NULL){
      tmp = malloc(sizeof(PROF));
```

```
      strcpy(tmp->type, type);
      strcpy(tmp->subtype, subtype);
      strcpy(tmp->dir, dir);
      strcpy(tmp->player, ply);
      tmp->narg = 0;
      if(i>2){
         for(j=3;j<=i;j++){
            if(!strcmp(comp[j],"-geometry")){
               strcpy(tmp->arg[j-3], comp[j]);
               tmp->narg++; j++;
            }else{
               if(strcmp(comp[j],"%f") && strcmp(comp[j],"%F")){
                  strcpy(tmp->arg[j-3], comp[j]);
                  tmp->narg++;
               }
            }
         }
      }
      tmp->next = NULL;
      *head_pro = tmp;
      return;
   }
   while(tmp->next != NULL) tmp = tmp->next;
   tmp->next = malloc(sizeof(PROF));
   tmp = tmp->next;
   strcpy(tmp->type, type);
   strcpy(tmp->subtype, subtype);
   strcpy(tmp->dir, dir);
   strcpy(tmp->player, ply);
   tmp->narg = 0;
   if(i>2){
      for(j=3;j<=i;j++){
         if(!strcmp(comp[j],"-geometry")){
            strcpy(tmp->arg[j-3], comp[j]);
            tmp->narg++; j++;
         }else{
            if(strcmp(comp[j],"%f") && strcmp(comp[j],"%F")){
               strcpy(tmp->arg[j-3], comp[j]);
               tmp->narg++;
            }
         }
      }
   }
   tmp->next = NULL;
}
```

```
/*-----------------------------------------------------------------*
 *  Function for checking available devices/players
 *  File name: checkpar.c
 *-----------------------------------------------------------------*/
#include "sce.h"

int check_device(head_pro, part, dir, ply, nargd, argd)
FLINK head_pro;
char part[], dir[], ply[], *argd[];
int *nargd;
{
   char *ptr, prt[35];
   char type[20], subtype[20];
   FLINK tmp;
   int i;

   strcpy(prt, part);
   ptr = strtok(prt, "/");
   strcpy(type, ptr);
   ptr = strtok(ptr+strlen(ptr)+1, "/");
   strcpy(subtype, ptr);
   tmp = head_pro;
   while(tmp!=NULL){
      if(!strcmp(subtype, tmp->subtype)){
         strcpy(dir, tmp->dir);
         strcpy(ply, tmp->player);
         *nargd = 0;
         if(tmp->narg > 0){
            *nargd = tmp->narg;
            for(i=0;i<=tmp->narg-1;i++){
               argd[i] = calloc(strlen(tmp->arg[i])+1, sizeof(char));
               strcpy(argd[i], tmp->arg[i]);
            }
         }
         return 1;
      }else if(!strcmp(tmp->subtype,"")){
       if(!strcmp(type, tmp->type)){
         strcpy(dir, tmp->dir);
         strcpy(ply, tmp->player);
         *nargd = 0;
         if(tmp->narg > 0){
            *nargd = tmp->narg;
            for(i=0;i<=tmp->narg-1;i++){
               argd[i] = calloc(strlen(tmp->arg[i])+1, sizeof(char));
               strcpy(argd[i], tmp->arg[i]);
            }
```

```
            }
            return 1;
          }
        }
        tmp = tmp->next;
    }
    return 0;
}
/*
 * Creating the list of parallel parts
 */
void par_list(head_par, time, part)
LINK *head_par;
int time;
char part[];
{
    LINK tmp;
    tmp = *head_par;

    if(tmp==NULL){
        tmp = malloc(sizeof(PARA));
        tmp->wait_t = time;
        strcpy(tmp->part, part);
        tmp->next = NULL;
        *head_par = tmp;
        return;
    }
    while(tmp->next != NULL) tmp = tmp->next;
    tmp->next = malloc(sizeof(PARA));
    tmp = tmp->next;
    tmp->wait_t = time;
    strcpy(tmp->part, part);
    tmp->next = NULL;
}
```

VITA

Tahar Agastani

Candidate for the Degree of

Master of Science

Thesis: MULTIMEDIA MAIL: AN IMPLEMENTATION PROVIDING A SCENARIO SERVICE

Major Field: Computer Science

Biographical Data:

Personal Data: Born in Kuningan, Jawa Barat, Indonesia, on April 13, 1962, the son of Syambas Hanafi and Rolinah Syambas.

Education: Graduated from High School, Kuningan, Indonesia, in 1981; graduated and received Bachelor of Engineering in Electrical Engineering from University of Indonesia, Jakarta, Indonesia in July 1988; completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December 1995.

Experience: Computer Programmer, Computer Science Center, University of Indonesia, August, 1988 to March 1989. Researcher Staff, the Agency for the Assessment and Application of Technology, Jakarta, Indonesia, since April, 1989.