Chapter 6

Semantics: Teleaction Objects

Teleaction Objects (TAOs) possess private knowledge specific to the object instances. The user can create and modify the private knowledge of a Teleaction Object, so that the Teleaction Object will automatically react to certain events to pre-perform operations for generating timely response, improving operational efficiency and maintaining consistency. Moreover, Teleaction Objects also possess a hypergraph structure leading to the effective presentation and efficient communication of multimedia information. The Active Multimedia System (AMS) is designed to manage the Teleaction Objects. In the AMS, the private knowledge of each Teleaction Object is realized by the index cells of Active Index. The applications to smart multimedia mail and multimedia information retrieval are described to illustrate the usefulness of Teleaction Objects.

1. INTRODUCTION

We describe the Teleaction Object (TAO) which is a multimedia object with associated hypergraph structure and knowledge structure. Recently distributed multimedia systems have become a common requirement for more and more applications [Berra90]. Although different media types have different characteristics in the size, resolution, storage method, transmission method, compression algorithm, presentation technique, etc., two central problems are common regardless of the application: presenting multimedia information effectively and transmitting multimedia information efficiently. It is therefore desirable to have a common approach for multimedia data modelling, which can lead to the solution of both problems [Little90a] [Znati93]. Teleaction Object, with its rich classifications of node types, media

types, and link types in the hypergraph structure, enables us to design algorithms to decide the priority in both communication and presentation for multimedia information according to the current environment and restrictions. With the different levels of knowledge specified by the system and the end-user, the TAO can react automatically to certain events.

An Active Multimedia System (AMS) is designed based upon the concept of TAOs [ChangH95b]. The AMS provides mechanisms for manipulating the TAOs so that the system can gather the private knowledge from a TAO instance and merge it into the knowledge base. The AMS provides mechanisms for maintaining the knowledge so that the TAO can automatically react to certain events to pre-perform operations for generating timely response, improving operational efficiency and maintaining consistency. The AMS also provides the tools so that the user can create his/her own TAO, and implement his/her own applications to handle the TAO. The Teleaction Object is a conceptual model. In actual implementation, the Teleaction Object can be implemented as objects in an object-oriented system. For a Multimedia Mail System, for example, a mail can be regarded as a Teleaction Object.

The approach of Teleaction Object model can support and improve many applications. For example, in the global information system such as the World-Wide Web (WWW) [Berners-Lee92], the user can navigate in the hyperspace supported by the network. But the major limitation is that the information-requesting mechanism constrains any transaction between the user and system to be passive. The Teleaction Object approach can be incorporated with the WWW browser such as the Mosaic. Therefore, the knowledge part will support the active actions in network information systems, such as two-way interaction [Dimit94], or pre-fetching.

Another application domain for Teleaction Object is delayed conferencing [Hou94]. The objective for delayed conferencing is to allow a group of participants to exchange information, including existing and newly created information, in a timely and consistent manner. Such delayed conferencing is based on the multimedia-based message delivery mechanism and exchanged information consisting of multimedia objects. Current multimedia mail systems plus groupware systems [Borenstin92] [Goldberg92] may support delayed conferencing. However, the limitation is that the system only executes/takes actions when the message is read by the recipient, or when the recipient answers certain questions. There are occasions when it is required to take action even before a message is read, or the recipient's environment is changed. The user should be allowed to define the event and the condition for actions to take place. The Teleaction Object approach can be applied to delayed conferencing because it supports

multimedia information exchange and maintains the knowledge at different levels to automatically react to the events.

This chapter is structured as follows. The next section gives the definitions and the motivation of the definitions for a Teleaction Object with both hypergraph structure and knowledge structure. Section 3 presents a scenario to explain how the Teleaction Objects work in the Active Multimedia System. The system architecture of the Active Multimedia System is described in Section 4. Based on the Active Multimedia System and the Teleaction Object approach, a working Smart Multimedia Mail application is illustrated in Section 5. Section 6 discusses application to multimedia information retrieval where an index cells hierarchy leads to user-specific pre-fetching of multimedia information. Section 7 discusses some of the contributions and identifies ongoing and future research.

2. DEFINITION OF TELEACTION OBJECTS

A Teleaction Object can be as simple as a single piece of information without connection or relation to any other objects. Or we can combine several TAOs in certain connections into a new complex TAO and/or add certain knowledge to a TAO. Basically, the TAO is further refined as two parts (G, K): hypergraph G and knowledge K. For a TAO, the hypergraph G is used to describe the connections and relations between the sub-TAOs within it. The knowledge K is used to describe the actions.

The Active Multimedia System AMS is a system that manipulates and maintains the Teleaction Objects and also provides the basic tools and methodology such that users can implement their own applications to handle the TAOs. The AMS is similar to the operating system of a computer, but instead of allocating and maintaining the resources AMS allocates and maintains TAOs. Section 4 will give the details of AMS architecture.

2.1 Definition of TAO Hypergraph Part

The hypergraph structure G plays an important part in TAO. Basically G is a graph structure (N, L), where N is a set of nodes and L is a set of links. Each node in G represents another Teleaction Object (TAO) and a link represents a specified relation or connection between these TAOs. By further refining the types of nodes, media and links, we can utilize this hypergraph structure for the dual purposes of regulating multimedia communication and generating multimedia presentations.

(a) Node types

There are essentially two types of nodes in the hypergraph structure G: basic node and composite node.

A *basic node* is defined as a terminal node in the hypergraph structure. The real media data is associated with the basic node and each basic node contains one and only one media data type. In other words, the basic node is the smallest unit of a TAO. For example, an image is a basic node. The media types for the basic node are defined below in (b).

A *composite node* is defined as a non-terminal node in the hypergraph structure. It contains a number of basic nodes and/or composite nodes. The composite node is a group of data instead of a single real media data. For example, a book chapter is a composite node in the hypergraph structure of a book because it contains sections, pictures and tables. By using the composite nodes, we not only can present the hierarchy of the multimedia object, but also can apply knowledge to a group of multimedia objects.

(b) Media types

Each media type has different characteristics of size, cost, user interface, storage hardware, interpretation, etc. Since applications choose different media types under different situations and considerations, knowing the media type is helpful for the system to optimize its performance. In our system, we define media types for the basic node in TAO as: text, graphics, image, moving-graphics, moving-image, audio, video, form and live-demo; while composition is for a composite node.

- *text* is coded alphanumeric data. It is the most basic media type for most multimedia applications.
- *graphics* is formatted picture data.
- *image* is pixel formatted picture data.
- *moving-graphics*, also called animation, is the formatted data of a graphics sequence.
- *moving-image* is a sequence of image frames.
- *audio* is formatted sound data.
- *video* is a combination of synchronized moving-image and audio.
- *form* restricts user input, possibly with additional formula to generate the content automatically.
- *live-demo* is a program that can be run to provide an interactive demo.
- *composition* is the media type of a composite node in TAO.

(c) Link types

We represent the specific relations and connections between nodes by using different types of links: attachment link, annotation link, reference link, location link and synchronization link.

An *attachment link* links a composite node A and another node B which is either a basic node or a composite node. It indicates that node B is a component of node A. This is the essential relationship between nodes in a multimedia hypergraph structure.

An *annotation link* links node A with node B, and both nodes can be basic node or composite node. This type of link specifies node B is an annotation associated with node A. The annotation has a different meaning from that of the attachment because the annotating node is an explanation or a synopsis of the annotated object, but it is not a necessary component required to construct the annotated node.

A *reference link* specifies there should be a navigation path from one node A to another node B when the user is browsing in the hypergraph structure G. Node B should be a node with no bundled node (to be explained later) as its ancestor.

A *location link* specifies the spatial relationship between nodes for their presentation. The number of nodes linked by a location link can be more than two and the type of each node can be basic node or composite node. We have developed a fuzzy relation language FRL [ChangH95a] to describe the spatial relationship in both horizontal and vertical directions. For example, image A and image B are located side by side and touching one another in the horizontal direction, which can be expressed by the location link with the relation language such as "X: A] == [B".

A *synchronization link* specifies the temporal relationship between nodes for their presentation. The number of nodes linked by a synchronization link can be more than two and the type of each node can be basic node or composite node. Similar to location links, we can use the same relation language to describe the temporal relationship for the synchronization links [ChangH95a]. For example, 5 seconds after image A is displayed we would like to play a moving-graphics B and an audio C at the same time, which can be expressed by the synchronization link with the relation language such as "T: A] < 5 [B; T: [B == [C".

For the purpose of facilitating multimedia presentation, multimedia communication, hyperlinking, and replacement, there is a special feature for nodes known as bundled nodes. A *bundled node* can be either a basic node or a composite node. A bundled composite-node serves as a single unit with all its components bundled together. For a bundled basic-node, its sole purpose is presentation. Using the bundling concept greatly simplifies the

specification of multimedia presentation. For instance, if a basic node containing audio is a bundled node, it can only be played from the beginning to the end, and it cannot be played half way. For another instance, if a composite node containing three basic components (text, image, and audio) is a bundled node, we cannot just present anyone of its components without presenting the other two.

A function a_bundle(N) is defined for node N as follows:

a_bundle(N) = nearest bundled ancestor of node N (excluding itself) in G when traversed along attachment or annotation links from N
= NIL if such node does not exist.

There are two constraints for bundled nodes. The first is regarding the relation links which are either location link or synchronization link: a relation link can be established between two nodes N and M only if (1) a_bundle(N) is the same as a_bundle(M); or (2) one of them is a_bundle of the other. The second constraint is regarding the reference links: a reference link should only link to a node N such that a_bundle(N) = NIL.

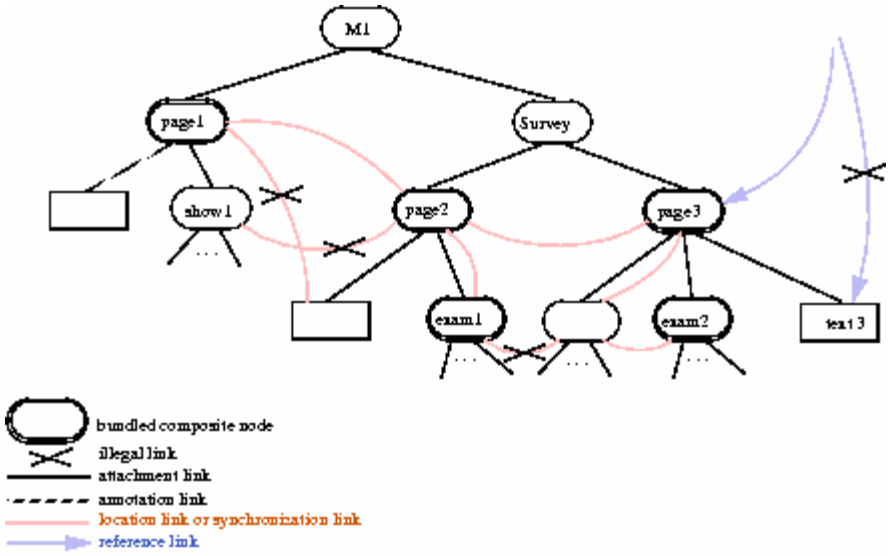

Figure 1. Example of legal and illegal links in a hyperstructure.

Bundled nodes have three purposes. First, restricting relation links at some nodes will retain only the meaningful relations expressed by Fuzzy Relation Language (FRL) Briefly speaking, FRL is used to express

temporal/spatial relations in presentation between nodes. Without bundled nodes to limit the use of relation links, many cases would occur which do not make sense. As explained in Figure 1, the double-edged nodes are bundled nodes, the lighter arcs represent the relation links, and the crosses indicate where the relation/reference links are illegal.

Second, restricting reference links at some nodes will maintain the integrity of presentation. As shown in Figure 1, the dotted arrow linking to the basic node `text3' is illegal because `text3' is one component of the bundled node `page3'. The presentation system cannot display `text3' without displaying all other components of `page3'. Thus, the reference link to `text3' should be moved up to `page3'.

Third, replacement of a bundled node by another representation-equivalent bundled node becomes possible. We can devise algorithms to replace a bundled node by another bundled node, as long as the presentation effects are comparable by some measurements. For example, the animation of a demo can be replaced by a live demo (actually running the program). Replacing a video by a still image with dubbed sound track is another possible replacement.

2.2 Motivation for TAO Hypergraph Part

In a distributed environment, communication and presentation of multimedia objects is both time consuming and space consuming. We need to transfer and display multimedia objects efficiently, effectively, and properly. For example, when the network traffic is high, we may send the main content of a multimedia object first, transmitting important images in low resolution, and leaving out the less important parts for later transmission. When we present multimedia objects, we need to know the order of presentation and possibly pre-fetch other data. In other words, we need to determine the priority of the transmission sequence and the presentation order for multimedia objects. Thus, a rich set of node types, media types and link types will enable us to use the hypergraph structure G to control communication and organize presentation. An example is given in Section 3. The different types of nodes and their structure in G provide useful information of different relationships between the TAOs. It is feasible to design algorithms to traverse the hypergraph G and determine the transmission sequence and the presentation order according to the currently available communication bandwidth, recipient's environment, link types, node types, media types and structure in G [Lin94].

Given a multimedia hypergraph G, we can apply the following algorithm to generate its Multimedia Data Schema (MDS) [Lin94], which controls the

synchronization between time-related data streams. The MDS is similar to the Object Composition Petri Net (OCPN) [Berra90], [Little90a].

1. Let g be the subgraph of the hypergraph G so that nodes in g can be traversed from the root of G via 'attachment' and 'annotation' links only.
2. For each node n in g, let level(n) be the level of n in the breadth-first spanning tree of g.
3. In g, for each synchronization link, which connects two nodes, say Mi and Mj:
   3.1. Let Mk defined as Nearest-Common-Ancestor(Mi, Mj), and let
   3.2. k = level(Mk).
   3.2. /* CASE 1: Ancestor-Descender relationship */
   If level(Mi) ≠ level(Mj) and Mk ∈ {Mi, Mj} /* and assume Mi = Mk */ then
   3.2.1. Let Mj' = Ancestor(Mj) at level k+1.
   3.2.2. Propagate temporal information from Mj up to Mj'; and create a new synchronization relation between Mi and Mj'.
   3.3. /* CASE 2: Cousin-Cousin relationship or */
   /* CASE 3: Distant relationship */
   If (level(Mi) = level(Mj) and level(Mk) + 1 ≠ level(Mj)) or (level(Mi) ≠ level(Mj) and Mk ∈ {Mi, Mj}) then
   3.3.1. Let Mi' = Ancestor(Mi) at level k+1; and
   Let Mj' = Ancestor(Mj) at level k+1
   3.3.2. Propagate temporal information from Mi up to Mi' and
   3.3.3. from Mj up to Mj'; and create a new synchronization relation between Mi' and Mj'.
4. Recursively generate MDS by using Algorithm described in [Lin94].

The transitions in MDS indicate points of synchronization and the places represent the media presentation processing. In other words, we should present the multimedia objects in a specified order according to the MDS. Again there are several considerations to generate an effective presentation sequence, such as the location and synchronization link relation between the TAOs, computer hardware, capability to display different media type, storage size etc. An example for MDS is given in Figure 5 of Section 3.

Given a Multimedia Data Schema (MDS), there is an algorithm to generate its Multimedia Communication Schema (MCS) [Lin94] for an efficient transmission sequence. There are several considerations to generate an efficient transmission sequence, such as the size of the TAOs, the type of the TAOs, the structure of the TAOs, the link relation between the TAOs, computer hardware, the capability to display different media type, the

storage size, bandwidth of communication, etc. For example, if we adopt the progressive transmission technique in the communication schema, the low-resolution image or the low-quality audio will be sent first if the network traffic is high. Then we try to transmit the higher quality data if possible. For those nodes connected by a synchronization link, the multimedia objects will be transmitted according to the synchronization requirements. Also, the nodes linked by an attachment link should have higher priority than the nodes linked by the annotation link and reference link because the user may want to see the content of top-level TAO first. Or the nodes closer to the currently viewed node should have higher priority than the nodes farther away from the currently viewed node in the transmission sequence.

Moreover, given a hypergraph G, we can design an algorithm to decide the pre-fetch sequence in order to improve the efficiency of multimedia browsing. Thus, only needed data become available based on the currently viewed TAOs. The pre-fetch sequence is changed dynamically and performed in the background and does not require monitoring by the user.

From the above discussion, it can be seen that the sets of node types, link types, and media types will provide the information needed for: (a) automatic scheduling of synchronization in communication; (b) automatic generation of proper presentation for specified spatial and temporal relations and (c) automatic pre-fetching of potential multimedia objects.

2.3 Definition of TAO Knowledge Part

Without specifying the part of knowledge K in a Teleaction Object, a TAO is just another hyper-media object. In fact, the simplest media object is a TAO whose hypergraph part is reduced to a simple node and whose knowledge part is empty. We can classify the knowledge of TAO into four levels:

System knowledge is associated with all TAO instances. It handles all generic operations that are applicable to all TAOs. This knowledge defines the default intent for each TAO. For example, checking the privilege for viewing each TAO; or keeping the history log; or pre-fetching other TAOs from remote servers when these TAOs are within a certain distance from the TAO currently being viewed in the hypergraph G. This knowledge is created by the system.

Environment knowledge is associated with all TAO instances belonging to a special user. A user may want to customize his/her AMS operations. So the user can add or remove some knowledge to his/her local knowledge base. For example, user A might add a new knowledge to purge TAOs whenever the system storage is low. The user can also overwrite some system knowledge, for example, he/she can change the distance criterion for pre-

fetch. The environment knowledge can be generated by either the system or the user.

Template knowledge is associated with a group of TAO instances in a predefined format. For frequently used TAO formats, a hypergraph as well as the associated knowledge can be provided, such as the time scheduler, the weekly work report generator, the resume, etc. It can be generated by either the system or the user.

Private knowledge is associated with one special TAO instance. This is the most important knowledge for the user because it carries individualized knowledge, which the user has created for the single TAO instance. It is generated by the user.

There is a local knowledge base for each user. When a user registers for the first time, AMS will create a local knowledge base for this user and initialize it with the system knowledge. After that, the environment knowledge is merged, withdrawn or overwritten into the local knowledge base. As time goes by, lots of template knowledge and private knowledge will be merged into or removed from local knowledge base when the TAOs become alive/dead. Therefore, the knowledge base is local because after the initialization, the inclusion of different environment knowledge, template knowledge and private knowledge leads to different user's local knowledge base.

For a Teleaction Object TAO instance, as discussed in previous sections, it is represented by the (G, K) pair. The knowledge part K of a TAO instance could be either the template knowledge if the TAO is a predefined template instance, or the private knowledge if the TAO is an individual TAO instance. When the Teleaction Object (G, K) becomes available, K is merged into the user's local knowledge base. Similarly, when the TAO becomes unavailable, the corresponding K should be withdrawn from the user's knowledge base. When a TAO moves from one user A's local knowledge base to another user B's local knowledge base, K will be copied into B's local knowledge base. With a better algorithm, part of the K will be merged into A's local knowledge base and part of K will move along with the TAO and be merged into B's local knowledge base.

Conceptually, there is a priority for overriding the existing knowledge as follows: private knowledge, template knowledge, environment knowledge, and system knowledge, where the private knowledge has the highest priority and the system knowledge has the lowest priority.

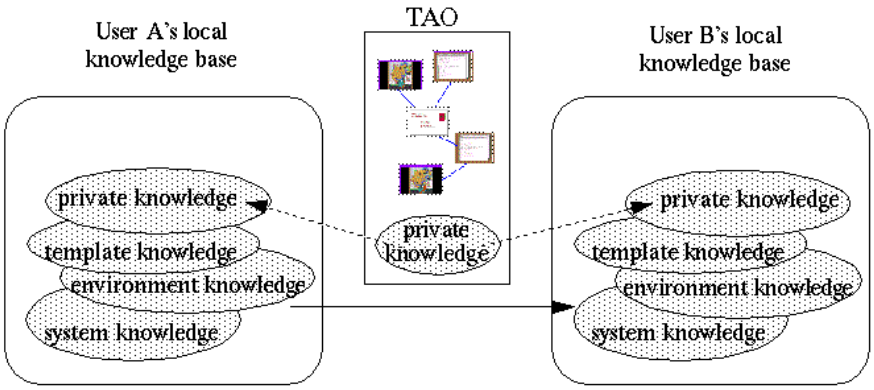

Figure 2. User A sends a Teleaction Object to user B.

As illustrated in Figure 2, the solid line indicates the transmission of the TAO from user A to user B, and the dotted lines indicate the sharing of private knowledge of TAO in both system A and system B. Private knowledge has the highest priority to override others in the local knowledge base.

From the object-oriented point of view, each TAO instance and each knowledge piece has its own class name. The hierarchy class name is indicated in the hierarchy, such as "root.TAO.mail". Therefore, once a TAO instance is available in the AMS system, knowledge with the same class name and super-class name will be attached to it. If the knowledge pieces have naming conflicts, the system will use the one with the longest class name inheriting and overriding knowledge at different levels. For example, once a TAO instance with the class name "root.TAO.mail" is available in the AMS, the system will attach the knowledge with class name "root", "root.TAO", and "root.TAO.mail" to it. And if two knowledge pieces have the same name, "pre-fetching", but with different class names, one being "root.TAO" and another being the class name "root.TAO.mail", the system will attach the one with "root.TAO.mail". In other words, it overrides part of the inherited knowledge.

In our model, the system knowledge in AMS is implemented with class name "root" and/or "root.TAO". All the instances of TAO will be attached to it since each TAO instance has a class name beginning with "root.TAO". Environment knowledge has the ability to modify/add the knowledge piece with class name "root" and/or "root.TAO", such that the user can customize his/her own local knowledge base. Template knowledge is used to build up new knowledge pieces with a nesting class name, such as "root.TAO.mail" or "root.TAO.mail.resume". Therefore all TAOs with the same class name in

depth, will be attached with this knowledge piece. Finally the private knowledge is attached to only one single TAO instance. We can implement the private knowledge piece with the class name ended with the identical TAO Id number, e.g. "root.TAO.mail.resume.1234". That means only this single TAO instance belongs to this class. Therefore, the private knowledge is attached to this TAO instance only.

2.4 Motivation for TAO Knowledge Part

Although we can abstract some useful information from different types of nodes, media and links of the TAO to improve the communication and presentation for multimedia data, without the knowledge part the TAO is just a complex hypermedia object to be used in a passive way. In order to change the TAO from being passive to active, we add the knowledge part for a TAO. Therefore, once a TAO becomes available in the AMS, it will add its own knowledge to the local knowledge base in AMS. And whenever a specified event occurs, related actions take place according to the knowledge. In other words, TAO is an active object.

By using the concept of system knowledge level and environment knowledge level, we can accomplish a customized system for individual users. After user's AMS is initialized by the system knowledge, it can be customized by overriding the system knowledge with environment knowledge. Since both knowledge levels are associated with class name "root" and/or "root.TAO", these knowledge pieces are applied to all available TAO instances in the AMS. Basically, these knowledge pieces are more related to the environment changing, pre-fetching, or pre-processing.

In certain cases, only the user has the best knowledge on how to manage the object. Therefore, we allow users to specify the knowledge within one TAO, which means some special knowledge is associated only with this TAO but not with any other TAO. This is the object's private knowledge. On the other hand, some TAOs are expected to share the template knowledge. Both template knowledge level and private knowledge level are for a special group of TAOs, the only difference is that a single TAO is in the groups of private knowledge. Basically, the template knowledge is more related to the application-specific features, such as mail or resume, while the private knowledge is related to really personal matter for a special single TAO.

From the above discussion, using knowledge in the TAO model will allow the objects to become active, and using different knowledge levels will allow the user to customize AMS and to specify private information. These features enable the Teleaction Object to become an intelligent active object, and the AMS to be more flexible.

3. A SCENARIO

Let us now present a scenario as an example. A project manager Smith is preparing a proposal for his boss Kessler and his group members Wang and Larson. His proposal M1 contains several pages of text, audio and images and also links to a confidential report M2 for his boss Kessler only. Two annotations about the image are also included in his proposal. The hypergraph structure of the proposal is shown in Figure 3.

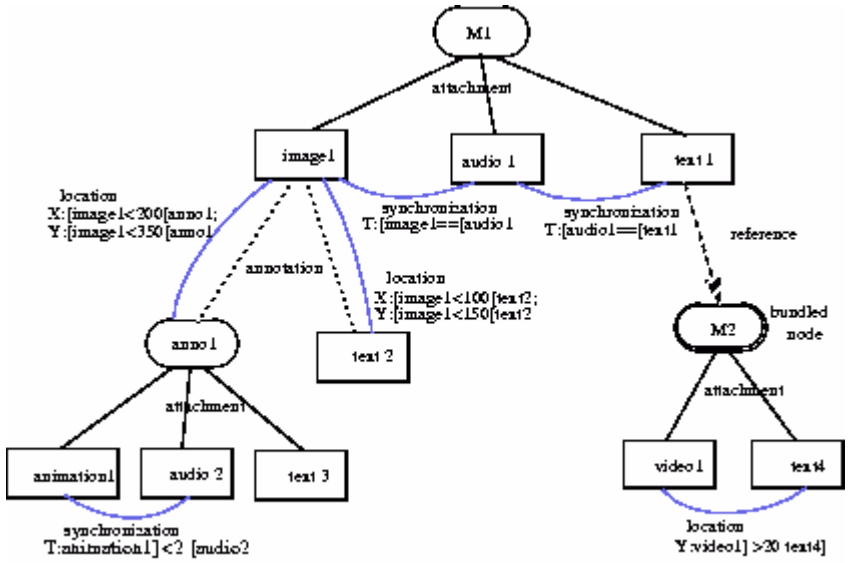

Figure 3. The hypergraph structure of the proposal.

In Figure 3, the rectangle denotes a basic node and the rounded rectangle denotes a composite node. The attachment links indicate the composition of the composite nodes. For example, the proposal M1 is composed of text, audio and image data; the confidential report M2 is a bundled node composed of text and video data; and the annotating object "anno1" is composed of text, audio and animation data. Two annotation links in this example specify that the image has two annotations on it; one annotation is a basic node for text data and the other annotation is a composite node. The reference link indicates there is a navigation path from text of M1 to the bundled confidential report M2 (in this case, for boss Kessler's eyes only). Location links specify where the annotations, anno1 and text2, are with respect to image1 and specify the layout of the confidential report M2. In the former case, the location links are between parent nodes and child nodes, while in the latter case, the location link is between sibling nodes and it

specifies that video1 is 20 units above text4. The synchronization link specifies the temporal ordering in the presentation. In this example, the synchronization link between animation1 and audio2 specifies that the audio2 should be delayed 2 seconds after having finished playing the animation1 while the two synchronization links between image1, audio1 and text specify they should be started simultaneously for M1.

According to the hypergraph structure, we develop the Multimedia Data Schema, shown in Figure 4, when any one of the group members decides to browse the proposal M1. Token flow in a Multimedia Data Schema illustrates the presentation of multimedia objects, while as token flow in a Multimedia Communication Schema illustrates the transmission of multimedia objects.

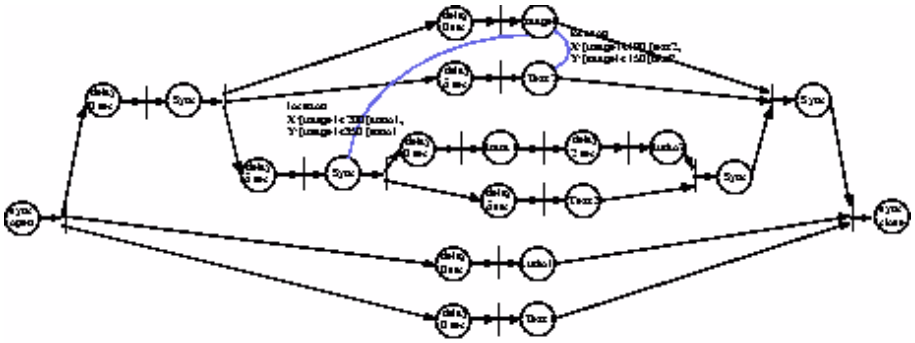

Figure 4. Multimedia Data Schema.

After Smith creates a Teleaction Object M1 with the hypergraph structure shown in Figure 3, he may want to accomplish several tasks just for this special proposal M1 only. The tasks are: (pic.a) the boss Kessler and the two group members should read this proposal in two days and send back their responses; (pic.b) the responses from Kessler and at least one of the two group members are needed in order to proceed to the next step in proposal preparation; and (pic.c) this proposal should become obsolete after one week. In this case, Smith will include the private knowledge with the TAO.

In our example, there are four local knowledge bases for the four users: Smith, Kessler, Wang and Larson, respectively. When these four users first join the AMS, all have the same local knowledge bases that are initialized with the system knowledge. For example it may include the following system knowledge: (sic.a) If the user has read part of a multimedia object, pre-fetch all the information within two hypergraph links from the object being viewed, in the associated hypergraph structure; (sic.b) If the user intends to read a part of the multimedia object, check whether the user has

the privilege or not; (sic.c) If any annotation part is added, do the version control and history checking; and (sic.d) If any multimedia object is obsolete, remove it.

After each user has the initial local knowledge base, the three other kinds of knowledge will be merged into, overwritten on, or removed from the local knowledge base. For example, the following environment knowledge is added for Smith: (eic.a) If any new TAO contains the keyword "FYI", make a copy to folder "FYI-1994"; and (eic.b) If the user has read part of a multimedia object, pre-fetch all the information within three hypergraph links from the object being viewed, in the associated hypergraph structure (perhaps because Smith has a larger storage allocation in his system).

The environment knowledge eic.a is merged into the user Smith's local knowledge base, while the environment knowledge eic.b overwrites the system knowledge sic.a. Therefore, all the TAO instances available in Smith's local system are different from that of the other three users because when a new "FYI" TAO becomes available, Smith's local knowledge base will have a backup copy in a folder while other's local knowledge base will not; and TAOs are pre-fetched within three links distance instead of two links distance in Smith's local system.

Returning to Smith's proposal M1, he creates the private knowledge for M1 and sends it to the other three users. Part of the private knowledge may go with the message while some knowledge may still be kept in the knowledge creator's local knowledge base. In our example, the two private knowledge, pic.a and pic.b, will be sent along with the M1 to the recipient's local knowledge base. In other words, these two private knowledge pieces will be merged into Kessler's, Wang's, and Larson's local knowledge bases. As for the last private knowledge pic.c of proposal M1, it is not sent along with M1, instead it is merged into Smith's local knowledge base. The concept is also illustrated in Figure 3.

Also in this example, we can expect that the private knowledge pic.c will override the system knowledge sic.d. If this proposal M1 is expired, all information of M1 is stored instead of being removed.

4. THE AMS ARCHITECTURE

We have discussed the two important components of a TAO: the hypergraph structure G and the knowledge K. In this section, we describe the AMS architecture that handles and maintains all the TAOs.

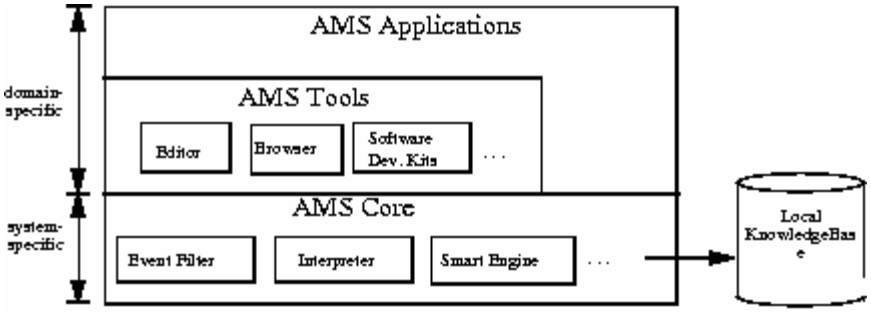

Figure 5. The architecture of AMS.

The AMS consists of two subsystems: a domain-specific part and a system-specific part as shown in Figure 6. The system-specific part performs the generic functions. The application programmers can use the tools and the generic functions to implement his/her own application with TAOs. After the TAOs have been generated by this application and passed to the AMS core, they will be translated, maintained and operated by the system-specific part.

The AMS provides the basic tools upon which users can implement their own applications. A browsing tool is provided for the user to browse the hypergraph G of a TAO. An editor tool is also provided for the user to edit the hypergraph G of a TAO. Other tools allow users to change the system knowledge and build up their own environment knowledge. The template knowledge and private knowledge of TAOs are generated by application-specific tools. For example, we can implement a Smart Multimedia Mail application, which is like a mail system but deals with multimedia mail and also can be associated with private knowledge. In other words, the Smart Multimedia Mail System generates and handles the TAO as a mail object, and all the TAOs generated by this mail system are maintained by AMS, similar to the way files are maintained by an operating system's file manager. The only difference is that AMS will maintain the TAOs as active objects. Another example is a simple multimedia calendar. In this case, the hypergraph part G is not so important, but we can set alarm and a list of tasks to be performed at appropriate times. For example, instead of only beeping, a TAO generated by this calendar can also automatically invoke specified operations.

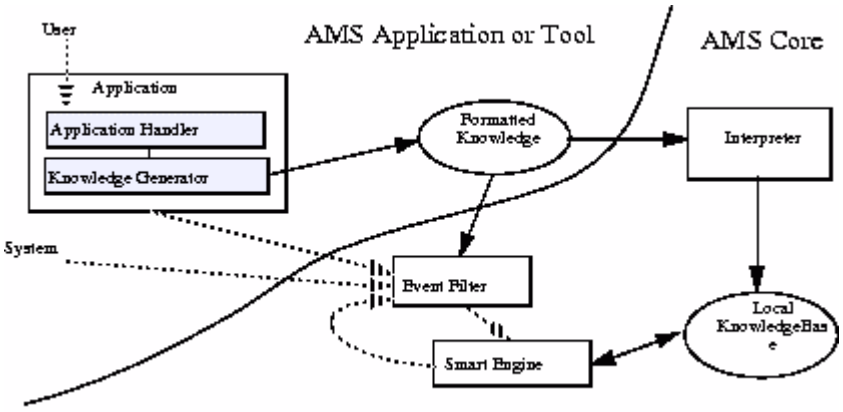

Figure 6. Relation between the AMS application and the AMS Core.

Each AMS tool or user implemented application has the basic components and the relation with the AMS core, as shown in Figure 6. In the above diagram, the line separates the architecture into two subsystems according to the dependence on domain knowledge. The left-hand side is an application-dependent subsystem, including tools provided by AMS and applications developed by programmers. Rectangles represent processes while ellipses represent formatted data. Solid arrows represent knowledge flow while dashed arrows represent event flow. The application-specific Application Handler allows the user to create special purpose TAOs and the Knowledge Generator transforms the knowledge part of these TAOs to the Formatted Knowledge defined in AMS. Therefore, in the right-hand side, the application-independent subsystem AMS itself, when a TAO becomes alive in AMS the Interpreter obtains the formatted knowledge and converts or merges it into the Local Knowledge Base. The local knowledge base is modelled by the Active Index that is a collection of index cells (ICs) [Chang95a] in AMS. At each local AMS system, the Event Filter will monitor the environment, user behavior, specified events, and internal messages, then generate the corresponding messages to the Smart Engine, while the Smart Engine distributes the messages to corresponding ICs in the local knowledge base to invoke the corresponding actions.

4.1 The Local Knowledge Base

The basic concept of AMS is to respond to the environmental changes and take corresponding actions according to user defined knowledge, and to

provide a way such that each TAO has it own private knowledge. In other words, "active" and "private" are the two key ingredients of AMS.

The local knowledge base is defined to be a set of ICs connected by messages [Chang95a]. An IC accepts input messages and performs some computation. It then activates another group of ICs, and posts the output message to these output ICs. If some of these output ICs have already been activated, they may simply accept the output from the current IC. The first output cell that accepts the output message will remove it from the output list of the current cell. After its computation, the IC may remain active (live), or de-activate itself (dead). An IC will also become dead, if it remains inactive for a certain period of time, i.e., if no other cells (including itself) send messages to it. An IC consists of a finite number of ICs. When the IC is in actual computation, it consists of a time-varying collection of ICs in different states, accepting certain input messages and posting output messages to the output lists.

Each IC has two functions: f is an acceptance function that determines when the IC is enabled and ready to fire. Once all the required messages become available, f will remove the messages from the output lists of the message-sending ICs and enable the IC. g is a knowledge function that performs the firing procedure for the IC. Once the f enables the IC, g will take over the control and fire the IC. According to the messages accepted by f, the firing procedure will decide (1) the next state of the IC; (2) how to generate new messages for specific ICs; and most importantly (3) how to perform a specific action sequence.

The IC is the local knowledge base in AMS. And for each TAO in AMS, there are corresponding ICs in the AMS

4.2 Formatted Knowledge for TAOs

According to the definition of an IC, g is the knowledge function in each IC which contains its own finite-state-machine. Basically the g function accepts the input set of messages, computes the next state, generates any new message to other ICs, and performs the corresponding action-sequence. The following BNF definitions show the essential syntax of formatted knowledge used in AMS. By using the hierarchy class name in the < Class Name > of < IC Def > production, we can decide which class of TAO the knowledge function g should be applied to.

< Formatted Knowledge > ::= < Message Def > < Action Def > < IC Set Def >
< Message Def > ::= {EVENT < Event Name > = < Event Expr. > }*
< Action Def > ::= {ACTION < Action Name > { < Parameter List > } }*
< Parameter List > ::= (< Parameter > {, < Parameter > }*)

```
< IC Set Def >        ::= < IC Def > { ; < IC Def > }*
< IC Def >            ::= IC < Type > < ID > < Class Name > < Max Life Time >
                          < FSM >
< FSM >               ::= FSM: < Number of State > < State Trans List >
< State Trans List >  ::= < State trans > { , < State trans > }* ;
< State trans >       ::= ( < Current_State > , < Next State > , < Input > , < Output > ,
                          < Action > ) | NIL
< Input >             ::= ( < Message List > )
< Output >            ::= ( { < IC-Message > }* )
< IC-Message >        ::= ( < IC Set > ; < Message List > )
< IC Set >            ::= < IC > { , < IC > }*
< Message List >      ::= < Message > { , < Message > }*
< Action >            ::= ( < Action-process List > )
```

4.3 The Application Handler and Knowledge Generator

For each application, we need an application-specific Application Handler for the end user to work on, and also to provide an easy way to collect the special knowledge given by the end user and this application. After the application creates the TAO, the Knowledge Generator transforms the knowledge gathered from the user to a format suitable for the Interpreter in AMS.

Generally speaking, we can use the multimedia editor tool provided by the AMS to generate the hypergraph structure of a TAO. The knowledge part of a TAO is more application-specific. For example, in a multimedia mail system, the knowledge is about reading, replying, editing, forwarding a mail, etc. While in a medical image system, the knowledge is about diagnosis studies, different image modalities for diagnosis studies, image processing, etc.

4.4 The Interpreter

For each newly created TAO, the Interpreter will transfer the knowledge part K from formatted form to ICs and merge into the local knowledge base. Since the rule set is in a well-defined format, the interpreter can easily generate customized IC dynamically in AMS. In other words, we can use the AMS interpreter to generate a new knowledge function g in an IC dynamically, if necessary.

4.5 The Event Filter

There are numerous events occurring in the system, but for different applications only some events are meaningful while others are not. Therefore, we can use an event filter to filter out events and that can be ignored and allow only meaningful events to enter the AMS.

4.6 The Smart Engine

The Smart Engine maintains the current ICs according to the messages. Since the message is generated by some events, the AMS system is event-driven and can respond to the environmental changes automatically. The message is generated either internally, a result from ICs, or externally, a result from the event filter. The Smart Engine operates as follows: it takes messages and distributes to specific ICs, then checks the corresponding acceptance functions f. If the acceptance function is satisfied, it removes the messages and invokes knowledge function g to fire the IC and performs the action sequence.

When the TAO is not available, it indicates that the corresponding IC is in a dead state, so the Smart Engine needs to remove the IC from the Active Index, and that IC is then withdrawn from the local knowledge base.

5. APPLICATION TO MULTIMEDIA MAIL

Based upon the AMS, we can implement different applications, e.g. smart medical image system, home shopping system, real-estate survey system, etc. In this section, we describe a Smart Multimedia Mail system (SMM) which is based on both AMS and e-mail system.

In SMM, each multimedia mail is a Teleaction Object. SMM provides a simple user interface so that the end user can create his/her private knowledge for individual mail messages. For example, the user can set alarms for his/her important mail (illustrated by the example described in Section 3); collect statistical information; re-route the mail to other recipients; or determine the schedule for a group. The user can also define template mail with associated template knowledge, such as a weekly report, sign-up sheet, etc.

The SMM is implemented in four functional blocks as shown in Figure 7. The Mail-Editing and Mail-Browsing blocks can be implemented using the AMS Editor and AMS Browser tools respectively. Just two blocks are left for the application designer: Mail-Handling and Mail-Knowledge-Editing.

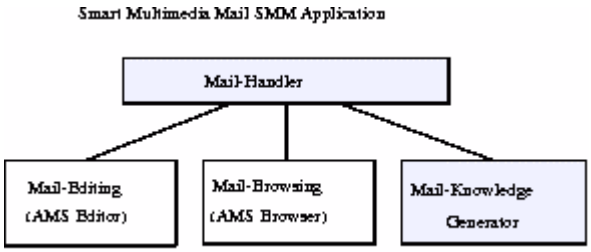

Figure 7. Function diagram of the Smart Multimedia Mail system SMM.

5.1 User Interface of SMM

The SMM begins with an ordinary mail-board. The user can view and create a multimedia message. When receiving a SMM mail, SMM will pass the private knowledge associated with this mail to AMS and merge it into the reader's local knowledge base. Therefore, whenever an event corresponding to the mail occurs, the IC in AMS will trigger and perform the related actions. When the user views a mail, SMM uses the browsing tool provided by AMS for display and browsing of the mail (a TAO) by the hypergraph layout window and zoom-in windows for detailed browsing, as shown in Figure 8. When the user wants to create a new multimedia mail, SMM also uses an editing tool provided by AMS which supports several ways to get the media data: from file, from database or from live sources.

Besides creating the multimedia mail, the user can also add his/her private knowledge to the mail. The knowledge is formally defined in Section 4, but the user should not be burdened with this detailed format. Instead, the user is provided with an application-specific, easy and friendly user interface. Therefore, SMM needs to provide its own application-specific user interface for gathering the information, using its generator to create formatted knowledge for the private knowledge. Later, the AMS interpreter transfers and merges the knowledge into the local knowledge base. Therefore, in SMM two windows are provided for knowledge acquisition. After the user has composed the new mail content, he/she can add his/her own knowledge to only this mail. Thus, he/she needs to open the knowledge window first, as shown in Figure 9.

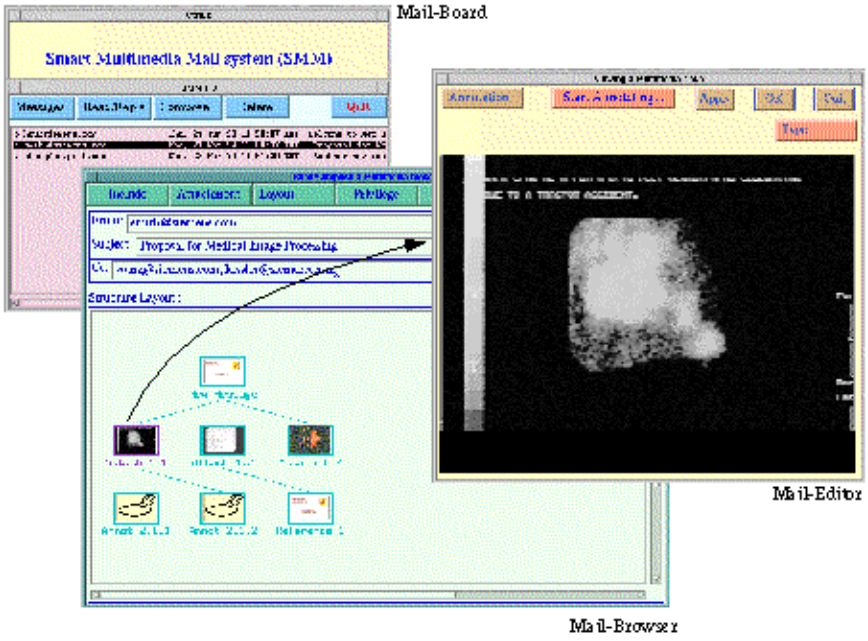

Figure 8. Browsing/Editing in Smart Multimedia Mail system (SMM).

There are two parts for specifying the private knowledge: event sub-panel and action sub-panel. The event sub-panel specifies the events, and the action sub-panel specifies the actions. Each panel can accept more than one statement in `and' connection. In Figure 9, another dialog window is shown to gather such information. Basically, we constrain the knowledge solicited from the user into the following four elements: Who, Doing, To Which Object, and When. (Currently, 'Who' is restricted to have a single value in the conjunctive logic clauses for all event statements in the same event sub-panel.)

The user can use the menu to specify the desired behavior. Moreover, the user can use the mouse (or cursor) to point at objects to specify what specific part of the message he/she refers to. This menu-based interface for specifying the private knowledge of a smart object in SMM is easy to use. All the user needs to do is to select menu items and to point at the objects without worrying about the detailed format of the knowledge. Then, SMM will automatically transform the user's specification into the formatted knowledge by applying the following algorithm.

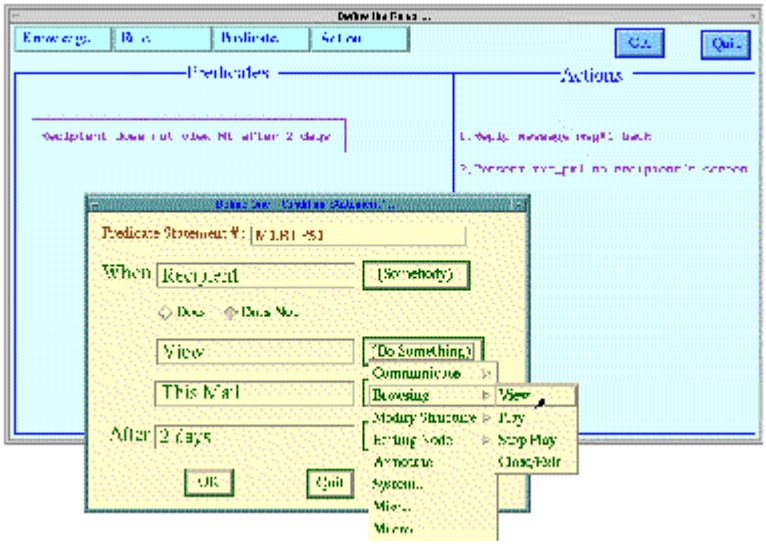

Figure 9. Dialog windows for the private knowledge in SMM.

1. Generate EVENT definitions of SMM.
2. Generate ACTION definitions of SMM.
3. Generate the IC Title with the class name "root.TAO.mail" concatenated with the TAO Id.

/* Assume each sub-panel has one or more statements that are conjunctive */

4. For each statement with "When", use the value of "When" to generate an alarm and place it into the Sender_ALARM set or Recipient_ALARM set according to "who".
5. If Sender_ALARM set is not empty, generate the FSM trans:
   (State0, State1, (SEND), (NIL), (a list of Set_Alarm for all alarms in the Sender_ALARM set))
6. If Recipient_ALARM set is not empty, generate the FSM trans:
   (State0, State1, (ARRIVE), (NIL), (a list of Set_Alarm for all alarm in the Recipient_ALARM set))
7. If both Sender_ALARM set and Recipient_ALARM set are empty, generate the FSM trans:
   (State0, State1, (NIL), (NIL), (NIL))
8. For each event sub-panel /* private knowledge*/
   8.1. For each statement with "When", generate the FSM trans:
   /* setup the alarm & actions*/
   (State1, State1, (event in this statement), (NIL), (Cancel_Alarm of related alarm))
   8.2. For all statements without "When", generate the FSM trans:

/* setup actions */
(State1, State1, (collection of events in all of these statements and all alarms), (NIL),
(all actions in the action sub-panel))

9. generate the final FSM trans:
(State0, Dead, (DEL), (NIL), (NIL))
(State1, Dead, (DEL), (NIL), (Cancel_Alarm of all alarms in all sub-panels))

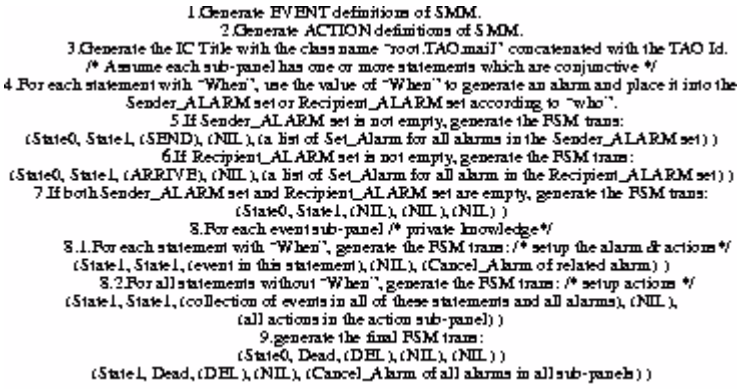

Figure 10. Heuristic algorithm to generate private knowledge in SMM.

5.2 The Knowledge Generator of SMM

After the user fills out the information in the event sub-panel and action sub-panel, the knowledge generator of SMM will gather all the information and transform it to the formatted knowledge defined in AMS. Following our example in Figure 9, the piece of knowledge is: "If the recipient does not view this mail within 2 days after the mail arrives, then send a message back to the sender and beep a message to the recipient". By applying the heuristic algorithm in Figure 10, the corresponding pieces of formatted knowledge are generated as shown below:

```
EVENT ARRIVE =   ...
EVENT VIEW =  ...
EVENT ALARM2 = ...
EVENT DEL =    ...
. . .
```

```
ACTION    Set_Alarm(own_IC_Id, duration, event_label, ...)
ACTION    Cancel_Alarm(IC_Id, event_label, ...)
ACTION    Reply_Message(user_Id, message_no, ...)
ACTION    Beep_Message(user_Id, message_no, ...)
. . .
IC    M1 #1234 "root.TAO.mail.#1234" INFINITE_TIME
   FSM :{State0, State1, Dead}
      /* setup alarm when arrival in recipient */
      (State0, State1, (ARRIVE), (NIL),
      (Set_Alarm(own_IC_Id(), 2_days, ALARM2, ...))
      /*Cancel alarm when recipient views it */
      (State1, State1, (VIEW), (NIL), (Cancel_Alarm(own_IC_Id(), ALARM2, ...)))
      /* Reply and beep if expired */
      (State1, State1, (ALARM2), (NIL), (Reply_Message(sender_Id(), mesg1, ...),
      Beep_Message(own_Id(), mesg1, ...) ))
      (State0, Dead, (DEL), (NIL), (NIL))
      (State1, Dead, (DEL), (NIL), (Cancel_Alarm(own_IC_Id(), ALARM2, ...) ))
```

6. APPLICATION TO MULTIMEDIA INFORMATION RETRIEVAL

To retrieve information in the hyperspace which is represented by a hyperstructure (i.e. a hypergraph structure), we can associate an IC with every recently accessed node in this hyperstructure. Thus the IC is a finite set of recently accessed nodes. The IC can be constructed as follows: It accepts a query and activates the adjacent ICs, and in turn posts queries to them. The action performed is to pre-fetch information items satisfying the query. A further refinement is to pre-fetch information items above a certain size. The justification is that we need only pre-fetch large information items, small information items need not be pre-fetched. At the University of Pittsburgh, we have added the IC to Mosaic, creating a new version called Mosaic-IC which has pre-fetching capabilities. An example of Mosaic-IC is illustrated in Figure 6 of Chapter 5, where the background window on the right displays the trace of executions of the IC, and the action_icons in the upper-right corner show the actions performed.

From the above application examples, it can be seen that the two important features we introduced in the Active Multimedia System - `active' and 'private' - are both realized by the Active Index. In AMS, the users can specify their private knowledge and then combine that with the system's knowledge, resulting in greater flexibility in AMS's adaptive behavior.

The `private' knowledge means polymorphism - certain objects can obtain reactions different from reactions to other objects even in the same environment. For example, in the AMS mail system described in Section 5, the users can specify their private knowledge such as the importance of different message classes. Therefore, the sender specifies his/her private knowledge (the importance of message classes) so that if the recipient forgets to view a particular message, a reminder will show up on both the sender's and the recipient's screen. Another interesting example is the inclusion of different levels of pre-fetching methods in Mosaic-IC, which is the Mosaic browser equipped with ICs. The AMS will provide two basic levels of pre-fetching methods. For a particular application, users can add their own pre-fetching methods based on their special considerations.

Basically, the Active Index maintains the `active' knowledge of AMS. In order to combine private knowledge with system's knowledge, in the Active Index we can divide the ICs into groups to form a hierarchy. In this hierarchy, one class of ICs can share the same methods. Messages sent to higher level ICs will be handled by the higher level methods. Only when there is no higher level method, will the message be sent to ICs at lower levels and handled by lower level methods.

Going back to the previous example of customized pre-fetching methods in Mosaic-IC, as shown in Figure 11, there are different classes of ICs in the hierarchy. XIC1 uses the simplest pre-fetching method of sequentially retrieving objects. XIC2 uses a smarter pre-fetching method by consulting both the current system profile and the user profile. XICT is designed to perform catalog searching, and can pre-fetch information from a special catalog. Therefore, most users using the Mosaic-IC will only need one or two levels of pre-fetching. But once the users get into the catalog searching application, they may need one additional level of pre-fetching.

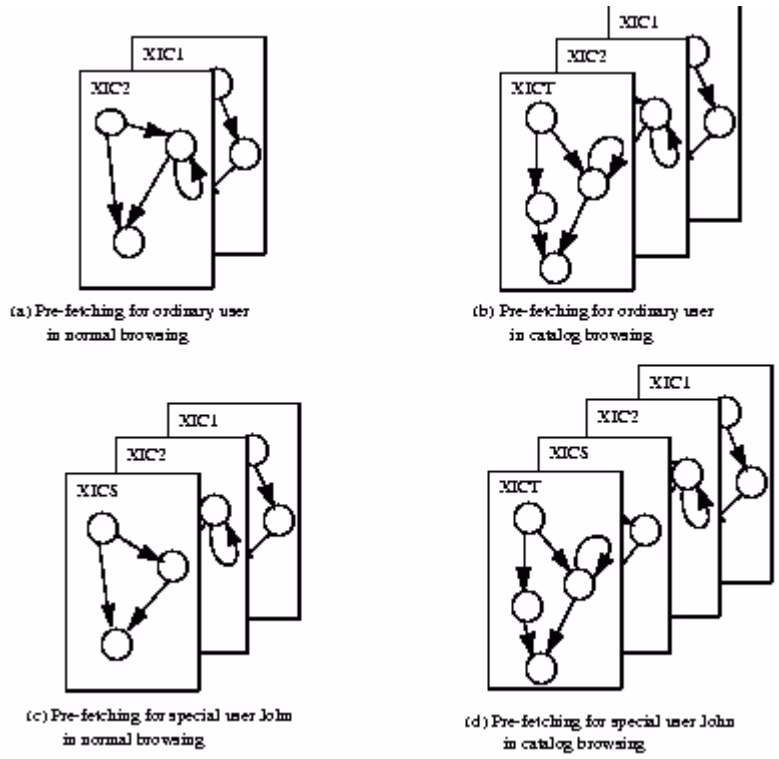

Figure 11. Different prefetching situations.

For different applications, different ICs from the hierarchy can be used. XIC1 pre-fetches in sequence, XIC2 pre-fetches by user/system profiles, XICS pre-fetches based upon the result of learning from users' past behaviors, while XICT pre-fetches by special knowledge in catalog searching.

Of course it is possible to have more levels of pre-fetching methods. For example, the AMS can use a learning monitor to observe the Mosaic-IC users' behavior. Then the AMS can develop a specialized IC at a new intermediate level with a customized pre-fetching method. For instance, XICS is for a particular user such as John who checks the technical reports on multimedia database very often using Mosaic-IC. Then John's normal navigation in Mosaic-IC will use three levels of pre-fetching methods: XIC1, XIC2, and XICS, where XICS has the highest preference in `multimedia'. But when John steps into the catalog searching application, then the pre-fetching methods may include four levels: XIC1, XIC2, XICS, and XICT. Combined, the pre-fetching will prefer the current interests in catalog

searching, then in `multimedia' information, and then according to user/system profiles and finally the sequential accessing of all objects.

In implementation, the message M is sent to the hierarchical group of ICs, not to an individual in the group. With a tag in the message M, the message M is either a normal message or a broadcast message. For the normal message M sent to the hierarchical group, the highest level will catch this message first. If the highest level cannot handle this message, it is passed to the next higher level until one IC catches it, and finally to the lowest level IC. For the broadcast message, it will be passed to the next level IC whether the current IC catches it or not. But in both cases, the message passing is in a sequential order, not in a non-deterministic order, as shown in Figure 12. For example, a KILL message for the hierarchical group of pre-fetching should be a broadcast message, while the PROFILE_CHANGED message could be a normal message that is caught only by XIC2.

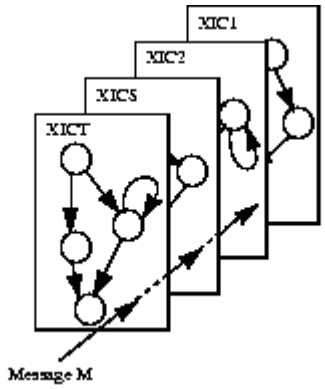

Figure 122. Message passing through a hierarchy of ICs.

7. DISCUSSION

A desired requirement for multimedia applications is that the user be able to interact with multimedia objects that are aware of environmental changes and moreover are able to dynamically incorporate unanticipated information to better react to environmental changes.

In this chapter we have presented the Teleaction Object model for the design of an Active Multimedia System. This model emphasizes a unified approach for the modelling of multimedia applications, presentation and communication, as a collection of interacting objects. Teleaction Objects are formally specified as (G, K) pairs. With the types of node, media, and link in

the hypergraph G, we can design algorithms for the efficient communication and effective presentation of multimedia information. Using multiple levels of knowledge realized by ICs, the user could customize the AMS and also apply private knowledge to certain group of TAOs.

Our approach has the following advantages as compared to the traditional AI approach: (1) The AMS system is not a rule-based system. It is based on the IC technique that will dynamically modify itself to perform various operations. (2) The TAO incorporates hypergraph structures that represent domain knowledge not easily captured by expert system rules. (3) The IC technique enables the system to localize a small set of rules in its operations. The AMS system, therefore, is more efficient than a general purpose expert system. ICs also share some characteristics of intelligent agents [ACM94], but the fundamental difference is that an IC is a data structure to facilitate information access and knowledge processing. We can use an IC to realize B-trees and other conventional data structures efficiently. An Active Index may have millions of ICs, whereas agent-based systems typically have at most hundreds of agents.

A prototype Active Multimedia System (AMS) for Smart Multimedia Mail application (SMM) has been implemented on SUN workstations. Currently, our implementation effort for AMS is taking a direct approach. It has been our concern for the standardization regarding the implementation of AMS. As mentioned in Chapter 2, an ISO standard for programming environments for the presentation of multimedia objects, called PREMO [Herman94], has drawn a lot of attention. PREMO addresses the issues of configuration, extension, and inter-operation of and between PREMO implementations. From its conceptual framework, PREMO is based on an object model in which object operations can be synchronous, asynchronous, or sampled. Besides active objects, events are used as the basic building block for its event model. Standards like PREMO will eventually provide a standardized development environment for active multimedia systems such as AMS.