Media Banks: Entertainment and the Internet

by A. Lippman
R. Kermode

There are two emerging models for delivering high-density, synchronized audiovisual presentations: video-on-demand and the Internet. The first is based on long-lived connections and guaranteed timeliness; the second assumes short spurts of low-bandwidth data on demand. We present the design for a Media Bank that intermediates between them. It provides on-demand access to media elements that are assembled on-the-fly by the recipient to reproduce synchronized audiovisual presentations. The Media Bank uses a fully distributed architecture that assumes a community of viewers. Any member citizen can request or deliver sound, picture, descriptive annotations, or programs to control assembly. Data are redundantly stored in small segments and are cataloged by content and format to facilitate personalized and interactive retrieval.

The notion of a Media Bank first arose at MIT in 1991 during discussions about building a virtual communications research program from diverse activities already in progress throughout the campus. An under-researched area was a "bit-bucket"—a simple repository for all sorts of data where large-scale, synchronous content could be easily deposited and later withdrawn. Video servers were not widely discussed in 1991; however, constructing such a bank would yield potentially valuable insights into new uses for digital video and permit diverse groups across the campus to cooperate.

It was a small step to distribute the functions through a large set of machines and invert the notion of a concentrated video server from a single (or small) set of large machines to a large set of very small ones. But this small step has ramifications. Video is a "dinosaur" of a data type. Not only is it big and unwieldy, it is notoriously slow to evolve because of the engineering lag and a large installed base. Solutions for digital video processing and storage require new technology and new investment. Economies of scale and construction have given it a history like that of broadcasting, where a few sources synchronously drive a great many, simplified receivers. The very success of the medium for mass entertainment and entrenched commercial interests impedes change and begs the question of why one would care to fix something that is demonstrably not broken.

However, in spite of fifty years of minimal engineering optimization, television is now becoming a digital medium. The sheer increase in program capacity afforded by compression motivates such a change, as does the opportunity for new distribution channels and quality options. Commercial ventures that are more than technologically motivated are in place in North America (DIRECTV** system) and emerging in Europe (Astra digital video broadcasting). The Federal Communication Commission (a United States government agency) inquiry into advanced television services (ATV) is nearing completion and will guide the transition of the UHF (ultra high frequency) broadcast band from sparsely populated analogue telecasts to up to one gigabit per second digital radiation. (It remains to be seen whether that bandwidth will be

0018-8670/96/$5.00 © 1996 IBM

allocated solely to higher definition services or whether it will be generalized and opened to any commercial use.) International organizations such as the Digital Audio Visual Council (DAVIC) are developing open standards for networked access to television and sound. As the last data type to fall, digital television now flows through the same channels as the rest of the media that we encounter in our daily communicative lives.

At the same time, we are witnessing the meteoric rise of the Internet as a *de facto* global infrastructure for both personal and commercial information. This network was originally envisioned as reliable support for distributed computing (1969). By the mid-1970s, an Advanced Research Projects Agency (ARPA) study found that three-quarters of all ARPANET (ARPA network) traffic was electronic mail (e-mail), and in the 1980s the network became a locus of commercial and personal interactions. By combining the network with consumer use of the personal computer, the network has now become an entity in itself ("the Internet"). In addition to becoming a foundation for global information infrastructures targeting commerce, research, education, and medicine, the Internet has become a democratized, participatory society. Its growth demonstrates the power of *community* versus *interactivity*; participation eclipses transaction-based full-service networks and asymmetric client-server designs.

In contrast to video service, the Internet is inherently symmetric. Any computer connected to it has the same potential access and abilities as any other—there is no systematic bias or barrier built into it. Many researchers attribute its popular growth to this simple feature. Equality of access is the basis for international communities, newsgroups (forums for discussion and information postings), chat rooms, and the plenitude of home pages (user-owned and -maintained nodes for public access of personal, commercial, and reference data) on the World Wide Web (a portion of the Internet where data exist in an *ad hoc* standard format). Whereas library retrieval and interactive information and transactional services have been slow to diffuse through society, the community aspects of the Internet and the reduced cost of entry now accelerate its use.

This impels us to consider mechanisms for media distribution that do not emulate broadcast. In particular, we address extending the ethos of the Internet and its hyper-library nephew, the World Wide Web,[1] to accommodate high bandwidth, real-time, synchronized data such as audio and video. In effect, we build a bridge between the fully distributed nature of the Internet and the requirements of audio and video media (see Figure 1).

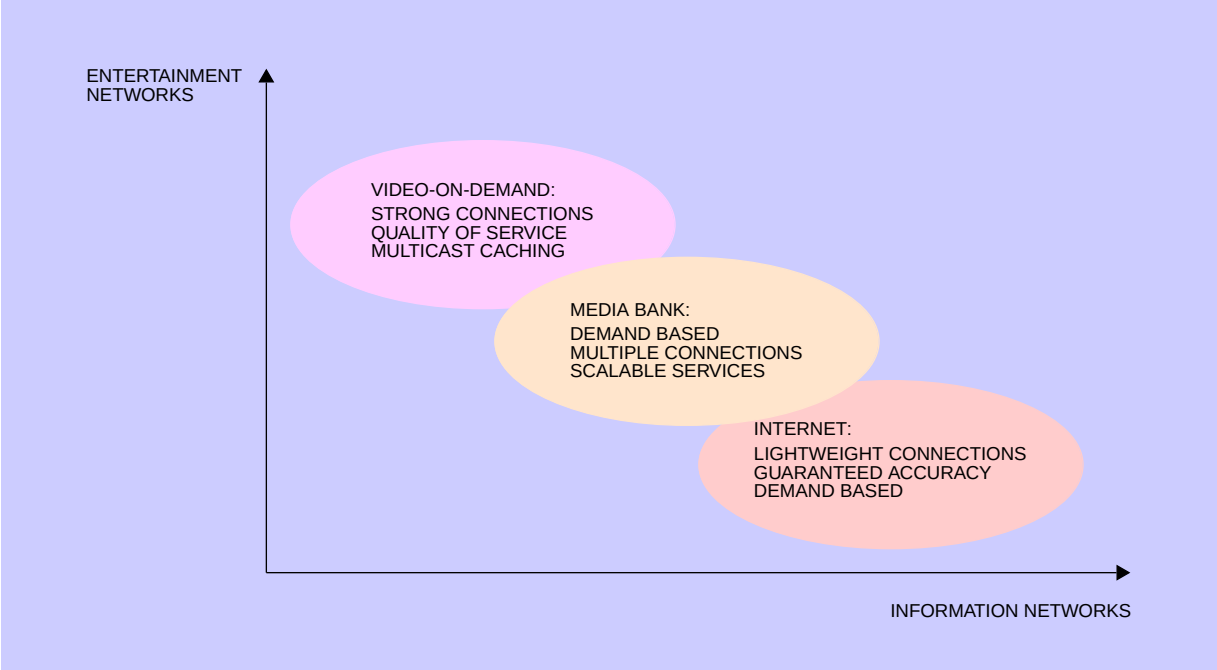
The primary design point is the need for synchronous delivery. Commercial data can be lost in transit with no ill effect—the datagram is simply retransmitted milliseconds later. Accuracy is more important than timeliness (at least at small-enough scale). Conversely, the denouement of a narrative cannot suffer delay. It can be distorted or degraded by transmission errors, but the definition of video is in its continuous flow. Maintaining continuity from distributed resources in the face of errors, sporadic system availability, and variable demand preoccupies designers of video servers.

The second fundamental challenge is that the video of today is inherently formatted information. Unlike text, which can consist of individual characters whose pagination, typeface, and layout are often packaged separately from the message, a video sequence embeds the data in a predefined raster size, frame rate, aspect ratio, and color space. The demands of these parameters ripple throughout the system, placing constraints on channels, storage media, processing hardware, and display technology. This mitigates strongly against content dispersion and reinforces a necessity for concentration of information.

In 1996, these concerns can no longer be considered in a vacuum. Video service of all sorts has broken the surface in discussions of national and global infrastructures and as a new consumer entertainment opportunity. *Video dialtone,* the local telephone entry into television distribution, is being tested in all regions of the United States and many European countries. There are at least 57 small- and medium-scale advanced service tests throughout America, and all seven regional operating companies have filed construction permits with the Federal Communication Commission to install broadband networks. Salem, Massachusetts, site of Alexander Graham Bell's first long-distance telephone call (from the Lyceum auditorium to *The Boston Globe* newspaper office) is poised to be an early test site for long-distance video service. At least two digital direct broadcast satellite systems are commercially deployed; both are showing early signs of success.

Other researchers have embarked on similar paths. In particular, Rowe et al.[2–4] have built distributed video

Figure 1 Comparison of differing network requirements



servers for on-demand access to large audiovisual archives. Their goal is scalable storage of bulk audiovisual data such as movies and classroom lectures. Data in their system are represented in a variety of formats with annotations and reference information included in the design. A "Compound Media Object" model allows flexible assembly of component information into coherent presentations. Synchronization is shared between the servers and the clients.

The design ethos used for building the MIT Media Bank is similar. It is driven by four main ideas:

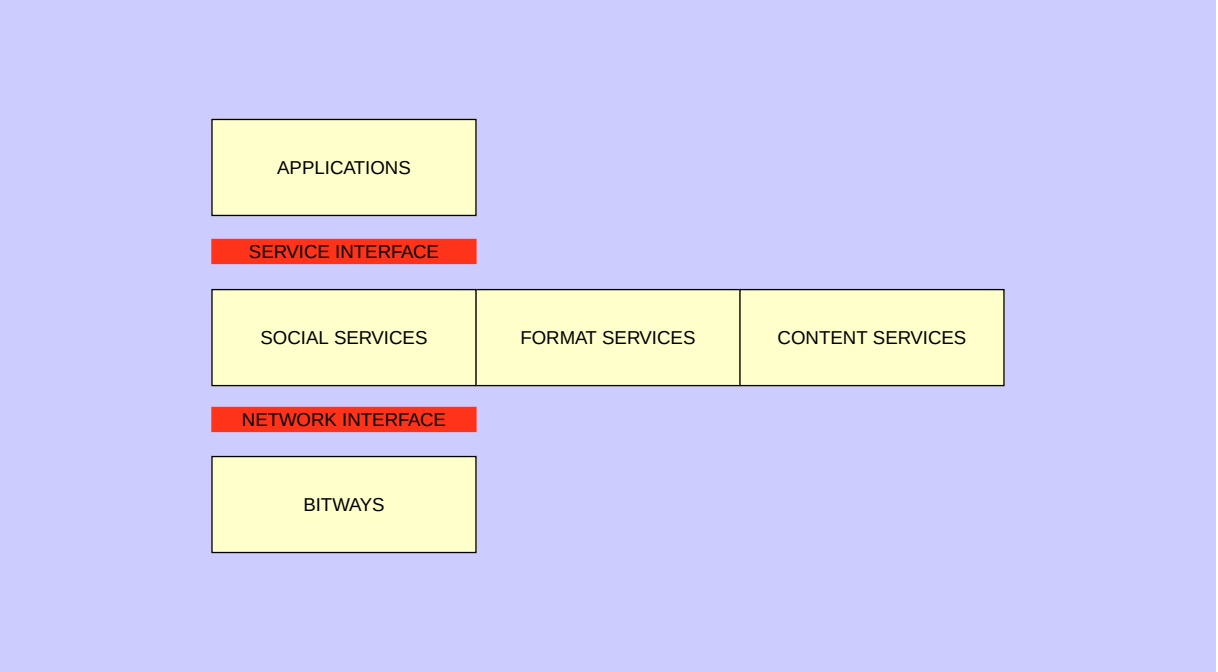
- Cross network operation
- Seamless distribution of services
- Stored descriptions
- Multiple format primitives

Entertainment program assembly illustrates many of these principles. Elements of a movie consist of a separately stored series of images, sounds, and descriptive data. These data may be contained in a single storage system or be replicated on several. Also, the data may exist in a diverse set of formats, for example, JPEG (Joint Photographic Experts Group),[5] MPEG (Moving Pictures Experts Group),[6,7] AC-3 (audio), and other formats. A separately stored, list-based structure contains the information needed to assemble the diverse parts into a coherent presentation and maintains a directory that locates the original material.

For example, in the case of a movie, the editing instructions to assemble diverse versions of the plot (X-rated; PG, or parental guidance required; 30-minute showing; etc.) may exist at one site but the images may be stored separately from language-dependent sound tracks. The instructions for billing and authentication of the viewer may reside elsewhere. The movie becomes a coherent event when it is requested—its display draws upon resources as needed. Just like the leaves on trees that "make no sound" if they fall in an unoccupied forest, this movie literally does not exist unless someone demands a viewing.

Seamless distribution of services implies that any of the processing functions described can occur in any of the systems attached to the network of the Media

Figure 2 The Media Bank multilayered network structure



Bank. Many may occur in a hypothetical digital television receiver in the home itself, but they could as well be done by authentication servers, editing systems, dubbing servers, storage servers, and the like. Note that this model for scalable storage and distributed processing extends familiar computing notions to entertainment—the World Wide Web does the same thing for paginated text. We argue that we can provide most services and functions by carefully crafting the data formats and basic elements to suit existing storage and processing models, rather than the other way around. A unique storage architecture for time-sensitive data may not be mandatory.

We adopt the style of many emerging national and global information infrastructure (NII/GII) efforts: a multilayered network structure that simplifies *applications*, by providing generic and specific *services* that are delivered through multiple *bitways* (see Figure 2). This model implies open interfaces between the layers, and stresses the generic intermediate or *services* layer. For the special case of media service, we expand that layer into three components, called respectively, social services, format services, and content services. Each has different characteristics.

Social services include billing, authentication, and any other information services (lexicographical, e-mail, etc.) that are normally associated with networked computing.

Format services are operations performed on the data on behalf of any client—for example, translation from MPEG to another encoded representation, stereo to monaural conversion, or recording of any kind. These are operations that occur on the interface to access the data and facilitate use in a diverse environment.

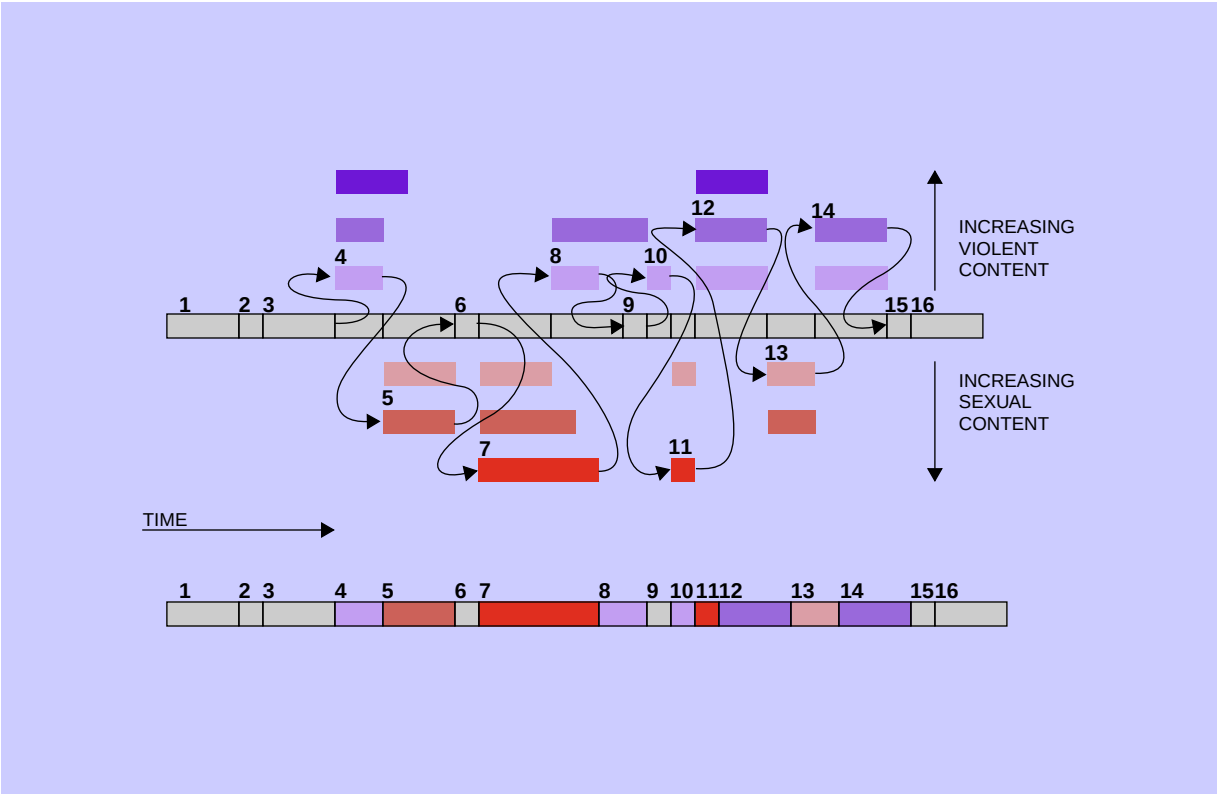
Content services are distributed processing tasks that relate to annotations and assembly rules.

There are only three elements needed to develop a Media Bank:

- An indexing and storage scheme
- A suite of client programs to access data
- A delivery protocol

In essense, the indexing system is optimized for representation of audiovisual elements stored in elemental or byte-sized chunks. These are called the Media

Figure 3 Personalized movie constructed from scenes of varying violent and sexual content



Bank protocols, and they are overlaid onto a suite of access and delivery mechanisms that are themselves optimized for networked delivery of randomly sized and located elements. Scalable and layered video are prime examples. Part of this component of the protocol provides for file access into variable sized arrays, needed, for example, to access selected frames of a coded video representation.

Access programs are the applications. These provide means to peruse the information distributed through the bank and to simulate television receivers. A growing repertoire of such programs is available to summarize information, transform it between coded formats, merge text and visual data, and incorporate authentication and payment mechanisms.

The delivery protocol is divided into two domains: networked distribution of synchronous information and specific protocols for access to *meta-data*[8] about the data.

Indexing and data representation

Choosing an efficient, malleable, and intuitive way to index and represent data is one of the most important design tasks of the Media Bank. Without an appropriate representation, indexing becomes slow, and delivering data from multiple isochronous sources to a client becomes impossible.

The representation chosen separates media objects into two parts. The first part consists of meta-data about the object. For example, the meta-data for a video clip object would describe the length of the clip, its resolution, its compression format, and information needed for indexing and retrieval. The meta-data could also include a function that describes a browser for viewing the video. The actual data for the video clip are not stored with the meta-data. Instead, they are stored in the second part, and referenced from the meta-data part. Several key benefits are afforded by segregating an object's meta-data and data.

The first benefit is that one need not retrieve the entire object to learn its characteristics, one need only retrieve the lightweight meta-data. Thus, one avoids the overhead associated with retrieving the entire object for the purposes of determining whether or not it is what is wanted, only to discard it upon finding that it is not. In other words, the overhead of filtering on the basis of content is greatly reduced.

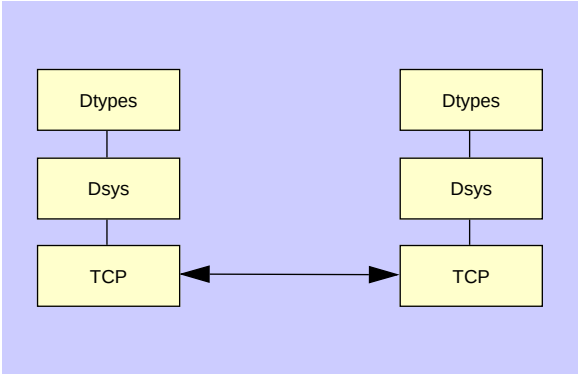
The second benefit arises out of the ability to derive objects from other objects. Consider the end result in the production of a movie or a television documentary. Typically hundreds to thousands of feet of film are logged and whittled down into a finely tuned, highly crafted linear result. The content of the movie becomes fixed and opaque. One cannot examine scenes from a different camera angle, tone down violent content, or request the 16:9 aspect ratio formatted version. Nor can one repurpose the material without having access to the material itself. Having the ability to derive objects from other objects removes all of these restrictions. Now, instead of the movie being viewed as a fixed single product, it may be assembled at the time of viewing. The decision of which particular clip and which parts of the clips are used becomes a function of the viewer's own preferences (see Figure 3).

The third benefit is really a corollary of the second taken from the content provider's point of view instead of that of the consumer.

Content providers have major concerns regarding access to their products. Clearly, controlling access to the bits of an object is important because it controls consumption of the object and hence enables the content provider to charge for the privilege of access. On the other hand the content provider also wants consumers to consume and to reuse the bits as often as possible, because greater use and reuse leads to greater revenue. Unfortunately, these conflicting goals are ill-served by the current broadcast model employed by practically all content providers today. As soon as a newspaper leaves the press or a television show is aired, access to the original material and editing decisions is lost and the material is rendered opaque for repurposing.

In our model one need not have access to the original footage in order to derive new content. Instead, all one need do is to derive a new object by combining select pieces of the proprietary one that remains secure on the content provider's own server. The end result is a new object that is an *edit decision list*, an object that can be exchanged privately without breaking copyright laws or losing functionality. Should a client wish to use the new object, the client still has to access the content provider's secure servers for the object data, and pay the appropriate royalty or access fee for the privilege.

Figure 4 Layered transportation stack for Media Bank object delivery



Media Bank objects are defined and distributed via a layered set of protocols (see Figure 4). At the lowest layer, delivery of meta-data is via Transmission Control Protocol (TCP). Above TCP is a protocol for the distribution of objects where the object type is carried along with the data, called *Dsys*.[9] Dsys allows diverse objects to be distributed, without advance knowledge at the receiver about the specific storage formats or requirements of each data element. Data for this protocol are a set of list-processing-like (LISP-like, actually *Scheme*[10]) vectors called *Dtypes*.[11] Dtypes include basic elements such as numbers and text arrays as well as extensions to define special-purpose types such as encoded bit streams.

Media Bank objects are accessed, located, and manipulated by a set of Scheme functions interpreted both at the point of origination (the server) and at the receiver.[9] This generalization allows all Media Bank objects to be both data elements as well as programs, and permits them to be interpreted and executed both locally and by remote agents. In operation, this means that a viewer for any object can be a program specific to that object, with user controls carried along with the data. Objects can therefore be assembled to provide a different television receiver or user interface for each movie, or could be organized into networked

Figure 5 Set of three objects for the trailer to the movie *Bad Boys*

```
#((uniqueid . "<badboys-video-mjpg>")
 (name . "badboys-video"
 (type . (item image mjpg))
 (version . 8166411078)
 (owner . "Henry Holtzman")
 (hardware-formats . #(mme))
 (data . "x-mbd://bad-taste.media.mit.edu:41503/raid4/mbdata/trailers/badboys/badboys.movie")
 (frames . 3277)
 (width . 320)
 (height . 240)
 (fps . 24))

#((uniqueid . "<badboys-audio-raw>")
 (name . "bad-boys-audio")
 (type . (item audio raw))
 (version . 8166411078)
 (owner . "Henry Holtzman")
 (data . "x-mbd://rocky-horror.media.mit.edu:41503/mbdata/trailers/badboys/badboys.audio.raw")
 (duration . 24100476/176400))

#((uniqueid . "<badboys-movie>")
 (name . "badboys-movie")
 (type . (item application playlist))
 (version . 8166411078)
 (owner . "Henry Holtzman")
 (script .
 #(
   (play-multiple
    #(
    #(
       #(play #((object . "<badboys-video>"))))
    #(
       #(play #((object . "<badboys-audio>")))))))))
```

games. Further, the execution of the code associated with any object can be delegated, facilitating automatic translation of data types, accounting, and security.

An example of a set of three simple Media Bank objects for the trailer to the movie *Bad Boys* is shown in Figure 5. Notice that the first two objects contain the meta-data for video and the audio to the movie, respectively, while the third object indicates that the two are to play synchronously to form a single presentation. It should be noted that these objects contain the base minimum in terms of information for their retrieval and playback. Extra name value pairs can be added at will (with the exception of a few reserved names), so that the description about the object is free to grow over time.

It should be noted that fixing the location of the object data in the data field of the meta-data for the object does not result in the same problems of brittleness as for the Internet World Wide Web Uniform Resource Locators (URLs). The reasons for this will become clear upon reading the section on directory servers.

Data delivery protocols

Central to the design of the data delivery protocols[12] is the notion of pull-based delivery. Pull-based delivery can be defined as "data delivery only in response to a request." While this delivery method is currently used by the World Wide Web for text and still images, it has only recently emerged as a contender for the delivery of large data types such as audio and video. Traditionally these data types have been delivered in an asymmetric broadcast fashion from a central head end that pushes data onto the network regardless of whether a client can handle the data or not. By employing the pull-based paradigm, one breaks nearly all of the usual assumptions made regarding the delivery of digital video and audio. Instead of a distribution mechanism that broadcasts from a centralized location to low-capacity clients, uses bandwidth asymmetrically, and puts the synchronization control at the server, the Media Bank uses an architecture in which

- The data are fully distributed across many servers in different locations.
- Multiple instances of the same objects can be duplicated at different servers.
- Bandwidth is considered to be used in a symmetric fashion.
- The clients are endowed with a degree of intelligence to perform a range of tasks that were previously the sole purvey of the server.

Two main types of servers in the Media Bank provide the bulk of the functionality listed above. The first kind of server is the directory server that provides a directory service for finding out where an object is located. The second kind is the object server on which the actual data are stored. Between them the object and directory servers provide access to objects stored in the Media Bank in a seamless fashion that:

- Is fault tolerant
- Allows both object and directory servers to be added and removed from the Media Bank at will
- Manages the load of both computation and network usage across the servers of the Media Bank and the network links that connect them

Directory servers

Directory servers provide the means for locating objects stored in object servers in a manner akin to that used by the UNIX** Domain Name Service for resolving machine names to Internet Protocol (IP) addresses. The idea of directory service is not new,[13,14] and is the subject of ongoing research particular amongst the Internet World Wide Web community. This section describes our implementation of a directory service.

Figure 6 Example of a network unique identifier (NUID)

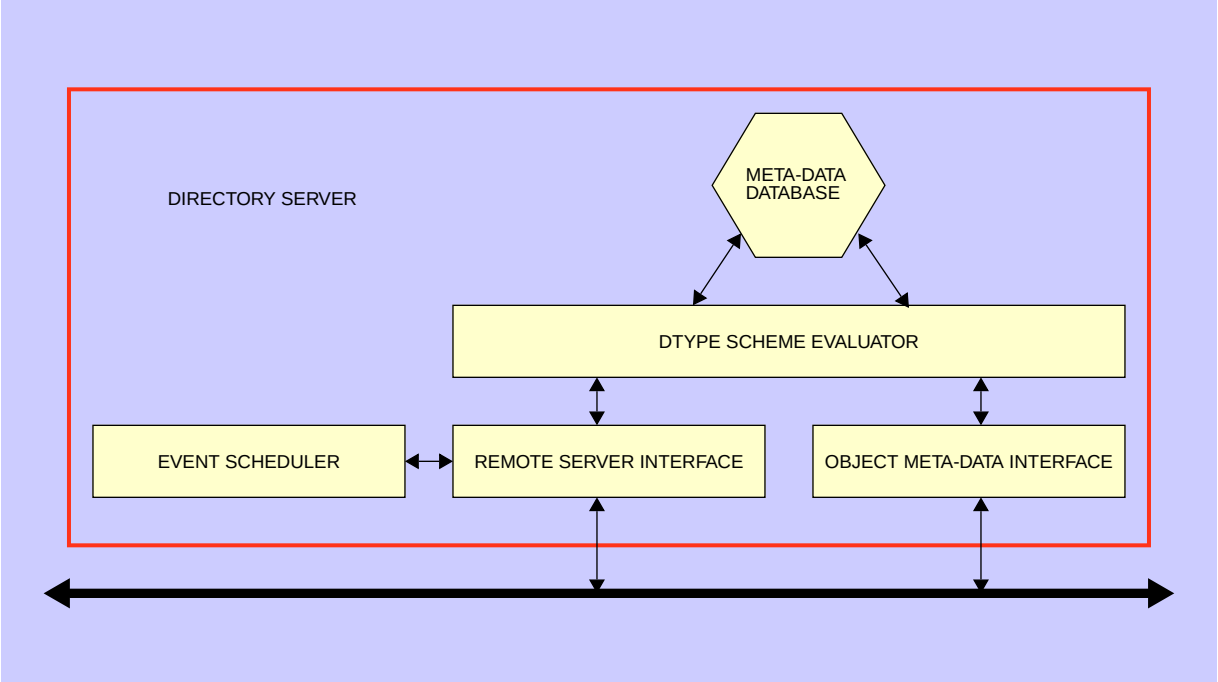
```
#((uniqueid . "badboys-video-mjpg")
      (host . "rocky-horror.media.mit.edu")
      (port . 41501))
```

Objects in the Media Bank are identified by one of two mechanisms: names or unique identifiers (UIDs). The difference between the two is that there may be multiple objects with the same name in the Media Bank but only one object instance per unique identifier. Object equivalence is assigned on the basis of name. In addition, unique identifiers come in two flavors: simple and network. A simple unique identifier assumes that the host and port of the object server is already known and that the unique identifier identifies a single object stored on that particular object server. A network unique identifier (NUID) differs from a unique identifier (UID) in that it includes the host and port number of the object server with the simple UID to form an identifier that is unique to the entire Internet, not just a particular machine. An example of a NUID for the object with the name "badboys-video" is shown in Figure 6.

Apart from the abstraction of names and unique identifiers, Media Bank directory servers provide a number of additional services. Specifically, they provide:

- The ability to detect dead or unresponsive object servers and remove references to them when responding to requests to resolve object names

Figure 7 Directory server functional block diagram



- The capability to store a subset of the meta-data information about an object that may be used by the client to prune unnecessary accesses to object servers for an object that does not fully meet the preferences specified by the client
- The facilities to manage the computational load between servers capable of serving equivalent objects

The inclusion of the these three additional services, along with the naming abstraction scheme are key features of the Media Bank as they solve the problem of brittle URLs that is inherent with the current implementation of the World Wide Web. No longer will the going down of one server automatically result in catastrophic blocking of a request for an object, since the network unique identifier of another equivalent object will be returned, should one be available.

The implementation of these services is reasonably straightforward. Each directory server consists of five main functional units, as shown in Figure 7.

At the core of the directory server is the Dtype Scheme evaluator. Built upon the Dtype data manipulation and Dsys networking libraries, the Scheme evaluator executes Scheme code applets for the other four functional blocks. It should be noted here that the Dsys networking library (through which the remote server interface and object meta-data interface connect to the network) uses a connectionless protocol on top of a modified TCP/IP stack in which connection establishment timeouts have been shortened to fractions of a second. While in general TCP/IP may not have the best performance in terms of latency, for local networks its performance has been found to be acceptable for real-time access. Work is currently in progress to replace the TCP/IP portion of the Dsys library with a real-time networking protocol that uses multicasting, distributed caching, and Application Level Framing (ALF).[15] This new protocol may also use RSVP[16] and eventually be used in an IPv6[17] environment.

Object servers communicate with the directory server via the remote server interface. Through this interface they inform the directory server of their presence, their current state, and make requests for directory service via regular remote procedure calls (RPCs). Should the directory server accede to a request by an

object server, it will reply in the affirmative indicating when the object server should next check in. In addition, the directory server may also schedule an update event to synchronize its database with that of the object server should the version number of the object server included in the original request for directory service not match the one inside the database of the directory server for that object server. Should an object server fail to re-request directory service by the time indicated in the reply, then the directory server will mark the object server's objects as "temporarily unavailable."

When a directory server requests an update from an object server, it stores the result in a meta-data database and, at the very minimum, stores the NUID for each object in a database keyed on the name of the object. Thus resolving a name into a list of object NUIDs becomes very simple indeed. In addition to the NUID, additional meta-data may be stored with each entry for the object. For example, the object type (e.g., image/mjpg, audio/raw, application/playlist) is currently appended to the NUID vector to speed up pruning unsuitable objects on the basis of type without having to contact the object server for the object.

Clients wishing to access the Media Bank initially do so through the object meta-data interface module. This module takes the client's request in the form of an RPC (remote procedure call) and then extracts and queries the meta-data database on the client's behalf. Among other things, the interface removes the NUIDs of objects on object servers that failed to check in with the directory server. A typical exchange between a client and the directory server is shown in Figure 8.

Notice that the directory server responds to the client's request for information regarding objects that match the name *t2-movie-browser* with an object with type set to *item multipart locator* and the data set to a vector of length two, where each entry corresponds to an object stored on another server. The two entries in the data vector are in no specific order and must be further examined before the client can determine which one is the more appropriate. The mechanisms involved in this process will be discussed in the section detailing the operation of Media Bank clients.

Figure 8 Typical client/directory server interaction

```
Client to directory server
      (item-fetch "t2-movie-browser")

Directory server response
      #((type item multipart locator)
       (data . #(
         #((name . "t2-movie-browser")
          (uniqueid . "<t2-movie-browser>")
          (type item application sprog)
          (host . "alphaville")
          (port . 41501)
          (load-average . 1.36155525677))
         #((name . "t2-movie-browser")
          (uniqueid . "<t2-movie-browser>")
          (type item application sprog)
          (host . "bad-taste")
          (port . 41501)
          (load-average . 3.26552888373)))))
```

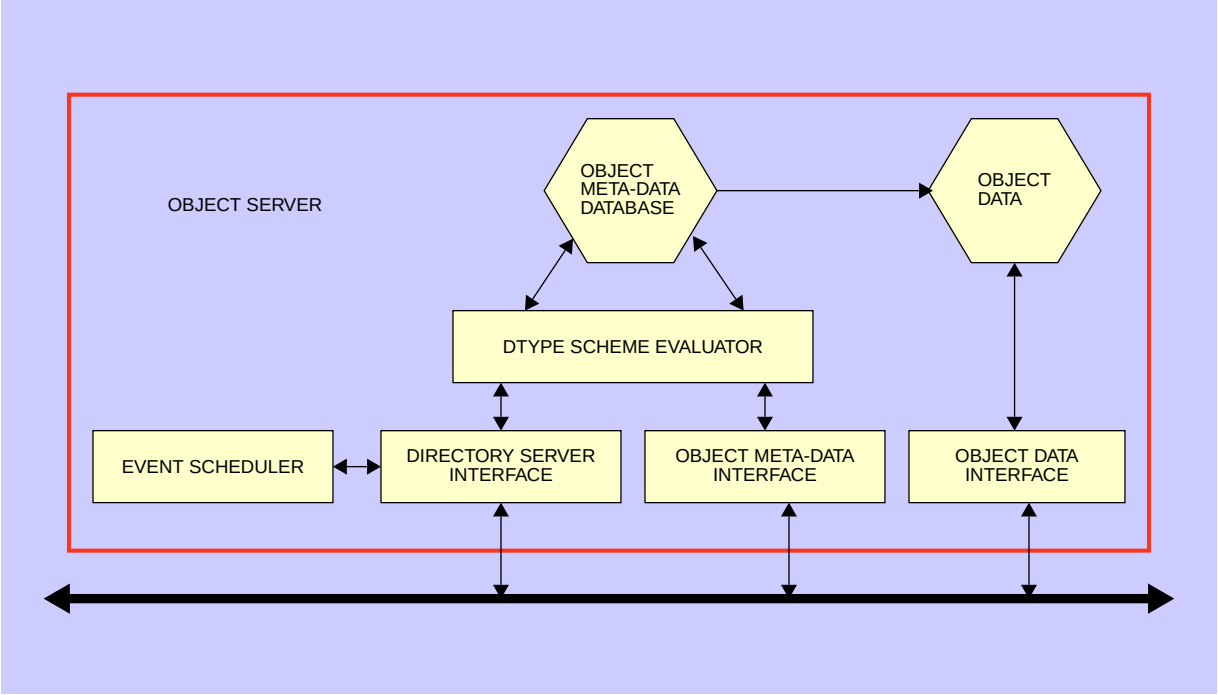
Object servers

Object servers do exactly what their name suggests, deliver objects. However, in keeping with the design philosophies of pull-based delivery and segregating an object's meta-data from its data, the fashion in which they do so is a little different from normal. A block diagram showing the internal pathways of a Media Bank object server is shown in Figure 9.

Like the directory server, the object server's core consists of a Scheme evaluator built on top of the Dtype and Dsys libraries. There is also an event scheduler that is used to schedule requests for directory service to the various directory servers that provide directory service for the object server. The directory server interface is somewhat different from the remote server interface found in the directory server. For one, it initiates object server/directory server communications. The directory server will not contact the object servers unless the directory server initiates the first contact. This is best shown by describing the communication exchanges that result when a new object server is brought on line.

Upon startup, the object server first attempts to find a directory server from a local list of possible directory servers. As soon as the object server locates a directory server that is active, it issues a joint request for its objects to be served and also for the most recent list of directory servers. Assuming that the response is in the affirmative, the directory server returns the time for the next subsequent check-in along with a list of other

Figure 9 Object server functional block diagram



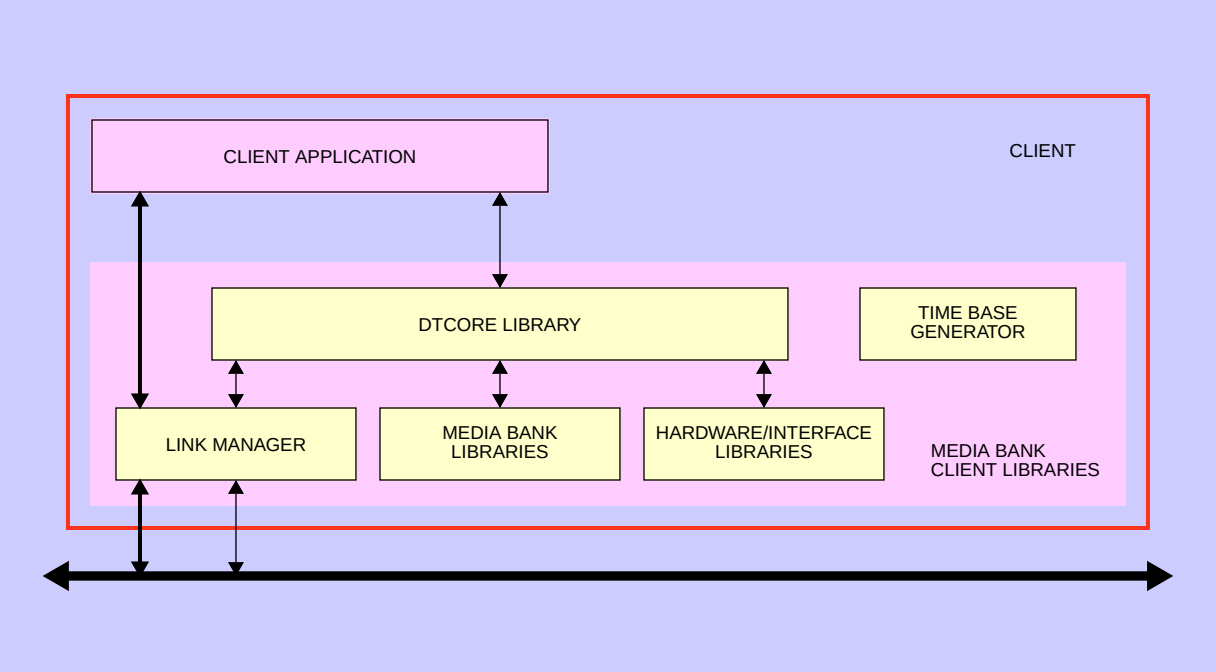
directory servers that this object server should contact. At this stage the object server schedules the check-in event and replaces the original list of directory servers with the new one. The object server then repeats the process contacting each directory server in the new list, this time requesting directory service only.

When the directory server receives and subsequently accedes to the request for directory service, it schedules an update event to occur shortly after sending the reply back to the object server. When the event scheduler determines that it is time for the update to occur, the directory server contacts the object server and requests an update based on the difference between the version number of the update request and the current version number of the object server. The response of the object server is generated using the version number difference and results in a list that selectively adds and deletes objects from the meta-data database of the directory server.

The meta-data interface of the object server is identical to that of the directory server, because the communications protocol for meta-data access is symmetric. This is important as it means that any media bank server or client can access any server and use the same suite of RPCs to extract meta-data information regarding objects known to that server.

Perhaps the biggest difference between directory servers and object servers is the ability of the object server to serve the actual object data via the object data interface. Two different protocols are currently supported at the object data interface. The first is the networked file system (NFS), but the amount of control possible over the latency and loading of the network is somewhat restricted. In fact the grade of service possible in accessing the object data is inferred from the state of the meta-data interface. The second protocol is a custom protocol developed at the Media Laboratory called the Media Bank data protocol (MBDP). MBDP transmits data from the object server by packetizing within Dtype wrappers and transmitting them using a slightly modified TCP/IP in which the timers are tuned to expire considerably quicker than normal. This design achieves two things, the first being a greater control over the use of network bandwidth by clients,

Figure 10 Media Bank client functional block diagram



and second the ability for the Media Bank to provide remote access beyond the Media Lab's local network. Ongoing work is being undertaken to create the next revision of MBDP that will shift from TCP/IP to multicast UDP (user datagram protocol) as the primary means for delivering object data. This new version of MDBP will be completely distributed and will blur the distinction between clients and object servers by allowing the clients to cache object data and subsequently apply the data to their peers as requested. This merging of functionality should also enhance scalability and performance, and may lead to a single unified architecture that could be used to deliver both real-time communications and prerecorded entertainment on a global scale.

The directory and object servers for the Media Bank may not find wide deployment around the Internet in the near term; however, we believe that their architecture is representative of what servers will look like once gigabit per second networking, connectivity, and capabilities become commonplace. In such an environment, the need to store media on large centralized servers directly connected to the primary Internet backbone diminishes significantly. Objects will be spread and duplicated over the entire net. Furthermore, the ability to publish multimedia and have it seen will continue to diffuse further from publishing houses and into the home. However, changes to servers are only half the story as the clients will change dramatically as well.

Clients

One of the key differences between Media Bank clients and practically every other current Web client or network video client is the endowment of sufficient intelligence to perform *synchronized assembly of multiple isochronous objects* via pull-based delivery. The decision to shift to pull-based delivery stems from the realization that a client that buffers object data from multiple servers is in a better position to manage its buffers and synchronize playback from them than any of the servers would be. An example of this kind of presentation is combining several dependent video and audio streams from different locations during a sports broadcast. The ramifications of shifting these responsibilities from the server to the client are far-reaching. Basically, the client now becomes an extremely powerful entity that can operate autono-

mously from the network. No longer is the client tied to a single server, but the client is free to switch between servers whenever desired. Server dropouts and broken networks can both be detected and handled by the client as appropriate. In addition, the ability to access and pull data from multiple servers gives clients the ability to create a whole suite of heretofore unforeseen presentations that exploit both the wealth of information available on the network and the user's own preferences and tastes.

Accordingly, Media Bank clients are given the ability to execute smart objects that define procedures and applets in a similar fashion to those found in the Tcl/Tk (tool command language/toolkit) clients of the Berkeley VoD (video-on-demand) project[2,3] and the Java**-capable clients of Sun Microsystems and Netscape Communications Corporation. The Media Bank clients execute objects that are written in Scheme, the same language used to represent the object meta-data. Correspondingly, Media Bank clients tend to be very similar internally to the object and directory servers they access. A diagram showing the internal structure of a Media Bank client is shown in Figure 10.

There are five main modules in the Media Bank client library suite. As with the Media Bank servers the Dtcore (Dtype and Dsys) libraries and Scheme evaluator lie at the center. However, in addition to the Dtcore libraries there are several other libraries that augment the functionality of the Scheme evaluator.

The second module is the link manager, which manages the client's interactions with Media Bank servers. Like the object server, the link manager actively seeks out a directory server that will provide access to the Media Bank and cache the list containing any alternate directory servers that may be present. In addition, the link manager also keeps track of which servers are not available and provides automatic error recovery during server dropout.

The third module, the Media Bank libraries, consists of a suite of dynamically loadable shared libraries that extend the functionality of the Scheme evaluator by providing procedures to manipulate Media Bank objects. Negotiating access to objects and servers, parsing playlists, object type checking, name resolution, sorting objects on preferences, and executing smart objects are all handled by the Media Bank libraries. Written as a mixture of Scheme-only functions and Scheme hooks into optimized C++ code, these libraries are dynamically loadable, and as such can be replaced by more recent versions or upgrades without having to recompile the client application.

The hardware libraries make up the fourth module. Basically, these libraries provide access via Scheme to any specialized hardware on the local machine. Examples of hardware- and architecture-specific code supported are access routines to MJPEG cards, access routine to the native audio server, and X library support.

The fifth and final module is the time base generator. Derived from the audio server clock, the time base generator generates periodic events signals that are then sent to any objects that have been instantiated and are in use. For example, a movie player may be fed 24 update events per second that cause frames to be displayed and audio to be played in strict synchronization.

The beauty of performing synchronization in this manner is that implementation of fast forward, fast reverse, and slow motion becomes trivial. All one needs to do is to change the rate at which the update events are generated. If one wants to play the movie at twice the normal speed, then the time base generator needs only to issue updates at 48 updates per second or at 24 updates per second, with the difference in time code between each update being doubled to one-twelfth of a second. Likewise half-speed playback will result if the time base generator issues only 12 updates per second. Because the client's display is completely decoupled from both the playback rate and the recorded rate, this does not necessarily mean that the required network bandwidth is tied to the playback rate. If a client is only capable of decoding and displaying 24 frames per second, then every second update frame event will be skipped for the double-speed playback at the 48 updates per second example just mentioned. A more powerful client may set an upper bound on the number of updates per second that is willing to process in deference to other users.

Sitting above all these modules is the client application. Usually, the client application will be written in one of two ways. The first is as a precompiled C++ program that uses Dtype hooks to execute the necessary Scheme code to access the five modules just described. The second option is to write the application as a Scheme program that can then be suitably wrapped up to become a Media Bank object. There

are advantages and disadvantages to both of these methods. The C++ option will result in faster code, particularly for clients that perform any sort of memory or compute-intensive manipulation of video or audio data, while the Scheme route offers the benefit of portability and accessibility at the expense of speed. However, there is a provision for a way around this shortcoming; one can extract the compute-intensive Scheme code, rewrite it in optimized C++, and put it in a Scheme wrapper inside a dynamically loadable library that can then be sent to the client as an object as needed.

In order to best demonstrate the operation of Media Bank clients, we have included the following four examples that detail some typical client actions, and we also show screen shots from several Media Bank clients.

The first example shows the steps taken by a client that wishes to access the movie *Terminator 2* (see Figure 11).

First the application issues a request to the Media Bank client libraries requesting the location of objects with a name that matches *Terminator 2*. Other information may be present in the form of preferences that will be used to rank any matching objects on the basis of a series of tests between any of the fields contained in the objects. Upon reception of the request, the Media Bank client libraries will contact the first available directory server and request a list of objects that match the name *Terminator 2*. The directory server will return a list of objects that match (Figure 11, top).

Upon receiving this list, the Media Bank client then sorts the objects in the reply according to the users' preferences and also a number of system preferences (e.g., the preferred video format is MJPEG, or motion JPEG, because the client has an MJPEG card). This process is performed in four stages. In the first stage, *prefiltering*, any objects that mismatch on type are pruned from the list of possible matches (e.g., all still images will be discarded here). Assuming that there is still more than one object in the list, it is *sorted* on the basis of the information currently known about each object and the combined user and system preferences. At this stage one object may satisfy the preferences sufficiently well that no other object need be considered. In this case the meta-data for the object are fetched from the appropriate object server and then remaining objects are placed into a field tagged "backup-info" that is then appended onto the meta-data for the object. Should insufficient information be available to make a decision, the Media Bank client libraries will enter an iterative phase of *resolving* unresolved objects: fetching the meta-data about them from the object server that serves them, re-evaluating the preferences for the now resolved object, and *resorting* by inserting that object into the list (Figure 11, middle).

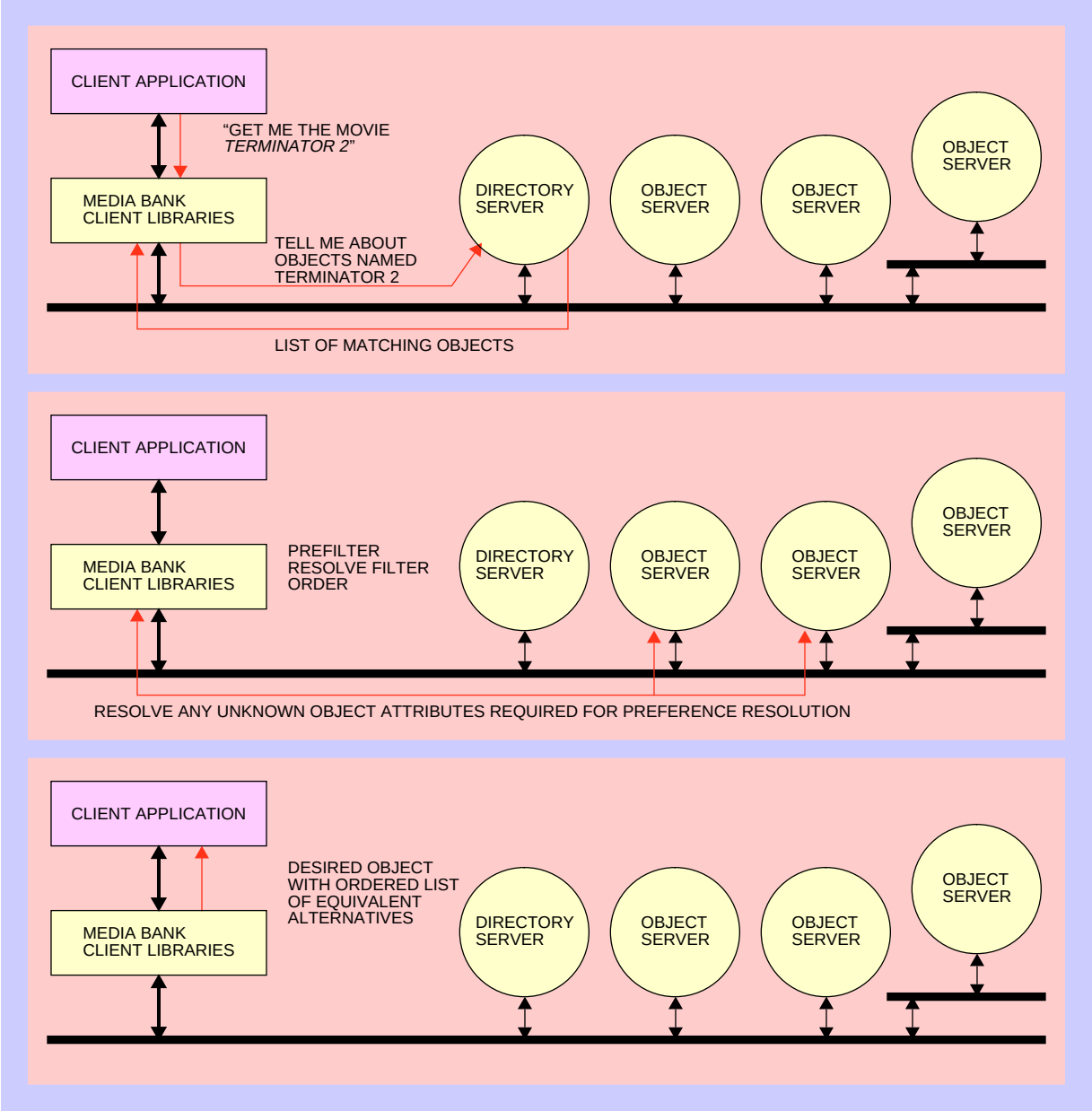
Finally, having resolved the matching objects, fetching the meta-data about them, and sorting them according to preferences, the Media Bank client libraries pass the best object (with the others in the backup-info field) to the client application (Figure 11, bottom).

The second example shows how a Media Bank client retrieves and synchronizes data from multiple isochronous sources once the objects have been located (see Figure 12). Perhaps the most trivial case that illustrates this capability is the playback of a movie. However, before delving into the exact mechanism for playback within the client it is worthwhile to examine the object format that makes synchronization possible.

Movies in the Media Bank are generally constructed from a number of objects that correspond to separately stored snippets of video and audio. Obviously the act of de-interleaving the audio and video requires that measures are taken to maintain the temporal dependencies between corresponding audio and video objects. Within the Media Bank this is achieved through the inclusion of offset, duration, number of samples or frames, and playback rate information in every audio or video object. Because some of this information is redundant, it is not required to include a value for each of these parameters with each object, as long as the missing parameters can be derived from the ones that are included. In addition, the Media Bank also allows the derivation of objects that are subranges of other larger objects. This design is particularly useful for synchronizing audio and video that have been digitized independently. An example of an audio object that is subranged from a large one is shown in Figure 13.

Having defined the storage format for individual objects, one needs to define a mechanism for synchronizing multiple streams. The Media Bank achieves the goal through the use of playlist objects that define the temporal relationships between multiple pieces of data at the object level. Two examples of a playlist are

Figure 11 Negotiating access to the movie *Terminator 2*



given in Figures 14 and 15. Figure 14 shows how three audio objects can be concatenated into a single larger one by issuing consecutive play commands, while Figure 15 shows how two separate temporal objects can be played simultaneously through the use of the play-multiple command.

As was briefly explained earlier in this section, the client coordinates the simultaneous playback of multiple objects via a local time base generator. The time base generator exploits the fact that objects "know" about their own temporal properties by sending updates that specify an offset in terms of time to a handler that is

Figure 12 Fetching the meta-data about an object from the directory server

```
Client request to directory server
      (object-fetch “Terminator 2”)

Director server reply to client
      #((type item multipart locator)
       (data .#(
         #((name . “Terminator 2”)
          (uniqueid . “<t2-movie-browser>”)
          (type item application sprog)
          (host . “alphaville”)
          (port . 41501))
         #((name . “Terminator 2”)
          (uniqueid . “<t2-movie-browser>”)
          (type item application sprog)
          (host . “bad-taste”)
          (port . 41501)))))
```

Figure 13 An object derived by subranging from a larger one

```
#((uniqueid . “<t2-audio-ch1-ch28-adjusted>”)
    (name . “t2-audio-ch1-ch28-adjusted”)
    (type . (item application subrange))
    (object . “<t2-audio-ch1-ch28-raw>”)
    (offset . 60.8)
    (duration . 81038000/23987))
```

Figure 14 Example of using the play command to concatenate multiple objects

```
#((uniqueid . “<t2-audio>”)
    (name . “t2-audio”)
    (type . (item application playlist))
    (version . 0)
    (script .#(
       (play #((object . “t2-audio-ch1-ch28-
adjusted”)))
       (play #((object . “t2-audio-ch29-ch57-
adjusted”)))
       (play #((object . “t2-audio-ch58-ch78-
adjusted”))))))
```

Figure 15 Example of using the play-multiple command to synchronize multiple objects

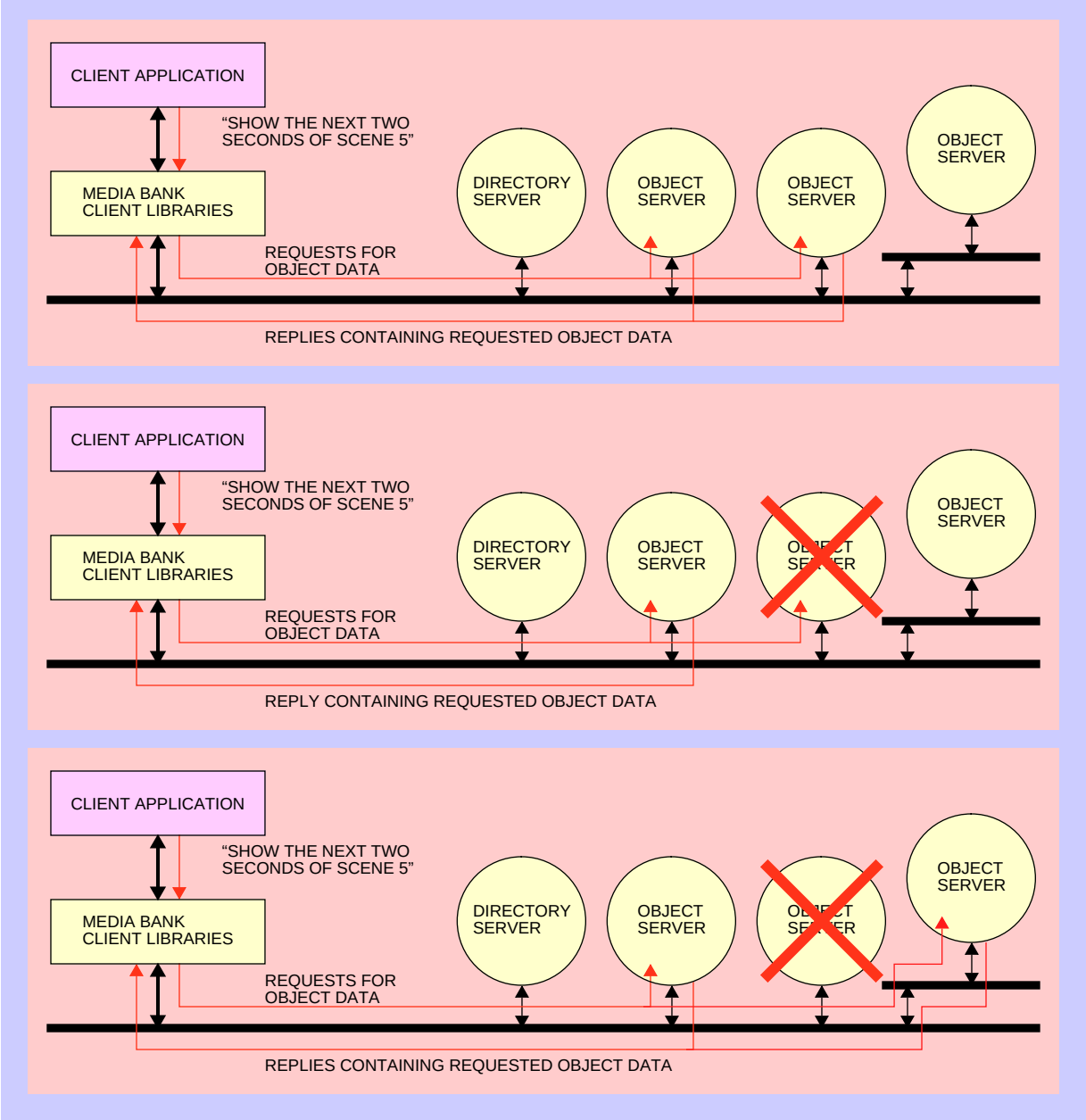
```
#((uniqueid . “<t2-movie>”)
    (name . “t2-movie”)
    (type . (item application playlist))
    (script .
      #((play-multiple
          #(
            #((play #((object . “t2-video”))))
            #((play #((object . “t2-audio”)))))) ))))
```

instanced for each object. The handler then performs the necessary actions to display the portion of the object at the specified time. Compound objects result in a hierarchy of handlers where the top or parent handler passes down the update event to the relevant children. A new update event is not generated until a handler has finished processing the previous update event. In this way, each object is informed in unison as to where it should be in the playback process. Obviously, the more objects one has, the greater load one places on the system, which can lead to the total handling time for all the handlers exceeding the time between update events. In these cases audio objects are given preference over video objects for resources, because the human ear is much more sensitive to temporal discontinuities than the human eye.[18]

An interesting side effect that arises out of using this particular mechanism for playback is the decoupling of the playback rate from the recording rate. Consider a video object recorded at 24 frames per second and sent updates as though it was recorded at 30 frames per second. In this case the handler would choose the frame nearest in time to the desired sample for playback, thereby performing automatic (albeit less than optimal) frame rate conversion.

The third example demonstrating the client’s capabilities is the ability to perform *autonomous error recovery* (see Figure 16). This is an especially useful feature, because it ensures that the quality of service given to a client is not bound to a single central server, as is the case with most solutions today. Thus the

Figure 16 Autonomous error recovery



Media Bank provides a much more robust means for delivery whereby the quality of service is much more closely tied to the client under the user's control, thereby minimizing the possibility of having a perfectly healthy client that is incapable of accessing data due to the failure of a single node in the network.

Having followed the first example that showed how a client negotiates access to the Media Bank, it should be no surprise that the means for error recovery lies in the *backup-info* field of the returned object. This field contains a list of alternate equivalent objects that may be used in lieu of the parent object, should access to

Figure 17 Screen shots of an object accessed via the Smart Client (left) and via Netscape/HTTP Gateway (right)

the parent fail for some reason. The second thing that makes rapid error recovery possible is the pull-based manner in which the client extracts object data from the object servers. As the client never has a continuous open stream to an object server, and pulls in only relativity small pieces of an object for any given request, detection of a defunct object server occurs almost instantaneously. Therefore, should an object server ever fail to respond to a request for data for an object within a specified time, the client may safely assume that object server is incapable of supplying further data. Consequently, the client also makes all future requests for data for that object to an alternate object server defined in the backup-info for that object. A diagram showing the normal operation (Figure 16, top), a dropped request due to a server crash (Figure 16, middle), and the subsequent recovery (Figure 16, bottom) for a client accessing data is shown.

The fourth and final example highlights the ability of the Media Bank to support a variety of clients with varying capabilities.

One could be forgiven for thinking that the screen shots shown in Figure 17 appear to be only standard pages of an ordinary World Wide Web browser. In reality, the browser shown on the left has been slightly modified to support the Media Bank libraries. The fact that the Mosaic browser was chosen for modification is not important; the choice made reflected the fact that it provided the most expedient means available

for generating a window for the "widgets" used by the Media Bank client libraries. The total modifications made to the Mosaic browser source code total roughly forty lines, but those forty lines provide a single point of access into the Media Bank client libraries that any smart object can use. The simplicity of this interface gives it its power, as clients can be written solely in Scheme and stored as objects that can be downloaded as needed. In effect, the Mosaic browser has been transformed from a simple display engine into a powerful virtual machine upon which any Media Bank object can be run.

Not all clients will be Media-Bank-capable, however (efforts to create a Media Bank plug-in for the Netscape browser client are ongoing at the time of writing); therefore a mechanism is needed to afford these clients access to the Media Bank. To that end a gateway application has been written that performs all of the negotiated accesses and object retrieval on the behalf of standard Hypertext Markup Language (HTML) clients. An example of the gateway operation is shown in the righthand screen shot of Figure 17. In this image, an ordinary browser is displaying the same "page" as the Media Bank enhanced Mosaic browser on the left. Unable to interpret the reference to the embedded Media Bank object in the page, this browser has been directed to an HTML gateway that acts as a proxy Media Bank client for the standard Netscape browser. The gateway itself is a client that does its best to retrieve the best possible approximation to the object requested. For example, if the object is a movie then the gateway will provide a key still from the movie, or it may instead send a small audio clip and a still. At present the backup data are supplied by the author of the object but future versions will be fully automatic and be sufficiently intelligent to tailor the approximated response according to the type of browser, and the bandwidth of the link.

Conclusions

In many respects, the Media Bank is a merger of Internet distribution techniques with television. Our goal is to find a common ground between the synchronous broadcasting of full packaged information associated with filmed entertainment and the demand-based dynamic assembly associated with computer networks. At the same time, we use the flexibility of programmable clients and servers to embed new techniques for indexing and assembling video as they arise. New applications will emerge when large populations have access to sufficiently powerful tools and databases. The Media Bank provides the foundation upon which to build these tools and databases, as it is optimized for programmability and user control.

As a vehicle for entertainment, the bank can be the basis for a television receiver. In our demonstration systems, synchronized video images appear simultaneously within a modified World Wide Web browser window and full screen on an external television. The browser can be the program guide, the device that builds a unique program for each viewer, or simply a controller for scanning through material. Future work will enrich this interface.

For research, we are using the Media Bank to explore ways to analyze, index, and deliver high-bandwidth isochronous information such as sounds and pictures. The Media Bank model of distributed storage and pull-based delivery is a relatively new one, and as such its capabilities have yet to be fully explored. At its most basic level, multimedia is nothing more than a unified delivery channel capable of containing diverse data types. To date, by dint of its size, rigidity, and complexity, video has been excluded from the same digital pipe that has carried stills, sound, and text. Our naming structure, storage method, and access paradigm allows a diverse set of representations to be jointly cataloged, potentially simplifying the problem.

We do not envision a system such as the Media Bank replacing broadcast for entertainment in the near term. Broadcasting is both bandwidth-efficient and resonates with a great many commercial and social interests. Until our networks increase their capacity, we cannot scale the Media Bank up to the level of the audience reached by popular television shows. However, the existing model of broadcast television has never been seriously challenged by any robust, computational alternative. Further, the expanded repertoire of available programs made possible by digitally compressed delivery and the implicit presence of computing at the receiver beg the question of how the two media can work together.

Cited references

1. T. J. Berners-Lee, R. Cailiau, and J. F. Groff, "The World Wide Web," *Computer Networks and ISDN Systems* **25**, 4–5, 454–

459 (November 1992).
2. L. A. Rowe et al., "A Distributed Hierarchical Video-on-Demand System," *Proceedings of the 1995 International Conference on Image Processing*, Washington, DC (October 1995).
3. L. A. Rowe, D. A. Berger, and J. E. Baldeschwieler, "The Berkeley Video-on-Demand System," *Multimedia Computing—Proceedings of the Sixth NEC Research Symposium*, T. Ishiguro, Editor, SIAM (1996).
4. D. Berger, *Video-on-Demand Metadata Query Interfaces*, master's degree thesis, University of California Berkeley, Computer Science Division (June 1995), ftp://roger-rabbit.cs.berkeley.edu/pub/thesis/dvberger-masters.ps.
5. ISO/IEC DIS 10918-1, ITU-T Rec.T.81 (JPEG), *Information Technology—Digital Compression and Coding of Continuous-Tone Still Images* (1992).
6. ISO/IEC 11172, *Information Technology—Coding of Moving Picture and Associated Audio for Digital Storage Media as up to about 1.5 Mbit/s* (1993).
7. ISO/IEC CD 13818-2, Recommendation H.262, *Generic Coding of Moving Pictures and Associated Audio*, Committee Draft, Seoul (November 1993).
8. R. Jain and A. Hampapur, "Metadata in Video Databases," special issue on *Metadata for Digital Media*, ACM SIGMOD (1994).
9. K. Dienes, *Information Architectures for Personalized Multimedia*, master's degree thesis, MIT Media Laboratory, Cambridge, MA (June 1995).
10. W. Clinger et al., *Revised Report on the Algorithmic Language Scheme*, MIT AI Lab Memo 848b, MIT, Cambridge, MA (November 1992).
11. N. Abramson, *Context-Sensitive Multimedia*, master's degree thesis, MIT Media Laboratory, Cambridge, MA (June 1993).
12. A. B. Lippman, R. G. Kermode, H. N. Holtzman, and K. Dienes, *Media Bank—Communications Protocol Definition*, Internal Document, MIT Media Laboratory, Entertainment and Information Systems Group, Cambridge, MA (October 1995).
13. K. Sollins, *Plan for Internet Directory Services*, rfc1107 (July 1, 1989).
14. T. Berners-Lee, *Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as Used in the World-Wide Web*, rfc1630 (June 9, 1994).
15. D. Clark and D. Tennenhouse, "Architectual Considerations for a New Generation of Protocols," *Proceedings of the ACM SIGCOMM '90* (September 1990), pp. 201–208.
16. L. Zhang, R. Braden, D. Estrin, S. Herzog, and S. Jamin, *Resource ReSerVation Protocol (RSVP)—Version 1 Functional Specification*, ftp://www.isi.edu/internet-drafts/draft-ietf-rsvp-spec-08.txt.
17. S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, Internet Draft, ftp://ds.internic.net/internet-drafts/draft-ietf-ipngwg-ipv6-spec-02.txt (June 1995).
18. W. R. Neuman, A. Crigler, S. M. Schneider, S. O'Donnell, and M. Reynolds, *The Television Sound Study*, MIT Media Laboratory, Cambridge, MA (1987).

Accepted for publication April 4, 1996.

Andrew Lippman *MIT Media Laboratory, 20 Ames Street, Cambridge, Massachusetts 02139-4307 (electronic mail: lip@media.mit.edu).* Dr. Lippman received both his B.S. and M.S. degrees in electrical engineering from the Massachusetts Institute of Technology. In 1995 he completed his Ph.D. studies at the École Polytechnique Fédérale de Lausanne, Switzerland. He is currently associate director of the MIT Media Laboratory and a lecturer at MIT. He holds seven patents in television and digital image processing. His current research interests are in the design of flexible, interactive digital television infrastructure.

Roger Kermode *MIT Media Laboratory, 20 Ames Street, Cambridge, Massachusetts 02139-4307 (electronic mail: woja@media.mit.edu).* Mr. Kermode is a doctoral student at the MIT Media Lab where he earned a master of science degree in 1994. He also holds undergraduate degrees in electrical engineering and computer science from the University of Melbourne. After completing his undergraduate studies in 1990 he worked at the Telstra (Australia) Research Laboratories as a research engineer investigating cell loss recover techniques for packetized digital video. Since commencing his studies in the United States he has interned at Silicon Graphics Incorporated. He is a Fulbright Scholar and has received fellowships from AT&T and Motorola, Inc. His research interests include video coding, object-oriented representations for multimedia, computational media, and interactive network infrastructure.

Reprint Order No. G321-5606.