Integrating Access and Collaboration for Multimedia Applications

Gunnar Teege and Michael Koch
Institut für Informatik, Technische Univ. München
80290 München, Germany
E-mail: [teege/kochm]@informatik.tu-muenchen.de

Abstract

Existing models and systems which integrate multimedia documents and collaborative distributed editing have a major drawback. They do not support media specific types of collaboration and distribution in a satisfactory way. We present a framework model for collaborative multimedia editing in which each media type can use its own type of collaboration and distribution. When used in system design, the model will strongly improve the flexibility and adaptability of systems supporting collaborative work on mixed-media documents.

Keywords: Multimedia, Collaboration, Distributed Editing, Hypermedia, Document Structure

1 Introduction

Within the last few years the traditional field of document editing has met the emerging fields of collaborative work, distributed systems, and multimedia documents. The different fields have grown together to become a single field dealing with collaborative distributed editing of multimedia documents. This field addresses the following aspects of editing:

- **collaboration**: user-awareness of other user’s activities; coordination of access and changes
- **distribution**: access of distributed user agents to shared data, e.g., using replication
- **multimedia**: the document contains portions of different media types
- **structure**: the document has an explicit structure, e.g., a hypermedia structure
- **persistence**: storing and retrieving the document between editing sessions

Existing systems and models do not yet integrate all of these aspects in a satisfactory way. There are several collaborative editors, such as Quilt [Fish et al., 1988] or GROVE [Ellis et al., 1990]. They support collaboration, distribution and persistence, sometimes including structural aspects as well. However, if different media types are handled at all, the handling is done in a simple uniform way. On the other hand, there are multimedia editors, such as MAEstro [Drapeau and Greenfield, 1991], and hypermedia database platforms, such as HyperBase [Wiil and Leggett, 1993]. They support multimedia, structure, and persistence. However, they deal only rudimentary with collaboration and distribution.

We present a model for an access layer which smoothly integrates all five aspects into a common framework. The major idea of our approach is a media specific view of collaboration and distribution. In multimedia editors it is typical to use different mechanisms for persistent storing and the user interface, depending on the media type. We propose an extension of this media dependency to the aspects of collaboration and of distribution.

2 Motivation

In an environment for collaborative work on shared information (“documents”) an important notion is that of awareness of other users’ activities. The environment has to support this awareness by broadcasting notifications about user activities. In our model we concentrate on awareness of activities which are related to the shared document’s content. This implies that each notification refers to one or more positions in the document or to parts
of the document. We do not consider additional mechanisms of collaborative editing support which are independent of the content, such as the use of a video or an audio connection for informal coordination.

It has been recognized by several authors, e.g., by Williams et al. [1992], that there is a need for an additional separate layer between the application and the layer for multimedia transport and storing in collaborative distributed systems. In our model we call this layer the access layer.

The application layer is situated on top of the access layer. The application’s task is to interact with one user. It presents to the user the document content and group activities performed on the content. The latter is done by displaying notifications from the access layer in an adequate way. On the other hand, the application accepts actions initiated by the user which may change the content and may generate notifications to other users.

The access layer coordinates the access of a group of application instances to a common content part. Hence, the access layer knows for every content part the set of application instances which are working on that part (the part’s work group). For this reason the access layer is the right place for managing the distribution of notifications related to content parts. Additionally, if the access layer is common to all types of applications, the distribution of notifications by the access layer results in inter-application notifications. If, e.g., a content part is currently both viewed in a browser and edited in an editor, all notifications generated by the editing process are distributed to the browser and may be displayed to the user.

Different schemes for accessing the data, such as replication, voting, floor passing or optimistic schemes are hidden from the application by the access layer. The application may only read and write data, announce its interest on a data portion, or announce the completion of a job concerning a data portion. The choice of access scheme and the handling of these tasks by the access layer depends mainly on the installed system’s environment (node availability, connection reliability).

Apart from the environment, an important criterion for the kind of access scheme is the medium type. Not all access schemes make sense for all media. As an example, a replication scheme may be useful for text portions, whereas it may cause problems when used for video data or large images. For this reason, in addition to the “vertical” separation between application and access layers, we propose a “horizontal” separation according to the medium type on both levels. This approach results in an access layer which is built from separate medium-specific “access specialists”. Every specialist may use its own kind of access scheme. However, in a specific environment only one scheme is supported for each medium type. By selecting a specific medium type, the application implicitly selects the access scheme which is used by the corresponding specialist. Since we only deal with collaboration aspects related to the document context, this approach does not imply any restriction on the kind of application.

As a consequence of the horizontal separation, notification distribution is handled in a medium-specific way as well. Mainly, this concerns the way in which the “work group” of a content portion is defined. For certain medium types it may be interesting to have several types of work groups, such as browsing and editing groups, with the work group type influencing the notification distribution.

As an example of the use of applications in our architecture consider the following scenario:

Peter is an engineer and currently working on a CAD drawing for a new automobile. He uses a special CAD editor that is customized to the needs of his task. The program receives the data of the vector graphic from the local instance of the enterprise access layer (from the vector graphic specialist). Anke is working on the same drawing and has also started an editor (not necessarily the same as Peter) on her workstation. The data is accessed by Anke’s local instance of the vector graphic specialist. The specialist has replicated the drawing because it is not very large in size and read access should always be possible. The engineers can follow the actions of each other because each completed modification of the drawing causes the access layer to notify all applications in its work group.

Now Anke wants to add a video annotation to a part of the drawing. The application transfers the video data to the local instance of the global access layer (to the video specialist). The difference to the drawing data is, that the video data shouldn’t be replicated because of its size. So the video specialist stores the data in a central video server.

3 Architecture

On the application layer each application supports one or more media types. In the simplest case there is a specific application for each medium type. Due to the common access layer, they can be used together to work on the different content portions of a common document. A more elaborate application may support integrated work on several different media types.

The access layer consists of media dependent access specialists which all offer a similar interface to the applications. To support full availability in the context of site and communication failures the specialists are fully
replicated. Hence, whenever an application process accesses a portion of a specific medium, a corresponding access specialist process runs on the same machine. The resulting overall structure is depicted in Figure 1.

The multimedia information is maintained by the specialists in the form of objects. Each object is of a specific medium type, hence it is managed by the corresponding specialist. An object is a component of the medium in a freely definable granularity. These objects may have an internal structure of sub-objects (e.g. a vector graphic has lines, circles, ...).

The specialists offer the following functions to the applications:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td></td>
</tr>
<tr>
<td>get_object:</td>
<td>objectID → access-object</td>
</tr>
<tr>
<td>put_object:</td>
<td>access-object → result</td>
</tr>
<tr>
<td>management</td>
<td></td>
</tr>
<tr>
<td>create_object:</td>
<td>→ access-object</td>
</tr>
<tr>
<td>delete_object:</td>
<td>objectID → result</td>
</tr>
<tr>
<td>synchronization</td>
<td></td>
</tr>
<tr>
<td>announce_intention:</td>
<td>objectID, suppl → result</td>
</tr>
<tr>
<td>announce_completion:</td>
<td>objectID, suppl → result</td>
</tr>
<tr>
<td>notification</td>
<td></td>
</tr>
<tr>
<td>send_asyncmsg:</td>
<td>notification → result</td>
</tr>
<tr>
<td>receive_asyncmsg:</td>
<td>→ notification</td>
</tr>
<tr>
<td>mask_asyncmsg:</td>
<td>notification-mask → result</td>
</tr>
</tbody>
</table>

The main task of the first two functions is the transfer of multimedia data between the application and the access layer. This is done with the help of “access objects”. An access object provides functions and data to access a portion of a specific media type. The implementation of an access object is media dependent (e.g. a text access object can hold a copy of the text and functions for reading and writing this text, a video access object can hold a reference to a video server and the protocols for reading and modifying the video sequence). Hence, depending on the media type, an invocation of get_object may either retrieve the actual data or simply provide a handle for data retrieval.

For synchronizing concurrent operations on objects, the access layer needs information about the applications’ intentions of changes. The function announce_intention is used by an application for announcing its intention to write an object or a part of the object. The application passes an object ID and supplementary medium specific information to the access layer. If a pessimistic concurrency handling protocol is used by the specialists announce_intention can be implemented by locking the object or parts of the object.
Notifications are usually generated and distributed by the access layer. A notification always consists of an object identifier, a type, such as "locked for update", and additional data depending on the medium type of the object and the notification’s type. An application may instruct the local access specialist to subscribe to specific notification types using the function mask_asyncmsg(). The function send_asyncmsg() can be used by the application to send its own notifications or to request information which will also be delivered in the form of notifications.

To offer this functionality the replicated specialists have to communicate. The main tasks of the communication protocol are

- **notification**: the creation and distribution of notifications
- **access**: reading objects
- **consistency**: writing objects and maintaining consistency

The realization of these tasks in the protocol depends upon the chosen concurrency control scheme. Possible schemes are optimistic ones like those described in [Aks cyn et al., 1988] and [Ellis and Gibbs, 1989] or pessimistic ones like the voting algorithms [Thomas, 1979; Jajodia and Mutchler, 1987] or floor control (as used in MultimETH [Lubich and Plattner, 1990]).

Peter marks several sub-objects in the drawing to be changed by him. His application uses the announce_intention call to inform the access layer about the situation. The vector graphic specialist locks the sub-objects. This fact is also distributed by a notification, so Anke’s application can mark the sub-objects as not changeable. If Anke nevertheless wants to change one of the sub-objects before Peter announces the completion, she gets a negative response by her application.

4 Application

Now we demonstrate the use of our model as a basis for a collaborative multiuser multimedia document editor. Note that the architecture presented in Section 3 only handles isolated multimedia objects. A document, however, is not a set of isolated objects. It is a collection of objects organized by a structure. This structure may be linear (ODA, SGML) or non-linear as in the case of hypermedia structures.

Therefore, the application has to manage a structure as well as the multimedia objects. The advantage of our open architecture is that we can handle the structure as another “medium type”.

We implement a **structure specialist**. It deals with objects which represent the structure of whole documents and contain references to the actual content portion objects. The kind of structure and the kind of pointers used for referencing content objects from within the structure is internal to the specialist and the corresponding application (a structure editor or a structure browser).

Anke accesses the drawing through the project document, which structures the whole project (specification, meeting protocols, drawings, calculations). She uses a structure browser to browse to the drawings section in the construction chapter. With the help of the structure browser she can see who else is working on the project and in which parts they are working (since notifications about operations on the structure are distributed as for all other medium types). By selecting the drawing in the structure browser, Anke is registered in the structure and the CAD editor is started with the object-ID of the drawing as parameter. The editor uses this ID to contact the access layer and get access to the drawing. Other users may now see in their structure browser that Anke is working on the drawing. Since Peter has started a CAD editor on the drawing, he may additionally see the single vector graphics specific actions performed by Anke on sub-objects in the drawing.

An editor system which is based on our model is the cooperative multimedia editor system IRIS [Borghoff and Teege, 1993]. The central part of the IRIS application is the structure editor and the corresponding structure access specialist. They use the ODA model for representing document structures.

Upon the start of IRIS the structure editor is loaded. It contacts the structure access specialist and retrieves the structure of the whole document. This information is displayed in the form of structure views (see Figure 2). Within these views the user may navigate within the structure, edit the structure and select a basic or composite logical object for editing.

If a part of the document is selected for editing in the structure view (by selecting a basic or composite logical object) a medium specific editor is started. This editor uses the structure access specialist for determining all content portions belonging to the selected structure node. It then instructs the corresponding media specialists to access the content portions.
Figure 2: Structure Views in IRIS (Shaded Parts are Locked).

Note that in our framework it is possible to use another application which structures the same contents into a hypermedia structure, instead of an ODA structure. The big advantage of our model is the fact that both applications can work collaboratively on the same shared content portions.

5 Related Work

There are several 'multimedia toolkits', such as Comet [Anderson and Chan, 1992], HeiMAT [Käppner et al., 1992], or the system of Gibbs [1992]. However, they are mainly concerned with the transport and storing of media data. Hence, they only provide a basis for implementing media specialists and access objects.

The need for a separate layer between multimedia transport and storing on the one side and applications on the other side has been recognized by several authors, e.g., by Williams et. al. [1992]. However, the model of Blair et al. [1994] only provides distributed multimedia objects and does not treat collaboration aspects. The model of Karmouch [1993] treats collaboration with the help of cooperative agents and delegates the access of multimedia information to media-specific shared services, thus separating collaboration and access.

Other approaches were made in the field of storage layers designed for hypermedia systems (so called hyperbases). One approach is that of Grønbæk et al. [1994] which adds notification generation to a media-independent common object-oriented database. Other systems are LINCKS [Lambrix et al., 1993], HyperBase [Wiil, 1993], CoMEdiA [Hornung and Santos, 1991b; Hornung and Santos, 1991a], Intermedia [Haan et al., 1992] and ABC [Shackelford et al., 1993]. The access layers of all these systems have in common that they handle objects of different media types with the same functions. As in the approach of Grønbæk et al., these layers are mostly realized on top of a centralized or distributed database system. The classic database functions are often extended with a simple notification mechanism which informs clients of changes in the database. None of the above systems uses different access or storage techniques for objects of different media.

Another hypermedia database system is the Hyperform system of Wiil and Leggett [1992]. It also fails to support different handling of different media types. Unlike the other systems it has the advantage of integrated extensibility. All components, such as concurrency control and notification control, are tailorable and may be extended to handle objects of different media types in a different way. The problem is that the Hyperform system is built as a centralized server system with caching on the client side. Thus there is not much flexibility in the managing of access to objects of different media types.
In this paper we presented a new model for a multimedia access layer that supports collaboration and can be used concurrently by different applications.

Our main goal was to allow the use of media dependent methods for the handling of collaborative access and updates without forcing the applications to deal with this access. We defined an intermediate access layer with an interface that hides all the protocol specific tasks and allows transparent access to the multimedia information and notifications. Notification generation and distribution is integrated with access functions.

The proposed model has several advantages:

- The application developers need not be familiar with the methods used in order to coordinate concurrent access. Hence, they can concentrate on the design of the user interface.

- Different applications can work concurrently with the same data.

- The access layer is modular and independent of the applications. Each part can be tailored to its specific environment and medium type. Hence, either optimistic or pessimistic protocols, replication or centralized storage can be used. Existing storage systems for multimedia data (such as video servers) can be integrated. New protocols can be implemented without changing the applications.

- The distribution of medium-specific notifications is separated from the applications and implemented by the access layer.

Currently we are investigating the applicability of our model by integrating different concurrency control methods into it.

References


