Multimedia Tools and Applications, 21, 243–260, 2003

SMILAuthor: An Authoring System for SMIL-Based Multimedia Presentations

CHUN-CHUAN YANG ccyang@csie.ncnu.edu.tw
YI-ZHENG YANG
Multimedia and Communications Laboratory, Department of Computer Science and Information Engineering, National Chi Nan University, Taiwan, Republic of China

Abstract. In this paper, an authoring tool named SMILAuthor for SMIL-based multimedia presentations is proposed. SMILAuthor adopts standard SMIL language as the format of the presentation to generate reusable and easily accessible presentations. Moreover, powerful editing functions such as cut, copy, and paste are supported by the system in a timeline-based manner. In order to support timeline-based editing functions, the playback duration of each object in the input SMIL script is first calculated by the parsing process of the system. The parsing process extracts and converts the temporal relationship of the input script to Real-Time Synchronization Model (RTSM), and the playback duration of each object in the script is then computed by traversing the RTSM. Editing results are converted to the SMIL format and saved in the output file. Language structure of SMIL is hidden by the system and the temporal information is visualized in the timeline manner to provide users an easy way to understand and control the timing of each object. Implementation of the system provides a friendly WYSIWYG environment and multiple views/windows are provided by the system to help authors compose multimedia presentations efficiently.

Keywords: authoring system, multimedia presentation, Synchronized Multimedia Integration Language (SMIL)

1. Introduction

Providing a powerful and attractive mechanism, multimedia is currently changing the style of digital life. The impact of multimedia is spreading out over every way in life, including entertainment, education, and business. For example, multimedia presentations provide a good tool for business demonstrations as well as distance learning. Multimedia presentation is concerning with the integration of multimedia objects, which may be located at remote data servers. In order to provide the useful tool for authors in designing multimedia presentations, an authoring system is necessary. A good authoring system for multimedia presentations should provide easy-to-use editing functions and meet the requirement of easy learning and generating reusable presentations [13].

Design issues of the authoring system for multimedia presentations may include the design of the format or internal structure for specifying the multimedia presentations [2, 4, 8, 23], the design of the user interface [12, 20], and the mechanisms for supporting the editing functions [3, 6, 10, 11, 14, 15], etc. The format for specifying/storing the presentations affects the popularity of the presentations. If the authoring system adopts a popular script language supported by browsers like *Internet Explorer* or *Netscape Navigator*, the readers do not have to install a special viewing program for the presentations. Unfortunately, different authoring systems [3, 6, 10, 11, 14, 15, 23] usually adopt different and proprietary formats. It would be better to adopt a popular script language as the format of multimedia presentations [2].

From the popularity point of view, *HTML* seems to be the best candidate. However, the lack of the ability in integrating synchronized multimedia for HTML makes it improper to be the language of multimedia presentations. *Synchronized Multimedia Integration Language* (*SMIL*) [1, 5, 9, 21, 22] was developed by the *WWW Consortium* (*W3C*) to allow for multimedia over WWW. It provides an easy way to compose multimedia presentations. With the efforts of W3C, SMIL is becoming the most popular language in authoring multimedia presentations, and in fact, it is currently supported by the newest versions of commercial browsers such as *Internet Explorer 6.0*. We make a brief introduction to SMIL in the following.

SMIL could be used to describe both the spatial relationship and temporal relationship of a multimedia presentation. The spatial relationship is concerning with the visual layout of media objects in the presentation, while the temporal relationship is concerning with the timing control of media objects. The elements for spatial relationship in SMIL include the ⟨*layout*⟩ element and the ⟨*region*⟩ element. The ⟨layout⟩ element determines how the elements in the document's body are positioned. The ⟨region⟩ element controls the position, the size, and scaling of media object elements.

The synchronization elements in SMIL for temporal relationship include the ⟨*seq*⟩ element, the ⟨*par*⟩ element, and the class of media object elements such as ⟨*img*⟩, ⟨*video*⟩, ⟨*audio*⟩ and ⟨*text*⟩, etc. The ⟨seq⟩ element defines a sequence of elements in which elements play one after the other. The ⟨par⟩ element defines a simple parallel time grouping in which multiple elements can play back at the same time. Both ⟨seq⟩ and ⟨par⟩ allow the nested structure that means the children element of them could be any of the synchronization elements. The media object elements allow the inclusion of media objects into an SMIL presentation. Media objects are included by reference (using a *URI*). Besides, some synchronization related attributes such as "*begin*", "*dur*", and "*end*" could be associated with these synchronization elements. A sample SMIL document is shown in figure 1.

```
<smil>
   <head>
      <layout>
         <root-layout width="500" height="250" />
         <region id="R-1" top="10" left="10"  height="200" width="200" />
         <region id="R-2" top="10" left="250" height="200" width="200" />
      </layout>
   </head>
   <body>
      <par>
         <audio id="lifecycle" src="lifecycle.ra" begin="1" dur="25" />
         <seq>
            <img region="R-1" src="egg.jpg"       dur="5" />
            <img region="R-2" src="larva.jpg"     dur="5" />
            <img region="R-1" src="pupa.jpg"      dur="5" />
            <img region="R-2" src="butterfly.jpg" dur="5" />
         </seq>
      </par>
   </body>
</smil>
```

Figure 1. A sample SMIL document.

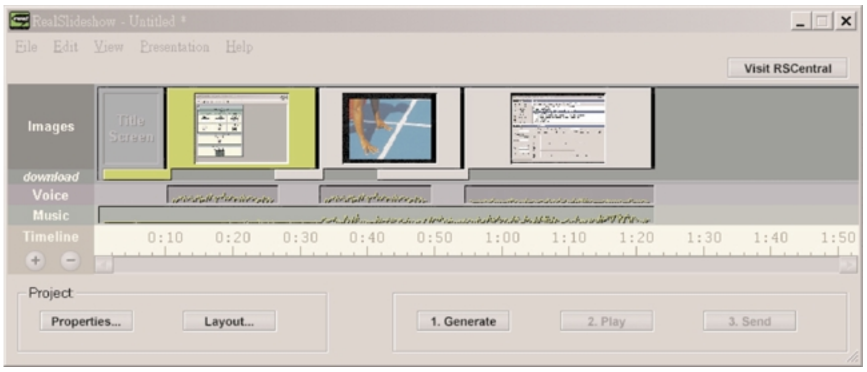


Figure 2. Snapshot: RealSlideshow.

The direct way to create a SMIL document is to use a text editor and start writing SMIL tags as most of the programmers do. For non-professionals, it is much better to have an authoring system that helps users compose any SMIL documents in a visualized (WYSIWYG) way. The SMIL-based authoring system should also provide easy-to-use editing functions. There are some existing authoring systems for SMIL-based presentations. Some of them only provide simple functionalities by which the users can only create simple SMIL documents. For example, *RealSlideShow* [18] (figure 2) can only generate presentations according to the static template provided by the tool.

Most of the existing authoring systems visualize SMIL tags and provide drag-and-drop functions to ease the composition of SMIL documents, but they adopt a recursive (nested) structure, in either icon-based or text-based way, which is similar to SMIL language structure to display the temporal information in the document. From the nested structure, the user cannot fast and easily understand the temporal relationship of all multimedia objects in the document. It is better to display the temporal information in a timeline manner so that users can easily understand the playback duration for each object. Moreover, none of them provides powerful editing functions like *cut*, *copy*, and *paste* as in the word processors. For instance, *GRiNS* [7] (icon-based, figure 3) and *SMIL Composer* [19] (text-based, figure 4) allow users to compose arbitrary multimedia documents, but both they use the nested structure for displaying the timing of objects and only provide simple editing functions such as insertion, deletion, and changing attributes of objects. No functions like *cut*, *copy*, and *paste* are supported in both systems.

Therefore, to address the popularity of the multimedia presentations and to provide powerful editing functions, we designed and implemented a SMIL-based authoring system, which is called *SMILAuthor*. As will be presented in the following sections, SMILAuthor could accept existed SMIL1.0 [21] scripts as the imported presentations being included in the new presentation. The SMILAuthor system parses the input SMIL script and represents the presentation in a timeline-based form as in *Director* [16]. Editing functions such as insert, clear, cut, copy, and paste, etc. could be used to compose the new presentation in the timeline-based manner. The result of the editing process is finally represented in the SMIL format and could be used as the imported presentation in another composing process.

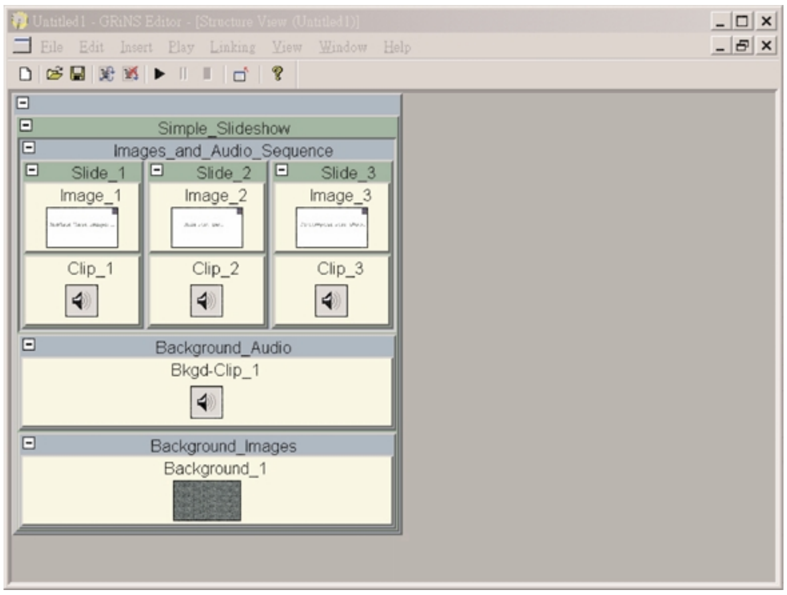
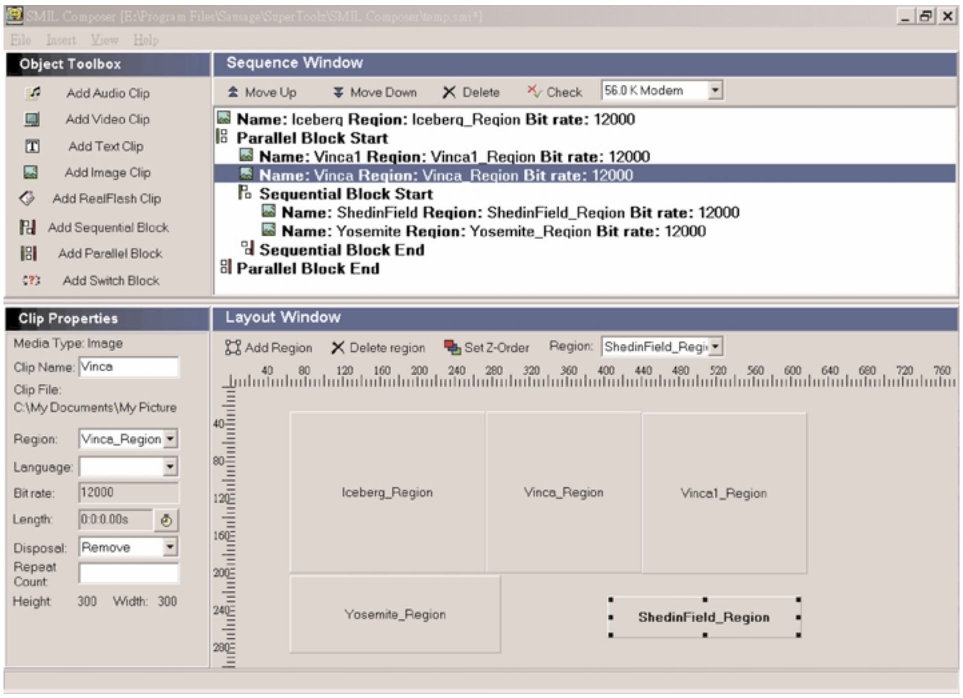


Figure 3. Snapshot: GRiNS.



Name: Iceberg **Region:** Iceberg_Region **Bit rate:** 12000
Parallel Block Start
  Name: Vinca1 **Region:** Vinca1_Region **Bit rate:** 12000
  Name: Vinca **Region:** Vinca_Region **Bit rate:** 12000
  Sequential Block Start
    Name: ShedinField **Region:** ShedinField_Region **Bit rate:** 12000
    Name: Yosemite **Region:** Yosemite_Region **Bit rate:** 12000
  Sequential Block End
Parallel Block End

Media Type: Image
Clip Name: Vinca
Clip File: C:\My Documents\My Picture
Region: Vinca_Region
Language:
Bit rate: 12000
Length: 0:0:0.00s
Disposal: Remove
Repeat Count:
Height 300 Width: 300

Figure 4. Snapshot: SMIL composer.

The remainder of the paper is organized as follows. First of all, the overview of SMIL-Author is presented in Section 2. Kernel mechanisms supporting the editing functions of SMILAuthor are then presented in Section 3. Implementation issues of the system are described in Section 4. Finally, Section 5 concludes this paper.

2. System overview

The goal of SMILAuthor is to provide useful functions for users to compose multimedia presentations. We could roughly divide the authoring process into three main steps, i.e., input, editing, and output, as illustrated in figure 5. Kernel mechanisms supported by SMI-LAuthor include (1) importing an existing presentation, (2) editing functions such as clear, cut, copy, and paste, and (3) saving the result.

Since the editing functions play a significant role in the authoring system, the mechanism supporting the functions affects the design of the other parts of the system. SMILAuthor focuses on the SMIL-based presentations; however, the editing functions are difficult to be realized in the manner of language, since these functions always involve the manipulation of time for each object. Therefore, it is better to perform the editing functions in the time domain, instead of the language domain. This is the reason why the input step in figure 5 requires computing the playback duration for each object for the input SMIL presentation.

In order to compute the playback duration for each object in the presentation, the authoring system has to parse the SMIL script. In our previous work, an algorithm to calculate the playback duration was proposed [24, 25]. In the proposed algorithm, the temporal relationship of the objects in the file is extracted and is represented by the *Real-time Synchronization Model* (*RTSM*) [26]. The playback duration for each object is then obtained by traversing the model. The parsing algorithm is briefly explained in Section 3.1.

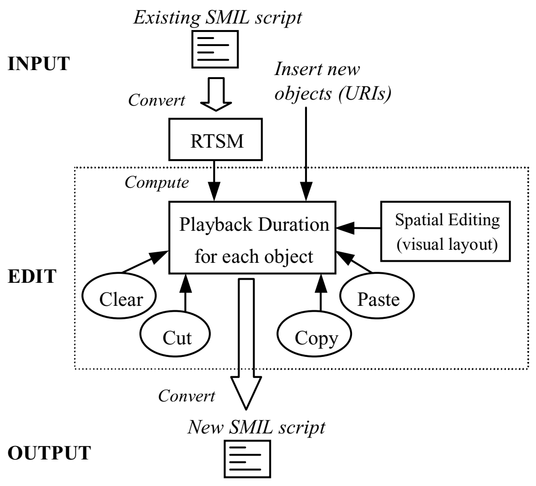


Figure 5. Overview of the authoring process.

The editing functions supported by the system include the functions of spatial editing and those of temporal editing. The spatial editing is concerning with the visual layout of the presentation, while the temporal editing is concerning with the timing property for each object. Visual objects (such as video, image, text, etc.) must be associated with the display region defined by the spatial editing functions. Hence, we could treat the display region of a visual object as one of its attributes while performing the temporal editing functions. As shown in figure 5, the editing functions, such as clear, cut, copy, and paste are performed on the playback duration of the object. In other word, these editing functions are timeline-based. The final result of editing is represented in the SMIL format and is saved to a file.

3. Kernel mechanisms of SMILAuthor

3.1. Parsing SMIL

As mentioned in Section 2, the temporal relationship among the objects of the input SMIL file is first represented by RTSM [26]. After the simplifying and reduction processes for the obtained RTSM, a reduced RTSM is obtained. The reduced RTSM is then traversed to calculate the playback duration for each object. We use an example to explain the algorithm briefly, and please refer paper [25] for the details.

The SMIL script in figure 6 requires the player to play the audio object *URI-1*, the video object *URI-2* and text object *URI-3* synchronously since these three objects are contained in a ⟨par⟩ element. The value of the "*endsync*" attribute in the ⟨par⟩ element requires ⟨par⟩ to end with the end of the audio object *URI-1*. In other words, once the audio object *URI-1* finishes playing, the video object *URI-2* and the text object *URI-3* must also stop playing at the same time. After the ⟨par⟩ element, the player has to display the image object *URI-4* for 5 seconds, and then play the audio object *URI-5* for 10 seconds. The obtained RTSM for the sample SMIL document after the converting process is shown in figure 7. The enforced place (denoted by double circle) is defined in RTSM as the dominated place for the firing of the following transition, and the virtual places is a place with zero duration.

Simplifying process removes the redundancy of the obtained RTSM, so the simplified RTSM is logically the same as the original RTSM. The simplified RTSM for the example

```
<seq>
        <par    endsync = id(URI-1)>
                <audio src=URI-1 />
                <video src=URI-2 />
                <text src=URI-3 />
        </par>
        <img src=URI-4 dur= "5s" />
        <audio src=URI-5, dur= "10s" />
</seq>
```

Figure 6. A SMIL code snippet.

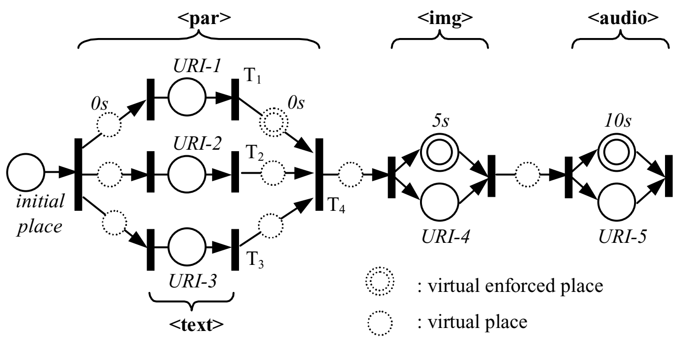


Figure 7. RTSM for the sample SMIL snippet.

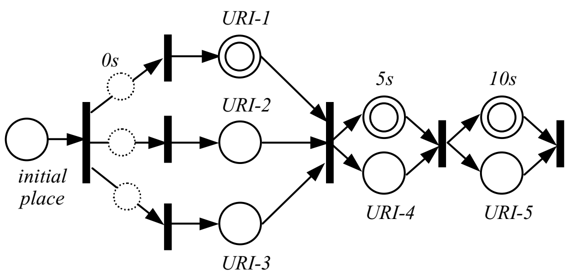


Figure 8. Simplified RTSM for the example.

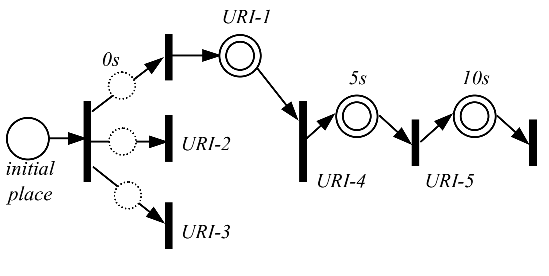


Figure 9. Reduced RTSM for the example.

is displayed in figure 8. Reducing process removes places that feed into the same transition with an enforced place, and the reduced RTSM for the sample is illustrated in figure 9.

The playback duration for each object, which is actually the duration from the firing time of the start transition to firing of the end transition of the object in the reduced RTSM, is computed by traversing the reduced RTSM started from the initial place. We illustrate the playback duration for each object of the example in figure 10. The playback duration for each object is then recorded for the editing functions presented in the next subsection.

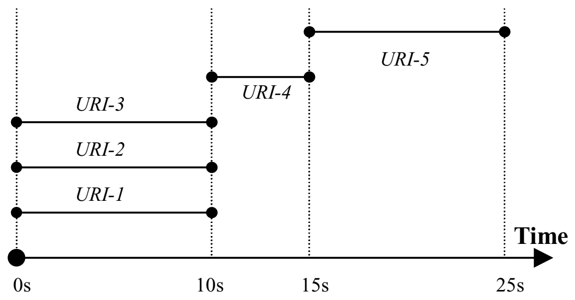


Figure 10. Object playback duration for the example.

Note that although only the handling of the temporal relationship is presented in this section, the parsing process also records the spatial information such as *region* and *layout* for each object in the input script. The spatial information is treated as one of the attributes for each object in the system to ease the implementation.

3.2. Editing functions

When the parsing process presented in last section is finished, all media objects with corresponding attributes such as *URI*, *display region* (created by spatial editing), *playback duration*, etc. in the input SMIL file are stored in the *object table*. The playback duration for an object is denoted by (T^s_{obj}, T^e_{obj}) in the paper. The playback duration means that the object should be played out in the time interval after the presentation is started, and the length of the duration is $T^e_{obj} - T^s_{obj}$. As mentioned in Section 2, the playback duration of objects provides the operating domain for the editing fuctions. In the following, we present the mechanism for each editing function supported in the system. Note that only the mechanism for realization of the function is presented, the design of the user interface is presented in Section 4.

3.2.1. Insert new objects. Inserting a new object to the presentation means to add a new object to the object table. The user has to provide the values of attributes (URI, display region, playback duration, etc.) for the inserted object. For visual objects, we define the *spatial-temporal conflict* as the case that visual objects with the same display position have overlap in their playback periods. Usually, the visual objects that will be played out concurrently should not occupy the same display position. But there are cases that the spatial-temporal conflict is allowable in a multimedia document for special purpose. Therefore, the authoring system makes it an option for the user to decide if the spatial-temporal conflict is allowed or not. If the spatial-temporal conflict is not allowable, the authoring system rejects the insertion request if there is a spatial-temporal conflict between the new visual object and those in the object table.

3.2.2. Modify object's attributes. The user could change the attributes of the objects in the object table. For example, the user may change the playback duration of an object by setting new values of (T^s_{obj}, T^e_{obj}) for the object, which reflects the action of moving, enlarging, or shortening the playback period of the object along the time line. Again, any modification of attributes for visual objects has to pass the test of spatial-temporal conflict.

3.2.3. Clear. The clear function is used to clear (part of) an object in the object table. In addition, the system also allows the user to clear a zone in the time line. A new object type named *zone* is defined to differentiate from the normal objects. When the user specifies a time zone to clear, all objects within the zone are cleared. The algorithm for the clear function is shown in figure 11 and an example is given in figure 12, in which case (a) shows

```
Function Clear (objID, $T^s_{clear}$, $T^e_{clear}$)

If  objID = zone then perform the following clear action on the
    objects whose ($T^s_{obj}$, $T^e_{obj}$) overlaps with ($T^s_{clear}$, $T^e_{clear}$).
    (two periods do not overlap when $T^s_{obj} > T^e_{clear}$ or $T^e_{obj} < T^s_{clear}$)
    if ($T^s_{obj} >= T^s_{clear}$ and $T^e_{obj} <= T^e_{clear}$) then delete the object
    else
        if $T^s_{obj} < T^s_{clear}$ and $T^e_{obj} > T^e_{clear}$ then split the object into
            two objects with periods ($T^s_{obj}$, $T^s_{clear}$) and ($T^e_{clear}$, $T^e_{obj}$)
        else if $T^s_{clear} < T^s_{obj} < T^e_{clear}$ then new $T^s_{obj}$ = $T^e_{clear}$
        else if $T^s_{clear} < T^e_{obj} < T^e_{clear}$ then new $T^e_{obj}$ = $T^s_{clear}$
Else
        Perform the above action on objID only
```

Figure 11. The algorithm for the clear function.

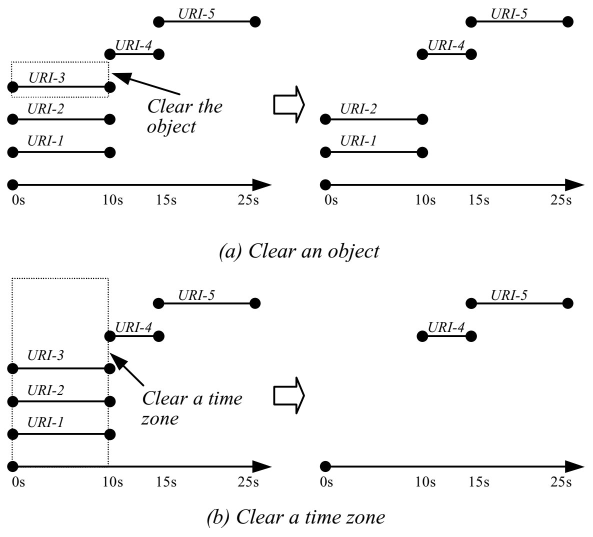


Figure 12. E.g., the ⟨Clear⟩ function.

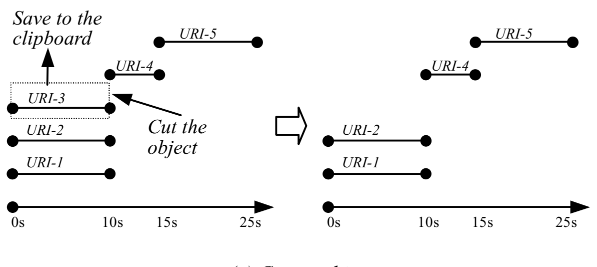
the case of clearing an object, and case (b) shows the case of clearing a time zone. Notice that the clearing action would sometimes result in the division of an object. For static objects like image and text, the division is merely reflected by setting new values for the playback periods. However, for continuous objects like video and audio, the division requires the authoring system to relocate the corresponding part of the medium data. Similar situation happens in other editing functions. For such cases, the system only needs to record the start point and end point in the object file and sets proper values of attributes ⟨*clip-begin*⟩ and ⟨*clip-end*⟩ when saving the result, which actually does not introduce much overhead to the authoring system.

3.2.4. Cut and copy. The cut function provides a way to cut (part of) an object or a time zone and to save the cutting part in the clipboard for future pasting. The cutting action for an object is similar to that of the clear function except that the cut part of the object is saved in the clipboard of the system. However, cutting a time zone not only moves all objects within the specified time zone to the clipboard but also advances the playback periods by the length of the time zone for the objects that are behind the time zone. The algorithm of the cut function is displayed in figure 13, and an example is given in figure 14.

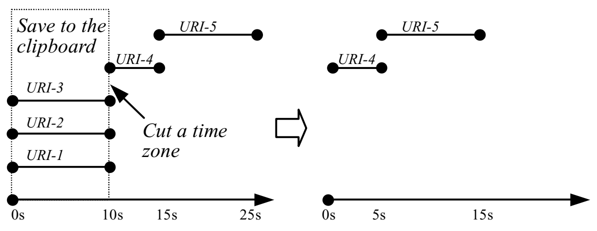
The copy function is similar to the cut function in the algorithm of saving objects in the clipboard, except that the copy function does not result in any change in the object table.

```
Function Cut (objID, $T^s_{cut}$, $T^e_{cut}$)

If objID = zone then
    For ∀ objects whose ($T^s_{obj}$, $T^e_{obj}$) overlaps with ($T^s_{cut}$, $T^e_{cut}$).
    if ($T^s_{obj} >= T^s_{cut}$ and $T^e_{cut} <= T^e_{obj}$) then save the whole
        object in the clipboard and delete the object from the
        object table.
    else
        if $T^s_{obj} < T^s_{cut}$ and $T^e_{obj} > T^e_{cut}$ then split the object into
            two objects with periods ($T^s_{obj}$, $T^s_{cut}$) and ($T^e_{cut}$, $T^e_{obj}$) in
            the object table, and save ($T^s_{cut}$, $T^e_{cut}$) of the object in
            the clipboard.
        else if $T^s_{cut} < T^s_{obj} < T^e_{cut}$ then
            save ($T^s_{obj}$, $T^e_{cut}$) of the object in the clipboard, and
            new $T^s_{obj}$ = $T^e_{cut}$ in the object table.
        else if $T^s_{cut} < T^e_{obj} < T^e_{cut}$ then
            save ($T^s_{cut}$, $T^e_{obj}$) of the object in the clipboard and
            new $T^e_{obj}$ = $T^s_{cut}$ in the object table.
Else
    Perform the above action on objID only

If objID = zone then advance the playback periods which are in
    back of the zone.
    For all objects in the object table with $T^s_{obj} > T^e_{cut}$,
    Set $T^s_{obj} = T^s_{obj} - (T^e_{cut} - T^s_{cut})$ and $T^e_{obj} = T^e_{obj} - (T^e_{cut} - T^s_{cut})$
```

Figure 13. The algorithm for the cut function.



(a) Cut an object



(b) Cut a time zone

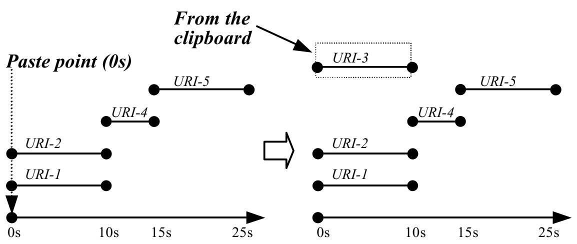
Figure 14. E.g., the ⟨Cut⟩ function.

3.2.5. Paste. The paste function provides the user to paste the objects in the clipboard at some time point of the presentation. The paste action depends on the types of object stored in the clipboard. If only one single object in the clipboard, the paste action is similar to that of inserting a new object, in which the user should specify the time point and the display position to insert. On the other hand, if a time zone object is stored in the clipboard, the paste function inserts the time zone (with all objects saved in the zone) at the specified time point. Figure 15 shows examples of the two cases for the paste function. The algorithm of paste function is shown in figure 16. Note that the algorithm does not allow the paste point T_{paste} within the playback period of some objects to reduce the complexity of the function.

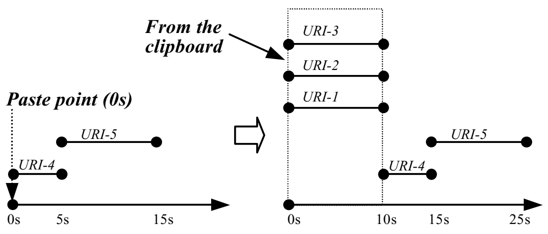
3.3. Saving the result

When user finishes the editing process and asks the authoring system to save the result, the system converts the objects in the object table to a SMIL file. The converting algorithm has to deal with both the spatial and temporal information of objects in the object table. The spatial information created by the spatial editing functions is concerning with the layout of display region for visual objects, and it is easy to convert the spatial information to the layout-related elements in SMIL, such as ⟨layout⟩ and ⟨region⟩. On the other hand, since the playback periods of objects spread over the time line, the temporal information is much more complicated than the spatial information. The converting algorithm thus focuses on the conversion of the temporal information to SMIL.

The temporal information consists of a set of media objects each with its playback period. The most straightforward way to convert the temporal information is to treat all

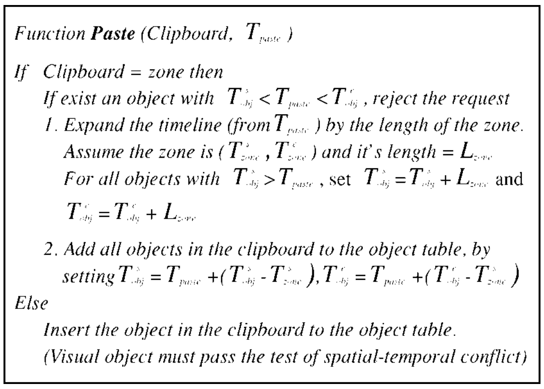


(a) Paste: an object in the clipboard



(b) Paste: a time zone in the clipboard

Figure 15. E.g., the ⟨Paste⟩ function.



```
Function Paste (Clipboard, $T_{paste}$)

If  Clipboard = zone then
    If exist an object with $T^s_{obj} < T_{paste} < T^e_{obj}$, reject the request
    1. Expand the timeline (from $T_{paste}$) by the length of the zone.
       Assume the zone is ($T^s_{zone}$, $T^e_{zone}$) and it's length = $L_{zone}$
       For all objects with $T^s_{obj} > T_{paste}$, set $T^s_{obj} = T^s_{obj} + L_{zone}$ and
       $T^e_{obj} = T^e_{obj} + L_{zone}$
    2. Add all objects in the clipboard to the object table, by
       setting $T^s_{obj} = T_{paste} + (T^s_{obj} - T^s_{zone})$, $T^e_{obj} = T_{paste} + (T^e_{obj} - T^s_{zone})$
Else
    Insert the object in the clipboard to the object table.
    (Visual object must pass the test of spatial-temporal conflict)
```

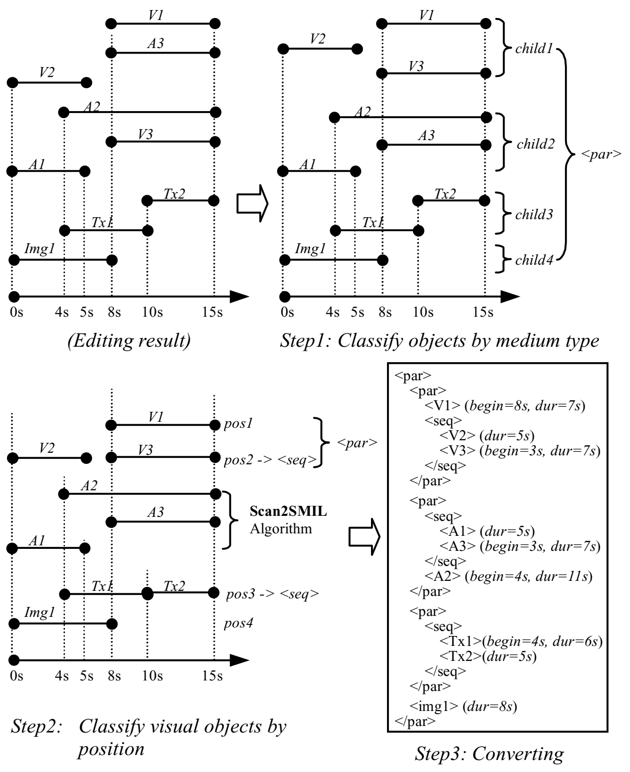
Figure 16. The algorithm for the paste function.

media objects as the children of a root ⟨par⟩ element. The "begin" attribute for each object is assigned to the playback time T^s_{obj} of the object and the "dur" attribute is assign to the length of the playback duration, i.e., $T^e_{obj} - T^s_{obj}$. The straightforward conversion is simple but introduces more overheads to the browser while presenting the SMIL file. The reason is that the straightforward conversion makes all media objects the children of a ⟨par⟩ element, and the ⟨par⟩ element, by the definition, requires the browser to deal with all its children concurrently. Hence, more processing overhead and more buffers are required for browsing the resulted SMIL file of the straightforward conversion.

Therefore, from the processing point of view of the browser, more sequential parts in the resulted SMIL file make the browsing more efficient. Unfortunately, it is not easy at all to find as fewer as possible sets of objects with disjoint playback periods from the temporal information after arbitrary editing process. Thus, we try to find some clues from the semantic level.

First of all, since the number of medium used in a presentation is limited, we could first classify the objects by their medium type. Objects of each medium type form a child element of the root ⟨par⟩ element in the SMIL file. Furthermore, if the user sets the option that the spatial-temporal conflict is not allowable, it implies that the visual objects with the same display position form a set of disjoint playback periods. Therefore, if we further classify the objects of a visual medium by the display position, we could determine all the sets of disjoint objects for that medium. More specifically, a set of disjoint objects forms a ⟨seq⟩ element for the same display position, and all sets of the same medium type further forms the children of a ⟨par⟩ element which is one of the children of the root ⟨par⟩. We illustrate the idea by the example in figure 17.

The classification by the display position does not work for non-visual objects like audio, so we developed the *Scan2SMIL* algorithm to convert non-visual objects. The *Scan2SMIL* does not consider any semantic relationship among the objects, but only provide a rule to determine a set of disjoint objects in each scan (iteration). The first step in *Scan2SMIL* is sorting the objects by T^s_{obj}. Next, the algorithm selects the object with smallest T^s_{obj} as the



```
<par>
  <par>
    <V1> (begin=8s, dur=7s)
    <seq>
      <V2> (dur=5s)
      <V3> (begin=3s, dur=7s)
    </seq>
  </par>
  <par>
    <seq>
      <A1> (dur=5s)
      <A3> (begin=3s, dur=7s)
    </seq>
    <A2> (begin=4s, dur=11s)
  </par>
  <par>
    <seq>
      <Tx1>(begin=4s, dur=6s)
      <Tx2>(dur=5s)
    </seq>
  </par>
  <img1> (dur=8s)
</par>
```

Figure 17. E.g., convert playback duration to SMIL.

```
Algorithm Scan2SMIL (for non-visual objects)
1. Create a <par> element for the medium type
       The <par> element is a child of the root <par> in the SMIL
       file
2. Sorting all objects by $T^s_{obj}$
3. While object exists in the object table do {
       Select the object with smallest $T^s_{obj}$
       Add the object to a new set S.
       Find the nearest object that is disjoint with all objects in S.
       Add the found object to S and continue to add disjoint
       objects to S until all objects are scanned.
       Remove the objects in S from the object table.
       The objects in S compose a <seq> element which is the
       child of the <par> element created in step 1.
   }
```

Figure 18. The *Scan2SMIL* algorithm.

first object of a new set of disjoint objects. The algorithm then searches the nearest disjoint object for the set of disjoint objects, i.e., the object with the smallest value of T^s_{obj} such that T^s_{obj} of the object $\geq T^e_{obj}$ of the first object in the set. The nearest object is added to the set of disjoint objects as the second object. The scanning process continues to search the next nearest disjoint object for the second object, the third object, etc. until all objects are scanned. The objects in the set obtained from the iteration obviously form a ⟨seq⟩ element and are removed from the object table. Similarly, following iterations create other sets of disjoint objects and form more ⟨seq⟩ elements. The algorithm stops when no objects in the object table. All ⟨seq⟩ elements from all iterations form the children of a ⟨par⟩ element of the non-visual medium.

The *Scan2SMIL* algorithm is displayed in figure 18, and the converting algorithm (*Convert2SMIL*) for the temporal information is displayed in figure 19. If the user requires the system to allow the the existence of the spatial-temporal conflict for the composing

```
Algorithm Convert2SMIL
1. Create the root <par> element
2. Classify objects by the medium type
     Objects of each type compose one child of the root <par>
3. For visual objects
     Classify objects by the display position
     Objects of each display position form a <seq> element
     All <seq> elements of different display positions form a <par>
     element.
     The <par> element is a child of the root <par>.
4. For non-visual objects
     Perform the Scan2SMIL algorithm
```

Figure 19. The *Convert2SMIL* algorithm.

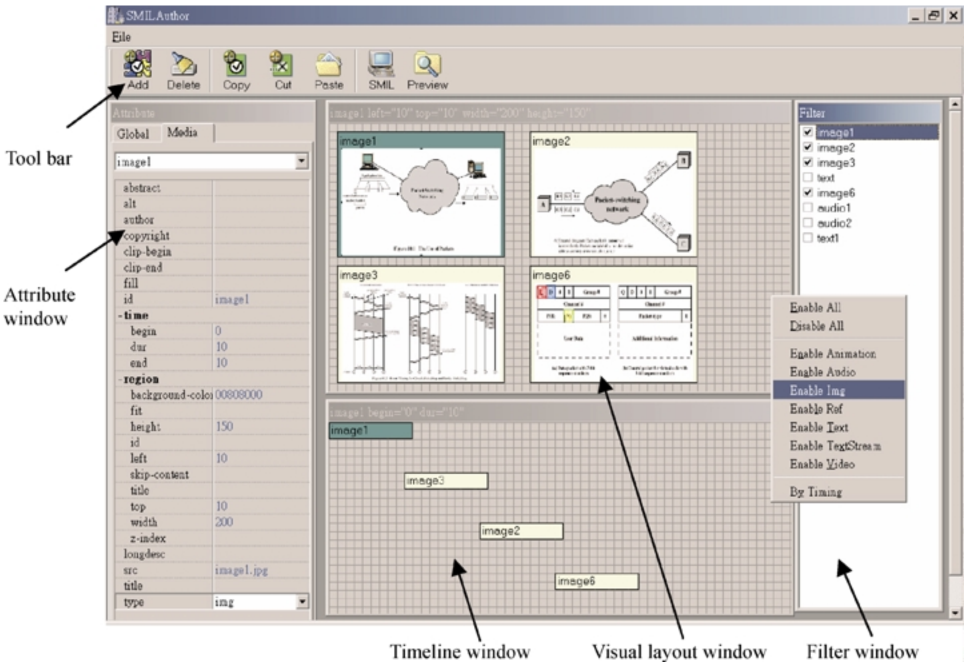
presentation, the *Convert2SMIL* algorithm should adopts *Scan2SMIL* for objects of each medium type, instead of considering the semantic level conversion.

3.4. Dealing with the hyperlinks

The link element ⟨*a*⟩ allows the description of navigational links that gives the user considerable flexibility to point and click on elements and go to the referenced URI during presentation. The functionality of the ⟨a⟩ element in SMIL 1.0 is restricted in that it only allows associating a link with a complete media object. Thus, the authoring system deals with the ⟨a⟩ element as the normal media object but records the link (URI) associating with the media object. Users can perform any of the editing functions on the media object associated with the ⟨a⟩ element. When the editing result is converting to a SMIL document, the media objects with hyperlinks is converted to the ⟨a⟩ element, instead of the media element.

4. System implementation

The implementation of SMILAuthor follows a similar concept as proposed in [12] to provide a "WYSIWYG" authoring environment. There are five major windows in the system to provide different views for the currently composing presentation. They are (1) *visual layout window*, (2) *timeline window*, (3) *filter window*, (4) *attribute window*, and (5) *preview window*. The display of the system on the monitor is shown in figure 20.



abstract	
alt	
author	
copyright	
clip-begin	
clip-end	
fill	
id	image1
-time	
begin	0
dur	10
end	10
-region	
background-color	00808000
fit	
height	150
id	
left	10
skip-content	
title	
top	10
width	200
z-index	
longdesc	
src	image1.jpg
title	
type	img

Figure 20. The display of SMILAuthor on the screen.

The visual layout window is used for the author to edit spatial relationship required by the presentation. The user could use the window to add, delete, resize, and move regions for the visual layout of the presentation. The timeline window displays the playback duration for objects. The users use the timeline windows to perform editing functions mentioned in Section 3. In order to reduce the large amount of information that has to display to the author, a filter window is used for the author to set displaying rule for both visual layout window and timeline window. The author could display selected information by specifying either medium type or time duration in the filter window. The attribute window is used to display and modify the attribute information for each object. Finally, the preview window is used to preview the presentation before saving the result to a file. Moreover, the preview window allows the author to preview only part of the presentation by specifying the preview duration, which is called *partial preview function* in the system. The preview window invokes an external player, RealPlayer [17] for the preview function.

Current version of the implementation only supports version 1.0 of SMIL [21]. We are going to support SMIL 2.0 [22] in the next version of the system.

5. Conclusion

A WYSIWYG authoring system for SMIL-based multimedia presentations named SMILAuthor is proposed and presented in this paper. Kernel mechanisms supported by the system include importing an existed SMIL script for the new presentation, useful editing functions such cut, copy, and paste for objects in the presentation, and saving the result in SMIL format. Since the editing functions supported in the system are timeline-based, the playback duration for each object in the input script has to be computed first in the parsing stage by traversing the RTSM representing the temporal relationship of the input script. Algorithms of editing functions and converting timeline-based information to SMIL format are included in the paper. Implementation issues of SMILAuthor are also presented in the paper. The features and advantages of SMILAuthor are listed as follows:

(1) SMILAuthor generates reusable and easily accessible multimedia presentations, since the standard SMIL language is adopted for I/O of the system, and users could use popular browsers like Microsoft Internet Explorer or Netscape Navigator for the playback of the presentations.
(2) Powerful editing functions, such as cut, copy, and paste, etc. are provided in the system to help authors compose new presentations in a more flexible way. These editing functions not only provide the manipulation for an individual (or part of) object but also the manipulation of a set of objects by specifying the time zone.
(3) Moreover, timeline-based editing functions with corresponding graphical user interface in the implementation provide users with an intuitive and easy-to-learn authoring environment. Language structure of SMIL is hidden by the system and the temporal information is visualized in the timeline manner to provide users an easy way to understand and control the timing of each object.
(4) Multiple views (windows) are provided in the implementation of the system to help users compose presentations friendly and efficiently. Partial preview function is provided by

the system so that the author could conveniently preview selected part of the composing presentation.

References

1. L. Bouthillier, “Synchronized multimedia on the Web—A new W3C format is all smiles,” Web Techniques Magazine, Vol. 3, No. 9, 1998.
2. S.-K. Chang, “Perspectives in multimedia software engineering,” in Proceedings. IEEE International Conference on Multimedia Computing and Systems, 1999, pp. 74–78.
3. D. Del Corso, G. Morrone, E. Ovcin, A. Truzzi, C. Scrizzi, and M. Gastaldi, “Interactive educational multimedia: A quick design and development tool,” in Proceedings., IEEE International Conference on Multimedia Computing and Systems, 1999, Vol. 2, pp. 841–845.
4. J. Emery and A. Karmouch, “A time-based multimedia document architecture,” in Proceedings of IEEE International Conference on Communications, 1995 (ICC ’95), Vol.1, pp. 555–559.
5. G. Flammia, “SMIL makes Web applications multimodal,” IEEE Intelligent Systems, Vol 13, No. 4, pp. 12–13, 1998.
6. J. Freire, R. Lozano, H. Martin, and F. Mocellin, “A STORM* environment for building multimedia presentations,” in Proceedings of 12th International Conference on Information Networking, 1998 (ICOIN-12), pp. 329–332.
7. GRiNS, http://www.oratrix.com/GRiNS/index.html.
8. L. Hardman, G. van Rossum, and Dick C.A. Bulterman, “Structured multimedia authoring,” in Proceedings of the First ACM International Conference on Multimedia, 1993, pp. 283–289.
9. P. Hoschka, “An introduction to the synchronized multimedia integration language,” IEEE Multimedia, pp. 84–88,1998.
10. S. Hudson and C.-N. His, “The walk-through approach to authoring multimedia documents,” in Proceedings, The Second ACM International Conference on Multimedia, 1994, pp. 173–180.
11. M. Jourdan, N. Layaïda, C. Roisin, L.S. Ismaïl, and L. Tardif, “Madeus, an authoring environment for interactive multimedia documents,” in Proceedings of 6th ACM International Conference on Multimedia, 1998, pp. 267–272.
12. M. Jourdan, C. Roisin, and L. Tardif, “Multiviews interfaces for multimedia authoring environments,” in Proceedings of Multimedia Modeling, 1998 (MMM’98), pp. 72–79.
13. J. Kelner, D. Hadj Sadok, F. Marques, and A. Neves, “The role of parametrization in the multimedia authoring process,” in Proceedings, IEEE Conference on Protocols for Multimedia Systems—Multimedia Networking, 1997, pp. 142–149.
14. J. Kim and Sun Shin An, “Design and implementation of IMAT (Internet Multimedia Authoring Tool) using a unified spatio-temporal relationship model,” in Proceedings of 3rd IEEE Workshop on Multimedia Signal Processing, 1999, pp. 617–622.
15. D. Lowe and M. Sifer, “Refining the MATILDA multimedia authoring framework with a visual formalism,” in Proceedings of 3rd IEEE International Conference on Multimedia Computing and Systems, 1996, pp. 291–294.
16. Macromedia, Director 8, http://www.macromedia.com/software/director/.
17. RealNetworks, “RealPlayer Plus,” http://www.real.com/, 2000.
18. RealNetworks, “RealSlideshow,” http://proforma.real.com/rn/tools/slideshow/index.html.
19. Sausage, SMIL Composer, http://www.sausagetools.com/supertoolz/toolz/stsmil.html.
20. J. Song, M.Y. Kim, G. Ramalingam, R. Miller, and B.K. Yi, “Interactive authoring of multimedia documents,” in Proceedings of IEEE Symposium on Visual Languages, 1996, pp. 276–283.
21. Synchronized Multimedia Integration Language (SMIL) 1.0 Specification, W3C Recommendation, June 1998, http://www.w3c.org/TR/REC-smil.
22. Synchronized Multimedia Integration Language (SMIL) Boston Specification, W3C Working Draft 20-August-1999, http://www.w3.org/TR/smil-boston.
23. M. Vazirgiannis, I. Kostalas, and T. Sellis, “Specifying and authoring multimedia scenarios,” IEEE Multimedia, Vol. 6, No. 3, pp. 24–37, 1999.

24. C.C. Yang, "User-interaction supported data-retrieving engine for distributed multimedia presentations," in Proceedings, IEEE International Conference on Communications, 2001 (ICC2001), pp. 3244–3250.
25. C.C. Yang, "On the design of the data-retrieving engine for distributed multimedia presentations," in Proceedings, IEEE International Conference on Communications, 2001 (ICC2001), pp. 3237–3243.
26. C.C. Yang and J.H. Huang, "A multimedia synchronization model and its implementation in transport protocols," IEEE Journal of Selected Area in Communications, Vol. 14, No. 1, pp. 212–225, 1996.

Chun-Chuan Yang received his B.S. degree in computer and information science from National Chiao-Tung University, Taiwan, in 1990 and Ph.D. degree in computer science from National Taiwan University in 1996. Since 1998, he has been an assistant professor in the department of computer science and information engineering, National Chi-Nan University, Taiwan. His research area of interests includes multimedia network protocols, multimedia synchronization control, and multimedia applications.

Yi-Zheng Yang received his B.S. degree in industry education from National Changhua University of Education, Taiwan, in 1999 and the M.S. degree in computer science from National Chi Nan University, Puli, Taiwan, in 2001. His current research topic includes multimedia authoring systems and Internet applications.