ELECTRONIC PUBLISHING, VOL. 8(2 & 3), 95–110 (JUNE & SEPTEMBER 1995)

Modelling multimedia documents

P. R. KING

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba
Canada R3T 2N2

e-mail: prking@cs.umanitoba.ca

SUMMARY

This paper discusses the need for models for multimedia documents and describes a particular formal model. The model makes use of an executable Interval Temporal Logic as its basis. The paper describes how temporal constraints among media items may be specified for subsequent manipulation and for use in prototyping. In particular, it uses the powerful notion of interval projection, both as a device for specifying variable display rates for media items and also for providing a scripting mechanism. The paper also outlines how this model may be used as the basis of an authoring tool for such documents.

KEY WORDS Multimedia Temporal logic Modelling Projection Authoring

1 INTRODUCTION

Our work is concerned with modelling electronic documents. Our ultimate goal is to make use of such formal models in designing authoring systems. Our specific interest is in multimedia documents, where by the term *multimedia*, we mean a document containing continuous or time dependent components, referred to as *media items* [1]. We are particularly interested in documents with rich sets of various types of temporal constraints. By providing a formal model for specifying such constraints, it will become possible to provide tools for authoring and for prototyping such documents.

Our model uses the Interval Temporal Logic (ITL) of Moszkowski [2] as a notation for expressing sets of temporal constraints between objects in multimedia documents. In earlier papers [3–5], we described some aspects of this use of ITL, and showed that ITL possesses the appropriate descriptive power. In the present paper we wish to present a number of more powerful features which are needed in multimedia documents, and demonstrate how our model can accommodate them. Two specific areas of concern will be *projection* and *scripting*. We will also discuss some of the problems associated with prototyping such documents, specifically the important notion of determinacy. For completeness, and so that this paper may be considered in isolation, we will also present summaries of the earlier results.

The remainder of this paper is organised as follows. In Section 2 we review the need for document models, and discuss the particular needs of such models in the case of multimedia documents. In Section 3 we review a first set of functional requirements for multimedia documents, review that part of Moszkowski's ITL needed to express this first level of functionality, and review how it is actually expressed. We also introduce a non-trivial example, which will be used elsewhere in the paper. In Section 4 we discuss the notion

CCC 0894–3982/95/020095–16

Received 1 April 1996
Revised 2 July 1996

of *projection* and describe various forms that this may take in multimedia documents. In particular, we describe the notion of *scripting* in multimedia documents, and we present a complete solution to modelling the example introduced earlier. In Section 5 we complete our discussion of the model. Section 6 indicates some of the problems associated with using this model in an authoring system, and our approach to their resolution. Section 7 presents some future directions and concludes.

2 ELECTRONIC DOCUMENTS AND FORMALISMS

2.1 Introduction

The use of formal models in document processing systems, that is the notion of a document as a structured object conforming to a chosen formal description, whereby the structure can be represented and manipulated, is a well known one. It is, however, somewhat inaccurate to talk about document models and formalisms (in the plural) for documents, since a single formalism has predominated, the *attribute grammar*. Attribute grammars have been used as the basis both of *document editors*, such as Grif [6], and of *standards* for electronic documents, such as ODA [7], SGML [8] and HyTime [1]. The essence of the approach in all these cases is to use the context-free portion of the grammar to define the component structure of a document, and to use the attributes to specify those quantities needed to perform the correct formatting. As just one instance, an SGML DTD (Document Type Definition) is, essentially, an attribute grammar, where the context-free portion serves to define the components of the class of document under discussion, and the attributes may be used for layout directives.

We should temper somewhat the remarks in the previous paragraph. While much of the work on document models depends on the attribute grammar, or some close variant of it, some authors have used other classes of grammars, see for example [9], or more powerful formalisms. Some researchers have, made use of programming languages as part of a model; Prolog has, for example, been used to perform attribute calculation. There is also work on object-based approaches to the provision of toolkits for both structured documents and multimedia documents [10].

At this stage it is important to realise that attributes of themselves possess no underlying formalism for their specification or subsequent computation. Essentially, attribute relationships in grammars are defined by arbitrary algebraic equations. Consequently, and in particular, there is *a priori* no formal method of verifying the consistency of a set of attributes. By way of contrast, a central property of the model chosen in the present work is that it must encompass a formalism which is capable of various sorts of manipulation, including consistency verification. This is important when considering our model as the basis of an authoring tool for the temporal aspects of such structured documents.

2.2 Document models and multimedia documents

The HyTime Standard [1] contains the following definitions:

> A document is a collection of information that is identified as a unit and that is intended for human perception.
> Multimedia is ... an attribute of a document indicating that the document might contain continuous or time dependent components.

To illustrate these definitions, and to illustrate the various aspects of our model in what follows, we now introduce a sample document specification containing a reasonably complex set of items and temporal constraints. We refer to this illustration as the *Beethoven Problem*:

> Play an audio containing the four movements of Beethoven's Fifth Symphony, opus 67, in c minor;
> Before each movement, display a video containing text with appropriate program notes;
> During the music, whenever there is a pianissimo passage, display a still video of a sleeping baby;
> During the second movement, display a video of a woodpecker at every staccato passage;
> At the end of the symphony, display a picture of Ludwig van Beethoven for 15 seconds, then display information about how this presentation can be purchased;
> During the video-text displays, the reader of the presentation is entitled to rewind the video at any stage, or to fast-forward to the start of the subsequent movement;
> The video-text displays should not take longer that 10 seconds each.

We now describe four general requirements of a model for multimedia documents, illustrating these requirements with this example.

2.2.1 Modularity and top-down decomposition

It should be possible to treat media items, and the specification of their temporal constraints, in a hierarchical manner. In this example, we should be able to treat the various components of the specification separately, composing them into the final result. Therefore, the first line of the example just given

> Play an audio containing the four movements of Beethoven's Fifth Symphony

may be regarded as the top level of the specification, the symphony, with the remaining items, for instance the four movements, below it in the hierarchy. This idea goes further, since it will usually be the case that the author wishes to add media items (MIs) and constraints to an existing set. For example, the item involving the woodpecker may have been added after the others. Sets of temporal constraints among discrete sets of media items should be *independently specifiable* and the formalism should provide a way of permitting MIs to be specified in a top-down fashion.

2.2.2 Compositionality, and bottom-up design

It should also be possible to proceed in the reverse manner from that just suggested. Given independent sets of constrained media items, it should be possible to assign names say, to those sets, and thereafter use these names in specifying further levels of constraints. This 'bottom-up' approach is, in a sense, the mirror image to the top-down approach just described. Both top-down and bottom-up are familiar design approaches in the area of

programming, and in other areas of formal specification. In our example, we should be able to compose the separate components of the specification into one specification in such a way that this composed specification preserves the consistency and correctness of the originals.

2.2.3 Well foundedness

The formalism should make use of a well defined, consistent calculus to express and manipulate formulae specifying temporal constraints. In particular, it should be possible to determine whether particular formulae expressing constraints are consistent. Well-foundedness is an important characteristic of any formalism which is to serve as a basis for formal specifications of whatever sort, but since one of the longer term goals of these particular temporal specifications is that they be transformable between documents, the notion of well-foundedness is a particular necessity in this work.

2.2.4 Executability

Since the formalism is to be the basis of an authoring tool, specifications written in it should be *executable*, in the sense that it should be possible to display the behaviour over time of free variables in formulae expressing temporal constraints. In this manner, a model could be developed for displaying the dynamic effect of sets of temporal specifications. The author could then make use of such a display to decide whether particular specifications are indeed what was intended. In the long term, it will be possible to use such a display as the basis of an interactive system, permitting the author to modify certain constraints and view the dynamic effect of such modifications.

3 REQUIRED FUNCTIONALITY IN THE MODEL

3.1 Functional requirements

As indicated earlier, we are restricting ourselves to *temporal* properties of media items. We are less concerned with their structural (context-free) composition, and not at all concerned with their spatial attributes. If we consider the temporal dimension of a multimedia document, the task for an authoring system is to place media items correctly on that axis. In [11], Erfle presents a comprehensive set of 18 issues, gleaned in part from existing published authoring systems and standards, which are needed in such constraints, and shows how each of these can be achieved in the HyTime standard. Our approach is somewhat similar. In [3], [4] and [5], we present in full a comparable set of functional requirements which encompass all of Erfle's issues, but which use a different taxonomy, and define seven constraint classes rather than eighteen. Here we summarise our taxonomy, with illustrations from the above example.

The first four of our set of seven issues involve constraints on absolute or relative start and finish times and durations of media items. These four cases are subdivided into the case of one, of two, and of more than two MIs. Consider the following component of the Beethoven problem:

> Before each movement, display text containing appropriate program notes.

This is a relatively simple example of *time* constraints involving more than two items; essentially this specifies sequential presentation of eight items, the four movements preceded by the four sets of program notes. Now consider the component:

> At the end of the symphony, display a picture of Ludwig van Beethoven for 15 seconds, then display information about how this presentation can be purchased.

This is a composition of several media items, this time involving *duration* as well as time.

Of the remaining three issues, one is concerned with what Erfle terms *adjustment* [11], which in programming languages is termed *exception handling*. Consider, for example, the following specification:

> The video-text displays should not take longer that 10 seconds each.

The issue is what happens if one of the displays does indeed require more than 10 seconds.

A further issue is concerned with the notion of *projection*, which will is a central one in this paper. Consider the two following instances:

> During the music, whenever there is a pianissimo passage, display a still video of a sleeping baby.
>
> During the second movement, display a video of a woodpecker at every staccato passage;

We will see later that these two are fairly complex examples of projection, and illustrate what we term *scripting*.

Finally, a special case of adjustment is *reader intervention*. Consider:

> During the video-text displays, the reader of the presentation is entitled to rewind video at any stage, or to fast-forward to the start of the subsequent movement.

This component involves a number of constraint classes, adjustment and reader interaction, projection, and a more complex set of compositions.

3.2 The model – Interval Temporal Logic

Our multimedia document model is based upon the Interval Temporal Logic of Moszkowski [2]. Interval temporal logic (ITL) is a first order predicate logic, to which is added the notion of an *interval* as finite, non-empty, unbounded sequence of *states*. The values of the free variables in temporal logic formulae may change from state to state in an interval. A temporal logic formula, therefore, is a predicate over an interval, which depends for its logical value upon the behaviour of free variables over the sequence of states in that interval. Temporal logic formulae may, alternatively, be regarded as imperative statements, which assign values to free variables over the sequence of states in an interval. This is indeed what Moszkowski does in defining his temporal language Tempura, to which we will make reference later. In this work we assume that the interval is an interval of *time*, corresponding to the temporal axis of a document, and that each component state represents *one and the same* unit of time – a clock tick of whatever granularity is required by the application at hand.

Moszkowski's ITL indeed satisfies the four general requirements of the model which were discussed in section 2. In the case of modularity and top-down decomposition the matter is quite simple. Briefly, the author may specify a constraint formula involving, say, variables *A*,*B*. Further, the desired set of constraints that exist among the variables $a_1, \ldots, a_i$ within *A* may be separately written, tested, and executed as a second set of constraints. It should also be noted that Moszkowski's ITL supports the definition of both *lambda expressions* and *functions*. Bottom up composition is also possible in ITL [12]. Composing two formulae with disjoint sets of free variables is achieved by a simple conjunction. Composing two formulae with overlapping sets of free variables is harder, and the details are beyond the scope of this paper. Some results appear in [4] and [5]. With regard to the requirement that the formalism be well-founded, it is sufficient to state that temporal logic is well-founded (sound), and permits manipulation, proofs of consistency, and so forth. Such manipulations and proofs can be regarded as analogous to static checks in a temporal logic program. The question of executability is more subtle, and will be deferred until section 6.

In [3] we have described that subset of Moszkowski's ITL which is of interest to us. For completeness and for ease of reference, we include some details in Appendix 1, but the reader needing a fuller explanation is referred to [3] or even [2]. In [3,4] we also indicate how some of the constraint classes outlined in the previous section can be specified in Moszkowski's ITL. In particular, the formalism provides a very natural mode of expression for constraints on absolute or relative times and durations of one, or more media items. Again for completeness these results are summarized in Appendix 2. In considering these expressions, it should be remembered that they provide what is in essence an existence proof, that ITL is sufficient for our purposes, not a suggestion that an author should use 'raw' ITL to author such documents. We emphasise that it is our intention to construct authoring tools on top of the ITL formalism.

In the next section of this paper, we turn to the use of projection for the remaining constraint classes.

4 PROJECTION AND SCRIPTING

The term *projection* refers to a change of scale. More precisely, in ITL the term denotes the definition of a second interval in terms of a given first interval, and the subsequent use of that second interval in some ITL formula or operation. The second interval may be of the same length or of a different length than the first. It may be of finer or coarser granularity than the first. The general form of an ITL formula involving the projection operator is:

$$w_1 \ \mathbf{proj} \ w_2$$

In this formula, the sub-formula w_1 is a *projector* which defines a second interval from the initial interval over which the sub-formula w_2 is specified. The composite formula w_1 **proj** w_2 is thereby specified over this new interval.

A definition of this projection operator together with some examples of its use, appears in [3]. In [2] Moszkowski, gives a formal semantics for the operator. For our purposes in this paper we can afford to be far less formal. We will first consider some uses of projection in multimedia constraints where a projector is used to change the rate of display of a MI. This we refer to as *multiplication*. We will then give an intuitive but useful meaning to the operator, and will then discuss how it may be used in what we term scripting. Finally, we will consider its use in providing for adjustment, exception handling.

4.1 Multiplication

For a number of its uses in multimedia constraints, a projection will be a *multiplication*, that is a projection which serves to either *magnify* an interval, by replacing each state in the interval by a given number of states, or to *contract* the interval, by replacing a given number of states by a single state. Such projections make use of the **len** operation to define the multiplication factor. A simple example in ITL may help. Consider first the ITL formula w_2 defined as

$$w_2 \equiv \mathbf{len}(4) \wedge (i = 0) \wedge (i \,\mathbf{gets}\, i + 1)$$

This formula specifies an interval of length 4, that is one with five states, in which the variable *i* takes on the following successive values: 0,1,2,3,4. Now consider the multiplication:

$$\mathbf{len}(2)\ \mathbf{proj}\ w_2 \equiv \mathbf{len}(2)\ \mathbf{proj}\ \mathbf{len}(4) \wedge (i = 0) \wedge (i \,\mathbf{gets}\, i + 1)$$

Here the original formula is multiplied by the projector $\mathbf{len}(2)$. By definition, this projection has the effect of magnifying the interval by a factor of 2. This is done by notionally replacing each single state in the initial interval by two states. Under the assumption that all states occupy the same unit of time, this particular multiplication would thus have the effect of incrementing the variable *i* at half the original pace.

In terms of media items such a multiplication may be used to specify a change in the rate of display of a media item, as the following five examples illustrate. Note that for clarity in this instance we are using an explicit presentation function *play* for the MI *a*. The second parameter of *play* controls the actual display.

Example 1: video *a* is to be played at half speed: $\mathbf{len}(2)$ **proj** *(play(a,* **true***))*
Example 2: video *a* is to be played at a rate specified by the quantity *desired_rate* relative to the normal rate of play (*play_rate*):
$$\mathbf{len}(\mathit{play\_rate}(a)/\mathit{desired\_rate})\mathbf{proj}(\mathit{play}(a,\mathbf{true}))$$
Example 3: video *a* to be played at double speed: **short***(2)* **proj** *(play(a,* **true***))*
Example 4: fast forward video *a*: $\mathbf{short}(\mathit{max\_rate}(a)/\mathit{play\_rate}(a))$ **proj** $(\mathit{play}(a,\mathbf{false}))$
Example 5: rewind video *a*: **short**$(\mathit{max\_rate}(a)/\mathit{play\_rate}(a))$ $\mathbf{proj}$ *(reverse(a,* **false***))*

In example 3, **short** is the inverse projection function to **len**. We will not give a separate definition for **short**. In fact, it would be possible to permit **len** to take fractional values, and dispense with **short** altogether.

4.2 More general projections – scripting

While multiplication represent a common use of projection, there are several other important applications of this concept. One such is scripting. Consider the 'sleeping baby' portion of the Beethoven problem:

> During the music, whenever there is a pianissimo passage, display a still video of a sleeping baby.

We assume that the predicate *play(B5)* may be used to present the audio of the symphony, that *pp(B5)* is true during the pianissimo passages, and that *show(baby)* will present the required video still. Since the sub-sequence of the states of the predicate *play(B5)* in which *pp(B5)* is true defines the states of an interval in the states of which we are to display

the video-still, this suggests that we use that sub-sequence to define a second projected interval for the predicate *show(baby)*. This is a more complex form projection than the multiplications thus far considered.

For our purposes an intuitively helpful way of describing a general projection of the form w_1 **proj** w_2 is to regard the projector as inserting a copy of w_1 between each successive state of w_2. As an example of this intuition, consider again the projection

len*(2)* **proj** *play(B5)*

It should be apparent that notionally inserting an interval of length 2 between each primitive unit of play of the music, is consistent with the earlier description of this multiplication as slowing down the presentation by a factor of 2.

For the scripting problem, we depict the playing of Beethoven's symphony and the pianissimo passages therein as follows:

*	*	*	*	*	*	*	*	*	*	*	*	*	*play*(*B*5)
F	T	F	T	F	F	T	T	F	F	T	F	F	*pp states*
	*		*			*	*			*			*new interval*

At each state in this newly defined interval, *show(baby)* is to be true. This is equivalent to saying that the predicate **always***(show(baby))* is to be true over this new interval. This new interval is a projection, according to our general notion, but it is not a multiplication. It which may be defined using the *scripting* projector **when**, whose definition is as follows:

w **when** b is true if w holds on the interval made up of just those states in which the Boolean expression b is true

This operator may be defined in terms of the operators listed in Appendix 1 together with the general projection operator. Details of this definition are beyond the scope of this paper, but are due to Roger Hale and appear in [13], where this operator is used in the specification of the behaviour of a reactive system, an elevator.

A simple ITL illustration of this operator, taken from [13], would be **len***(6)* **when** *(x=0)*. Here, **len***(6)* is true on the interval made up of those states in which *x=0* is true. This formula then is true over any interval in which x has the value 0 exactly 6 times.

With Hale's scripting operator we can specify this part of the Beethoven problem. We define in effect a single interval over which we play the symphony, *(play(B5))* and over which the predicate *pp* provides the appropriate scripting projector, making use of the **when** operator, for the predicate *show(baby)*. Thus we write the conjunction as

play(B5) **and** (**always** *show (baby)* **when** *pp(B5))*

Notice the use of **always** – we have a (projected) interval in *every state* of which the predicate *show (baby)* is to hold.

4.3 Modelling the Beethoven Problem

We now give a reasonably complete specification of the entire Beethoven Problem. with the exception of the final two requirements which involve intervention and adjustments.

We will use the following media-item types and objects:

MI TYPE	**MIs**	
audio_item	m_1,m_2,m_3,m_4;	– *movements of the symphony*
text_item	t_1,t_2,t_3,t_4;	– *notes between movements*
video_still_item	*baby*;	– *picture of sleeping baby*
vvideo_clip_item	*bird*;	– *woodpecker video*
video_still_item	*ludwig*;	– *picture of Beethoven*
video_clip_item	*commercial*;	– *purchasing information*

We will also assume the following presentation operators for the various types:

TYPE	**OPERATOR**
audio_item	*play*
text_item	*display*
video_still_item	*show*
video_clip_item	*play*

We now present the entire specification (in the keyword representation):

[
(display(t_1); *play* (m_1);
display(t_2); *play* (m_2);
display(t_3); *play* (m_3);
display(t_4); *play* (m_4))
and
*(***always** *show (baby)* **when** *pp(b5))*
where $b5 = \{m_1,m_2,m_3,m_4\}$
and
*(***always** *play (bird)* **when** *staccato*(m_2))
];
len *(15)* **and** *show (ludwig)*;
play (commercial)

The first four lines specify the sequential presentation of the program notes and the four movements of Beethoven's opus 67. These are followed by two scripting conjuncts, controlling the baby and the woodpecker in the manner described in the previous section. The next conjunct specifies the presentation of the portrait of the composer for an interval of length 15, followed, finally, by a presentation of the purchasing information.

4.4 Imprecision and adjustment

As Erfle [11] points out, it is not realistic to suggest that the author will want to express all constraints with absolute precision. Nor will the author necessarily be able so to do. Here are two examples:

> Start audio *a* shortly after video *b* ends;
> Start audio *a* and then start audio *b* after *a* has finished, but no more than 10 seconds after the start of *a*

In addition to imprecise specification, there must be provision to deal with exceptional conditions which would arise if, for instance, *a* in the second example just given has a duration equal to 12 seconds. In a similar manner to [11], we outline two ways whereby imprecision in the specifications may be modelled. Both make use of projection.

In the first approach, the degree of strictness may be determined by a multiplication projector. For this purpose we replace both **len***(n)* and **short***(n)* by a single multiplier function, as suggested earlier, to which we add an additional parameter. We write this new function as **mult**($\pm n$, *strict*) where the second parameter is an attribute whose value defines the strictness of the projection. The default is *strict*, but the author may use other values as required.

The second approach is again to use the multiplication **mult***($\pm n$, strict)*, but to interpret *strict* as a directive to the *application*, which will determine which projector to use.

Although adjustment is closely tied to the notion of imprecision, it is less a question for the document model we are introducing than it is for the *author*. It is for the author to indicate that an adjustment or, alternatively, a replacement is to occur if a media item does not fit the interval as specified (just as it is a programmer's responsibility to predict and resolve exceptional conditions). The formalism provides the means to do both of these. They provide a subset of the power normally associated with exception handling in programming languages, but it appears to be sufficient for this application area. We first note that a replacement may be specified using the implication operator, which defines a conditional construct:

if *dur(a)>10* **then** *b* **else** *c*

In this case, the 'normal' formula *c* is to be replaced by the 'special case' *b* if the duration of the object a exceeds 10 units. An adjustment would also make use of the conditional. In the two examples which follow, the abnormal condition is assumed to occur if the duration of the object a is less than *n* units.

if *dur(a)<n* **then** (*a;* **len** *(n-dur(a))*)
Here the author has decided to insert an additional delay to make up the *n* units of time.

if *dur(a)<n* **then** (**len** *(dur(a)/n)* **proj** *a*)
In this case the presentation of a is adjusted by a multiplication projector so that it indeed takes the correct amount of time.

Both case uses an empty else part for the case where no adjustment is needed. For the last part of the specification of the Beethoven Problem, it would be a simple matter to add the obvious conditional, and insert whatever adjustment is desired.

5 USER FUNCTIONS – USER INTERACTION

The term user interaction implies that the *reader* interacts with the document. The Beethoven problem contains two examples:

rewind the video
fast-forward to the start of the subsequent movement.

Such reader intervention implies two separate issues to be resolved within the formalism:

specification and manipulation of the user 'interrupt'
specification of the new action required by the user

The second of these requirements is relatively simple. The specification of each operation within the media item-type definition must include a specification of each permitted reader interaction. User actions are not, in this regard, special in any way, and nothing needs to be added to the formalism to accomplish these actions. The first of these two issues requires a specification of 'early termination' of an action. ITL provides an interval operator **prefix** which may be used for this purpose. In ITL, the predicate **prefix***(w)* is true over any interval which is a prefix of an interval over which *w* is true. Thus this operator permits 'early termination' of a statement, typically of a loop. ITL examples appear in [3,2].

To provide for reader intervention, therefore, the operation play within the media item type video-clip in the Beethoven Problem might actually be specified as

$$\begin{aligned} \mathit{play} \quad = \quad & \mathbf{prefix}(\mathbf{for}\ i < \mathit{frames}(b)\ \mathbf{do}\ \mathit{show}(\mathit{frame}(i,b))) \\ \wedge \quad & (\mathbf{if}\ \mathbf{user}(\mathit{stop}(b))\ \mathbf{then} \ldots) \\ \wedge \quad & (\mathbf{if}\ \mathbf{user}(\mathit{ff}(b))\ \mathbf{then} \ldots \mathbf{proj} \ldots) \\ \wedge \quad & \ldots \end{aligned}$$

This is an operation definition within a type definition. The intent is that 'normally' the play function will operate by showing all of the frames of a video-clip in sequence. That is the intended effect of the formula

$$\mathbf{for}\ i < \mathit{frames}(b)\ \mathbf{do}\ \mathit{show}(\mathit{frame}(i,b))$$

However, **prefix** indicates that this loop may terminate early. This would happen if and only if any of the other predicates in the formula are true. In this example we have added predicates that demonstrate two possible user actions. It is assumed that the predicates which start with *user* become true upon the specified user action, *stop* or *ff* (for fast-forward) in this instance, actually occurring.

6 DETERMINACY AND AUTHORING

In this section we will indicate fairly briefly how this model may be utilised to provide an authoring tool.

6.1 Executability of ITL formulae

In section 2, we indicated what we meant by requiring that the model be *executable*, so that the dynamic behaviour of formulae may be exhibited. Several things are to be said with regard to ITL. In the first place there is a notion of the *normal form* of a temporal logic formula. This is a formula, equivalent to the original formula, but consisting of a set of conjuncts which show the state by state behaviour of all free variables in the formula over the interval in question. Clearly, this normal form, if indeed it can be derived, could be used to display successive 'states' of a system of temporal constraints, and hence to build the sort of dynamic model of the effect of a set of constraint specifications that an author might

find useful. However, in temporal logic the general problem of finding such a normal form is unsolvable.

On the other hand, it is known there are subsets of temporal logic where a normal form always exists and can be found. Furthermore, there is at least one interval temporal logic programming language based on such an implementable subset. The language in question is known as Tempura, and is fully described in [2]. We have chosen to base our formalism very heavily on this particular interval temporal logic.

6.2 Execution and manipulation: the display form

In fact, a number of the ITL operators needed in this model fall outside this executable subset (though Moszkowski in fact makes no claim that his subset is the largest executable subset). For example, any operation involving the operator *or* will be non-deterministic, and therefore be outside the executable subset. Any formula involving a projection is similarly outside the executable subset.

This situation is, fortunately, a good deal less serious than it sounds. The fact that a certain formula may not be executable does not mean that it cannot be manipulated by the author. Indeed, the author will usually wish to manipulate specifications involving non-determinacies and inderminacies, so that selections and refinements can be made. Our intention is to develop an authoring system in which such manipulation can take place. For this reason, we have developed a *display form* of many of the ITL formulae which we are using. In brief, the display form make explicit the empty sub-intervals which are implicit in the ITL formulae. By *empty* we mean an interval in which no MI is displayed. We will consider a single example of what we mean. Consider first the binary relation *before* which appears in Appendix 2 as

An alternative way of representing this formula is to make explicit the empty interval between the MIs *a* and *b*, as in the display form:

This display form can be used to graphically display the effect of such a binary relation to the author. The author may then manipulate the form, selecting different values of the indeterminate quantity *n*. Once any specific value of *n*, is selected, the display form becomes deterministic and belongs to Moszkowski 's executable subset. In [5], we give a display form for all of the time and duration relationships between arbitrary sets of MIs. A particular area under current consideration involves an analogous display form for specifications involving projections, but this matter is outside the scope of the current paper.

7 DISCUSSION AND CONCLUSIONS

As suggested earlier, we have a number of reasons for selecting Tempura, or more accurately Moszkowski's ITL in this work. In the first place, we feel that using a well-founded formalism, rather than an ad hoc approach such as the attribute grammars described, is an important decision when dealing with the complex temporal constraints which can arise in

the documents we are interested in. Since we are ultimately interested in an authoring tool, we believe that it is important to have specifications which an author writes based upon a formalism which will permit consistency checking. We also require the ability to build complex specifications by composing them from smaller verified components; that is we require a powerful set of composition and decomposition mechanisms. Moszkowski's ITL provides very much what is required in this regard.

On a different matter, it is also worth remarking that although we have concentrated exclusively on the expression of temporal constraints, there is no *a priori* reason why such a formalism should not be used for other, related, problems. For example, the placement of text objects onto pages in document formatting systems may be regarded as an analogous constraint problem, where the interval in question is spatial rather than temporal. We have not studied the application of such a formalism to this problem in any detail. Other authors have investigated similar formalisms for problems concerned with system performance and quality of service [14].

With regard to future work on this formalism and its applications, we wish to proceed from the formalism to a higher-level specification language for temporal constraints in multimedia documents. We intend to treat the version of interval temporal logic which we arrive at as the 'assembler' for a specification language. Some details of what we have in mind have been hinted at in the above. We would then provide a translator for such specifications, which would comprise a translator from our specification language into its "assembler", which we hope will be very close to Moszkowski's temporal language Tempura. Then we would plan to make use of an existing interpreter for that language. This aspect of the work is currently in its relatively early stages.

The longer term provides us with a number of interesting problems and challenges. On the one hand, we wish to proceed towards a complete authoring tool. As suggested in section 6, we plan to combine our specifications with a suitable editor, so that it would be possible for the output of the interpreter to provide a graphic illustration of the static or dynamic affect of the author's specifications. We will then work towards an interactive tool, which would display the effect of changes to a set of temporal constraint specifications.

A second class of questions concerns document transformation. As mentioned in the case of static paper documents, there is already some interesting work in this area, and one can pose analogous problems in the case of multimedia documents with temporal relations, in addition to hierarchical and spatial ones, between component objects.

ACKNOWLEDGEMENTS

I am grateful to Helen Cameron for critically reading a draft of this paper, to Ben Moszkowski for some helpful discussions, and to Susan Harder for speedy and accurate typing.

REFERENCES

1. ISO 10744, Information Technology — Hypermedia/Time-based Structuring Language (HyTime), 1992.
2. B. Moszkowski, *Executing Temporal Logic Programs*, Cambridge University Press, 1986.
3. P.R. King, 'Vers un formalisme basé sur la logique temporelle pour l'expression de contraintes temporelles dans les documents multi-média', Technical Report Rapport de Recherche no 942, LRI, Université de Paris Sud, Orsay, France, (December 1994).

4. P.R. King, 'Towards an ITL Based Formalism for Expressing Temporal Constraints in Multimedia Documents', in *Proceedings IJCAI Workshop on Executable Temporal Logics*, pp. 63–75, Montréal, (August 1995).
5. P.R. King, 'A logic based formalism for temporal constraints in multimedia documents'. PODP 96 to appear.
6. V. Quint and I. Vatton, 'Grif, an Interactive System for Structured Document Manipulation', in *Proceedings of the International Conference on Text Processing and Document Manipulation*, ed., J.C. van Vliet, pp. 200–213. Cambridge University Press, (1996).
7. ISO 8613, Information Processing — Text and Office Systems — Office Document Architecture (ODA) and Interchange Format, 1989.
8. ISO 8879, Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML), 1986.
9. A. Brown, Jr., T. Wakayama, and H. Blair, 'A reconstruction of context-dependent document processing in sgml', in *EP92: Proceedings of Electronic Publishing 92, International Conference on Electronic Publishing, Document Manipulation and Typography*, eds., C. Vanoirbeek and G. Coray, pp. 9–25. Cambridge University Press, (1992).
10. F.C. Cole and H. Brown, 'Editing structured documents — problems and solutions', *Electronic Publishing Origination, Dissemination and Design*, **5**(4), 211–218, (1992).
11. R. Erfle, 'Specification of temporal constraints in multimedia documents using HyTime', in *EP94: Proceedings of the Fifth International Conference on Electronic Publishing, Document Manipulation and Typography*, eds., C. Hüser, W. Möhr, and V. Quint, pp. 397–411. J. Wiley, (1994).
12. B. Moszkowski, 'Some Very Compositional Temporal Properties', in *Proceedings of Programming Concepts, Methods and Calculi. IFIP Transactions A-56*, ed., E.-R. Olderog, pp. 307–326, Amsterdam, (1994). North-Holland.
13. R. Hale, 'Using Temporal Logic for Prototyping: the Design of a Lift Controller', in *Lecture Notes in Computer Science*, volume 379, pp. 375–408. Springer-Verlag, (1989).
14. H. Bowman, L. Blair, G.S. Blair, and A.G. Chetwynd, 'A formal description technique supporting expression of quality of service and media synchronization', in *Multimedia Transport and Teleservices International COST 237 Workshop*, Vienna, Austria, (November 1994). Appears as Vol 882, Lecture Notes in Computer Science, Springer-Verlag, 1994.
15. J.F. Allen, 'Maintaining knowledge about temporal intervals', *Communications of the ACM*, **26**(11), (November 1983).

Appendix 1 Operators from Moszkowski's ITL

We present two forms (representations) for many of these operators, one symbolic and one ASCII.

First order operators:

$\wedge$	$\sim$	**and**	**not**
$\vee$	$\equiv$	**or**	**equiv** (equivalent)
$\supset$		**impl** (implies) can also use **if** . . . **then** . . .	

Temporal quantifiers:

$\forall$	$\exists$	**for_all**	**exists**(there exists)

Temporal operators, such as:

$\bigcirc$	**next**	the value in the next state
$\Box$	**always**	across all states
$\Diamond$	**sometimes**	in some state
$\Diamond$ i	**first**	in some initial subinterval
$\Box$ i	**all_init**	in all initial subintervals
$\Diamond$ a	**arb**	in some arbitrary subinterval
$\Box$ a	**all**	in all initial subintervals

Assignment and equality operators:

gets	assignment from previous state
$=$	initial state assignment

Interval operators:

len	number of states in interval
empty	**len** = zero
more	**len** $\neq$ zero
halt	condition terminating interval

The *chop* operator, ';' where $a;b$ is true over an interval T, say, if T can be decomposed into two consecutive subintervals, $T = T_1T_2$ such that a is true over T_1 and b is true over T_2.

A set of *arithmetic operators*, which are not further specified.

Appendix 2 ITL expressions for MI constraints involving *time* and *duration*

An author will wish to select from a pre-defined set of media item (MI) types and media item operations, and use variables of these types. For the purposes of this paper, however, these types and variable declarations can largely be ignored; we almost always identify a media item and the set of temporal constraints to be specified for it, and will usually use simple identifiers a, b, ... to represent occurrences of media items.

Time and duration constraints involving one single MI:

MI a is to start at clock time $t = n$ units:	$\mathbf{len}(n);a$	sequence
Specify a particular duration, n say, for a:	$\mathbf{len}(n) \wedge a$	conjunction
Measure the duration of a:	$\exists i{:}\ \big[(i = 0) \wedge (i\ \mathbf{gets} i + 1) \wedge a\big]$	computed in i

Time and duration constraints involving two MIs, a and b, say:
J. F. Allen [15] gives a functionally complete set of thirteen such binary relations, which may be depicted as follows:

```
aaaaa      aEb    Equals
bbbbb

aaaaabbbbbaMb     Meets          aaaaa  bbbbb  aBb    Before

aaaaa      aSb    Starts         aaaaa          aOb    Overlaps
bbb                                 bbbbb

bbbbb      aDb    During         bbbbb          aFb    Finishes
 aaa                                aaa
```

together with the six inverse relations: **MI, BI, SI, OI, DI, FI**
ITL formulae for these seven relations are as follows:

a**E**b	$\equiv$	a $\wedge$ b			
a**M**b	$\equiv$	a ; b	a**B**b	$\equiv$	a; $\Diamond$ b
a**S**b	$\equiv$	a $\wedge$ $\Diamond_{\mathrm{i}}$ b	a**O**b	$\equiv$	$\Diamond_{\mathrm{i}}$ a $\wedge$ $\Diamond$ b
a**D**b	$\equiv$	b $\wedge$ $\Diamond_{\mathrm{a}}$ a	a**F**b	$\equiv$	b $\wedge$ $\Diamond$ a

Time and duration constraints involving three or more MI's:
There are two approaches:

1. Specify n-ary relations as compositions of binary relations. Example: start the item b immediately after a and item c immediately after b. Here we need to synchronise on the object b:$\exists x{:}\ \big[\big(\Diamond_{\mathrm{i}}\ (\mathbf{halt}\ x \wedge a)\big);b\big) \wedge (\mathbf{halt}\ x; b; c)\big]$
2. Code n-ary relations directly in the formalism without decomposing them first into binary relations.

Examples:

(i) start b immediately after a and c immediately after b: a; b; c
(ii) start a when first of b,c,d stops: **len**(**min**(**dur**(b), **dur**(c),**dur**(d)); a