Specification and Prototyping of Structured Multimedia Documents using Interval Temporal Logic *

Howard Bowman[1], Helen Cameron[2], Peter King[2] and Simon Thompson[1]

[1] Computing Laboratory, University of Kent at Canterbury, Canterbury, Kent, CT2 7NF, United Kingdom
[2] Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, R3T 2N2, Canada
{H.Bowman,S.J.Thompson}@ukc.ac.uk {prking,hacamero}@cs.umanitoba.ca

Abstract. This paper explores a formalism for describing a wide class of multimedia document constraints. We outline the requirements on temporal logic specification that arise from the multimedia documents application area. In particular, we highlight a canonical document example. Then we present the temporal logic formalism that we use. This formalism extends existing interval temporal logic with a number of new features: actions, framing of actions, past operators, a projection-like operator called filter and a new handling of interval length. A model theory and satisfaction relation are defined for the logic and a specification of the canonical example is presented.

1 Introduction

This paper explores a formalism for describing a wide class of multimedia document constraints. The term *multimedia* indicates that a document may contain continuous or time-dependent entities [9] known as *media items* [5]. Part of the task facing the author of such a document, therefore, is to describe the dynamic temporal relationships that are to hold between media items. We are interested in documents with rich sets of such relationships, and as a consequence are keenly interested in issues of consistency verification, modelling, proto-typing, and specification refinement. While a number of authoring systems for multimedia documents are extant [8, 3], little investigation of suitable formalisms for such temporal constraints has been done.

The formalism that we introduce is an Interval Temporal Logic (ITL) specification notation called *Mexitl* (*Multimedia in Executable Interval Temporal Logic*). While the anticipated application area for this notation is multimedia, it is also relevant to other areas of real-time specification. King [10, 11] proposed

* Travel grants to support the research presented here have been provided by the British Council. The second author is supported by an individual research grant from the Natural Sciences and Engineering Research Council of Canada.

the use of ITL to specify multimedia documents and the theory presented here stems from that earlier work.

A major difference between ITL and standard linear time temporal logic [13] is that it is interpreted over finite state sequences, called intervals, rather than over infinite models. A number of authors have investigated ITLs, e.g. [14] [7] [12] [4]. The restriction to finite states prompts consideration of a number of temporal operators not typically found in non-interval temporal logics, e.g. *chop* and *projection*. These turn out to be useful in the multimedia documents application domain and will be discussed in Section 3.1.

We anticipate that complete specifications of multimedia documents will have a number of elements. An *abstract data typing* notation will be used to describe the primitive *operations/actions* of a specification, such as *displayCaption* or *playVideo*. We will not consider this notation here; the specification language that we present takes the primitive actions as given. Mechanisms to define composite actions out of primitive actions can also be added.

We introduce a methodology for developing multimedia artifacts using *Mexitl*. Specifications are written in the logic, and refined according to the rules of the language. Implementations can be developed as either deterministic refinements or as proofs of *Mexitl* formulas interpreted in a constructive logic.

Structure of the Paper. The paper is structured as follows. Section 2 reviews the requirements associated with multimedia documents, and introduces a typical problem from the field. Section 3 presents the specification notation that we advocate. The operators of the language are presented, the model theory is highlighted, and the satisfaction relation is defined. Section 4 applies the defined notation to the requirements of Section 2. Related work is discussed in Section 5 and concluding remarks are presented in Section 6.

2 Multimedia Documents Requirements

Erfle [5] presents a set of eighteen issues, or functional requirements, which are regarded as being sufficient to describe multimedia documents. This set was obtained by a study of what is provided in many existing authoring systems and standards. King [10] presents an equivalent set of eight requirements. For the sake of completeness we will provide a summary of these requirements. We divide our summary into two, a set of general requirements, dictated by the authoring aspect of this application, followed by eight individual functional requirements.

General Requirements We first need to represent the *display* of a media item, both *standard* display, where an item is displayed in its normal fashion at its normal rate, and *variations*, such as displaying at half speed, rewind, fast-forward. We also require facilities for both serial and parallel *composition* of sets of constraints. Parallel composition also permits independent development of *channels* [8, 3], which may then be combined so that they occur in the same multimedia presentation. Our use of the term channel generalises its multimedia usage to *independent authorship* and is akin to the term *thread*.

Functional Requirements The following eight individual items are required:

1. Temporal placement of a media item at an absolute (time) point;
2. Specification of the duration of a media item;
3. Determination of the start and finish points of a media item;
4. Relative placement of two or more media items;
5. Repetitive display of a media item;
6. Conditional display of a media item;
7. Scripting, that is, using events or conditions occurring in one media item to control the display of a second; and
8. Exception handling, that is, controlling error situations which may occur during the display of a multimedia document.

We illustrate these requirements by an informal specification of a fairly elaborate multimedia document, to be known as the *Beethoven Problem*. It requires the development of a multimedia document containing an audio of Beethoven's Fifth Symphony, opus 67 in c minor, together with various other media items to illustrate the music. King [11] presents an earlier version of a portion of this example. The seven parts of the Beethoven Problem appear in Figure 1.

Beethoven's Fifth Symphony comprises four movements.

1. Play the four movements of the symphony in sequence with a gap of 20 seconds between each movement.
2. Before the symphony, play an audio which announces the name of the symphony, the composer, and the orchestra. Two seconds after this audio starts, display a video still of Beethoven. Stop the video still display as the first movement starts. After the last movement, wait 3 seconds, display a video of Ludwig van Beethoven, and then, after a further 5 seconds, display for 30 seconds information about how to order this presentation.
3. At the start of each movement display a video still of a title for 5 seconds; repeat this 5 second display every 3 minutes during the corresponding movement.
4. The audio introduced in 2 is actually in three parts, corresponding to the name of the symphony, the composer, and the orchestra. During this audio, display, in sequence, three video stills containing the same information for an appropriate time.
5. In the twenty second gap between the second and third movements, show a video/audio display describing the third movement. If this display takes longer than twenty seconds, truncate it.
6. During each crescendo passage of the first movement, display a looped video tape of a bug climbing an inclined plane.
7. Count the number of staccato notes in the first movement.

Fig. 1. The Beethoven Problem.

3 Introduction to *Mexitl*

We present a *core language* for *Mexitl*, which contains the primitive constructs of the notation. Then we describe the model theory underlying the language; this theory is based upon finite sequences of states (called *intervals*) and we define the satisfaction relation.

3.1 The Core Language

Expressions have the following form:

$$E \ ::= \ c \mid v \mid V \mid f(E) \mid \mathbf{mylen}$$

where $c \in \mathcal{N}$, $v \in \mathbf{Var}_{static}$, the set of static variables, $V \in \mathbf{Var}_{state}$, the set of state variables, and f is in a set of assumed functions. In addition, **mylen** is a distinguished variable which denotes the length of the current interval.

$P \in \mathcal{P}$ (the domain of logical propositions) is constructed as follows:

$$\begin{aligned} P ::= \ & a_X \mid p(E_1, ..., E_n) \mid E = E \mid \mathbf{False} \mid P \Rightarrow P \mid P \ ; \ P \mid \\ & P \ \mathbf{proj} \ P \mid P \ \tilde{;} \ P \mid (\exists x \leq E) P \mid P \ \mathbf{filter} \ P \end{aligned}$$

where $a \in \mathbf{Act}$ and $X \subseteq \mathbf{Act}$ is a 'framing' set; p is in a set of given predicates and E is an expression. Much of this logic will be well known to a reader familiar with interval temporal logic [14]; for instance,

- **;** is the sequencing operator, *chop*, familiar from [14]. An interval satisfies $P \ ; \ Q$ if the interval can be divided into two contiguous sub-intervals, such that P holds over the first subinterval and Q holds over the second.
- **proj** is the projection operator, also described in [14]. An interval satisfies P **proj** Q if it can be sub-divided into a series of sub-intervals each of which satisfies P - we call P *the projection formula* - and a new interval formed from the end points of the sub-intervals satisfies Q, which we call *the projected formula*.

The remainder of our operators are not standard, and require more explanation.

Actions. Actions in *Mexitl* are atomic, in the sense that they cannot be analysed into simpler components. Time is discrete, and an action is thought of as taking place in a single state.

In the full paper [2] we consider actions with data attributes; here we confine ourselves to basic actions, written a. We assume that actions are given; their definition is not part of the language. From a logical point of view we can think of an action a as an atomic proposition.

An action can appear a number of times in an interval, however, each of these represents a different instance of the action. At the level of interval states, actions do not have duration. However, durational behaviour can be obtained by defining composite actions, which are a shorthand for the occurrence of multiple primitive actions. In particular, primitive actions may correspond to indexing into a composite action. For example, an action *video*[500] (the 500th frame of the video) might be a constituent of the composite action *video*.

Although actions do not have duration, sets of (distinct) actions can occur at the same state. Such sets reflect simultaneous lock-step occurrence of the actions. In this sense, the model employs synchronous parallelism.

Framing of actions. One aspect which distinguishes our usual perception of logical propositions and actions is the idea of framing. An assertion of a, where a is a particular action, is often interpreted as 'a *and no other action* happens' whereas a logical interpretation simply reads this as a happening. The former interpretation, in which the action a is "framed", would lead to a non-monotonic logic were we to adopt it.

Instead of this, in our system we subscript the actions with sets X of actions. a_X is interpreted as 'a happens and none of the other actions in X happens'. The set X thus provides an explicit frame within which the action a takes place. Logically the interpretation of a_X is the conjunction of a and $\neg b$ for all b in $X - \{a\}$. We add a distinguished action – **null** – to the set of actions. This action has null effect, but can be used for framing purposes thus: $\mathbf{null}_X$.

Length and Next. In contrast to the standard approach to ITL we have not included the next operator, $\bigcirc$, directly in *Mexitl*. However, standard length operators and $\bigcirc$ can be derived from the expression **mylen**, as follows:

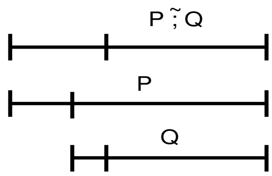
$$\mathbf{len}(E) \;\equiv\; \mathbf{mylen} = E \qquad \bigcirc P \;\equiv\; (\mathbf{mylen} = 1)\ ;\ P$$

We have included **mylen** as primitive for three reasons. It is useful to be able to refer directly to the length of an interval when defining various properties of component parts, such as the two halves into which it is cut by the 'chop' operator. Moreover, we use **mylen** as a bound for existential quantifications in derivations of many temporal operators from the core language. Finally, including **mylen** allows us to avoid a 'next' operator over expressions, thus avoiding expressions which might be undefined.

Quantification. In our core language we have included a limited form of existential quantification, namely quantification in which the value of the variable is *bounded* by (the value of) an expression: $(\exists x \leq E)P$. The effect of an existential quantification is to introduce a new variable in the scope of the quantifier. A bounded quantification has the property that its satisfaction over a given interval remains decidable; this is not the case for unbounded quantification over the natural numbers.

Past Operators. We include a single past operator in *Mexitl*, chop in the past, denoted $\tilde{;}$. An implication of the inclusion of past operators is that the past history of a computation must be recorded; we will show how this is done in Section 3.2. Using such a more sophisticated model theory, $P \;\tilde{;}\; Q$ is satisfied by an interval such that,

1. P holds over the larger interval resulting from moving the start of the interval into the past, and
2. Q holds over the original interval according to a past history that is truncated at the start of the interval over which P holds.



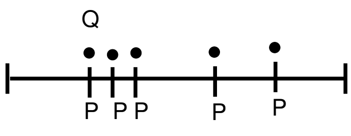
Intervals are depicted as line segments with three reference points. The leftmost point is the start of time, then moving to the right, the next point is the start of the current interval and the final point is the end of the current interval. Amongst other things, $\tilde{;}$ has the effect of chopping up the past, in a dual manner to which **;** divides up the future.

The more familiar past operators, since and previous, can be derived from chop in the past as follows. $P \;\tilde{\mathcal{S}}\; Q$ is given by

$$x = \textbf{mylen} \Rightarrow (((Q \ \wedge\ \textbf{mylen} > x) \ \wedge \bigcirc\Box(\textbf{mylen} > x \Rightarrow P)) \ \tilde{;}\ \textbf{True})$$

where $\Box$ is defined in Figure 3 and $\ominus P \ \equiv\ \textbf{False}\ \tilde{\mathcal{S}}\ P$. In fact, $\tilde{\mathcal{S}}$ is a strong since operator. The definition of strong since enables previously to be derived; weak since can also be derived in standard fashion. The incorporation of past operators is a significant departure from standard interval temporal logic. The motivation for their inclusion is that the specification of a number of examples is made substantially easier.

Filter. In filtering we have a variant of projection (**proj**). P **filter** Q has the effect of selecting all points which begin an interval over which the property P holds; the formula holds if over this new interval the property Q is true. In the following illustrative figure, propositions are associated with the initial point of the interval over which they hold.



filter generalises **when** as defined in [7]; in Hale's definition (based on **proj**) the property P needs to be a local property, that is, one without temporal operators. We see the generalisation as valuable – it allows us to express properties of the form 'whenever a musical performance is within 5 minutes of the end of a movement, show some more of a video sequence' (see for example Part 6 of the

Beethoven example presented in Section 2) – but we see no way of deriving it from the other *Mexitl* operators.

Types and definitions. We use a simple mechanism for naming types and values in the remainder of the paper. Note that '::' is used to mean 'is of type', and that definitions of types and values are not allowed to be (mutually) recursive.

3.2 Satisfaction

The order of presentation of this section is as follows. First we describe *computation structures*, which enhance intervals in order that past operators can be handled. Then we consider the interpretation of expressions and finally, we describe the satisfaction relation for the main temporal operators.

The major difference between our interpretation of *Mexitl* and the standard interpretations of interval temporal logics, to be found in [14] [7], for example, is that our satisfaction relation embraces past operators. Consequently, the sequence of states that have already been passed through must be recorded. This record is obtained by defining our satisfaction relation over what we call *computation structures*. These computation structures are pairs, (σ, i) where, in the normal way, σ is a sequence of states, with length $|\sigma|$, i.e., $\sigma = \sigma_0, \sigma_1, ..., \sigma_{|\sigma|}$ (we refer to this as the interval) and i is the *point of reference*. The point of reference identifies the past states of the computation (the *history*) and the interval under current consideration (the *current interval*). We introduce notation for prefix and suffix of intervals,

$$[\sigma]^i = \sigma_0, ..., \sigma_i \qquad (\sigma)^i = \sigma_i, ..., \sigma_{|\sigma|}$$

Since we have actions, our states are slightly more sophisticated than those of standard interval temporal logic. Specifically, each state σ_i is a pair; the first component of the pair A_i is a set of actions and the second component D_i records the current data state; it is a finite function from variables to data values (which come from $\mathcal{N}$, the natural numbers).

Expressions are interpreted in a standard manner apart from the operator **mylen**, which is interpreted as follows:

$$[\![\mathbf{mylen}]\!]_{(\sigma,i)} = |\sigma| - i$$

Our satisfaction relation, $\models$, interprets *Mexitl* propositions over computation structures. The notation $(\sigma, i) \models P$ denotes that the computation structure (σ, i) satisfies the proposition P. The most important clauses of the definition of satisfaction are in Figure 2; others which are standard can be found in [2].

Note that we say that $\sigma' \simeq_{x,E} \sigma$ if σ and σ' are the same length and they have the same value on all actions and variables except (perhaps) x, and that at each point the value of x is less than or equal to the corresponding value of E.

An arbitrary *Mexitl* proposition is interpreted relative to a zero point of reference. Thus, we define that an interval σ *satisfies* a proposition P if and only if $(\sigma, 0) \models P$. In addition, in the usual way, we state that P is *valid* if and

$$
\begin{array}{rll}
(\sigma, i) \models a_X & \text{iff} & a \in A_i \text{ and } \forall x \in X - \{a\}\ x \notin A_i \\[2ex]
(\sigma, i) \models P_1 ; P_2 & \text{iff} & \exists k \in \mathcal{N}\ (i \leq k \leq |\sigma| \text{ and } ([\sigma]^k, i) \models P_1 \text{ and } (\sigma, k) \models P_2) \\[2ex]
(\sigma, i) \models P_1 \ \mathbf{proj}\ P_2 & \text{iff} & \exists m \in \mathcal{N} \text{ and } \exists \tau_0, \tau_1, ..., \tau_m \in \mathcal{N} \\
 & & (i = \tau_0 < \tau_1 < ... < \tau_m = |\sigma| \text{ and} \\
 & & \forall j < m\ (([\sigma]^{\tau_j+1}, \tau_j) \models P_1) \text{ and} \\
 & & ([\sigma]^{i-1}.\sigma_{\tau_0}\sigma_{\tau_1}...\sigma_{\tau_m}, i) \models P_2) \\[2ex]
(\sigma, i) \models P_1 \ \tilde{;}\ P_2 & \text{iff} & \exists k\ (0 \leq k \leq i \text{ and } (\sigma, k) \models P_1 \text{ and } ((\sigma)^k, i - k) \models P_2) \\[2ex]
(\sigma, i) \models (\exists x \leq E) P & \text{iff} & (\sigma', i) \models P \ \text{ for some } \sigma' \simeq_{x,E} \sigma \\[2ex]
(\sigma, i) \models P_1 \ \mathbf{filter}\ P_2 & \text{iff} & \exists \tau_0, \tau_1, ..., \tau_m \in \mathcal{N} \\
 & & i \leq \tau_0 < \tau_1 < ... < \tau_m \leq |\sigma| \text{ and} \\
 & & \forall j < m\ ((\sigma, \tau_j) \models P_1) \text{ and} \\
 & & \forall k, i \leq k \leq |\sigma| \wedge (\sigma, k) \models P_1 \text{ implies } k = \tau_j \text{ for some } j \text{ and} \\
 & & ([\sigma]^{i-1}.\sigma_{\tau_0}\sigma_{\tau_1}...\sigma_{\tau_m}, i) \models P_2) \\
 & & \text{or no points satisfy } P_1
\end{array}
$$

Fig. 2. Satisfaction for *Mexitl*

only if for all σ in $\mathcal{I}$, $(\sigma, 0) \models P$, where $\mathcal{I}$ denotes the set of all possible intervals. If P is valid we write $\models P$.

[2] defines a full set of ITL derived operators from *Mexitl*. Due to space considerations we can only include the operators that we use in the Beethoven example of Section 4. These operators are presented in Figure 3.

3.3 Reasoning about *Mexitl*

We have developed a range of proof rules for the *Mexitl* logic. Details of these are given in [2]. The rules include

– characterisations of the basic operators of the language, such as

$$[P6] \qquad ((P \ \mathbf{proj}\ R) \vee (Q \ \mathbf{proj}\ R)) \Rightarrow ((P \vee Q) \ \mathbf{proj}\ R)$$

– derivation of common theorems of temporal logic from the definitions of operators given in [2];

$$\Diamond_t P \equiv \mathbf{mylen} = t \,;\, P \qquad \Diamond P \equiv \mathbf{True} \,;\, P \qquad \Box P \equiv \neg\Diamond\neg P$$

$$\overset{\mathrm{i}}{\Diamond}_t P \equiv P \,;\, \mathbf{mylen} = t \qquad \overset{\mathrm{i}}{\Diamond} P \equiv P \,;\, \mathbf{True} \qquad \overset{\mathrm{i}}{\Diamond}_X P \equiv P \,;\, \mathbf{null}_X^*$$

$$\overset{\mathrm{i}}{\Diamond}_{\leq t} P \equiv P \,;\, \mathbf{mylen} \leq t \qquad \overset{\mathrm{a}}{\Diamond} P \equiv \mathbf{True} \,;\, P \,;\, \mathbf{True} \qquad P^* \equiv P \;\mathbf{proj}\; \mathbf{True}$$

$$\mathbf{beg}\ P \equiv (\mathbf{len}(0) \wedge P) \,;\, \mathbf{True} \qquad \mathbf{halt}\ P \equiv \Box(P \Leftrightarrow \mathbf{len}(0))$$

$$\mathbf{halt}_X\ P \equiv \Box(P \Leftrightarrow \mathbf{len}(0) \vee \mathbf{beg}(\mathbf{null}_X)) \qquad \mathbf{fin}\ P \equiv \Box(\mathbf{len}(0) \Rightarrow P)$$

$$\mathbf{for}\ i := 1\ \mathbf{to}\ E\ \mathbf{do}\ P \equiv i := 1 \,;\, ((P \wedge i \leftarrow i+1 \wedge i \leq E)^* \wedge \mathbf{fin}(i > E))$$

$$P\ \mathbf{when}\ Q \equiv \neg\Diamond Q \vee (\mathbf{halt}(Q) \,;\, (\bigcirc\mathbf{halt}(Q) \;\mathbf{proj}\; P) \,;\, \mathbf{keep}\bigcirc(\neg Q))$$

Fig. 3. Derived Operators

- a characterisation of the framing of actions discussed earlier. From an action a_X we can deduce a and $\neg b$ for all b in $X - \{a\}$; the occurrence of *two* actions in X simultaneously results in a contradiction, and hence an unimplementable specification.

The rules have a number of purposes:

- Implication formalises refinement, so that a rule such as [$P6$] shows that a sound refinement of $(P \vee Q)$ **proj** R is given by a choice between (P **proj** R) and (Q **proj** R).
- We can use the rules to reason about specifications.
- Using the rules we can aim to find a normal form for *Mexitl* formulas (along the lines of Gabbay's work [6]) which may form the basis for an implementation of the language.

We do not claim that the logic we present is complete; we know of no complete axiomatisation of interval temporal logic with projection. Kono claims such an axiomatisation in unpublished work, but we have discovered that one of the rules in that system is unsound. See [2] for more details.

4 Applying *Mexitl*

We show how *Mexitl* can be employed for multimedia documents. We first describe a development methodology for *Mexitl* and then we show how the individual functional requirements are met by providing a complete formal specification of the Beethoven problem of Figure 1.

4.1 The *Mexitl* Development Methodology

The *Mexitl* interval temporal logic is designed to support the specification, design, prototyping and implementation of multimedia systems. Various aspects of the system are useful in different ways.

– The logical rules mentioned in Section 3.3 and [2] underpin a *refinement* strategy for specification development. A specification S_1 is refined by the specification S_2 if S_2 implies S_1, that is if $S_2 \Rightarrow S_1$ is valid. For instance, if $P' \Rightarrow P$ and $Q' \Rightarrow Q$ then

$$P' \ \mathbf{proj}\ Q' \Rightarrow P\ \mathbf{proj}\ Q$$

which states that we can refine a projection by refining either of its constituent formulae. We might, for instance, be more specific about the subdivision of an interval in a projection and replace **True** by $\mathbf{len}(2)$ in **True proj** Q, which is acceptable since $\mathbf{len}(2) \Rightarrow \mathbf{True}$. In a similar way we can refine a specification by conjoining other constraints into it. This meets our formal definition of refinement since $A \wedge B \Rightarrow A$ for any A and B.
– A specification written in *Mexitl* can be prototyped in two ways.
  - A general interval formula can be thought of as a *non-deterministic* description of an interval, since in general it will describe a collection of intervals rather than a single interval. A refinement of a specification will be more deterministic, in that it describes fewer intervals, and we can thus see the process of refinement as moving towards a deterministic specification implementable in standard ways.
  - Alternatively, taking a *constructive* view of logic as expounded in [17], we can view formulas of temporal logic as specifications with *proofs* being implementations. Preliminary work in this direction, which implements interval temporal logic in the Alf system, is reported elsewhere [18].
– As our first point suggested, we can build refinements of specifications by means of conjunction: we can think of this as a *parallel composition*, in contrast to the sequential composition given by chop.
If our actions are framed by giving an explicit framing set, as in a_X, then actions provide a means of *synchronising* between different conjuncts. For example in,

$$(a_X;b_X;c_X) \wedge (\mathbf{True};b_X;\mathbf{True})$$

where the framing set X is $\{a, b, c\}$, the only interval satisfying this formula must be one in which the b action in the second conjunct happens at the same point as the b action in the first conjunct. We achieve this synchronisation without adding any specific machinery for that purpose. We are also actively exploring the role that framing plays in the separate development of aspects of multimedia documents.
– We also conclude from our exploration of the sequence of examples in Section 4.2 that the *Mexitl* language is a low-level mechanism for representing such presentations; we aim to investigate higher-level approaches for which *Mexitl* as presented here might play the role of an intermediate language.

We introduce a methodology for developing multimedia artefacts using *Mexitl*. Specifications are written in the logic, and refined according to the rules of the language. Implementations can be developed as either deterministic refinements or as proofs of *Mexitl* formulas interpreted in a constructive logic.

4.2 Specification of Beethoven Example

We begin with a type definition of a symphony with four movements.

type symphony4 = (movement, movement, movement, movement)
type movement = interval – a finite interval of actions
m_1, m_2, m_3, m_4 :: movement

These definitions allow us to use m_1, m_2, etc. both as intervals in *Mexitl* formulae, and also as sequences of primitive actions into which we can index. We distinguish two elements of these sequences, $m_i[1]$ and $m_i[m_i.\mathrm{last}]$, where last is a predefined constant stating how many actions there are in such a sequence. The constants m_1, m_2, m_3 and m_4 are assumed to be set to the contents of the corresponding movements. In addition, the association of a framing set with a composite action, e.g. $m_{2\,X}$, implicitly frames all the primitive actions involved in the composite action, e.g. $m_2[5]_X$.

We now give a *Mexitl* specification of each part of the Beethoven problem as given in Figure 1.

1. In this specification, we frame each movement against any (other) actions from the set of actions comprising the four movements. The bounded temporal operator $\Diamond_{20''}$ is also framed to prevent any of the actions M from any of the movements from playing during the 20 second intervals.

   $$S_1 \equiv m_{1\,M} \ ; (\textbf{for } i := 2 \textbf{ to } 4 \textbf{ do } \Diamond_{20''\,M} \ m_{i\,M})$$

   where M is the set of all all audio actions forming m_1, m_2, m_3 and m_4.
2. We compose the items at the beginning and end of the presentation in a serial fashion and express the first of these as a parallel composition of two sub-channels. The video still itself is specified as the repetition (*) of an elementary one-state action displaying a picture of Beethoven.
   type still = action ; audio, video = interval
   a_1 :: audio ; Ludwig, orderinginfo :: still ; lvb :: video

   $$\begin{aligned} S_2 \equiv\ & (a_{1\,A} \ \wedge \ (\Diamond_{2''\,S}\mathrm{Ludwig}^*_S)) \ ; \ S_1 \ ; \\ & (\Diamond_{3''\,V}\mathrm{lvb}_V) \ ; (\Diamond_{5''\,S}(\mathrm{OrderingInfo}^*_S \ \wedge \ \textbf{len}(30''))) \end{aligned}$$

   where A is the set of audio actions, S is the set of still image actions, and V is the set of video image actions.
3. For each of the four movements we use $\textbf{halt}(m_i[1])$ and $\textbf{halt}(m_i[m_i.\mathrm{last}])$ to synchronise with the start and finish of each movement m_i. Actions are framed against the corresponding class of actions so as to avoid unwanted behaviour. Note that the earlier framing of actions in S_1 ensures, for example, that the occurrence of the first action in m_1 is uniquely defined.
   t_1, t_2, t_3, t_4 :: still – the four titles

   $$R_{=5''} \equiv (t^*_{i\,T} \ \wedge \ \textbf{len}(5'')) \qquad \text{– repeat title for exactly 5 seconds}$$

$R_{<5''} \equiv (t_i^*{}_T \;\wedge\; \mathbf{less}(5''))$ – repeat title for less than 5 seconds

$$S_3 \equiv S_2 \;\wedge\; \diamondsuit_T \;\mathbf{for}\; i \;:=\; 1 \;\mathbf{to}\; 4 \;\mathbf{do}\; (\; \mathbf{halt}_T(m_i[1]) \;;\; (\mathbf{halt}(m_i[m_i.\text{last}]) \;\wedge\; ((\diamondsuit_{2'55''} R_{=5''})^* \;;\; (R_{<5''} \;\vee\; (\diamondsuit_{\leq 2'55''} R_{=5''})))\;)\;)$$

where T is the set $\{t_1, t_2, t_3, t_4\}$.

4. The display of each still is synchronised with the end of the corresponding audio display. The framing here is more subtle. We cannot compose with the original S_2 in Example 2 above, since we deliberately framed S_2 against any other video stills occurring in that prefix interval. Hence, we have provided an alternative S_2' without framing on the item Ludwig.
   a_1, a_2, a_3 :: audio ; f_1, f_2, f_3 :: still

$$S_2' \equiv ((\mathbf{for}\; i \;:=\; 1 \;\mathbf{to}\; 3 \;\mathbf{do}\; a_{i\,A}) \;\wedge\; \diamond_2 \text{Ludwig}^*) \;;\; \cdots$$
$$S_4 \equiv S_3 \;\wedge\; \diamondsuit \;\mathbf{for}\; i \;:=\; 1 \;\mathbf{to}\; 3 \;\mathbf{do}\; (f_i^* \;\wedge\; \mathbf{halt}(a_i[a_i.\text{last}]))$$

5. We synchronise the display of the audio-video with an interval where the state previous to the first contains the last action of movement 2, and end the display when the next state contains the first action of the third movement. In the 20 second interval we either play the entire audio-video, which, if it is less than 20 seconds, will require a (strictly) initial interval, or play as much of it as we can, using a for loop with a bounded existential quantifier.
   type videoaudio = interval
   va :: videoaudio

$$S_5 \equiv S_4 \;\wedge\; \diamondsuit_{\{\text{va[i]}\}}(\mathbf{beg}(\ominus\; m_2[m_2.\text{last}]) \;\wedge\; \mathbf{halt}(\bigcirc m_3[1]) \;\wedge\; ((\diamondsuit\, \text{va}_{\text{VA}}) \;\vee\; (\exists\, t \leq \text{va.last})\; \mathbf{for}\; i \;:=\; 1 \;\mathbf{to}\; t \;\mathbf{do}\; \text{va}[i]_{\text{VA}}))$$

   where {va[i]} is the set of all actions corresponding to va and VA is the set of all video/audio actions.
   In Examples 6 and 7, we have chosen to omit all considerations of framing as they would confuse matters unduly. We assume that the condition staccato corresponds to a state variable whose value is supplied by the application. We assume that each note in the music occupies exactly one state in the interval.
6. C tests if we are in a crescendo by comparing the current volume with the volume in the previous and next states. The filter operation allows us to concatenate the crescendo passages as a single interval, over which we then play the video (of the climbing bug) as often and for as long as we can.

$$C \equiv (\exists x \leq \text{maxVol})\; ((\text{vol} = x) \;\wedge\; (\ominus(\text{vol} < x) \;\vee\; \bigcirc(\text{vol} > x)))$$
$$S_6 \equiv S_5 \;\wedge\; \diamondsuit\; (\; \mathbf{beg}(\ominus\; a_3[a_3.\text{last}]) \;\wedge\; \mathbf{halt}(\bigcirc\; m_1[m_1.\text{last}]) \;\wedge\; (C \;\;\mathbf{filter}\;\; (\text{video}^* \;;\; (\exists\, t \;\leq\; \text{video.last})\; \mathbf{for}\; i \;:=\; 1 \;\mathbf{to}\; t \;\mathbf{do}\; \text{video}[i])))$$

7. Point scripting conditions are specified with **when**. The definition of **gets** can be found in [2].

$$S_7 \equiv S_6 \;\wedge\; \diamondsuit\; (\; \mathbf{beg}(m_1[1]) \;\wedge\; \mathbf{halt}(m_1[m_1.\text{last}]) \;\wedge\; (\exists\; x \leq (\mathbf{mylen} + 1)((x = 1) \;\wedge\; (x \;\mathbf{gets}\; x + 1))\; \mathbf{when}\; \text{staccato}\;)$$

4.3 Serial Composition and Past Operators

In the foregoing we have commented on the issues associated with framing and parallel composition. These issues do not arise in the case of serial composition using the ; operator. Assuming P_A does not contain past operators, P_A implies nothing about the framing of Q in the formula P_A ; Q. Further, the past operators in the formalism allow a coauthor to create piece a of a document a ; b without providing information required by the author of piece b. For example, suppose that a definition P of "pianissimo" should be displayed exactly once. Instead of a having to signal b in some way whether or not a presents P, the author of b could specify **if** $\neg \diamondminus P$ **then** P.

As a further example, the author of b could use $\mathcal{S}$ (since) to specify ($\neg$ off) $\mathcal{S}$ on to determine whether or not a map display was left on by a.

$\ominus$ is useful for finding a transition point, that is, a point at which some property P becomes true: $\ominus\neg P \;\;\wedge\;\; P$. Example 6 above provides a specific example.

5 Related Work

There has been little previous work on applying temporal logic to the field of electronic publishing. King [10, 11] justify the use of ITL in this area and present examples. Furuta and Stotts [16, 15] make use of a temporal logic for specifying and tracing dynamic link paths in hypertext documents. Their logic is somewhat different from ours, and does not require, for example, any notion of projection, and their application is also rather different.

In contrast, interval temporal logic has a relatively extensive history, e.g. [14, 7, 12]. Our work builds upon this body of literature: *Mexitl* is derived from the ITL notation defined in [14]; we have used a number of the derived ITL operators defined in [7] and our proof theory is related to that of [12].

However, in terms of obtaining executability we have taken a rather different approach to other researchers. Ostensibly there are two approaches to obtaining executability. Firstly, a restricted logic (without operators such as eventually) can be used in order that specifications are deterministic and characterise just a single model. This is the approach employed by [14]. Alternatively, a richer set of logical operators can be allowed resulting in non-deterministic specifications having to be accommodated. This is the approach employed by [12] and, to take a non interval temporal logic example, by [1]. However, the consequence of such an approach is that more complicated execution strategies must be considered, in particular, backtracking must be employed. In contrast to these two alternatives, we are considering multi-levelled development strategies in which abstract *Mexitl* specifications are refined into concrete specifications. In general terms, the abstract specifications will be expressed in the full logic, while refinement entails evolving the specification towards a deterministic executable form. We are also exploring a constructive approach to this issue [18].

Duan's [4] approach to interval temporal logic is the closest to ours. Duan's work extends Moskowski's interval temporal logic in a number of respects: (1)

past operators are added, (2) a new projection operator is defined, (3) framing of variables is investigated, (4) infinite models are incorporated and (5) concurrency and communication primitives are considered. The last two of these are beyond the scope of this paper, however, the other extensions are of interest; we consider each in turn.

1. *Past Operators.* Firstly, we have incorporated past operators for somewhat different reasons to Duan. He is motivated by the desire to give an operational definition of assignment in the presence of framed variables. In contrast, our interest in past operators has arisen because they simplify certain of our example specifications. Importantly though, all the past operators that Duan uses can be defined from our chop in the past operator which, it should be pointed out, is different from the operator *Chopp* that Duan uses. These derivations are included in the full paper [2].
   In addition, Duan's chop (in the future) behaves differently to our chop in the presence of past operators in its second argument. Our reason for defining chop in the way we have is to fit with our motivatory examples and to allow $\Diamond$ and $\Box$ to be defined from it in the usual manner. Duan loses this interderivability. In addition, Duan's chop is derivable from ours as demonstrated in the full paper [2].
2. *Projection.* Duan defines a new projection operator, denoted *prj*. His motivation for defining this new operator is in order to model concurrency and interaction. However we have remained faithful to the original projection operator of Moskowski [14] which seems much more applicable to our application area. Furthermore, we have discovered how to derive Duan's *prj* operator from the standard projection. This derivation is also included in the full paper [2].
3. *Framing of Variables.* In the context in which we are working this has not been important; we have however introduced a notion of framing for actions, without using a non-monotonic logic.

6 Concluding Remarks

Our paper represents the first step in a programme of work looking towards formally based specification, refinement, prototyping and implementation of multimedia systems using interval temporal logic. We have shown how the *Mexitl* formalism is given a semantics; we have highlighted the fundamentals of a proof theory and we have shown how to use the formalism in the development of a substantial example. This is built in a series of steps using the operations of the logic to combine parts of the specification into a coherent whole.

For the future, we intend to investigate in a number of directions. We aim to develop the logic of *Mexitl*, working towards a system which is complete (relative to the theory of Peano arithmetic). We will also look further at a constructive implementation of *Mexitl*; the beginnings of which are reported in [18].

The methodology supported by *Mexitl* is also a topic of active interest for us: we aim to look further at the interaction between conjunction, framing and the separate development of channels in presentations. This will come from the investigation of further case studies as well as from looking at the theoretical issues involved. In particular, we aim to study higher-level languages for multimedia specification and their translation into the *Mexitl* logic.

References

1. H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM : A framework for programming in temporal logic. In *Lecture Notes in Artificial Intelligence, vol. 430*. Springer–Verlag, 1989.
2. H. Bowman, H. Cameron, P. King, and S. Thompson. *Mexitl*: Multimedia in Executable Interval Temporal Logic. Technical Report 3-97 (Kent), Computing Laboratory, University of Kent, 1997.
3. D.C.A. Bulterman and L. Hardman. Multimedia authoring tools: State of the art and research challenges. In *Computer Science Today: Recent Trends and Developments*, LNCS, No. 1000, pages 575–591. Springer-Verlag, 1995.
4. Z. H. Duan. *An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming*. PhD thesis, Univ. of Newcastle Upon Tyne, 1996.
5. R. Erfle. Specification of temporal constraints in multimedia documents using hytyime. *Electronic Publishing*, 6(4):397–411, December 1993.
6. D. Gabbay. The declarative past and imperative future. In *Temporal Logic in Specification*. LNCS 389, Springer-Verlag, 1989.
7. R. Hale. Using temporal logic for prototyping: the design of a lift controller. In *Lecture Notes in Computer Science, vol. 379*, pages 375–408. Springer–Verlag, 1989.
8. L. Hardman, G. van Rossum, and D.C.A. Bulterman. Structured multimedia authoring. *ACM Multimedia*, pages 283–289, 1993.
9. ISO 19744. *Information Technology – Hypermedia/Time-based Structuring Language (HyTime)*, 1992.
10. P.R. King. A logic based formalism for temporal constraints in multimedia documents. In *PODP 96*, September 1996. Revised version to appear in LNCS, Springer Verlag.
11. P.R. King. Modelling multimedia documents. *Elec. Pub.*, 8(2, 3):95–110, 1996.
12. S. Kono. A combination of clausal and non-clausal temporal logic programs. In *Lecture Notes in AI, vol. 897*, pages 40–57. Springer–Verlag, 1993.
13. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.
14. B. Moskowski. *Executing Temporal Logic*. Cambridge University Press, 1986.
15. P.D. Stotts, R. Furuta, and J.C. Ruiz. Hyperdocuments as automata: Trace-based browsing property verification. In *Proc. ACM Conf. on Hypertext*, pages 272–281. ACM Press, 1992.
16. P.D. Stotts, R. Furuta, and J.C. Ruiz. Hyperdocuments as automata: Verification of trace-based browsing properties by model checking. *ACM Transactions on Information Systems*, 1997. To appear.
17. S. Thompson. *Type Theory and Functional Programming*. Addison-Wesley, 1991.
18. S. Thompson. Constructive interval temporal logic in Alf. In *this volume*, 1997.

COVER SHEET

Specification and Prototyping of Structured Multimedia Documents using Interval Temporal Logic

Howard Bowman (Kent), Helen Cameron (Manitoba), Peter King (Manitoba) & Simon Thompson (Kent)

Computing Laboratory,
University of Kent at Canterbury,
Canterbury, Kent, CT2 7NF, United Kingdom

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

{H.Bowman,S.J.Thompson}@ukc.ac.uk
{prking,hacamero}@cs.umanitoba.ca

Area: (2) Specification and Verification

Keywords Interval temporal logic; actions; framing; past operators; semantics; multimedia; case study; Beethoven

Abstract: This paper explores a formalism for describing a wide class of multimedia document constraints. We outline the requirements on temporal logic specification that arise from the multimedia documents application area. In particular, we highlight a canonical document example. Then we present the temporal logic formalism that we use. This formalism extends existing interval temporal logic with a number of new features: actions, framing of actions, past operators, a projection-like operator called filter and a new handling of interval length. A model theory and satisfaction relation is defined for the logic and a specification of the canonical example is presented.

This article was processed using the LaTeX macro package with LLNCS style