A Sequence-Based Object-Oriented Model for Video Databases

Marlon Dumas (marlon.dumas@imag.fr)
Rafael Lozano (rafael.lozano@imag.fr)[*]
Marie-Christine Fauvet (marie-christine.fauvet@imag.fr)
Hervé Martin (herve.martin@imag.fr)
Pierre-Claude Scholl (pierre-claude.scholl@imag.fr)
LSR-IMAG, University of Grenoble, BP 72, 38402 St-Martin d'Hères, France

Abstract. Structuration, annotation and composition are amidst the most crucial modeling issues that video editing and querying in the context of a database entail. In this paper, we propose a sequence-based, object-oriented data model that addresses them in an unified, yet orthogonal way. Thanks to this orthogonality, the interactions between these three aspects are properly captured, i.e. annotations may be attached to any level of video structuration, and all the composition operators preserve the structurations and annotations of the argument videos. We also propose to query both the structuration and the annotations of videos using an extension of ODMG's OQL which integrates a set of algebraic operators on sequences. The overall proposal is formalized and implemented on top of an object-oriented DBMS.

Keywords: video databases, temporal databases, object-oriented databases, ODMG.

1. Introduction

For a long time, videos were managed by specific environments due to their huge size and demanding hardware requirements. Improvements in data compression (such as MPEG-2), continuously increasing network transfer rates and processing capacity, and advances in operating systems, now allow conventional computing systems to handle videos, thereby opening new application areas such as video-on-demand, video conferencing and home video editing.

The specificities of these applications have been addressed from several perspectives by a large body of research works, prototypes and commercial tools (see [9] for a survey). Many of these efforts have focused on important performance issues related to storage, real-time transfer in distributed environments, quality of service, synchronization, etc. However, as the needs of these applications evolve, other issues such as content-based retrieval, summary extraction, interactive editing and browsing, etc. become increasingly important. In particular, advances in automatic indexing techniques, together with video meta-data standardization efforts (such as MPEG-7), have accentuated the need for high-level representation models and querying tools for video annotations.

In this setting, we believe that database technology is likely to play an increasingly important role in video management, provided that it offers high-level concepts and

[*] Supported by SFERE–CONACYT, Mexico

interfaces for storing, retrieving and editing video data. Our work is intended to be a contribution in this direction.

Video modeling in the context of a database entail numerous issues which have been addressed separately in several works. In this paper we focus on three of them, namely *structuration* (also known as *segmentation*), *annotation*, and *composition*, and we address them in a comprehensive framework. This framework is essentially based on ideas developed in the context of Temporal Databases [29, 7, 26], and assumes an underlying object-oriented data model (e.g. the ODMG [4] one). However, modulo some restrictions, the basic concepts can be adapted to an object-relational setting.

Since the proposed data model is based on concepts stemming from temporal databases, it is possible to query it using operators defined in this context. Accordingly, in this paper we propose to query video annotations using a variant of a temporal extension of ODMG's OQL previously proposed by the authors [11]. This is an important feature, since it entails that most of the evaluation techniques developed in the context of temporal query languages may be adapted to efficiently evaluate queries on videos.

The main originality of the proposal is that it properly captures the interactions between the three addressed aspects of video modeling. In particular, annotations may be independently attached to any level of video structuration, and video composition operators are defined so as to preserve the structurations and annotations of the argument videos. To achieve this latter feature, the composition operators are defined using object reflection techniques.

The next four sections successively deal with structuration, annotation, querying and composition of videos. Throughout these sections, the proposed data model and query language are formally defined and illustrated through several examples. Next, in section 6, we describe the prototype that we have developed to validate the proposal's feasibility. Finally, a comparison with related works is included in section 7 and conclusions are drawn in section 8.

2. Video structuration

Video structuration is the process of partitioning the frames of a video according to their semantical correlations. For instance, a video can be structured into shots, scenes, and video sequences[1]. There is a strong similarity between the structuration of a video and the structuration of time in calendars. Accordingly, we propose a video structuration model based on the concept of *granularity*. This concept has been extensively studied in areas such as temporal reasoning in artificial intelligence, and temporal databases. The definition of granularity that we adopt here is not completely novel, but rather generalizes the one developed in [31].

[1] The term "video sequence" as used here is not to be mistaken with the datatype "sequence" introduced in section 3.1.

2.1. TIME-LINES AND GRANULARITIES

The basic concept of the video data model that we consider is that of *time-line*. At an abstract level, a time-line TL is a pair (D, $<_{TL}$), made up of a finite set of *time-marks* D, and a binary relation $<_{TL}$ over this set defining a total linear order.

Time-lines model independent time-coordinate systems, with respect to which data and meta-data may be temporally anchored. Each video has its own time-line, which captures the temporal relations between its elementary components (i.e. its frames).

A *granularity* over a time-line TL is defined as a partition of TL into convex sets: each of these convex sets is then seen as an atomic *granule* and every time-mark of the time-line is approximated by the granule that contains it. A granularity is intrinsically attached to a unique time-line. The function that maps a granularity into a time-line is subsequently denoted TimeLine. Thus, the expression TimeLine(G) denotes the time-line that granularity G partitions.

A hierarchical structure is defined over the set of granularities of a time-line through the definition of the following relation.

Definition 1 (*Finer-than relation*). A granularity G1 is *finer-than* another granularity G2 (noted G1 $\prec$ G2) if TL(G1) = TL(G2), and if each granule of G2 may be associated to a set of consecutive granules of G1. Formally, G1 $\prec$ G2 iff:

$$\forall g_2 \in G2,\, g_2 = \bigcup_{g_1 \in G1,\, g_1 \subseteq g_2} g_1$$

□

For any two granularities G1 and G2 such that G1 $\prec$ G2, two mappings are defined: one for *expanding* a granule of G2 into a set of granules of G1 (noted $\varepsilon_{G2,G1}$), and the other for *approximating* a granule of G1 by a granule of G2 (noted $\alpha_{G1,G2}$), as shown in figure 1. Formally:

- $\forall g_2 \in G2 \;\; \varepsilon_{G2,G1}(g_2) = \{g_1 \in G1 \mid g_1 \subseteq g_2\}$
- $\forall g_1 \in G1 \;\; \alpha_{G1,G2}(g_1) =$ the unique $g_2 \in G2$ such that $g_1 \subseteq g_2$

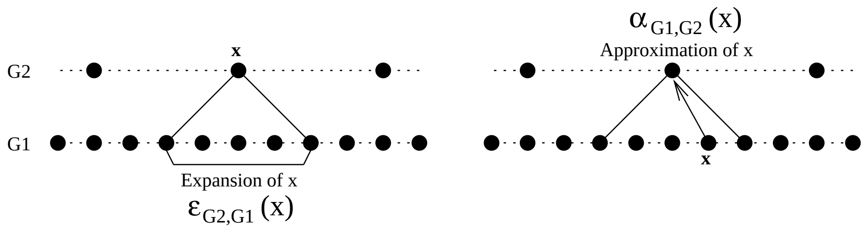


Figure 1. Granule expansion and approximation

2.2. Using granularities to model video structuration

Any level of structuration of a video is naturally modeled by a granularity over its time-line. Indeed, the set of shots of a video may be seen as a set of disjoint intervals of time-marks covering the whole time-line of the video. Notice that it is also possible to view the frames of a video as a "degenerated" partitioning of its time-line, in which each partition is composed of a single time-mark. The advantage of this modeling is that the notion of frame is seen as a structuration level of a video in the same way as the shots or the scenes. As a result, it is possible to attach annotations to the shots or the scenes in the same way as they are attached to frames, as detailed in the next section.

The following ODMG interfaces specify the foundations of the proposed structuration model.

```
interface TimeLine; /* not detailed here */
interface Granularity {
  TimeLine TimeLine();
  bool finerThan(Granularity other);
}
interface Video {
  attribute TimeLine VTL;        /* standing for Video Time-Line */
  attribute Granularity frames;        /* granularity corresponding to the frames */
}
```

For a given object V belonging to a class which implements the above Video interface, the following constraint apply[2]:

```
V.frames = MinGranularity(V.VTL)
```

Where MinGranularity denotes the function that retrieves the finest granularity of a time-line (this is the granularity whose granules are all singletons).

The Video interface is not intended to be used as is. Instead the application developers are expected to specialize it (through inheritance) to fit each specific situation. For instance, the following ODMG statements show how the above interface may be specialized to model the structuration levels defined in classical film theory, i.e. video sequences, scenes and shots.

```
interface Film : Video {
  attribute Granularity vsequences;    /* models the sequences of the video */
  attribute Granularity scenes;    /* models the scenes of the video */
  attribute Granularity shots;    /* models the shots of the video */
```

[2] Notice that it is the responsibility of the application programs to ensure all this as well as the other constraints introduced subsequently, since ODMG does not support general-purpose integrity constraints.

```
}
```

To capture the relationships between the structuration levels of a film, the application developer may impose the following constraints over the instances of classes implementing the Film interface (let F be such instance):

– TimeLine(F.vsequences) = TimeLine(F.scenes) = TimeLine(F.shots) = F.VTL.

– F.shots $\prec$ F.scenes $\prec$ F.vsequences.

The “finer than” relation over the granularities defining the structuration levels of a video is not necessarily a total order. In other words, it is possible to define two non-comparable structuration levels over a single video. This feature is not supported by most proposed video models (e.g. [14]).

To illustrate the gain of expressive power obtained through this feature, consider the following specialization of the Video interface, which is intended to model tennis match videos.

```
interface TennisVideo : Video {
  attribute Granularity shots;    /* models the shots of the video */
  attribute Granularity points;    /* models the scenes of the tennis match */
  attribute Granularity games;    /* models the games of the tennis match */
  attribute Granularity sets;    /* models the sets of the tennis match */
}
```

From their intended semantics, one may see that the three structuration levels points, games and sets, correspond to mutually comparable granularities, but none of these granularities is comparable with respect to the one modeling the shots of the video. For this simple reason, the model proposed in [14] cannot properly capture the meta-data of this kind of videos.

3. Video annotation

Given the current state of image processing technology, it is not reasonable to dynamically extract semantical information from a video during query evaluation over a video database. Therefore, in order to formulate content-based queries over videos, their semantical content must be previously described as *annotations*.

Annotations are generally stored separately from the “raw” video data. This approach is quite natural, since annotations are only needed during video querying[3], while access to raw video data is only required during video playing. In addition, this approach allows several “virtual” videos to share pieces of “raw” video data, without necessarily sharing any meta-data.

[3] Close captions are an exception, since they are annotations that must be displayed during video playing.

Our approach to video annotation is based on a single parametric type, namely Sequence that we describe in the sequel.

3.1. Basic abstract datatypes

On the basis of the granularity model introduced in the previous section, the following ADTs are introduced to model temporal values and associations.

3.1.0.1. *Instant.* An *instant* is a reference to a granule. It is represented by a pair made up of a granularity and a natural number (the *position* of the referenced granule).

3.1.0.2. *Duration.* A *duration* is a number of granules of a granularity. Durations are signed so as to differentiate forward from backward motion in time. A duration is described by a pair composed of a granularity and an integer (the *size*).

3.1.0.3. *ISet.* An ISet (standing for Instant Set) models a collection of instants with no assumption on its representation. The Interval ADT is defined as a restriction of the type ISet which includes all convex instant sets.

3.1.0.4. *Sequence.* A *sequence* is a function from a set of instants observed at some common granularity, to a set of values sharing a common structure. The domain of a sequence is not necessarily an interval. The Sequence ADT is parameterized by the type of the range of the sequence. Subsequently, we use the notation Sequence<T> to denote the instantiation of the parametric type Sequence with type T.

An example of a sequence is:

s1 → ⟨main_character: Linda, main_action: cook⟩
s2 → ⟨main_character: Linda, main_action: cook⟩
s3 → ⟨main_character: Linda, main_action: cook⟩
s6 → ⟨main_character: Joe, main_action: eat⟩
s7 → ⟨main_character: Joe, main_action: eat⟩
s8 → ⟨main_character: Joe, main_action: eat⟩

Where s1, s2, s3, s6, s7, s8 denote instants at the same granularity (e.g. a subset of a video's scenes), and ⟨att_name: value, att_name: value⟩ denotes an object.

Note that since sequences are instances of an ADT, their internal representation may be different from the one suggested by its definition. For instance, the above sequence could be internally represented in the following way:

[s1..s3] → ⟨main_character: Linda, main_action: cook⟩
[s6..s8] → ⟨main_character: Joe, main_action: eat⟩

Where [s1..s3] and [s6..s8] denote intervals.

It can be seen from the above definition, that the integration of the sequence datatype into an object-oriented data model, requires that this latter supports parametric polymorphism. Unfortunately, this is not the case of ODMG. In section 6, we will discuss how this limitation can be circumvented at the implementation level.

3.2. Using sequences to model annotations

We recall that one of the main objective of our work is to provide a video meta-data model fulfilling the following requirements: (1) the structure of the annotations is not constrained, and (2) annotations may be independently attached to each level of structuration.

The first of these requirements is essential since video annotations are generally domain-dependent. Indeed, in the context of a movie, annotations may deal with the characters or actors appearing on it, while in a news TV program, they may deal with the events being reported. In addition, as we will see later, videos may be obtained by composition of several videos, whose annotations may have heterogeneous structures.

The second requirement on the other hand, increases the expressive power of the video data model. Indeed, in most previous proposals (e.g. [24, 12, 1, 19]), annotations are only attached to the video frames[4]. This withholds the user from expressing properties which are true of a scene, without being true at each frame in this scene. For instance, saying that in a given scene two characters talk to each other, is something that is true at the granularity of that scene, without being true of each frame in this scene.

To achieve the first requirement, our model allows any object of a database to be used as a video annotation. It is not until the definition of the database schema, that the designer imposes type restrictions on the annotations, so as to model the specificities of a given application. Concretely, annotations may range from simple full text structures, up to arbitrarily complex objects. As a matter of fact, a video segment may even be annotated by another video segment (i.e. annotations may be used to simulate simple hyper-links).

To achieve the second requirement, a video is annotated through a set of independent sequences, each one corresponding to a given level of structuration. In particular, a video may be annotated by as many sequences of annotations as there are levels of structuration defined over it as illustrated in figure 2. More generally, several sequences may be attached to a single structuration level. For instance, a single video may be annotated by two sequences at the granularity of the scenes.

This approach has the limitation that an annotation may not be attached to a video segment which does not correspond to an element of the structuration hierarchy (e.g. a shot or a scene). Although it is possible to attach an annotation to an arbitrary interval of frames, the semantics of this association is the same as if the annotation was attached to each frame in the interval. Still, as stated above, our approach is more expressive than those in which annotations may only be attached to the frames, and not to the elements of the other structuration levels.

[4] In [1, 19] annotations are attached to intervals of frames, but the underlying semantics is that the annotation is valid at each frame in this interval.

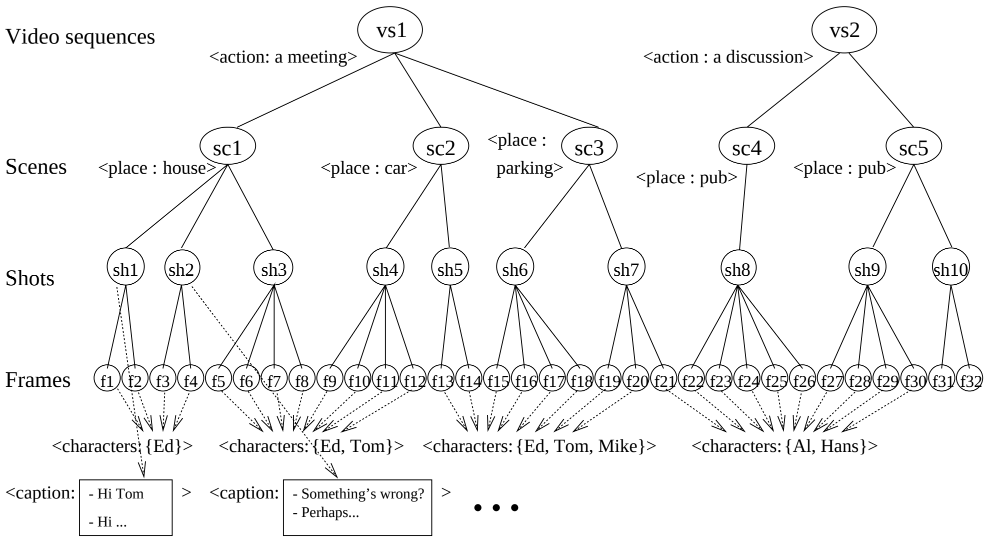

Video sequences
vs1
<action: a meeting>
vs2
<action : a discussion>
Scenes
sc1
<place : house>
sc2
<place : car>
sc3
<place : parking>
sc4
<place : pub>
sc5
<place : pub>
Shots
sh1 sh2 sh3 sh4 sh5 sh6 sh7 sh8 sh9 sh10
Frames
f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 f11 f12 f13 f14 f15 f16 f17 f18 f19 f20 f21 f22 f23 f24 f25 f26 f27 f28 f29 f30 f31 f32
<characters: {Ed}>
<characters: {Ed, Tom}>
<characters: {Ed, Tom, Mike}>
<characters: {Al, Hans}>
<caption: - Hi Tom - Hi ... >
<caption: - Something's wrong? - Perhaps... >
• • •

Figure 2. Sequences of annotations attached to a video. Video sequences are annotated with actions, scenes with locations, shots with close captions, and frames with the characters that appear within them.

3.3. USING SEQUENCES TO SHARE RAW VIDEO DATA

Up to now, we have modeled what is traditionally called the *meta-data* of a video, and have omitted to discuss about the images and sound samples that appear during video playing. Since this is generally the most bulky part of a video, any effort to compress or share it is extremely important. Video compression is a subject that has been thoroughly studied and most of the video formats (e.g. MPEG-2), are based on some compression technique. Sharing video data on the other hand, has received much less attention. In the context of an object database, this is relatively easy to achieve, since videos can be modeled as objects and object identifiers can be used to refer to them.

It is at this point that the distinction between raw videos and virtual appears. A *raw video* is a low-level concept modeling an image dataflow without any structuration nor indexing information (i.e. without meta-data). *Virtual* videos are then built from segments of raw videos, and may have some meta-data attached to them. Virtual videos do not "copy" the data contained in the raw videos, but rather reference them. Therefore, a given segment of a raw video may be shared by several virtual videos. This issue will be illustrated in section 5, where the creation and composition of virtual videos is discussed.

To model references to raw video data, we introduce the interface RVRef (standing for Raw Video Reference). An object of a class implementing this interface is composed of an instance of the RawVideo class (not described here), and a positive integer denoting the position of the referenced frame inside this raw video.

```
interface RVRef {
  attribute RawVideo source;
  attribute short frameNumber;
};
```

Note that since frame numbers are used as a time-base, it is not possible to know how much does a video last (in terms of seconds) until a presentation rate is fixed. We will discuss the issue of setting a presentation rate in sections 4.3 and 6.2.

The following interfaces (which complete those given in section 2) summarize the above discussion. The Film interface is intended to model the kind of videos depicted in figure 2.

```
interface Video {                    /* models structured and annotated virtual videos */
  attribute TimeLine VTL;
  attribute Granularity frames;
  attribute Sequence<RVRef> rawData;
}
interface Film : Video {                         /* a possible annotation schema for films */
  . . . /* attributes specified in section 2 go here */
  attribute Sequence<Action> actions;
  attribute Sequence<Place> places;
  attribute Sequence<String> captions;
  attribute Sequence<String> characters;
}
```

For a given object V belonging to a class which implements the Video interface, the following constraints apply:

– Granularity(V.rawData) = V.frames

– The domain of V.rawData is an interval, i.e. discontinuities are not allowed between the successive images of a video. This constraint does not restrict the set of operators that are applicable to the sequence of frame references attached to a video. It simply states that on the end, this sequence should have a convex domain. In particular, the composition operators that we present in section 5, are designed so as to respect this constraint.

Additionally, the database developer may introduce the following constraints on the instances of classes implementing the Film interface (let F be such instance).

– Granularity(F.actions) = F.vsequences, where Granularity(S) stands for the granularity of the domain of sequence S.

– Granularity(F.places) = F.scenes

– Granularity(F.captions) = F.shots

– Granularity(F.characters) = F.frames

4. VideOQL: a query language for video databases

In this section, we define an extension of ODMG's Object Query Language, namely VIDEOQL, that integrates the datatypes defined in the previous sections. We also illustrate through some significant examples, how this extension can be used to formulate structure-based and content-based queries on videos. The following notations are used throughout this section:

– T1 $\rightarrow$ T2 denotes the type of all functions with domain T1 and codomain T2.

– {T} denotes the type of sets of T.

– $\langle$T1, T2, ... Tn$\rangle$ stands for the type of tuples whose ith component is of type Ti (1 $\leq$ i $\leq$ n). Tuple components may be labeled using the notation $\langle$L1: T1, L2: T2, ... Ln: Tn$\rangle$.

– $\langle$v1, v2, ..., vn$\rangle$ denotes a tuple value whose ith component is vi. If the tuple attributes are labeled then the notation $\langle$L1: v1, L2: v2, ..., Ln: vn$\rangle$ is used instead.

It is worth noting that VIDEOQL is a variant of a query language for temporal databases that we described in [11]. This means in particular that at least some of the optimization techniques for temporal queries that have been reported in the temporal database literature [29, 10], could be adapted to build an efficient evaluation engine for VIDEOQL.

4.1. OPERATORS ON TEMPORAL VALUES

4.1.0.5. *Constructors* The instant constructor, noted P @ G, builds an instant from a natural number P (the position with respect to the origin) and a granularity G. The duration constructor is defined similarly and noted #. There are two interval constructors: [I1..I2] which builds an interval with lower bound I1 and upper bound I2, and [I | D] which yields an interval with lower bound I and duration D.

4.1.0.6. *Selectors* Two elementary selectors are defined on durations, namely Granularity and Size, such that Granularity(N # G) = G and Size(N # G) = N. Similarly, two selectors are defined on instants: Granularity and Position. The function Duration(IS) (IS being an ISet) yields a duration whose size is equal to the number of instants in IS.

4.1.0.7. *Conversions* Conversion operators on instants are defined on the basis of the approximation and expansion operators described in section 2.1. For instance, given two granularities G1 and G2 such that G1 $\prec$ G2, it is possible to expand an instant with granularity G2 into an interval with granularity G1, or to approximate an instant with granularity G1 by an instant with granularity G2. These conversions are performed by the following operators:

expand: Instant, Granularity $\rightarrow$ Interval
expand(I, G) = let G' = Granularity(I) in $\varepsilon_{G',G}$(Position(I)) @ G
/ precondition: G $\prec$ Granularity(I) */*
approx: Instant, Granularity $\rightarrow$ Instant
approx(I, G) = let G' = Granularity(I) in $\alpha_{G',G}$(Position(I)) @ G
/ precondition: Granularity(I) $\prec$ G */*

These operators may be straightforwardly generalized to instant sets by pointwisely applying them to each instant in the ISet.

4.2. Operators on sequences

The following specification introduces the selectors of the Sequence ADT.

Domain: Sequence<T> $\rightarrow$ ISet */* domain of the sequence seen as a function */*
Range: Sequence<T> $\rightarrow$ { T }
/ range of the sequence seen as a function */*
Value: Sequence<T>, Instant $\rightarrow$ T
/ Value(S, I) = value of S at I; precondition: I $\in$ Domain(S) */*
Duration: Sequence<T> $\rightarrow$ Duration
/ Duration(S) = Duration(Domain(S)) */*
Granularity: Sequence<T> $\rightarrow$ Granularity
/ Granularity(S) = Granularity(Domain(S)) */*

A set of algebraic operators is defined on sequences. These operators are divided into five groups: pointwise mapping, join, restriction, partitioning and splitting. The first three correspond to the classical projection, join and selection operators of the relational algebra, while the latter two are proper to sequences. In fact, partitioning and splitting are tightly connected to two important characteristics of sequences: granularity and order.

Modulo grouping, agregation and sequence splitting, the expressive power of these algebraic operators on sequences is that of the *sequenced queries family* defined in SQL/Temporal [27].

The pointwise mapping operator map, applies a function to each element in the range of a sequence while leaving the domain of the sequence intact. The join operator on the other side, merges two sequences into a single one by combining synchronous values.

There are two restriction operators on sequences. The first one (noted during) restricts the domain of a sequence to the instants lying in a given set of instants. The second one (noted when) restricts a sequence to the instants at which its value satisfies a given predicate. Such predicate is given as a boolean function whose parameter denotes a value of the sequence's range. Syntactically, the operator when is coupled with map, in the same way as OQL's select operator on collections is coupled with the where operator. The exact syntax of the constructs of the language is discussed below.

The partitioning operator, namely partitioned by is useful to express zoom-out operations. More precisely, S partitioned by G2, S being at granularity G1 (G1 $\prec$ G2), makes a partition of S according to granularity G2. The result is a sequence, at granularity G2, of sequences at granularity G1, such that the value of the main sequence at any instant I (at granularity G2) is the restriction of S to the interval expand(I, G1). This is illustrated in figure 3.

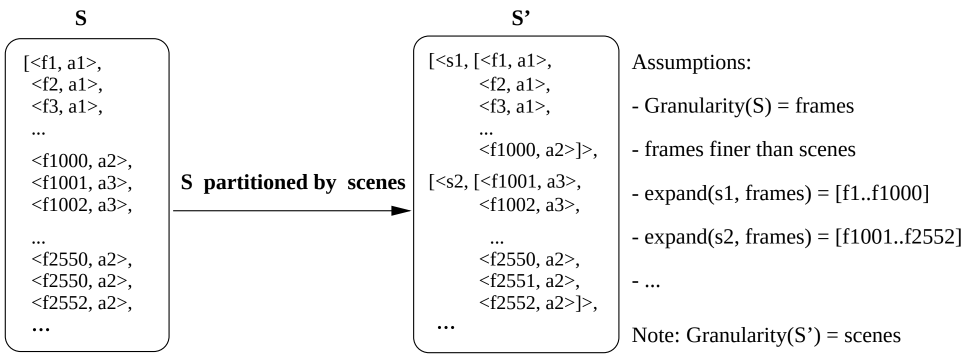


Figure 3. Sequence partitioning operator. Instants at the granularity called frames (respectively scenes) are denoted as f1, f2, ... (resp. s1, s2, ...). The condition frames $\prec$ scenes should hold.

To reason about successive values of sequences, four "splitting" operators are provided, namely afterfirst, beforefirst, afterlast and beforelast. S as x afterfirst P(x) yields the tail of S starting at the first instant at which the value of S satisfies predicate P, or the empty sequence if such instant does not exist. S as x beforefirst P(x), on the other hand, restricts S to the instants preceding the first instant at which the value of S satisfies P, or S if no such instant exists. For any sequence S and any predicate P, S is equal to the union of S as x beforefirst P(x) and S as x afterfirst P(x) (which are disjoint). Similar remarks apply to afterlast and beforelast.

Figure 4 provides a complete formalization of the above operators. the formalization of each language operator is made up of three parts:

- Syntax: in a BNF-like notation with terminal symbols typeset in **boldface**.

- Typing: a typing rule for the operator using the notation $\frac{\text{premise}}{\text{implication}}$. The notation q::t means that the query q has type t, while q[x::t']::t means that query q has type t assuming that variable x has type t'.

- Semantics: the semantics of the operator in terms of a first-order calculus-like expression defining the graph of the resulting sequence (which is a set of pairs $\langle$instant, value$\rangle$) in terms of that of the argument(s). The semantics of each

Syntax: <query> ::= **map** <query> **on** <query> **as** <identifier>

Typing: $\dfrac{q_2 :: \mathit{Sequence}{<}T1{>},\ q_1[x :: T1] :: T2}{(\textsf{map } q_1 \textsf{ on } q_2 \textsf{ as x}) :: \mathit{Sequence}{<}T2{>}}$

Semantics: $[\![\textsf{map } q_1 \textsf{ on } q_2 \textsf{ as x}]\!]_\nu = \{ \langle I, [\![q_1]\!]_{\nu[x\leftarrow v]} \rangle \mid \langle I, v\rangle \in [\![q_2]\!]_\nu \}$

Syntax: <query> ::= <query> **during** <query>

Typing: $\dfrac{q_1 :: \mathit{Sequence}{<}T{>},\ q_2 :: \mathit{ISet}}{(q_1 \textsf{ during } q_2) :: \mathit{Sequence}{<}T{>}}$

Semantics: $[\![q_1 \textsf{ during } q_2]\!]_\nu = \{\langle I,v\rangle \mid \langle I,v\rangle \in [\![q_1]\!]_\nu \wedge I \in [\![q_2]\!]_\nu\}$

Syntax: <query> ::= **map** <query> **on** <query> **as** <identifier> **when** <query>

Typing: $\dfrac{q_2 :: \mathit{Sequence}{<}T1{>},\ q_1[x :: T1] :: T2,\ q_3[x :: T1] :: \mathit{boolean}}{(\textsf{map } q_1 \textsf{ on } q_2 \textsf{ as x when } q_3) :: \mathit{Sequence}{<}T2{>}}$

Semantics: $[\![\textsf{map } q_1 \textsf{ on } q_2 \textsf{ as x when } q_3]\!]_\nu =$
$\{\langle I, [\![q_1]\!]_{\nu[x\leftarrow v]}\rangle \mid \langle I,v\rangle \in [\![q_2]\!]_\nu \wedge [\![q_3]\!]_{\nu[x\leftarrow v]} \}$

Syntax: <query> ::= **join (**<identifier> **:** <query> {**,** <identifier> **:** <query> } **)**

Typing: $\dfrac{q_1 :: \mathit{Sequence}{<}T1{>},\ q_2 :: \mathit{Sequence}{<}T2{>},\ \ldots q_n :: \mathit{Sequence}{<}Tn{>}}{\textsf{join } (l_1 : q_1, l_2 : q_2, \ldots, l_n : q_n) :: \mathit{Sequence}{<}\langle l_1 : T1, l_2 : T2, \ldots l_n : Tn\rangle{>}}$

Semantics: $[\![\textsf{join}(l_1\text{: } q_1,\ l_2\text{: } q_2, \ldots l_n\text{: } q_n)]\!]_\nu =$
$\{ \langle I, \langle l_1\text{: } v_1, l_2\text{: } v_2, \ldots, l_n\text{: } v_n\rangle\rangle \mid \langle I, v_1\rangle \in [\![q_1]\!]_\nu \wedge$
$\langle I, v_2\rangle \in [\![q_2]\!]_\nu \wedge \ldots \langle I, v_n\rangle \in [\![q_n]\!]_\nu \}$

Syntax: <query> ::= <query> **partitioned by** <query>

Typing: $\dfrac{q_1 :: \mathit{Sequence}{<}T{>},\ q_2 :: \mathit{Granularity}}{(q_1 \textsf{ partitioned by } q_2) :: \mathit{Sequence}{<}\mathit{Sequence}{<}T{>}{>}}$

Semantics: $[\![q_1 \textsf{ partitioned by } q_2]\!]_\nu =$
$\{ \langle I, VS \rangle \mid \exists I' \in \textsf{domain}([\![q_1]\!]_\nu)\ (I = \textsf{approx}(I', [\![q_2]\!]_\nu) \wedge$
$VS = [\![q_1]\!]_\nu \textsf{ during expand}(I, \textsf{granularity}([\![q_1]\!]_\nu))) \}$
/ precondition: Granularity($[\![q_1]\!]_\nu$) $\prec$ $[\![q_2]\!]_\nu$ */*

Syntax: <query> ::= <query> **as** <identifier> **afterfirst** <query>

Typing: $\dfrac{q_1 :: \mathit{Sequence}{<}T{>},\ q_2[x :: T] :: \mathit{boolean}}{(q_1 \textsf{ as x afterfirst } q_2) :: \mathit{Sequence}{<}T{>}}$

Semantics: $[\![q_1 \textsf{ as x afterfirst } q_2]\!]_\nu =$
$\{ \langle I, v\rangle \mid \langle I, v\rangle \in [\![q_1]\!]_\nu \wedge \exists \langle I', v' \rangle \in [\![q_1]\!]_\nu\ ([\![q_2]\!]_{\nu[x\leftarrow v']} \wedge I \geq I') \}$

Syntax: <query> ::= <query> **as** <identifier> **beforefirst** <query>

Typing: $\dfrac{q_1 :: \mathit{Sequence}{<}T{>},\ q_2[x :: T] :: \mathit{boolean}}{(q_1 \textsf{ as x beforefirst } q_2) :: \mathit{Sequence}{<}T{>}}$

Semantics: $[\![q_1 \textsf{ as x beforefirst } q_2]\!]_\nu =$
$\{ \langle I, v\rangle \mid \langle I, v\rangle \in [\![q_1]\!]_\nu \wedge \neg \exists \langle I', v' \rangle \in [\![q_1]\!]_\nu\ ([\![q_2]\!]_{\nu[x\leftarrow v']} \wedge I \geq I') \}$

Note: Operators beforelast and afterlast are defined symmetrically to afterfirst and beforefirst.

Figure 4. Syntax and semantics of VIDEOQL's algebraic operators on sequences

operator is parametrized by a valuation function which fixes the values of free symbols in the corresponding query. For instance, $[\![\mathsf{q}]\!]_\nu$ denotes the result of evaluating q under valuation ν. Finally, the notation $\nu[\mathsf{x} \leftarrow \mathsf{v}]$ denotes the valuation equal to ν except that it assigns value v to symbol x.

In addition, the following macros are introduced as a syntactical sugar; they combine up the map and the partitioned by operators and introduce a having operator on partitioned histories.

```
map q1 on q2 partitioned by q3 ≡ map q1 on (q2 partitioned by q3) as partition
map q1 on q2 as x when q3 partitioned by q4 ≡
  map q1 on ((map x on q2 as x when q3) partitioned by q4) as partition
map q1 on q2 partitioned by q3 having q4 ≡
  map q1 on ((q2 partitioned by q3) as partition when q4) as partition
map q1 on q2 as x when q3 partitioned by q4 having q5 ≡
  map q1 on (((map x on q2 as x when q3) partitioned by q4) as partition when q5) as partition
```

Notice that in all these macro definitions, the variable partition is used in the map and having clauses to refer to the sub-sequences generated by the partitioning phase. This convention is analogue to that adopted in OQL's "group by" clause [4].

4.3. QUERIES

To illustrate the ease of use and the expressive power of VIDEOQL, we consider a database managing a collection of movies. Each movie is identified by its title. The frames of each movie are annotated with the names of the actors appearing on them, while the scenes are annotated with the location where they take place. The following extended ODL statements describe the class Movie, which implements the Video interface described in section 3.2.

```
class Movie : Video (extent TheMovies, key title) {
  attribute string title;
  attribute Sequence<Location> location;
  attribute Sequence<Set<string>> actors;
}
```

The first two queries that we consider, involve the restriction operators on sequences. In the first one, a sequence of annotations attached to the frames of a video is restricted to a given interval. The annotations appearing in this portion of the sequence are then extracted and furtherly processed using standard OQL operators. In the second query, a sequence of annotations is restricted based on a condition on the values that it takes, and the result is aggregated using the Duration selector on sequences (see section 4.2).

Q1 : **Restriction (based on the domain)**

Retrieve the names of actors that are present at least once during the first 20 seconds of the movie "Hopes" (assuming a constant presentation rate of 30 frames/second).

```
flatten(select anActorSet
        from TheMovies as F,
             range(F.actors during [ 0 @ F.frames | (20 * 30) # F.frames ]) as anActorSet
        where F.title = “Hopes”)
        /* Operators @, # and | were introduced in section 4.1 */
```

Q2 : **Restriction (based on the range) and sequence aggregation**
In which movies, is John present during more that 15 minutes (assuming the same presentation rate as above).

```
select F from TheMovies as F
where duration(F.actors as anActorSet when “John” in anActorSet)
        >= (15 * 60 * 30) # F.frames
```

In both of these queries, we assumed a constant presentation rate when converting a duration expressed in seconds (or some other finer granularity) into a number of frames. However, virtual videos may involve several raw videos possibly recorded (and therefore presented) under different frame rates. In these situations, the correspondence between durations expressed in seconds, and frame numbers, becomes far more complex[5]. To the best of our knowledge, this problem has not been addressed by any of the existing video models. Indeed, the models which offer conversion functions between durations expressed in terms of seconds, and frame numbers, assume a constant presentation rate (see for instance [8]). We believe that studying this issue is an interesting and non-trivial perspective to our work.

One of the most important characteristics of videos lies on their sequenced structure. Therefore, queries dealing with this structure are expected to be frequent. To some extent, these queries may be expressed using the splitting operators on sequences as in the following query.

Q3 : **Succession: binary sequence splitting**
In which movies does John appear for the first time before Tom does so.

```
select F from TheMovies as F
where exists actorSet2 in range(F.actors as actorSet1 beforefirst “Tom” in actorSet1) :
                              “John” in actorSet2
/* i.e. there is an actor set containing “John”, within the range of the original sequence
restricted to the instants before the first time when “Tom” appears. */
```

However, these splitting operators (even combined with the other operators of the model) are not sufficient to express all queries involving succession. In particular, they cannot express the query *“For each actor appearing in a given movie, how many times does he/she appears in this movie”*, where *“an actor appears”* means that she/he is present in one frame, without being present in the previous one. The same is true of queries involving maximal intervals during which a fact is uninterruptedly

[5] If an analogue storage support is used, durations in seconds should first be converted into time-codes before being converted into frame numbers, therefore rendering the conversion procedure still more complex.

true, e.g. *"Who is present uninterruptedly in Freedom during a longest interval of time than Tom does"*.

Two approaches are possible to cope with these limitations. The first is to introduce new operators addressing these kinds of queries, and to study the expressive power of the resulting language. The alternative is to define a selector on the Sequence ADT which retrieves an interval-based representation of a sequence, i.e. a collection of pairs ⟨Interval, Object⟩. For instance, given the sequence whose graph is: {⟨1, v1⟩, ⟨2, v1⟩, ⟨4, v1⟩, ⟨5, v2⟩, ⟨6, v2⟩, ⟨7, v2⟩, ⟨8, v3⟩, ⟨9, v1⟩, ⟨10, v1⟩ } , its interval-based representation is: { ⟨[1..2], v1⟩, ⟨[4..4], v1⟩, ⟨[5..7], v2⟩, ⟨[8..8], v3⟩, ⟨[9..10], v1⟩ }. Once a sequence represented in this form, standard OQL operators on collections combined with an adequate set of operators on intervals, could be used to express the above two queries. The expressive power of the language regarding sequences would then exactly match the expressive power of OQL for handling collections of intervals, which, modulo aggregation and grouping, is essentially that of temporal relational complete languages as defined in [6]. Exploring and comparing the above alternatives is an interesting perspective to the work developed in this paper.

Since in our model, annotations may be independently attached to each level of video structuration, it is necessary to provide a mechanism for switching between these levels. The partitioned by operator provides this mechanism in our OQL extension.

Q4 : **Granularity change: sequence partitioning**
Retrieve the scenes of "Freedom" in which John is in at least half of the frames of the scene.

```
select domain(map partition
              on F.actors
              partitioned by F.scenes
              having duration(partition as anActorSet when "John" in anActorSet)
                     >= duration(partition) / 2)
from TheMovies as F where F.title = "Freedom"
```

5. Video composition

5.1. Building elementary virtual videos

The duality between raw and virtual videos introduced in section 3.3 enforces a separation between the raw data of a video and its meta-data (i.e. its structuration and annotations). The former are shared among several virtual videos, thereby avoiding their expensive duplication, while the latter are mainly duplicated, so that the user can edit them.

A question that remains to be answered is: how are virtual videos created?

This is performed in three steps. First, "elementary" virtual videos are created by extracting segments of raw videos. This step is carried out through a constructor of the class Video[6] called create, which takes as parameter a raw video and two integers denoting the beginning and the end of the extracted segment. Once an elementary virtual video created, it may be structured and annotated using the constructors and update operators on granularities and sequences (not described in this paper). Lastly, annotated elementary virtual videos are combined into complex ones through the set of algebraic composition operators presented below.

5.2. Virtual video composition operators

We consider five video composition operators: extraction, concatenation, intersection, union and difference. The specifications of all these operators are reflective, in the sense that the exact type of the generated video depends on the exact type of the effective parameters[7]. For instance, the composition of two instances of class Movie (see section 4.3), yields an instance of class Movie, while the composition of an instance of class Movie and one of a class implementing the interface TennisVideo (see section 2.2), yields an instance of class Video. In the former case, the structuration and the annotations of the parameters are propagated to the result, while in the latter case they are not. More generally, the result of any composition operator is an instance of the smallest common ancestor of the classes of the effective parameters, and, modulo their type compatibility, the structuration levels and annotation sequences appearing in the composed videos are propagated to the resulting one.

This latter feature distinguishes our proposal from that of [33, 14, 20], which define composition operators similar to ours, but do not consider the issue of “composing” the video meta-data in the same way that they compose the video raw data.

5.2.0.8. *Extraction.* This operator takes as parameters a virtual video V and an instant set IS, defined at the lowest granularity of V (i.e. V.frames). It creates a new video by restricting V to the instants in IS, as shown in figure 5.

The derivation of the granularities of the resulting video is carried out using the operator Restrict on granularities, which is formally defined in appendix A. Intuitively, Restrict(G, IS) derives a new granularity by restricting granularity G to the granules referenced in IS.

The derivation of the annotations, on the other hand, is performed in two steps. First each of the sequences of annotations of the argument video are restricted through the operator during on sequences. Then, the resulting sequences are transformed into equivalent sequences over the new time-line, using the operator Compact defined in appendix A. Intuitively, this operator maps a sequence with a non-convex domain, into one with a convex domain. For example,
$\text{Compact}(\{\langle 3, v1\rangle, \langle 5, v2\rangle, \langle 6, v2\rangle, \langle 9, v1\rangle\}) = \{\langle 1, v1\rangle, \langle 2, v2\rangle, \langle 3, v2\rangle, \langle 4, v1\rangle\}$

The full definition of the operator follows.

[6] The class Video provided by the model is an implementation of the interface Video described in section 3.

[7] Reflection is supported in ODMG through a meta-object model.

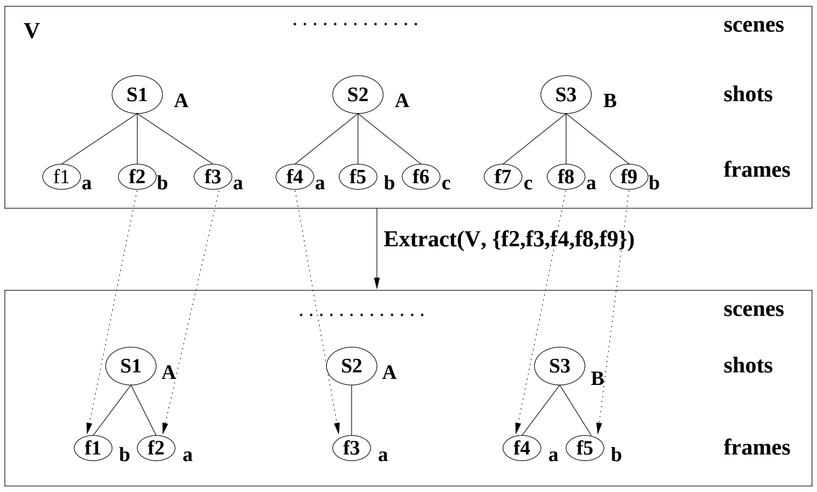


Figure 5. The video extraction operator. Characters a, b and c denote annotations attached to the frames, while A, B and C are annotations attached to the scenes.

Extract: Video, ISet $\rightarrow$ Video
V_r = Extract(V_o, IS) iff:
  V_r.VTL = [1..Size(Duration(IS))] */* a time-line is seen as an interval of integers */*
  V_r.GP = Restrict(V_o.GP, IS) for all properties GP of type Granularity
  V_r.SP = Compact(V_o.SP during IS) for each property SP whose value is a sequence defined over granularity V_r.frames
  Compact(V_o.SP during approx(IS, Granularity(V_o.SP))[8] for each property SP whose value is a sequence defined over a granularity other than V_r.frames

5.2.0.9. *Concatenation $\oplus_V$.* This operator takes two videos as parameters. The resulting video is composed of all frames belonging to the first video, followed by all frames in the second one.

The definition of this operator involves two auxiliary operators: translate and shift, both described in appendix A.

The operator translate on granularities, corresponds to a translation of the origin of a time-line. Intuitively, it shifts the time-marks contained in the granules of a granularity, by the number given as parameter. E.g.
translate({ [1..2], [3..6], [7..7] }, 5) = { [6..7], [8..11], [12..12] }.

The operator shift on sequences, forwardly "shifts" the domain of a sequence by the number of granules given as a parameter. For instance,
shift({$\langle$3, v1$\rangle$, $\langle$5, v2$\rangle$, $\langle$6, v2$\rangle$, $\langle$9, v1$\rangle$}, 5) = {$\langle$8, v1$\rangle$, $\langle$10, v2$\rangle$, $\langle$11, v2$\rangle$, $\langle$14, v1$\rangle$}

_ $\oplus_V$ _ : Video, Video $\rightarrow$ Video
$V_r = V_1 \oplus_V V_2$ iff:

[8] We recall that the conversion operator approx is defined in section 4.1.

V_r.VTL = [1 .. length(V_1.VTL) + length(V_2.VTL)]
V_r.GP = V_1.GP $\cup$ translate(V_2.GP, length(V_1.GP)) for all properties GP of type Granularity
V_r.SP $=$ V_1.SP $\cup$ shift(V_2.SP, length(V_1.SP)) for all properties SP of type Sequence
V_r.rawData = V_1.rawData $\cup$ shift(V_2.rawData, length(V_1.VTL))

5.2.0.10. *Intersection.* This operation creates a new video where only common footage of both videos is captured.

_ $\cap_V$ _ : Video, Video $\rightarrow$ Video
$V_1 \cap_V V_2 =$ Extract(V_1, Domain(V_1.rawData as x when x in range(V_2.rawData)))

5.2.0.11. *Difference.* This operation creates a new video which is composed of the frames of the first operand without the common frames of the second operand.

_ $-_V$ _ : Video, Video $\rightarrow$ Video
$V_1 -_V V_2 =$ Extract(V_1, Domain(V_1.frames as x when not (x in range(V_2.frames))))

5.2.0.12. *Union* This operation has the same features as the concatenation one, but common footage is not repeated. Formally:

_ $\cup_V$ _ : Video, Video $\rightarrow$ Video
$V_1 \cup_V V_2 = V_1 \oplus_V (V_2 -_V V_1)$

Figure 6 illustrates the meaning of the intersection and the union operators. This figure puts forward the raw video data sharing resulting from virtual video composition. The structuration and annotations of the videos are not shown in this figure.

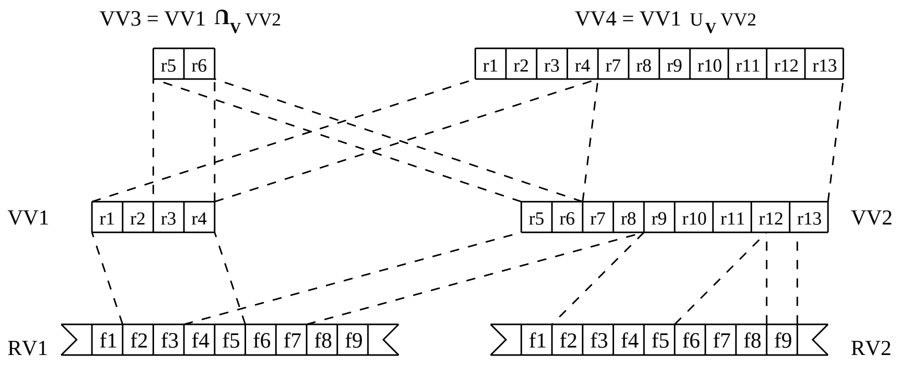

VV3 = VV1 ∩V VV2
VV4 = VV1 ∪V VV2
r5 r6
r1 r2 r3 r4 r7 r8 r9 r10 r11 r12 r13
VV1
r1 r2 r3 r4
r5 r6 r7 r8 r9 r10 r11 r12 r13
VV2
RV1
f1 f2 f3 f4 f5 f6 f7 f8 f9
f1 f2 f3 f4 f5 f6 f7 f8 f9
RV2

Figure 6. Virtual video composition and raw video data sharing: RV1 and RV2 denote raw videos, VV1 through VV4 denote virtual videos, f1, f2, etc. denote raw frames, and r1, r2, etc. denote raw frame references.

5.3. Video composition and querying

Embedded into the query language, the composition operators allow to build new videos by retrieving and combining segments of existing ones. Typically, the retrieval phase of this process involves a “select/from/where” query expression (which returns a collection of videos). To combine all the videos of the resulting collection into a single one, it is necessary to extend the composition operators so as to apply to lists of videos. The following operator generalizes the binary $\oplus_V$ operator to deal with lists of videos.

ConcatAll: [Video] $\rightarrow$ Video
ConcatAll([V_1, V_2, ... V_n]) = $V_1 \oplus_V V_2 \oplus_V \ldots \oplus_V V_n$

Operators IntersectAll and UnionAll are defined similarly.

Using these operators, most of the queries presented in section 4.3 can the be easily rewritten so as to generate a new video from the videos or video segments that they retrieve.

Q5 : **Video composition**
Create a new movie by concatenating all the movies where John appears for the first time before Tom does so.

```
ConcatAll(
select F from TheFilms as F
where exists actorSet2 in range(F.actors as actorSet1 beforefirst “Tom” in actorSet1) :
                              “John” in actorSet2
order by F.title)
```

6. Implementation

The video model presented has been implemented as a prototype on top of the O_2 DBMS. In this section, we present the overall architecture of the prototype, and detail its components.

6.1. Overall architecture

The overall architecture of the prototype is shown in figure 7. It is composed of seven modules. Four of them, namely the schema definition language preprocessor, the query language preprocessor, the video editing user interface and the video player, operate as front-ends to the DBMS. The other three modules (the schema manager, the video meta-data manager and the raw video repository) directly operate within the DBMS.

The schema definition and the query languages preprocessors were developed in the context of the TEMPOS temporal database system [11]. In the current state of the implementation, we have not yet considered any optimization issue: the query

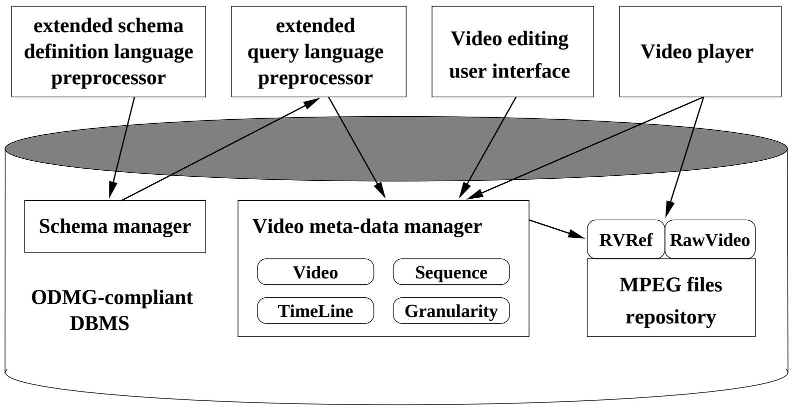


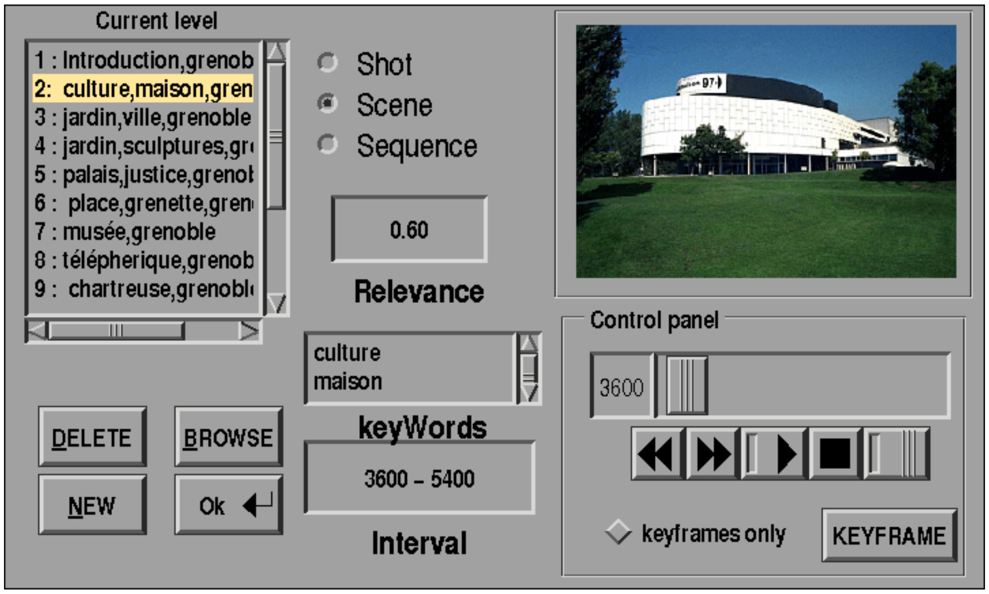
Figure 7. Overall prototype architecture

language preprocessor simply translates queries formulated in VIDEOQL, into OQL queries containing invocations to the methods implementing the operators of the Sequence ADT.

The video editing user interface and the video player (which we detail in section 6.2), are also adapted from previous work done by the authors in the context of the VSTORM video database model [20]. Together, these two modules provide an interactive environment for editing, browsing and playing videos stored in the database.

The screen shot in figure 8 illustrates the edition of a video sequence using the video editing interface. The box entitled "current level" lists all the existing scenes of the video sequence. When a scene is selected, all the data attached to it are displayed in the rest of the canvas, so that the user may browse or edit them. A new scene can be added by clicking on the "NEW" button. The shots composing the new scene are then entered either through the "Control panel" or directly by giving the cardinal numbers of the first and last shots of the scene. The "Browse" button is used to link objects with this scene (i.e. to annotate the scene). The "KEYFRAME" button allows the user to select a representative frame of the scene (i.e. a key-frame). The set of key-frames of a given structuration level may subsequently be viewed by selecting the "keyframes only" button. This functionality is useful for video browsing. Finally, the "relevance factor" entry is intended to measure the current scene's relevance with respect to all other scenes in the current video sequence. This factor is subsequently used for summary extraction as discussed in [21].

The schema manager has two functionalities. First, it acts as a mediator between the external preprocessors and the DBMS catalog. This mediation ensures some independence between the preprocessors' design and the underlying DBMS. Indeed, the ODMG standard does not define an interface to update the DBMS catalog, although it defines an interface to access it (which, by the way, is currently not implemented by the object DBMS vendors). Second, this module keeps track of the instantiations of the parametric class Sequence used within a given database. This is necessary to



1 : Introduction,grenob
2: culture,maison,gren
3 : jardin,ville,grenoble
4 : jardin,sculptures,gr
5 : palais,justice,grenol
6 : place,grenette,gren
7 : musée,grenoble
8 : télépherique,grenob
9 : chartreuse,grenobl

Shot [✓ Scene] Sequence

Relevance: 0.60

keyWords: culture maison

Interval: 3600 - 5400

Figure 8. Screen shot from the video editing user interface

simulate parametric polymorphism on top of the ODMG object model as discussed in section 6.3.

The video meta-data manager is a library of classes implementing the interfaces for modeling video structuration and annotation that we described in sections 2 and 3. In particular, this module provides implementations of the interfaces TimeLine, Granularity, Sequence and Video.

Finally, the raw video repository contains the compressed streams containing the images and sound samples of each physical video. These are stored using the “bytes” datatype provided by the O_2 DBMS.

Although performance was not a major concern during the implementation of the prototype, the response time of most of the operators that it offers are acceptable. Still, some performance improvements could probably be achieved if we integrated the numerous implementation techniques developed both in the settings of temporal databases [29, 10] and video databases [9]. Moreover, we have not studied the performances of the prototype when the data are distributed among several servers, since O_2 is based on a single server architecture.

6.2. Implementation notes related to the video player

The video player is implemented as an extension of an MPEG player developed at the University of California[9]. Three features have been integrated into the original MPEG player:

- The original video player was designed to read its input from a single physical file, while virtual videos may stem from several raw videos stored in separate files. To avoid generating the entire MPEG data corresponding to a virtual video, and dumping it into a physical file before starting its presentation, we modified the original player so as to accept reading data from a dynamically generated file (i.e. a pipe). Several bufferization problems were tackled at this point.
- A virtual video may stem from several physical videos having different window sizes. The player was therefore modified so as to dynamically resize the video presentation window in such situations. An alternative approach that we considered, is to fix a window size for the whole virtual video presentation, and to dynamically rescale each involved physical video so as to fit on it. However, we did not implement this latter solution since it involves complex modifications into the video player.
- Similarly, several physical videos involved in a single virtual video may be recorded with different frame rates. The original video player was therefore modified so as to accept changing the frame displaying rate on the fly. Otherwise, the user would perceive some portions of the virtual video as going faster or slower than the others.

In the future, we plan to decompose the video player into two subcomponents: one which decompresses video data, and another that displays it under some presentation parameters (e.g. window size and speed). In this way, it will be quite straightforward to consider other compression formats than MPEG (e.g. M-JPEG, AVI and H.261), and even to play virtual videos composed of physical videos stored under different formats. To this end, we believe that it would be useful to integrate into our implementation, the ideas developed in some playback control systems such as Windows Media Player 7 [22].

6.3. Implementation notes related to the Sequence ADT

Perhaps, the major problems that we faced during the design of the prototype, were those related to the lack of parametric classes in the O_2 model (which is true of the ODMG object model as well). Indeed, the Sequence ADT could be naturally mapped into a parametric class.

[9] Downloadable at ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/mpeg. This video player only plays images.

A first solution that we envisaged, is to generate a class for each kind of sequence involved in an application. In our example database, this implies generating a class for sequences of Location, another for sequences of Set<string>, etc. We believe that in realistic situations, these would rapidly lead to an undesirable proliferation of classes. In addition, some operators, such as the sequence joins, cannot be satisfactorily implemented using this approach, since the type of the resulting sequence intrinsically depends upon that of the argument sequences.

Instead, we decided to partially simulate parametric classes by exploiting the preprocessors included in the architecture. In this approach, a single non-parametric class Sequence, corresponding to sequences whose values are of type Object (the top of the ODMG's class hierarchy), is first implemented. Then, during schema definition, each sequence-valued attribute is declared as being of type Sequence by the data definition language preprocessor, while its exact type specification is stored in the schema manager. Since the schema manager is accessed by the VIDEOQL preprocessor, this latter knows the exact type of the sequences involved in a query. With this knowledge, the VIDEOQL preprocessor adds explicit downcastings in the translated query expression, whenever a value of a sequence is extracted. In this way, the user of VIDEOQL manipulates sequences as if they were parametrically typed.

The above solution has several drawbacks. First, adding explicit downcastings in the translated queries introduces a burden during query evaluation, since the OQL interpreter performs a dynamic type checking whenever an object is downcasted. Second and foremost, the above solution does not take into account that the database objects are not only accessible through the query language, but also, through any of the programming language bindings (e.g. the C++ and Java bindings defined in the ODMG standard). An outcome of this limitation is that, in the current implementation of the prototype, the application programmer must introduce downcastings and explicit dynamic type checkings when manipulating sequences.

7. Related works

There is a wealth of works dealing with semi-structured formats for representing semantical video contents (i.e. annotations). The results of these works are currently being exploited by ongoing standardization efforts, such as MPEG-7 [17] and the Dublin core meta-data proposal [32]. In both of these approaches, annotations are represented as "segment descriptors", i.e. hierarchically structured entities attached to a particular segment of a document, or possibly to an entire document. MPEG-7 is intended to be flexible, in the sense that user-defined annotation types are allowed in addition to the ones provided by the standard.

Our proposal is complementary to the above ones, since it is not intended to define a format for video metadata representation, but rather a data model and a set of tools for browsing, editing, composing and querying videos using the functionalities of an object DBMS. Stated otherwise, the objective of our proposal is to provide a set of abstractions for manipulating video metadata regardless of its internal representation, while MPEG-7 aims to provide a standard format for this internal representation.

The idea of using DBMS functionalities to store, browse, and query video contents is not novel; it has been applied in many commercial tools and research data model proposals [9]. In the sequel, we compare our contributions with respect to these two families of related works.

7.1. Commercial tools

Several commercial DBMS such as Oracle [25] and Informix [16] (in partnership with MEDIAstra) provide solutions both for storage and manipulation of video data. Thanks to these extensions, the DBMS is capable of efficiently storing MPEG-2 files, by implementing adequate clustering strategies and avoiding some kinds of replication. Additionally, the Video Foundation Datablade developed by MEDIAstra, allows the application developer to index MPEG videos and to automatically segment a video into scenes.

In addition to these DBMS extensions, there exist many ad hoc tools for end-user digital video manipulation. Since this software market evolves rapidly, it is difficult to get an exact picture of the functionalities offered by these systems. Nonetheless, some representative systems such as Ulead MediaStudio Pro 6.0 [30] and Adobe Premiere 5.1 [3], provide an idea of the current state of the art.

MediaStudio Pro 6.0 is an application consisting of several more or less independent modules. Each of these modules addresses a specific functionality (e.g. video editing, audio editing, video painting and captioning). Adobe Premiere is a tool for professional digital video editing. It proposes various interfaces to produce broadcast-quality movies for video, film, multimedia, and the Web. Premiere 5 features a new editing interface and QuickTime 3 support. Moreover, Premiere 5 handles a wide range of audio formats. Both of the above tools communicate with a digital video device through built-in or pre-installed IEEE-1394 bus protocols [28].

All the above DBMS extensions and ad hoc tools support simple video editing operations. However, their focus is rather on efficiently capturing and managing raw video data, than on providing high-level abstractions for video metadata browsing, querying and composition. In particular, the issue of orthogonally modeling video structuration and annotation addressed by our proposal, is far ahead from the abstractions currently supported by the above systems.

7.2. Research proposals

With respect to previous research proposals in the area of video databases, the main innovation of our approach lies on the orthogonality with which the different aspects of video modeling are tackled. For instance, in our proposal annotations may be independently attached to each level of video structuration, whereas in most of the existing video data models (e.g. AVIS [1] and CVOT [19]), annotations may only be attached to the frames. Moreover, in most existing approaches, the structure of the annotations is restricted (generally to full-text keywords), while in our proposal, any object may be used to annotate any video segment.

The fact that a single video may be annotated by arbitrarily complex and heterogeneous objects, raises the problem of querying it without precisely knowing its underlying schema. We have studied in a previous work [20] how heterogeneous annotations may be queried through generalized path expressions [5]. We strongly believe that the combination of an open annotation schema as the one that we propose, with the use of generalized path expressions for querying, provides a good tradeoff between the flexibility of full-text annotations [18], and the semantical precision of typed ones.

[14] is perhaps one of the closest works to ours. This paper describes a framework for modeling video and audio through the notion of *media stream*. Annotations are attached to intervals over these streams, and temporal queries are formulated by using comparison operators on the lower and upper bounds of these intervals. Unfortunately, the issue of defining high-level temporal query operators is not addressed by this work. In addition, the proposed query language uses an ad hoc syntax, and is not integrated into a general-purpose query language. This remark also applies to other related proposals such as VideoSQL [24] and VIQS [15]. In contrast, the query language that we propose is a fully compliant extension of OQL, therefore allowing to formulate both video and "conventional" queries in an unified framework.

The idea of using temporal database concepts to model video data has been explored in [19]. In this work, the authors suggest to handle videos using histories as defined in the TIGUKAT object model [13]. Queries on videos are then expressed using high-level operators on histories. Unfortunately, some important features of video annotations are not taken into account. For instance, it is not possible to attach different kinds of annotations to different levels of video structuration as discussed above. Furthermore, the authors assume that video annotations are located with respect to a global time-line (i.e. each annotation is conceptually attached to an anchored date such as "14/8/1999 at 5:00 a.m."), so that the modeling issues arising from the unanchored nature of time in videos are skirted.

In some respect, the work reported here is close to that of [26], which proposes an ADT-driven model for sequences and an associated query language. However, this latter work focuses on discretely-varying data such as time-series, and the authors do not discuss how it may be extended to deal with stepwisely varying data such as video annotations.

Most of the video composition operators that we considered in this paper are inspired from those studied in [33]. These operators also appear, with slightly different semantics, in other related proposals such as [24] and [14]. However, none of these works discusses how the logical structuration of the composed videos is reflected in the resulting one, nor how these composition operators can be integrated into a query language.

Finally, other related works include the numerous data model proposals for multimedia presentations (e.g. [2]). Nevertheless, these proposals are not directly relevant to ours, since they do not attempt to model the internal contents of videos, but rather to provide support for building interactive presentations by composition of multimedia documents.

8. Conclusion

We have presented a data model that provides high-level support for storing, browsing, querying and composing videos on top of an object-oriented DBMS. Through a detailed analysis of related works, we have shown that this model features several important novelties that ensure an increased user-friendliness, flexibility and expressive power. Specifically, our video data model has the following advantages with respect to its competitors:

– It is possible to define non-comparable structuration levels over a single video. For instance, a video about a tennis match may be structured into shots one the one side, and into points, games and sets on the other.

– Structuration and annotation are modeled as orthogonal aspects. Concretely, this means that an annotation attached to a scene is not treated in the same way as an annotation attached to every frame of that scene.

– The composition operators output structured and annotated videos, i.e. the structuration and annotations attached to the composed videos are propagated to the resulting one. This feature is achieved using the reflection capabilities of the ODMG type system.

– The sequences of annotations attached to a video may be manipulated without refering to a specific representation of them, whereas in previous proposals, the data manipulation language imposes an interval-based representation.

The soundness and feasibility of the proposed data model and its query language have been validated through a complete formalization and a prototype implementation on top of a commercial object-oriented DBMS.

Since our proposal is based on concepts stemming from Temporal Databases [29, 10], we believe that several techniques developed in this latter setting may be adapted to design an efficient and scalable implementation of it. Validating this claim is a perspective to the work reported in this paper.

There are several other research avenues that we would like to explore:

– **Summary extraction**. In section 6.1, we mentioned the possibility of annotating a video segment with a measure of its relevance with respect to other video segments (i.e. a relevance factor). A practical application of this kind of annotations may be to extract a “condensed” version of a video fitting some given duration constraints. We have started to examine this issue in [21].

– **Database-driven collaborative video edition**, i.e. studying how the distribution, concurrency, and security facilities provided by current DBMS may be exploited to support collaborative edition of structured video documents.

For a long time, videos have been managed as unstructured documents. Recent works, such as the one reported in this paper, show that accurately and formally modeling this media significantly improves the expressive power of the tools used to handle it. This demonstrates that it is not because a media is complex that it should be modeled fuzzyly. On the contrary, it is in such situations that the structuration efforts are of greater concern.

Acknowledgments: thanks to the anonymous reviewers of Multimedia Tools and Applications for their highly relevant comments on a draft of this paper.

References

1. Adali, S., K. S. Candan, S. Chen, K. Erol, and V. Subrahmanian: 1996, 'The Advanced Video Information System: data structures and query processing'. *Multimedia Systems* **4**, 172 – 186.
2. Adiba, M.: 1996, 'STORM : An Object-Oriented, Multimedia DBMS'. In: K.Nowsu (ed.): *Multimedia Database Management Systems*. Kluwer Academic Publishers, Chapt. 3, pp. 47–88.
3. Adobe: 2000, 'Adobe Premiere 5.1'. www.adobe.com/products/premiere.
4. Cattell, R. and D. Barry (eds.): 1997, *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann.
5. Christophides, V., S. Abiteboul, S. Cluet, and M. Scholl: 1994, 'From Structured Documents to Novel Query Facilities'. In: *Proc. of the ACM SIGMOD conference on Management of Data*.
6. Clifford, J., A. Croker, and A. Tuzhilin: 1994, 'On completeness of historical relational query languages'. *ACM Transactions on Database Systems* **19**(1), 64 – 116.
7. Clifford, J. and A. Tuzhilin (eds.): 1995, 'Proc. of the workshop on Recent Advances in Temporal Databases'. Zurich, Switzerland:, Springer Verlag.
8. Dionisio, J. and A. Cárdenas: 1998, 'A unified data model for representing multimedia, time-line and simulation data'. *IEEE Transactions on Knowledge and Data Engineering* **10**(5).
9. Elmagarmid, A., H. Jiang, A. Abdelsalam, A. Joshi, and M. Ahmed: 1997, *Video Database Systems: Issues, Products and Applications*. Kluwer Academic Publisher.
10. Etzion, O., S. Jajodia, and S. Sripada (eds.): 1998, *Temporal Databases: Research and Practice*. Springer Verlag, LNCS 1399.
11. Fauvet, M.-C., M. Dumas, and P.-C. Scholl: 1999, 'A representation independent temporal extension of ODMG's Object Query Language'. In: *actes des Journées Bases de Données Avancées*. Bordeaux, France.
12. Gandhi, M. and E. Robertson: 1994, 'A data model for audio-video data'. In: *Proc. of the 6th Int. Conference on Management of Data (CISMOD)*. Bangalore (India).
13. Goralwalla, I. A. and M. T. Ozsu: 1993, 'Temporal extensions to a uniform behavioral object model'. In: *proc. of the 12th International Conference on the Entity-Relationship Approach - ER'93, LNCS 823*.
14. Hjelsvold, R., R. Midtstraum, and O. Sandsta: 1995, 'A Temporal Foundation of Video Databases'. in [7].
15. Hwang, E. and V. Subrahmanian: 1995, 'Querying Video Libraries'. Technical report, Univ. of Maryland.
16. Informix Inc.: 2000, 'Informix Video Foundation Datablade'. www.informix.com/datablades.

17. ISO/IEC coding of moving pictures & audio working group: 1998, 'MPEG-7: Context and Objectives'. http://drogo.cselt.stet.it/mpeg/standards/mpeg-7.
18. Jiang, H., D. Montesi, and A. Elmagarmid: 1999, 'Integrated Video and Text for Content-based Access to Video Databases'. *Multimedia Tools and Applications* **9**(3), 227–249.
19. Li, J., I. Goralwalla, M. Ozsu, and D. Szafron: 1997, 'Modeling Video Temporal Relationships in an Object Database Management System'. In: *Proc. of IS&T/SPIE Int. Symposium on Electronic Imaging: Multimedia Computing and Networking*. San Jose, CA (USA), pp. 80 – 91.
20. Lozano, R. and H. Martin: 1998, 'Querying Virtual Videos with Path and Temporal Expressions'. In: *proc. of the ACM Symposium on Applied Computing*. Atlanta, Georgia (USA).
21. Martin, H. and R. Lozano: 2000, 'Dynamic generation of video abstracts using an object-oriented video DBMS'. *International Journal of Networking and Information Systems – Special Issue on Video Data* **2**(8).
22. Microsoft: 2000, 'Windows media player 7'. www.microsoft.com/Windows/windowsmedia.
23. Milner, R., M. Tofte, R. Harper, and D. MacQueen: 1997, *The Definition of Standard ML – Revised*. MIT Press.
24. Oomoto, E. and K. Tanaka: 1993, 'OVID : Design and implementation of a video-object database system'. *IEEE Transactions on Knowledge and Data Engineering* **5**(4).
25. Oracle Inc.: 2000, 'Oracle Video Server'. www.oracle.com/itv/ovs.html.
26. Seshadri, P., M. Livny, and R. Ramakrishnan: 1996, 'The Design and Implementation of a Sequence Database System'. In: *Proc. of the Int. Conference on Very Large Databases (VLDB)*. Bombay, India, pp. 99 – 110.
27. Snodgrass, R., M. Böhlen, C. Jensen, and A. Steiner: 1998, 'Transitioning Temporal Support in TSQL2 to SQL3'. in [10].
28. Steinberg, D. and Y. Birk: 2000, 'An Empirical Analysis of the IEEE-1394 Serial Bus Protocol'. *IEEE Micro* **20**(1).
29. Tansel, A. U., J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodggrass (eds.): 1993, *Temporal Databases*. The Benjamins/Cummings Publishing Company.
30. Ulead Systems: 2000, 'MediaStudio Pro 6.0'. www.ulead.com/msp/features.htm.
31. Wang, X. S., S. Jajodia, and V. Subrahmanian: 1995, 'Temporal modules : an approach toward federated temporal databases'. *Information Systems* **82**.
32. Weibel, S., J. Kunze, C. Lagoze, and M. Wolf: 1998, 'Dublin core metadata for resource discovery'. http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2413.txt.
33. Weiss, R., A. Duda, and D. Gifford: 1995, 'Composition and Search with a Video Algebra'. *IEEE Multimedia* **2**(1), 12 – 25.

Appendix

A. Functions used by the video composition operators

In this appendix, we formally describe the operators on sequences and granularities used to define the composition operators in section 5.

We adopt the following representation conventions: (1) a granularity is represented as a list of disjoint contiguous intervals of integers in ascending order; (2) an instant set is represented as a list of ordered integers.

To manipulate lists, we use a notation similar to that of the functional programming language ML [23].

Restrict: Granularity, ISet $\rightarrow$ Granularity

```
/* Restrict(G, IS) derives a granularity by restricting granularity G to the granules referenced in IS */
/* Granularities and Isets are represented using the conventions above */
fun Restrict(G, IS) = RestrictBis(G, IS, 1)
fun RestrictBis([ ]) = [ ]
    RestrictBis(g :: R, LI, k) =        /* "g" is an interval of integers and "LI" is a list of integers */
              [k .. k + length(g ∩ LI) − 1] :: RestrictBis(R, LI, k + length(g ∩ LI))

Compact: Sequence<T> → Sequence<T>
/* Compact(S) creates a sequence with a domain [1..n] (n being the number of instants in S's
domain), and containing the same succession of values as S */
fun Compact([ ]) = [ ]
    | Compact(⟨i, v⟩ :: R) = ⟨1, v⟩ :: Shift(Compact(R), 1)

Shift: Sequence<T>, integer → Sequence<T>
/* Shift(S, N) is the sequence obtained by forwardly shifting the domain of S by N units. */
fun Shift([ ], n) = [ ]
    | Shift(⟨i, v⟩ :: R, n) = ⟨i + n, v⟩ :: shift(R, n)

Translate: Granularity, integer → Granularity
fun Translate([ ], n) = [ ]
    Translate(g :: R, n) = [g.lowerBound + n, g.upperBound + n] :: Translate(R, n)
```

Marlon Dumas received his PhD from the University of Grenoble in June 2000 for his works on Temporal Databases. He is currently a Postdoctoral Fellow at the Cooperative Information Systems research center, Queensland University of Technology, Australia. His research interests include Temporal Data Management, Object-Oriented Modeling, and their applications to Electronic Commerce.

Rafael Lozano received his PhD from the University of Grenoble in April 2000 for his works on Video Databases. He is currently an Assistant Professor at the Technological Institute of Monterrey, Mexico City Campus. His research interests include Object and XML Databases and their applications to Multimedia Systems Development.

Marie-Christine Fauvet received her PhD in Computer Sciences from the University of Grenoble in 1988 for her works on Version Management for CAD Applications. Since 1988, she has successively been Assistant and Associate Professor at the University of Grenoble. Her current research interests include Schema Evolution in Object Databases, Temporal Databases, Databases for the Web, and Electronic commerce.

Hervé Martin received his PhD in Computer Sciences from the University of Grenoble in 1991 for his works on Consistency and Integrity Constraints in Object Databases. He is currently an Associate Professor at the University Pierre-Mendès France, and conducts several research projects at the LSR-IMAG laboratory within the areas of Multimedia Databases and Web-based Information Systems.

Pierre-Claude Scholl received his PhD in Computer Sciences from the University of Grenoble in 1979 for his works on Programming Methodology. He is currently a Full Professor at the University of Grenoble. His research interests include Temporal Databases and Object-Oriented Models and Languages.