Multimedia in Views

M. Bordegoni

Computer Science/Department of Algorithmics and Architecture

CS-R9263 1992

Multimedia in Views

Monica Bordegoni

CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
Email: Monica.Bordegoni@cwi.nl

Abstract

This work aims at defining and implementing a multi-media model within the TAXATA (Things Are eXactly As They Appear) model used in the Views project, which supplies a consistent and integrated open-architecture application environment. Dynamic objects, having a time dimension, structuring operators, presentation and accessing methods have been integrated into the current Views model. Relationships among the media objects have been defined using temporal relations instead of a time-line approach.

The
V I E W S
System

1991 Mathematics Subject Classification: 68U99, 68N99.

1991 CR Categories: H.5.1, H.5.2, D39, H.5.0.

Keywords and Phrases: multimedia information systems, user interfaces, software engineering, information interfaces and presentation.

Note: This work is performed during a 6 months ERCIM (European Research Consortium in Informatics and Mathematics) fellowship at the Algorithms and Architectures Department of CWI.

1 Introduction

Multi-media information is commonly integrated in current applications. A multi-media application consists of a set of independent displayable pieces of information (text, graphics, animation, video and audio) which have to be integrated, combined, stored and displayed. Applications have to deal with temporal, spatial and content-based aspects of multi-media information: temporal relations among media have to be identified in order to synchronize and compose in time media information; media information requires to be composed in space and displayed in some form, and finally the description of the structure of each media information has to be provided.

This research work aims at defining and implementing a multi-media model within the TAXATA (Things Are eXactly As They Appear) model used in the Views project [1]], which supplies a consistent and integrated open-architecture application environment. According to the Views project philosophy, Views applications environment has to support the implementation of multi-media application where the basic structure of the application itself is already defined in the kernel of Views.

This research aims at defining and implementing:

1. basic types of multi-media information
2. methods and operators to give a structure to pieces of media information
3. methods to present, access and store media information.

Dynamic objects, having a time dimension, structuring operators, presentation and accessing methods have been added to the current Views model.

Multi-media applications require the “concept of time" to synchronize the objects, that is to allow more media to play at the same time. There is a need to define an easy way to describe media synchronization at the application level. Low level synchronisation, dealing with real-time performance, is an implementation issue, connected to media (hw and sw) and out of the interest of this research. Some functions operating on predefined data format (video, text, ...) such as fast forward, rewind, pause, stop, volume up and down, speed (data sample rate) have to be supported into the kernel of the system.

Many authoring systems are based on the concept of synchronization as a simple event time-line. Given a time-line, actions on media are associated with points in time within the time-line. When a media object is moved, all references and links with the other media objects are lost. A time-line concept does not seem sufficient and a more structured approach, so that events are triggered not only by the advancement of the “real time”, is needed. In such a case, actions on media do not only rely on a schedule but also on events on other media. It helps maintaining constraints among the media objects.

This report describes a study of an alternative approach to the time-line. Starting from a taxonomy of temporal relations between two time intervals, a set of possible relations among media objects has been derived. Hierarchical relations and conditions have been used to define a model for structuring media objects within an application.

The second part of the report presents a notation and a language for the description of a media application, defined on the base of the media model. The language has been integrated in the Views environment.

2 Multi-media

A multi-media application consists of a set of multi-media objects. The objects are linked together by means of some relations.

2.1 Media objects

It is possible to identify two kinds of multi-media objects: time-invariant and time-variant [2].

2.1.1 Time-invariant objects

The presentation of these objects is not temporally related to the representation of other objects and they have not an intrinsic concept of time. These objects are:

- text;
- picture (still picture, graphics).

2.1.2 Time-variant objects

The presentation of these objects is temporally related to actions or to other objects, and a synchronization of their presentation is needed. These objects are chunks of information with a time interval associated with:

- animation (single sequence, series of sequences);
- video (single frame, complete scene);
- audio (sound/voice fragment).

2.2 Media length

A time-invariant object becomes time-variant when a length in time is associated with it. For example, it is possible to display a text or a picture for 10 seconds.

Time-variant media has a length associated with. The length can be measured as a length in time (for example, in seconds) or can be measured by a unit of measure, characteristic of the media. For example, we might want to play a chunk of video for 12 seconds or play 30 frames of it, play a sound for 15 seconds or play 30 audio units at a given sampling rate.

2.3 Temporal location of a media within a time period

When a time-variant media M (a video, an audio or an animation) has to be played over a period T, the actual length of the chunk M may be shorter that the time value T. In this case it is possible to identify some other styles of playing the object within the interval of time. They are shown in Figure 1.

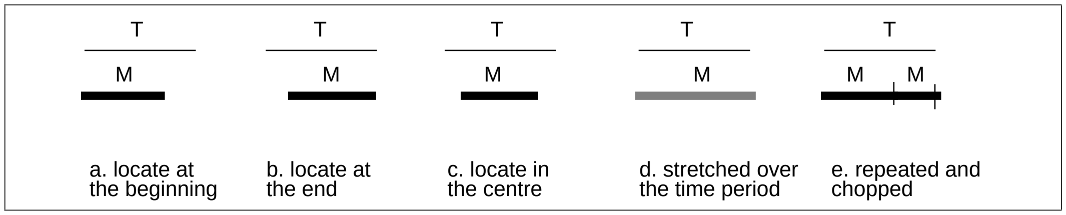


Figure 1. Styles of playing an object within a period T.

3 Temporal relations

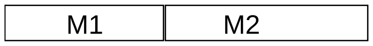
A taxonomy of temporal relations between two intervals of time has been defined in [3]. As a media is performed over a period of time, it is possible to use the taxonomy to identify temporal relations between two media.

Relations are subdivided in *sequential* (where two media are played in sequence, without overlapping), and *parallel* (where two media are played with an overlap of their time intervals).

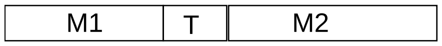
The following picture shows sequential and parallel relations between two generic media M1 and M2, where T is an arbitrary period of time.

1. *Sequential*

1.1. Meets

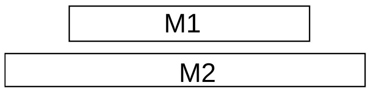


1.2. Before

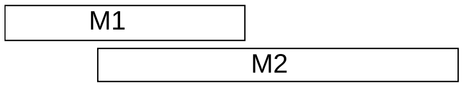


2. *Parallel*

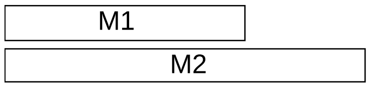
2.1. During



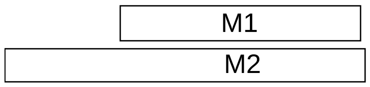
2.2. Overlaps



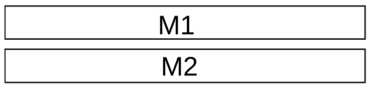
2.3. Starts



2.4. Ends



2.5. Equal



3.1 Startpoint and endpoint

Starting from the taxonomy, it is possible to analyze the identified temporal relations between two media considering their *Startpoint* and *Endpoint*.

Startpoint is the point in time when a media starts playing, Endpoint the point when a media stops playing. Both can:

A. be related to other actions on media;
B. depend upon a time value;
C. depend upon an external event (user's interaction, signal from an application, etc.).

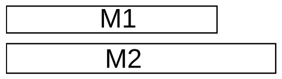
Some examples are:

A. Play M1 when M2 finishes or play M2 when M1 is playing for 25 seconds;
B. Play M1 at 11:33:20;
C. Play M1 when the user selects the icon button “start”, or show a “mail” icon when an email message arrives.

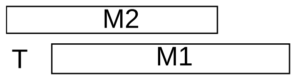
3.1.1 Startpoint

A startpoint determines when a media starts playing. A startpoint of a media M1 is defined by one of the following events (in brackets the corresponding case described in the taxonomy is expressed):

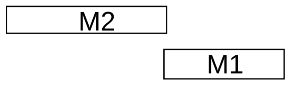
1.A1 **M2 starts** *(2.3 - 2.5)*



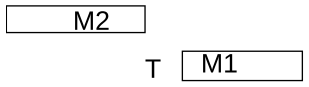
1.A2 **M2 is playing for T** *(2.1 - 2.2)*



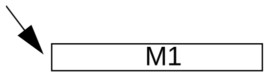
1.A3 **M2 finishes** *(1.1)*



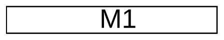
1.A **M2 finishes + T** *(1.2)*[1]



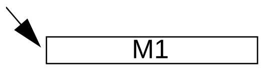
1.A5 **other events**



1.B1 **T = hh:mm:ss**



1.C1 **users’ event**

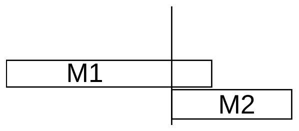


1. The case “M2 finishes - T” can be expressed by the case 1.A2, where the length of M2 is known

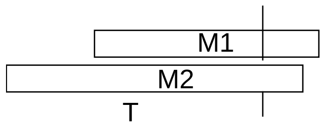
3.1.2 Endpoint

An endpoint determines when a media stops playing. An endpoint of a media M1 is defined by one of the following events (in brackets the corresponding case described in the taxonomy is expressed):

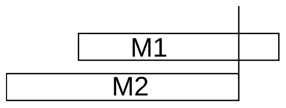
2.A1 **M2 starts**



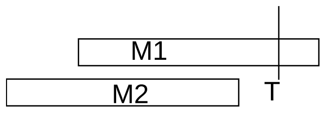
2.A2 **M2 is playing for T**



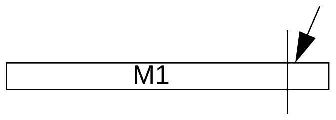
2.A3 **M2 finishes** *(2.4 - 2.5)*



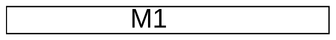
2.A4 **M2 finishes + T**[1]



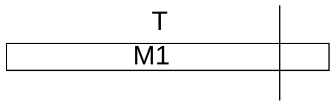
2.A5 **other events**



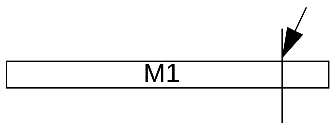
2.B1 **no events (till the end)**



2.B2 **for T**



2.C1 **user event**



1. The case “M2 finishes - T” can be expressed by the case 2.A2 where the length of M2 is known.

3.2 Time-line versus temporal relation

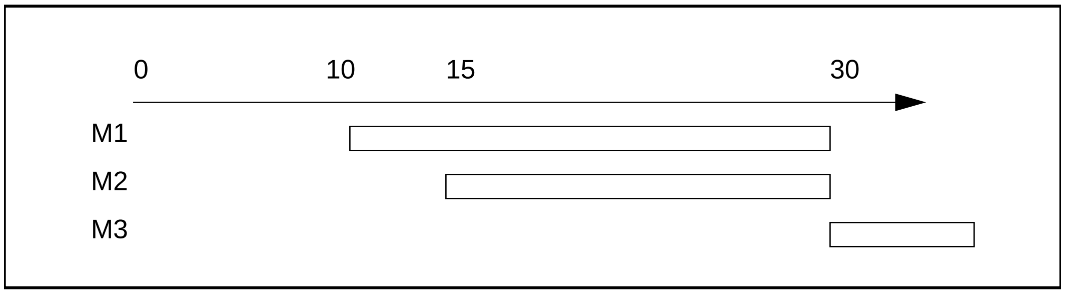
Some systems are based on the concept of synchronization as a simple event time-line. Given a time-line, actions on media are associated with points in time within the time-line. When a media object on the time-line is moved, all references and links with the other media objects are lost. So a time-line concept does not seem sufficient and a more structured approach is needed, so that actions on media are triggered not only by the advancement of the "real time". In such a case, those actions do not only rely on a schedule but also on actions on other media.

As an example, let consider the following media application:

Play M1 after T=10secs from now and stop it after T=20secs. Play M2 after T=5secs M1 is playing and stop it when M1 stops. Play M3 when M1 and M2 stop.

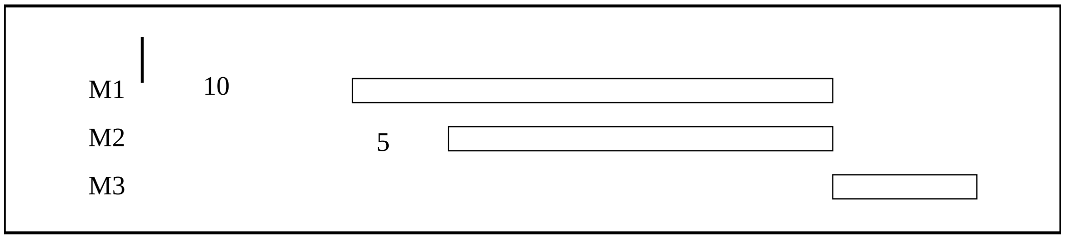
The description of this application can be given by a *time-line* or a *temporal relation* approach.

3.2.1 Time-line



1. Play M1 at T=10 and stop at T=30
2. Play M2 at T=15 and stop at T=30
3. Play M3 at T=30 till its end

3.2.2 Temporal relation



1. Play M1 after T=10 from now for T=20
2. Play M2 while M1 is playing for T=5 and stop when M1 stops
3. Play M3 when M1 and M2 stop

What happens if we want M1 to start at T=5?

We have to change all the specifications given in the first case, while we actually need to change only the first specification in the second case.

The application may adopt a time-line approach as an internal representation of events, but this does not affect user's specification.

4 Multi-media relations

This chapter describes the relations which can be defined among two or more media, derived from the temporal relations described above.

Temporal relations described above have been subdivided in: hierarchical relations and conditional relations. Hierarchical relations are used to describe a sequential or a parallel relation of type 1.1 and 2.3 presented in chapter 3. Conditional relations have been introduced to describe all the other cases.

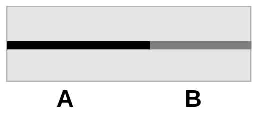
Some of these relations are constraint relations, where the relationship among the objects must be maintained when their state changes, and the relationship has to satisfy itself multidirectionally [4].

4.1 Hierarchical relations

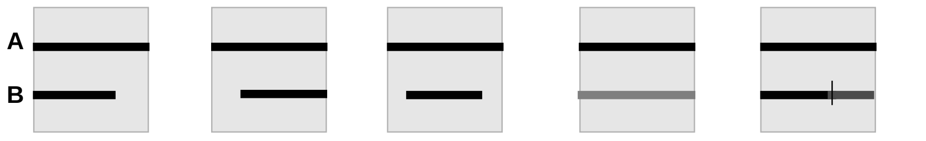
A hierarchical relation is applied to a set of objects and states that the objects have to be played in series or in parallel. If the objects are linked by means of a *sequential relation*, when an object finishes playing the following starts playing. In the case of a *parallel relation*, all the objects start playing at the same point in time.

All the objects related by sequential or parallel relations form a *compound object*.

A compound object, containing objects linked by a sequential relation, finishes when the last object of the sequence finishes. An example is shown in the following picture (compound object is visualized by a halftoned box).

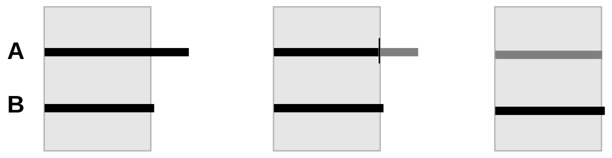


In the case of two objects linked by a parallel relation, the compound object can be considered finished when the longest object finishes. Sometimes, it is useful to define alternative styles of playing the shortest object. They are shown in the following picture.



a. locate at the beginning
b. locate at the end
c. locate in the centre
d. stretched over the longer
e. repeated and chopped

Another possibility is to consider the compound object finished when the shortest object finishes. The following picture shows how the longest media can be played.



a. go on playing
b.chopped
c. compressed

Of course, the styles compress, chop and repeat have no meaning in the case of time-invariant objects.

4.2 Conditional relations

A conditional relation between two media objects, media1 and media2, links a state of media1 to a state of media2, according to some rule. The state of media1 is associated with a *condition* and the state of media2 with an *action*. When the condition on state/media1 is verified, then the relation executes the action on state/media2. For example, if the relation is stop/media1 $\rightarrow$ play/media2, then, when media1 stops playing, media2 starts playing.

The actions that can be performed over a media can be: start playing or stop playing the media. Conditions are described in the following sections.

4.2.1 Deterministic and non-deterministic conditions

A media condition is *deterministic* if the event changing the state of object involved in the condition is fixed so that it is possible to say exactly if and when the condition is satisfied. Otherwise, the condition is *non-deterministic*.

Because *Startpoint* and *Endpoint* can be related to other actions on media, depend upon a time value or depend upon an event external to the application, it is possible to define the following relations, where the condition is on the left side of the relation and the action on the right:

- state/media $\rightarrow$ state/media
- time $\rightarrow$ state/media
- external-event $\rightarrow$ state/media

The first two kinds of conditions, depending on the state of a media or on an absolute value of time, are deterministic. Whereas, the condition linked to a general external event is non-deterministic, as it is not possible to know when or if it happens.

4.2.2 Simple conditions

Simple conditions state when an action can be performed on an object. As seen above, a simple condition can be related to a state of a media, to a time value or to an event. Simple conditions are as follows:

- media starts playing
- media starts playing plus a delay
- media stops playing
- media stops playing plus a delay
- media is/is not playing
- event
- time

4.2.3 Compound conditions

Conditions can be compounded in order to obtain more complex and structured conditions. Single conditions can be compounded using the AND and OR operators, obtaining *ANDConditions* and *ORConditions*.

The following application is an example of the use of a compound condition (Figure 2): a text (M1) is shown for 10 seconds and a picture (M2) is displayed in parallel for 20 seconds. A user’s event (E1) can start a piece of music (M5) lasting 5 seconds when the text M1 is displayed. When the piece of music finishes, a video (M3) in parallel with an audio (M4) are shown. (If there is no user event E1, when the picture M1's display time is over, the video M3 starts). An ANDCondition is required to start the piece of music M5 when M1 is playing *and* the event E1 occurs.

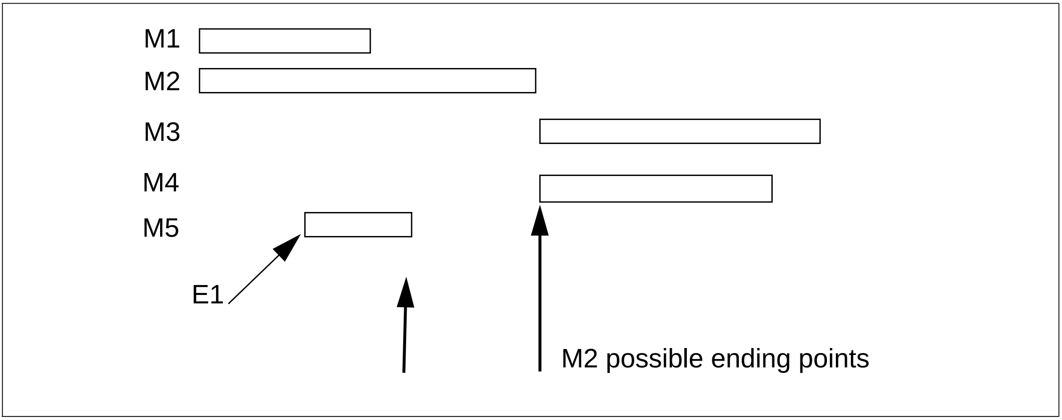


Figure 2. Example of a media application using an AND condition

Another example, using an ORCondition, is the following (Figure 2): play a text (M1) and a video (M2) in parallel. Play a sound (M3) and a text (M4) in sequence. Play the sequence M3, M4 *or* when the external event E1 arrives or when the time T occurs.

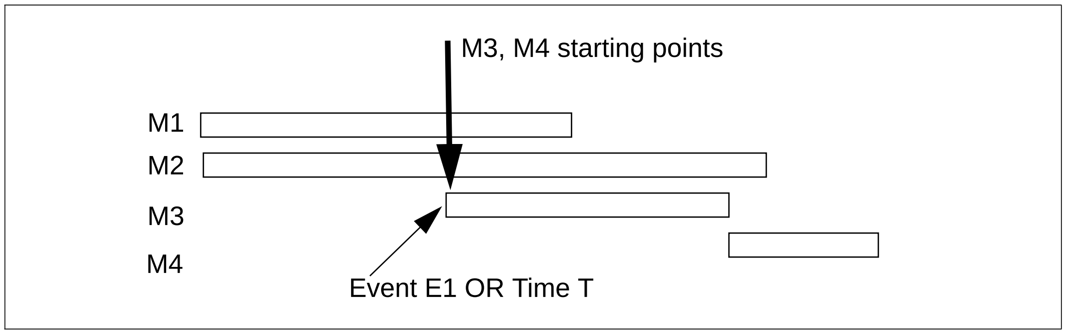


Figure 3. Example of a media application using an OR condition

4.3 Constraint relations

A constraint relation is a relationship between two or more objects which has to be held when the states of the objects change. Typically, constraint relations are multidirectional, which means that if some constraint is defined among some objects, a change of the state of one object implies the change of the state of all other related objects [4].

A bidirectional constraint relation is applied to sequential relations. The relation which links two siblings is a constraint, so that when the first object finishes playing, the following starts playing and if the following object starts playing (due to any condition), the previous object stops playing.

A multidirectional constraint relation is defined among the Startpoints of media objects linked by a parallel relation. When one object starts playing all its siblings do the same.

Finally, some constraints can be defined for some of the conditional relations. In particular, it is possible to define the constraints shown in Figure 4 (relations and their inverse are shown):

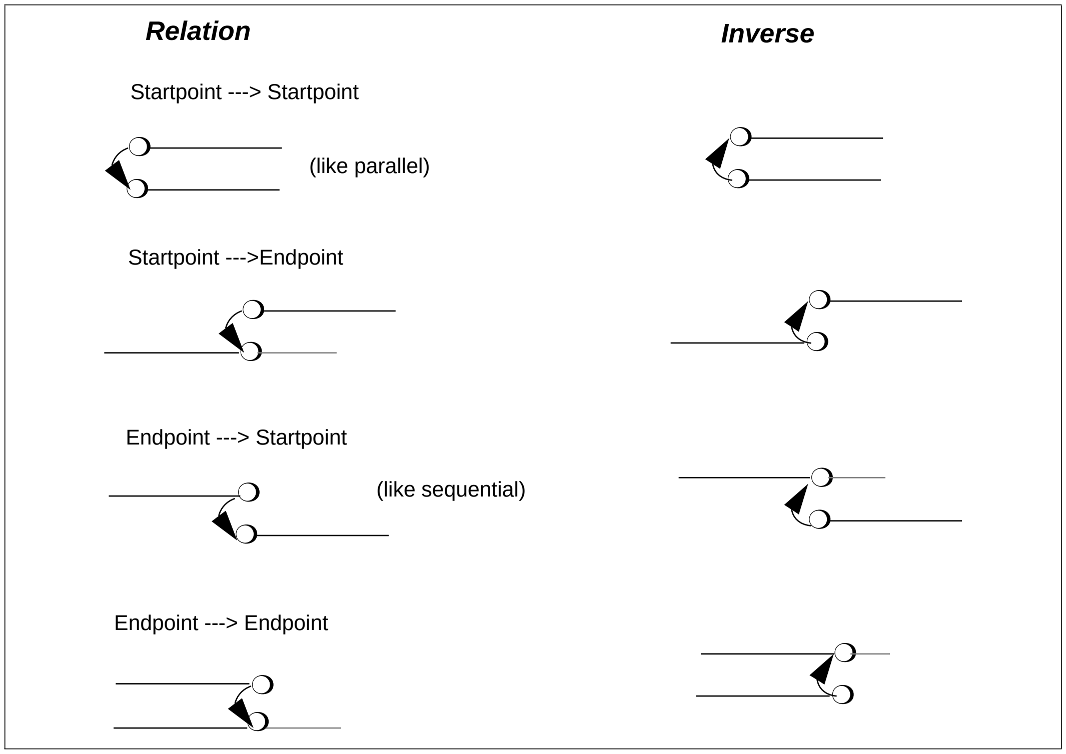


Figure 4. Constraint relations.

Constraint relations cannot always be applied and maintained in relations implying a delay. For example, consider the relation “play the object M2 10 seconds after the object M1 has finished playing” (Figure 5). The inverse relation, “10 seconds before the object M2 starts playing, stop the object M1”, can be maintained only if it is possible to know exactly when M2 starts playing (i.e., it has not to depend on a non-deterministic condition).

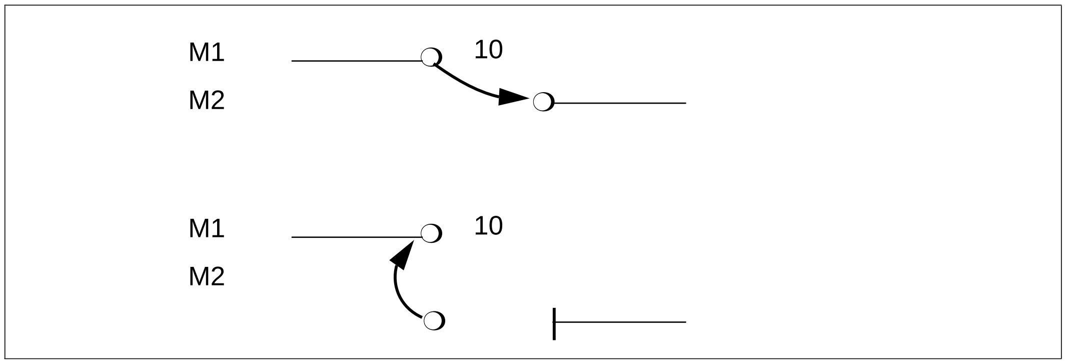


Figure 5. Example of a constraint relation.

Constraint relations can be useful in some cases, but not in every case. Moreover, it is difficult to control constraints and their propagation within the application. This may lead to undesired effects. For example, consider the application (Figure 4) “play the media M1 when an external event arrives and stop the media M2 when M1 starts”. The inverse relation “start M1 when M2 finishes” is in conflict with the other one, requiring also an external event for starting.

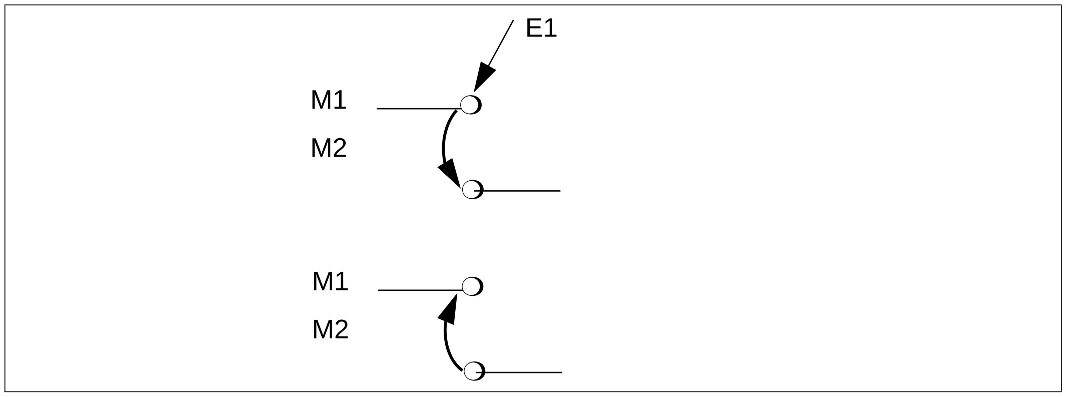


Figure 6. Another example of a constraint relation.

5 Multi-media in Views

This research work aims at defining and implementing a multi-media model within the TAXATA model used in the Views project, which supplies a consistent and integrated open-architecture application environment. According to the Views project philosophy, Views application environment has to support the implementation of multi-media application where the basic structure of the application itself is already defined in the kernel of Views. Views should allow users to define multimedia applications by means of a high level notation or by means of a programming language. In the first case, the notation to describe the media objects and their relationships uses some simple operators; in the second case, Views should provide a predefined framework (compound of media objects and functions to structure the objects).

This research aims at identifying (Figure 5):

1. Notation to describe media objects and their relations;
2. A programming environment which provides a programmer with functions to create multimedia objects and to structure them.
3. Views Internal Structure. It consists of some functions which interpret the programming language and build a permanent internal structure for the media application, and some other functions which convert the permanent structure into a low level structure when the media application (or some objects of the application) is played.

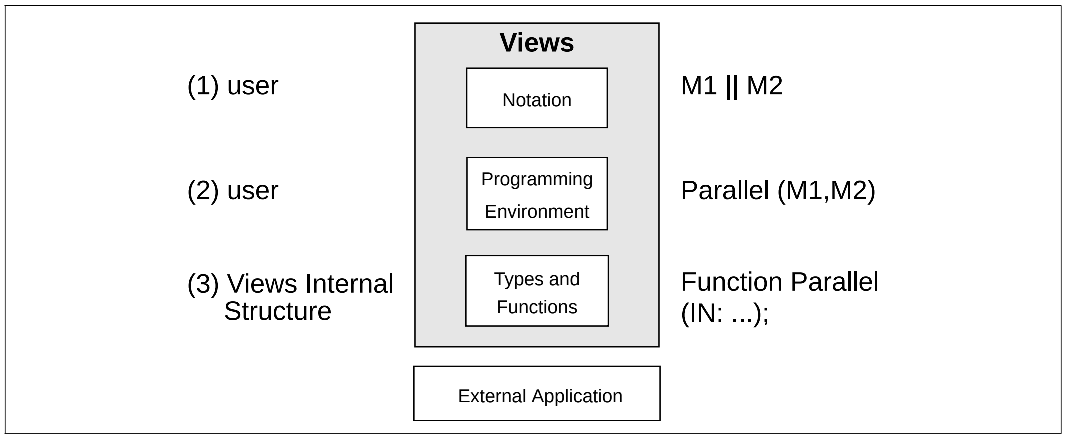


Figure 7. Another example of a constraint relation.

5.1 Multi-media objects in Views

The multi-media objects supported by Views are the following.

- **Text**
  It consists of a text string and a length value defining how long the text has to be displayed.
- **Picture**
  It consists of a data object containing the description of the picture (dimensions, colour planes, data) and a length value defining how long the picture has to be displayed.
- **Animation**
  It consists of a data object containing the description of the animation (number of frames, colour planes, data) and a length value defining how long the animation has to be played.
- **Audio**
  It consists of a data object containing the description of the chunk of audio (sapling rate, length of the chunk), a length value defining how long the audio has to be played and a volume value.
- **Video**
  It consists of a data object containing the description of the chunk of video (sapling rate, length of the chunk, number of frames) and a length value defining how long the video has to be played.

5.2 Multi-media model

It is possible to structure media objects by means of hierarchical relations and conditions (simple and compound) described previously. Moreover, some other objects (described in the following), called blocks can be used to structure the objects.

5.2.1 Hierarchical relations

Hierarchical relations give a hierarchical structure to the application, allowing the definition of *compound objects* consisting of a set of objects to be played in series or in parallel, or a composition of them.

- **Sequential relation**
  Play two or more objects (or compound objects) in series: the following object starts as soon as the previous is finished or, for the bidirectional relation between two siblings, when the following object starts, the previous stops playing.
- **Parallel relation**
  Play two or more objects (or compound objects) in parallel. The system starts playing the objects all together, at the same point in time (the objects are related by means of multidirectional relations).

5.2.2 Conditional relations

Conditions are used to define more structured and powerful applications, by defining some conditions which specify when a media starts or stops playing, which cannot be expressed by means of hierarchical relations. Conditional relations state that an action (play or stop) on a media his performed when one or more events occur.

The possible conditions are:

A. an external event occurs (for example, a user selects an icon button);
B. it is a certain time (for example it is 3:20:00);

C. a media starts playing plus a delay (which can be nil);
D. a media stops playing plus a delay (which can be nil);
E. a media is playing.

It is possible to require the system to verify only one condition, one condition within a group, or all conditions within a group. According to these cases, the following three types of condition have been identified.

Simple condition

An action is performed on a media when a condition is verified.

♦ **Starting relation**

A media can play when the condition required (of type A, B, C or D) is verified. In particular, it is possible to notice that:

Condition C:

- if the delay is zero, this case can be defined as a parallel relation;
- if the delay is not zero, then it might be defined as a sequence of time T= delay and media played in parallel with the other media;

Condition D:

- - if the delay is zero, this case can be defined as a sequential form;
- - if the delay is not zero, this case can be defined as a sequence of the first media, time T= delay and the other media.

Examples:

1. Play M1 when Event: start playing M1 when the Event occurs.
2. Play M1 when T = 3:20:00: start playing M1 when it is 3:20:00.
3. Play M1 when M2 starts: Play in Parallel (M1; M2).
4. Play M1 when M2 is playing for T=30secs: Play in Parallel (M2; Play in Sequence ((T = 30); M1)).
5. Play M1 when M2 finishes: Play in Sequence (M1; M2).
6. Play M1 when M2 finishes + T = 20secs: Play in Sequence (M2; (T = 20); M1)).

♦ **Ending relation**

A media can stop playing when the condition required (of type A, B, C or D) is verified.

Examples:

1. Stop M1 when Event: stop playing M1 when the Event occurs.
2. Stop M1 when T = 3:20:00: stop playing M1 when it is 3:20:00.
3. Stop M1 when M2 starts.
4. Stop M1 when M2 is playing for T=30secs.
5. Stop M1 when M2 finishes.
6. Stop M1 when M2 finishes + T = 20secs.

OR Condition

An action on an object is performed when one condition, within a set of two or more conditions, is verified.

Example:

Play/Stop M1 when a user selects an icon button or when the media M2 starts playing.

AND Condition

An action on an object can occur when all the conditions, within a set of two or more conditions, are verified.

Example:

Play/Stop M1 when the media M2 is playing and an icon button is selected.

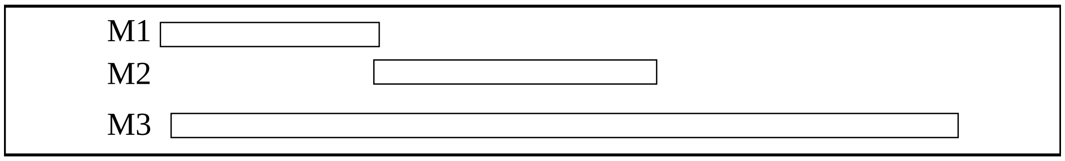
5.2.3 Blocks

Blocks are structuring objects used to group some media objects and conditions together. They have been introduced to simplify the definition of the application and the visiting of its components. The utility of block objects while visiting the application is described later.

6 Notation

The defined notations try to reproduce the way users think when they are defining a media application. They draw a picture of the application and then try to describe the picture by describing which are the objects involved and the relations among them using a simple language.

For example, let us try to describe the following media application:



We would say: play M1 and M2 in sequence and M3 in parallel with them. The developed language try to allows users to define their applications in a similar way, by means of a natural language.

6.1 Description of the notation

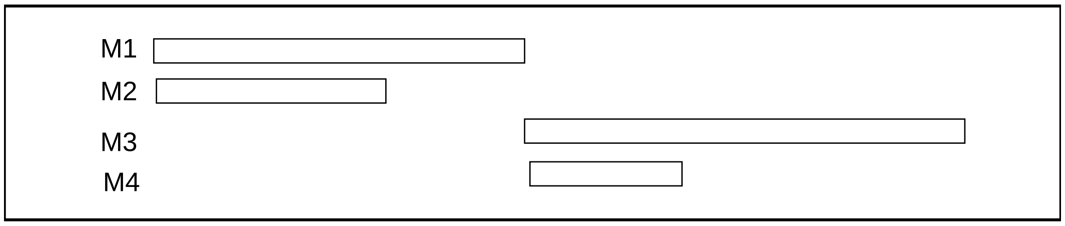
The high level notation presented in Table 1 uses some simple symbols to define media objects, hierarchical relations, conditions and blocks described previously. In the description, M denotes a single media, MM a compound object, T a time period, C a condition, A an action object and B a block.

6.2 Examples

The following examples show how a media application can be defined using the notation described above. These examples will be also used in some following chapters.

Example 1. Hierarchical relations

Play M1 and M3 in parallel. When they finish playing, play M3 and M4 in parallel:



Type of Symbol	Symbol	Description
Declaring	`M = <TYPE>:object_name`	declare an object
	`M | T`	set the length of an object
	`@@M = n.n`	set the sampling rate
Querying	`L = #(M)`	query the length
	`S = @@(M)`	query the sampling rate
Hierarchical Relations	`MM = (MM1;M2;MM3)`	define a sequential object
	`MM = (MM1||M2||MM3)`	define a parallel object
Actions	`A = > MM1`	play an object
	`A = <> MM1`	stop an object
Simple Conditions	`:: MM1>`	when an object starts
	`:: MM1> +T`	when an object starts+ delay
	`:: MM1<>`	when an object stops
	`:: MM1<> +T`	when an object stops + delay
	`:: MM1@`	when an object is playing
	`:: MM1!@`	when an object is not playing
	`:: X`	when an external event occurs
	`:: T`	when a time T occurs
Compound Conditions	`C = (C1 OR C2 OR … )`	define an OR condition
	`C = (C1 AND C2 AND …)`	define an AND condition
Structuring Relation	`B = <<MM, A1, A2>>`	define a block object

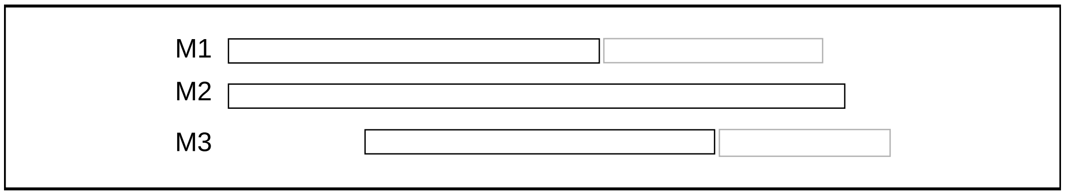
Table 1. Notation commands

Or in our notation:

```
P1 = ( M1 || M2 )
P2 = ( M3 || M4 )
S = ( P1 ; P2 )
```

Example 2. Simple condition - start/stop

Play M1 and M2 in parallel. Play M1 for 10secs and M2 for 20secs. Play M3 when M1 is playing for 5secs and stop when M2 is playing for 15secs:

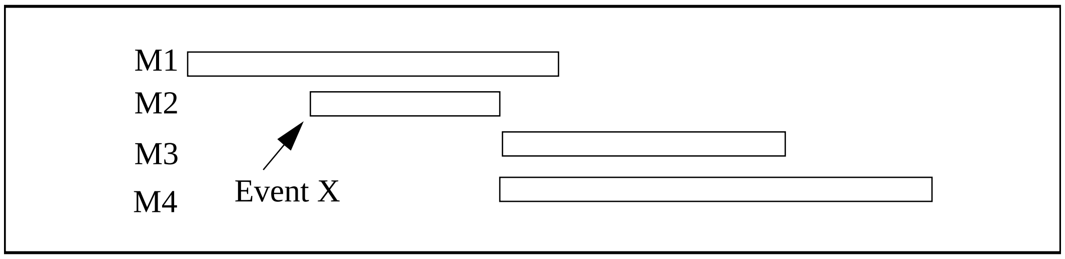


Notation:

```
M1 | 10
M2 | 20
P = ( M1 || M2 )
A1 = > M3 :: ( M1> + 5 )
A2 = <> M3 :: ( M2> +15 )
B = << P, A1, A2 >>
```

Example 3. Simple condition - event

Play M1. Play M2 when the external event X arrives. When M2 finishes, start playing M3 and M4 in parallel:

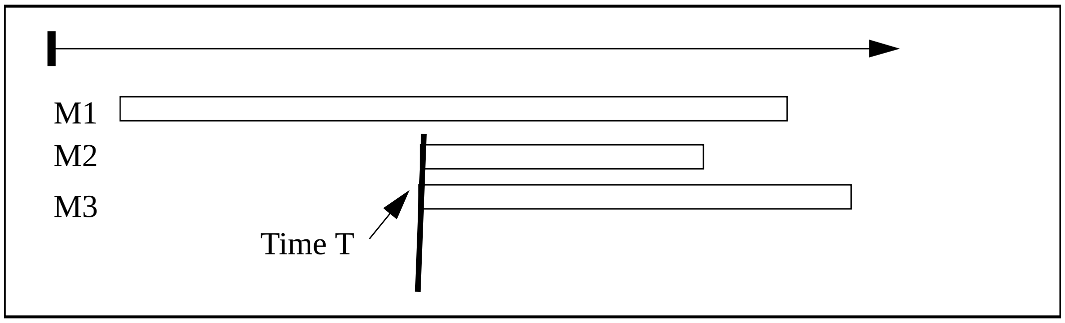


Notation:

```
P = ( M3 || M4 )
S = ( M2; P )
A = >S :: (X)
B = << M1, A >>
```

Example 4. Simple condition - time

Play M1; play M2, M3 in parallel when Time T occurs:

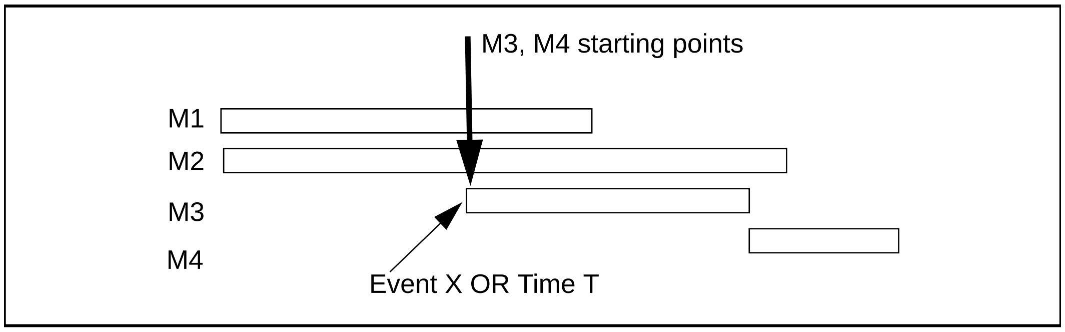


Notation:

```
P = ( M2 || M3 )
A = >P :: (T)
B = << M1 , A >>
```

Example 5. Compound condition - OR condition

This example is the one presented in section 4.2.3 - Figure 2: play a text (M1) and a video (M2) in parallel. Play a sound (M3) and a text (M4) in sequence. Play the sequence M3, M4 *or* when the external event E1 arrives or when the time T occures:

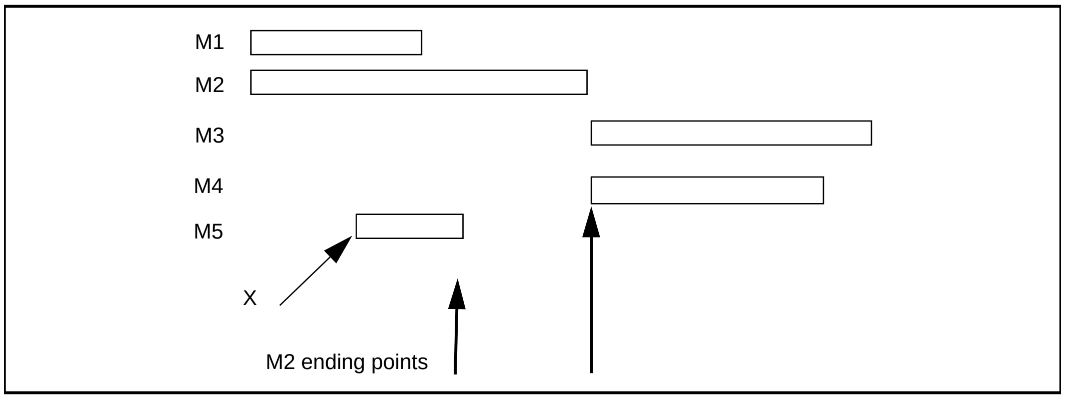


Notation:

```
P = ( M1 || M2 )
S = ( M3 ; M4 )
A = > S :: ( ( X) OR ( T ) )
B = << P, A >>
```

Example 6. Compound condition - AND condition

This example is the one presented in section 4.2.3 - Figure 1: a text (M1) is shown for 10 seconds and a picture (M2) is displayed in parallel for 20 seconds. A user’s event (X) can start a piece of music (M5) lasting 5 seconds when the text M1 is displayed. When the piece of music finishes, a video (M3) in parallel with an audio (M4) are shown. (If there is no user event E1, when the picture M1’s display time is over, the video M3 starts). An ANDCondition is required to start the piece of music M5 when M1 is playing *and* the event X occurs:



Notation:

```
M1 | 10
M2 | 20
M3 | 5
P1 = ( M1 || M2 )
P2 = ( M3; M4 )
A1 = > M5 :: ( ( M1@) AND ( X ) )
A2 = <> P1 :: ( M5 <> )
B = << P1 , A1 , A2 >>
S = ( B; P2 )
```

6.3 Temporal location of media played in parallel

These relations deal with the cases described in section 4.1, where it is required to detect the end of an object compound of two or more media played in parallel.

An example is:

Let M1 and M2 be two media. M1 is shorter that M2. They are played in parallel:

```
S1 = M1 || M2
```

A possibility is that the media starts at the same time and the object P is considered finished when the longest, M2 in this case, finishes (the notation is `M1 || M2`). But other relations can be defined.

1. P finishes when M1 finishes.

   1.1. M2 is normally played:

```
M1 || M2 [-]
```

   1.2. M2 is chopped:

```
M1||M2 [|]
      M2 | (# M1 )
```

   1.3. M2 is compressed, which means M2 is speeded up:

```
M1 || M2 [ <- ]
   @@M2′ = ( (# M2 ) / (# M1 ) ) * @@M2
```

2. P finishes when M2 finishes.

   2.1. M1 is stretched, which means M1 is slowed down:

```
M1 || M2 [ -> ]
   @@M1′ = ( (# M1 ) / (# M2 ) ) * @@M1
```

   2.2. M1 is repeated and chopped:

```
M1 || M2 [ ^^ ]
   n1 = ( ( # M2 ) / ( # M1 ) )
   n2 = ( ( # M2) MOD ( # M1 ) )
   S1 = (M1 ; M1 ; M1 ; ... ) // x1 times
   S2 = ( T1; ( M1 | n2 ) )
   S3 = S2 || M2
   > S3
```

   2.3. M1 is located in M2:

```
(at the beginning:)
M1 || M2 [ M* ]
   x1 = ( # M2 ) - ( # M1 )
   T1 = ( M1 ; x1secs )
   T1 || M2

(at the end:)
M1 || M2 [ *M ]
   x1 = ( # M2 ) - ( # M1 )
   T1 = ( x1secs ; M1 )
   T1 || M2

(centred:)
M1 || M2 [ *M* ]
   x1 = (  ( # M2 ) - ( # M1 )) /2
   T1 = (x1secs ; M1 ; x1secs)
   T1 || M2
```

6.4 Temporal location of media within a time period

These relations deal with the cases described in section 2.3, where it is required to play media for a time longer than the length of the media itself.

An example is:

Let `M` be a media and `T` a time period longer than the length of `M`. We want to play `M` over T: `M | T`.

A possibility is that the media finishes playing according to its length (the notation is `M | T`). But other relations can be defined:

1. `M` is stretched over `T` (`M` is slowed down):

   ```
   (M->) | T
   ```

   Let `L1` be the length of `M`, `F1` its sampling rate, the new sampling rate will be:

   ```
   (L1/T)*F1
   ```

2. `M` is repeated and chopped:

   ```
   (M^^) | T
   n1 = T / L1 (times)
   n2 = T mod L1 (seconds)
   S1 = (M ; M ; M ; ... ) n1 times
   S2 = (S1; M | n2 )
   ```

3. `M` is located in T; at the beginning, at the end or it is centred in T:

   ```
   (at the beginning:)
      (M*) | T
      x1 = T - ( # M )
      S1 = ( M ; x1secs )

   (at the end:)
      (*M) | T
      x1 = T - ( # M )
      S1 = ( x1secs ; M )

   (centered:)
      (*M*) | T
      x1 = ( T - ( # M )) /2
      S1 = ( x1secs ; M ; x1secs)
   ```

7 Programming environment

The programming environment consists of a set of data types and functions that programmers can manipulate to define their applications.

The programming environment provides:

1. set of functions to declare the objects of each predefined basic type; (example, `M1 = audio (Audio)`)
2. set of functions to define some attributes of the objects (example, length of a chunk of video, volume of an audio fragment)
3. set of functions to query the objects (example, query the length of a chunk of video)

4. set of functions to structure the objects and conditions (example, `Parallel (M1,M2)`)

7.1 Functions to declare the objects

A set of functions are provided to define the multimedia objects and their length in time. These functions are:

```
MEDIA = video ( data, length )
MEDIA = audio ( data, length )
MEDIA = text ( “<text>”, length )
MEDIA = animation ( data, length )
MEDIA = picture (data, length )
```

7.2 Functions to define some attributes of the objects

These functions are used to set some attributes (length, sampling rate, …) of objects.

- **VIDEO**
  ```
  VideoLengthSet (media, #, <options>)
  VideoSpeedSet (media, n.n)
  ```
- **AUDIO**
  ```
  AudioLengthSet (media, #, <options>)
  AudioVolumeSet (media, <volume>)
  AudioSpeedSet (media, n.n)
  ```
- **TEXT**
  ```
  TextLengthSet (media, #)
  ```
- **ANIMATION**
  ```
  (not implemented)
  ```
- **PICTURE:**
  ```
  PictureLengthSet (media, #)
  ```

The parameter `option` in the functions `VideoLengthSet` and `AudioLengthSet` is used to define the cases described in 4.2. It can assume one of the following values: `ASIS, STRETCHED, REPEAT, BEGIN, END, CENTRED`.

7.3 Functions to query the objects

These functions are used to query the attributes (length, sampling rate, …) associated with the objects.

- **VIDEO**
  ```
  objVideoLength
  ```
- **AUDIO**
  ```
  objAudioLength
  objAudioVolume
  ```
- **TEXT**
  ```
  objTextLength
  ```
- **ANIMATION**
  ```
  (not implemented)
  ```
- **PICTURE**
  ```
  objPictureLength
  ```

7.4 Functions to structure the objects

In the following, a set of functions to give a structure to the media objects is presented. These functions are used to define the hierarchical relations among media objects, conditions and blocks.

Table 2 shows the functions. `M` denotes a single media, `MM` a compound object (parallel or sequential object), `T` a time period and `C` a condition, `A` an action object and `B` a block.

Type of Function	Function	Description
Hierarchical Relations	MM = Sequential(MM1,M2)	define a sequential object
	MM = Parallel (MM1, M2)	define a parallel object
Actions	A = Play(MM, C)	play an object when a condition is true
	A = Stop (MM, C)	stop an object when a condition is true
Simple Conditions	C = ConditionStart(MM, 0)	when an object starts
	C = ConditionStart(MM,delay)	when an object starts plus a delay
	C = ConditionStop(MM,0)	when an object stops
	C = ConditionStop(MM,delay)	when an object stops plus a delay
	C = ConditionPlaying (MM)	when an object is playing
	C = ConditionNotPlaying (M)	when an object is not playing
	C = ConditionEvent(X)	when an external event occurs
	C = ConditionTime(hh:mm:ss)	when a time T occurs
Compound Conditions	C = ORCondition(C1,C2,...)	define an OR condition
	C = ANDCondition(C1,C2...)	define an AND condition
Structuring Relation	B = Block(MM,A1,A2)	define a block object

Table 2. Language functions

7.5 Examples

The following examples describe how some media applications can be defined using the commands described above. The examples are the ones presented in section 6.2.

Example 1.

```
P1 = Parallel ( M1, M2 );
P2 = Parallel ( M3 , M4 );
S = Sequential ( P1 ; P2 );
```

Example 2.

```
M1 = video ( Video1, 10;);
M2 = picture ( Picture1, 20;);
M3 = audio ( Audio1, 0 );
P = Parallel ( M1, M2 );
C1 = ConditionStart ( M1, 5 );
C2 = ConditionStart ( M2, 15 );
A1 = Play ( M3, C1 );
```

```
        A2 = Stop ( M3, C2 );
        B = Block ( P, A1, A2 );
```

Example 3.

```
        P = Parallel ( M3, M4 );
        C = ConditionEvent ( event X );
        S = Sequential ( M2, P );
        A = Play ( S, C );
        B = Block (M1, A);
```

Example 4.

```
        C = ConditionTime ( Time T);
        P = Parallel ( M2, M3 );
        A = Play ( P, C );
        B = Block ( M1, A );
```

Example 5.

```
        P = Parallel ( M1, M2 );
        S = Sequential ( M3, M4 );
        C1 = ConditionEvent ( event X );
        C2 = ConditionTime ( T );
        C3 = ORCondition ( C1, C2 );
        A = Play ( S, C3 );
        B = Block (P, A );
```

Example 6.

```
        S1 = Parallel ( M1, M2 );
        P2 = Parallel ( M3, M4 );
        C1 = ConditionEvent ( event X );
        C2 = ConditionPlaying ( M1 );
        C3 = ANDCondition ( C1, C2);
        A1 = Play (M5, C3);
        C4 = ConditionStop ( M5, 0 );
        A2 = Stop ( P1, C4 );
        B = Block (P1, A1, A2 );
        S = Sequential (B; P2 );
```

8 Views internal structure

A media application specified by means of the language is converted by the system into a *permanent structure* containing the description of the objects and their relations. The application is completely structured, forming a tree: the root of the tree can be a compound object or a block. A textual description of the structure is automatically provided by Views. A user can visit the complete application, or a sub-part of it. When a user visits an object of the media application, the structure is converted into a simpler structure, called *playing structure*, compound of basic elements. At this level, Views invariant technique is used to create relations among objects. Invariants create a relationship between the content of an object and one or more other objects. If an object content changes, any connected invariants are called to update connected objects. In the case of a multimedia application, invariants are used to manage synchronization among objects, to manage conditional objects and to create links to external objects (external application).

This chapter describes the data types needed to create the media objects, to organise the objects in a structure and to define hierarchical relations and conditions among the objects. In particular, it presents the data types used in the permanent structure, the data types and the invariants used in the playing structure and how to generate the second structure starting from the first one.

8.1 Permanent data types

The system provides a set of basic data types to structure the application. The types are shown in Table 3.

Permanent Type	**Content**
Multi-media	media (picture, text, audio, video, animation) length value
Parallel	sequence of children
Sequential	sequence of children
Time	hh:mm:ss
Event	description of the event
ConditionEvent	event
Condition Time	time
Condition Start	object (media, parallel or sequential)
Condition Start + delay	object (media, parallel or sequential) delay
Condition Stop	object (media, parallel or sequential)
Condition Stop + delay	object (media, parallel or sequential) delay
Condition Playing	object (media, parallel or sequential)
Condition Not Playing	object (media, parallel or sequential)
OR Condition	sequence of simple conditions
AND Condition	sequence of simple conditions
Play	object (media, parallel or sequential) condition
Stop	object (media, parallel or sequential) condition
Block	object (media, parallel, sequential or block) sequence of play and stop objects

Table 3. Permanent media data types

The described application is structured by means of these types. The result is a tree where the root can be a compound object or a block, and it is the application itself.

8.2 Playing data types

The system creates a new structure every time an object of the application, or the application itself, is visited (or played). The new structure links the media objects to a timer (Time object) and uses Views invariants to relate the objects. Views invariants define links between objects by creating a relationship between the content of an object and the content of one or more other objects. If an object content changes, any connected invariants are called to update connected objects. (As the state of media objects changes over time, a playing structure allows more users to visit the same application at the same time.)

The objects of the playing structure contain the same information as the basic object. Moreover, playing objects contain some boolean variables describing the state of an object (start playing, playing, stop playing). Views invariants are used to manage synchronization among objects, to manage conditional objects and to create links to external objects (external application).

These types (named playing types) are described in Table 4.

The two booleans, PlayBool and StopBool, contained in Multi-media, Parallel and Sequential objects are used to indicate when the object has started playing and when it has finished playing. These booleans are set by other Views objects or by the external application, through the Views interface. For example, to play an object, PlayBool has to be set to true. When the media finishes playing, the StopBool is set to true.

The boolean Time Bool (in the Time object) is set to true when the time is satisfied, Event Bool (in the Event object) is set to true when the expected event occurs.

The booleans contained in the Simple Condition objects have the following meaning: Enable Bool states if the object is enable to receive events, Arrived Bool is set to true if the expected event arrives and Playing Bool is used to manage an AND Condition where the condition needs to check if an object (media or compound) is playing.

The booleans Enable and Arrived Bool used in OR Conditions and AND Condition objects have the same meaning described above. In addition, AND Condition objects contain two integers: Number of Conditions that indicates the number of conditions linked to the object, and Counter indicating the number of conditions that have been satisfied.

Playing Type	**Content**
Multi-media	permanent multi-media Play Bool / Stop Bool
Parallel	sequence of playing multi-media Play Bool / Stop Bool
Sequential	sequence of playing multi-media Play Bool / Stop Bool
Time	Time Bool
Event	Event Bool
Simple Condition	Enable Bool Arrived Bool Playing Bool
OR Condition	Enable Bool Arrived Bool
AND Condition	Enable Bool Arrived Bool Number of Conditions Counter

Table 4. Playing media data types

8.3 Multi-media invariants

As described above, booleans are used to indicate the state of an object. Invariants are used to link these booleans for playing the application. They are used as input and output parameters of media invariants.The change of the value of a boolean variable of an object can lead to updating other objects in order to maintain invariants. An example is shown in Table 8. It shows the playing structure of the following sequential object:

```
        S1 = (M1; M2; M3)
```

Views invariants are bidirectional. This facility can be used for the implementation of constraint relations.

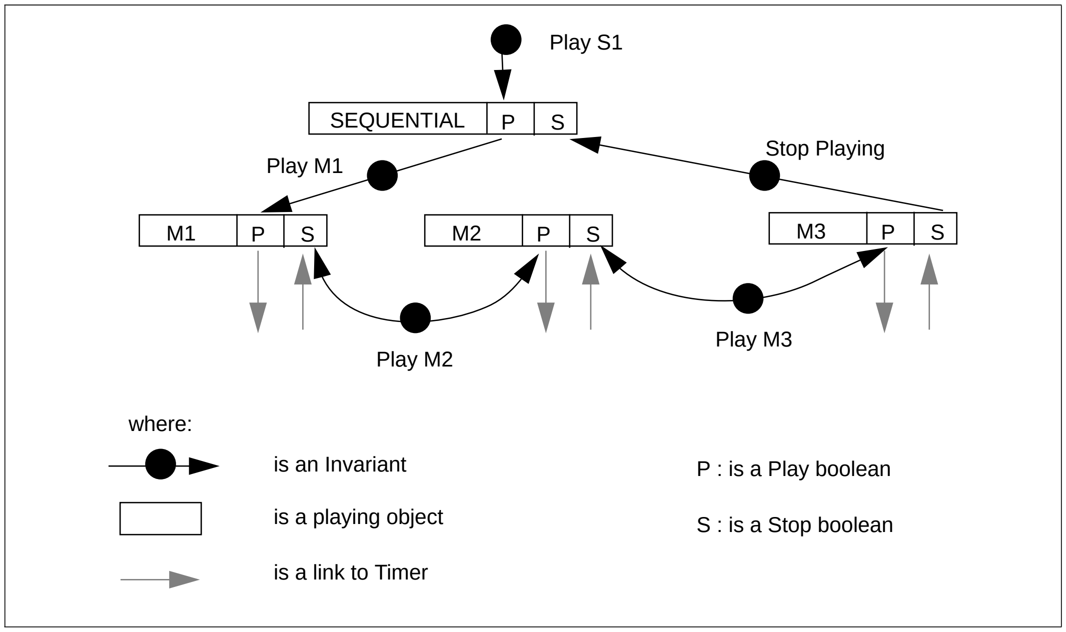


Figure 8. Example of the playing structure of a sequential object.

8.4 From basic to playing data structure

Basic objects are converted into a playing data structure. The conversion consists of the creation of some objects and invariants to link them.

In particular, some relations expressed in the basic structure can be reduced to simpler ones by means of some other objects and invariants. For this purpose a special object has been defined: the Silence object. It is considered as a media object which does nothing for a certain time, i.e. delay. It is connected to the system clock by means of an invariant. It can ask the clock to start counting seconds and receive a message of “time over” from the clock.

This object is used, for example, to describe the relation “play M1 and stop it when M2 has played for 15 seconds”:

```
        <>M1 :: (M2> +15)
```

The structure is shown in Figure 7.

Particular reductions are used to describe the following cases (the right hand side shows the reduced notation):

Case 1:

```
        >M2 :: (M1>)---> ( M1 || M2 )
```

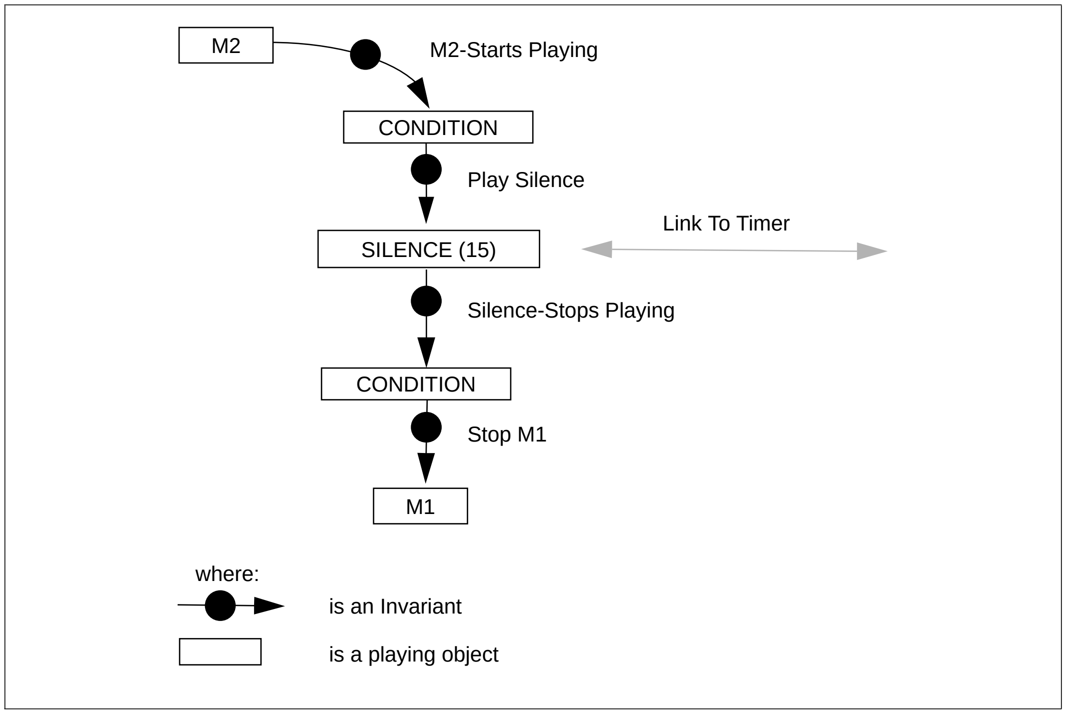


Figure 9. Example of reduction using the object silence.

Case 2:

```
      >M2 :: (M1> + delay)---> ( M1 || (Silence(delay) ; M2) )
```

Case 3:

```
      >M2 :: (M1<>)---> ( M1 ; M2 )
```

Case 4:

```
      >M2 :: (M1<> + delay)---> ( M1 ; Silence(delay) ; M2 )
```

8.5 Visiting the application

It is possible to visualize the internal structure of Views objects by *visiting* them. In the case of a multimedia application, “visit” an object means “play it”. A textual description of the structure is automatically provided by Views Figure 10. It is possible to play the complete application, a single media object, parallel or sequential objects and blocks. Visiting a block gives the possibility of playing an object and conditions associated with it. As an example, let us consider the media application described in Section 4.2.3 - Figure 1. It is possible to visit the parallel objects P1 (Figure 10-a) or the block B (Figure 10-b) which comprises P1 and the conditional object. In the first case, the objects M1 and M2 are played in parallel, in the second case, if the user performs the event, the object M5 is also played.

9 Conclusions

This report has presented the integration of multi-media objects in the Views system. The system provides some basic data types for defining and structuring a media application. The structure of the media objects is defined by means of a set of functions which allow the definition of hierarchical and conditional relations among the media.

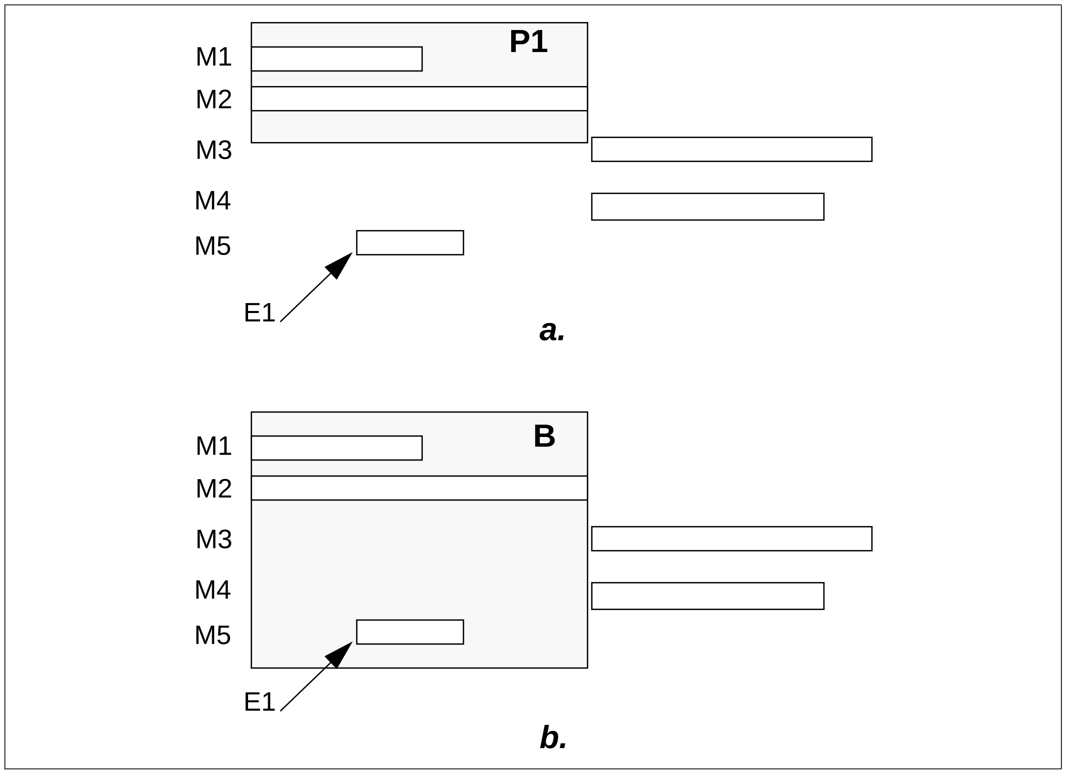


Figure 10. Example of visiting the views structure of a media application.

The defined application can then be visited by playing the complete application or some objects of it. Moreover, thanks to some Views functionality, it is possible to change some parameter values of the application (for example the length of a chunk of video) and play again the application.

A possible extension would be the study and implementation of a module for parsing the notation proposed in this report for the definition of a media application and the implementation of a visual presentation of the application (objects and their relations).

10 References

[1] S. Pemberton, *Views: An Open-architecture User Interface System*. In *Proceedings International Conference "Interacting with Computers: Preparing for the Nineties"*, Noordwijkerhout, The Netherlands, December 1990.

[2] Hoepner P., *Synchronizing the Presentation of Multimedia Objects - ODA Extensions-*, SIGOIS Bulletin, vol. 19, n. 1, pp. 19-32.

[3] J.F. Allen, *Maintaining Knowledge about Temporal Intervals,* Communication of the ACM, vol. 26, no. 11, pp. 832-843, November 1983.

[4] Borning A., Duisberg R., *Constraint-Based Tools for Building User Interfaces*, ACM Transaction on Graphics, vol. 5, n. 4, pp. 345-374, October 1986.

Appendix: Implementation Architecture

11 Views internal structure

The media application specified by means of the language is converted by the system into a permanent structure containing the description of the objects and their relations. The application is completely structured, forming a tree. The root of the tree can be a compound object or a block. When the user visits an object of the media application, the structure is converted into a simpler structure compound of basic elements. A user can visit the complete application, a sequential object, a parallel object or a single media object. At this level, Views invariant technique is used to create relations among objects. In the case of a multimedia application, invariants are used to manage synchronization among objects, to manage conditional objects and to create links to external objects (external application).

This report presents the data types used in the permanent structure, the data types and the invariants used in the playing structure and how to generate the second structure starting from the first one.

12 Permanent data types

The basic data types are subdivided into Multi-media Types, Structuring Types and Conditional Types, Time and Event Types.

12.1 Multi-media types

It consists of a set of data types describing the internal structure of the multi-media objects.

An object of type multi-media is compound of two objects: an object chosen among the five available types of multi-media (picture, text, audio, video and animation) and a length value. The structure is shown in Figure 1.

The correspondent implementation of this structure in Views is shown in Figure 2. It consists of an object of type Multimedia compound of an object of type MediaType (a choice of one among the five available media), a Length object (an integer value containing the length of the media) and some Option saying how to play a media.

Each media object is described by some information typical of the type of media itself. The types of media are described in the following.

12.1.1 Video

The object Video contains some data describing the chunk of video (type of video, header, video data) and a float value indicating the sampling rate (number of frames per second).

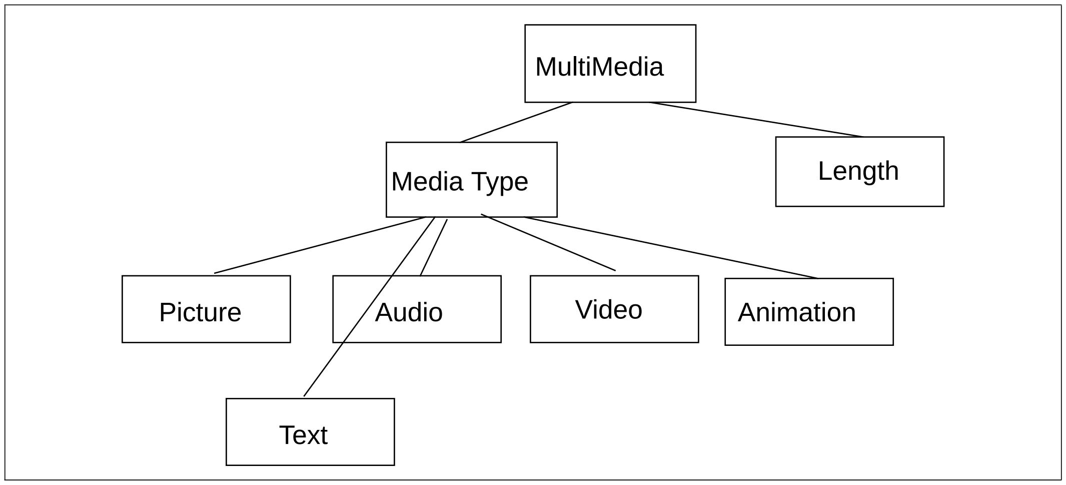


Figure 11. Multi-media Structure

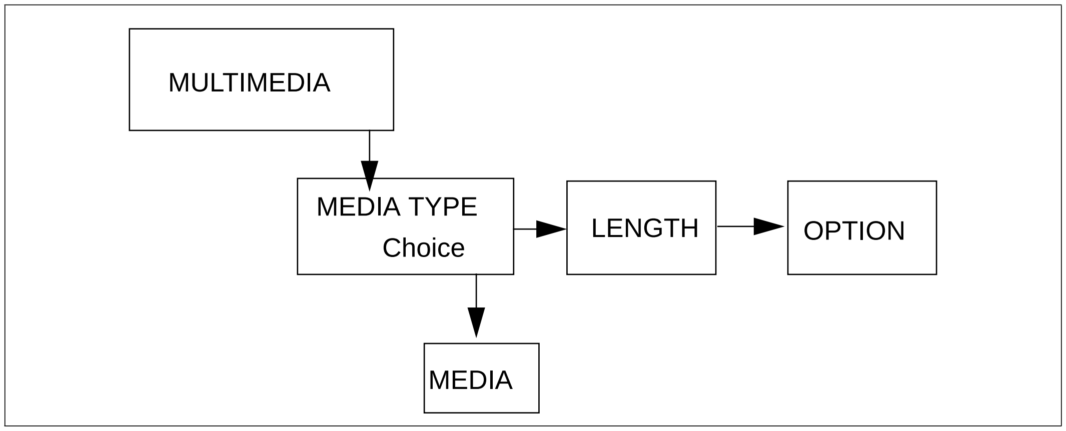
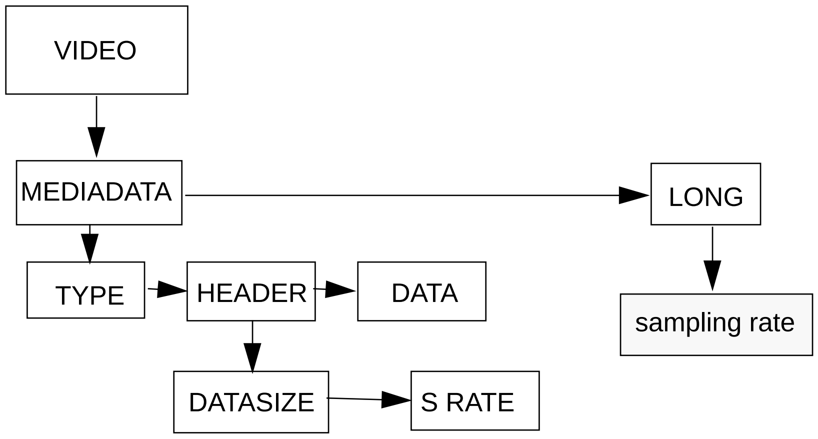
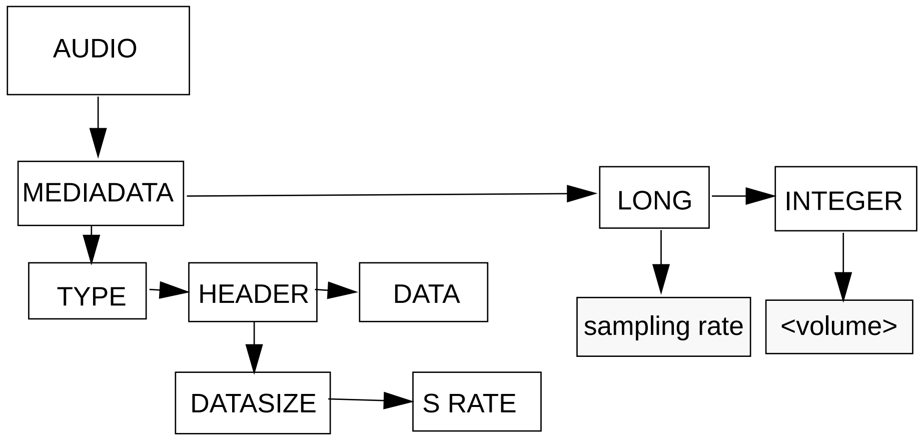


Figure 12. Multi-media Type in Views



12.1.2 Audio

The object Audio contains some data describing the chunk of audio (type of audio, header, audio data),a float value indicating the sampling rate and an integer value indicating the volume.

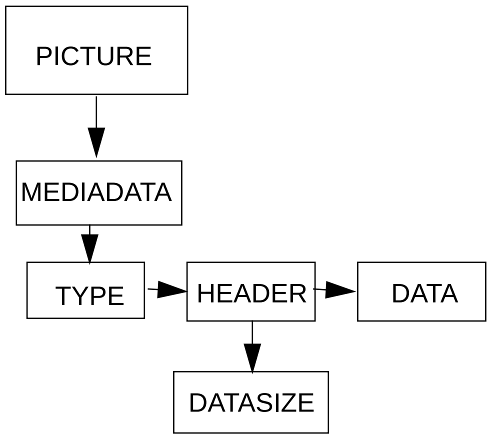


12.1.3 Text

The object Text simply contains a text string.

12.1.4 Picture

The object Picture contains some data describing the picture (type of picture, header, video data)



12.1.5 Animation

It has not been implemented yet.

12.2 Structuring types

It consists of a set of data types used for hierarchically structuring the multi-media objects. The structured objects (played in parallel or in sequence) are called compound objects.

The main data types used are:

- *MM_OBJECTS*
  it is a sequence of media objects (of type Video, Audio, ...) and compound objects.
- *PARALLEL*
  it is an object which executes in parallel the list of objects.
- *SEQUENTIAL*
  it is an object which executes in series the list of objects.

The last two objects consist of an object of type MM_OBJECTS. Figure 3 and Figure 4 show the Parallel and the Sequential types; M indicates a media object or a compound object.

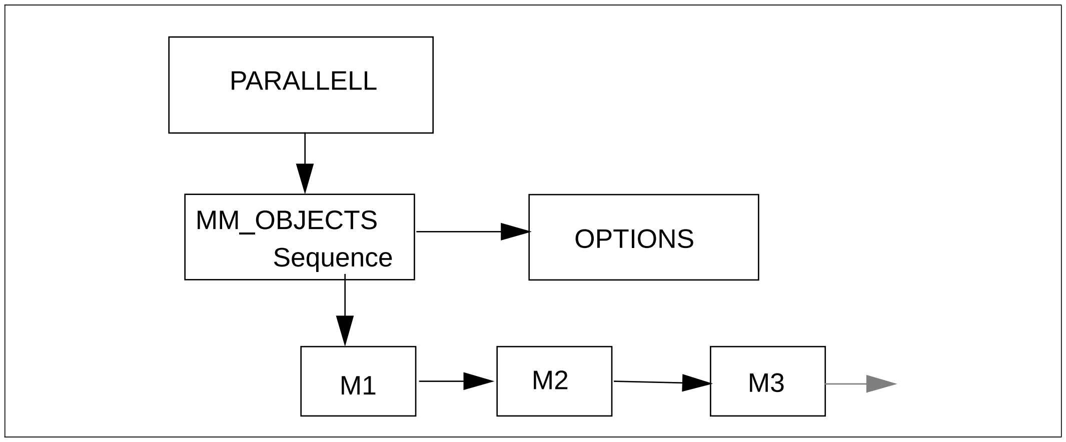


Figure 13. Parallel Type

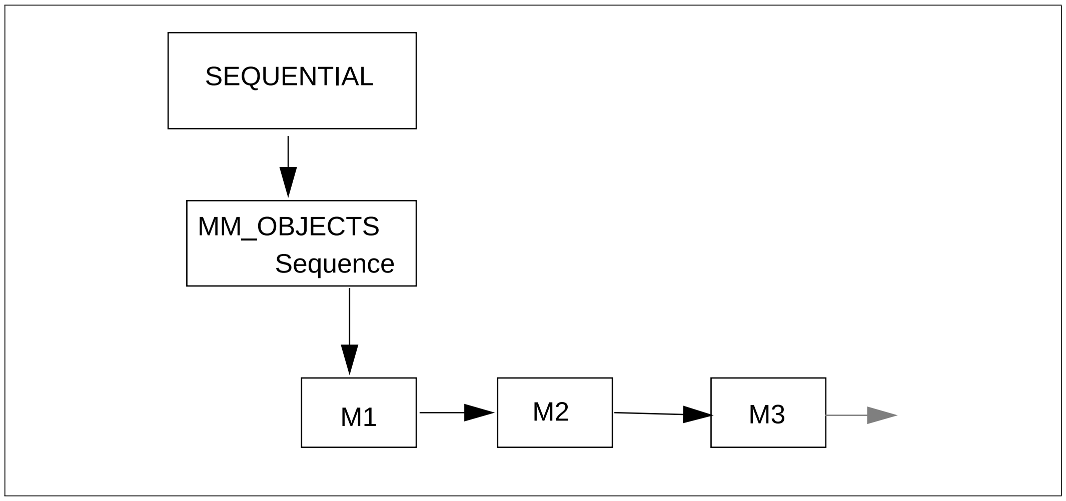
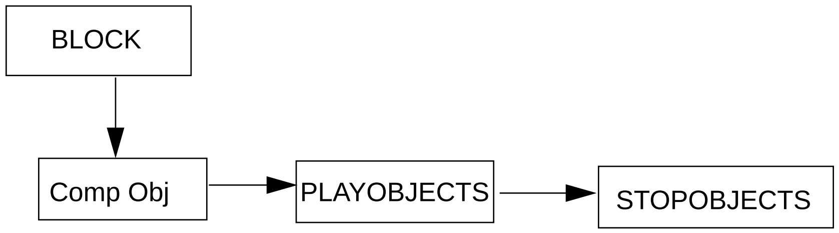


Figure 14. Sequential Type

12.3 Block type

It contains an object of type parallel, sequential, multimedia or block and a list of play and stop objects.



12.4 Conditional types

Figure 5 shows all Conditional Types. M_COMPOUND object can be a MULTIMEDIA, a PARALLEL or a SEQUENTIAL object.

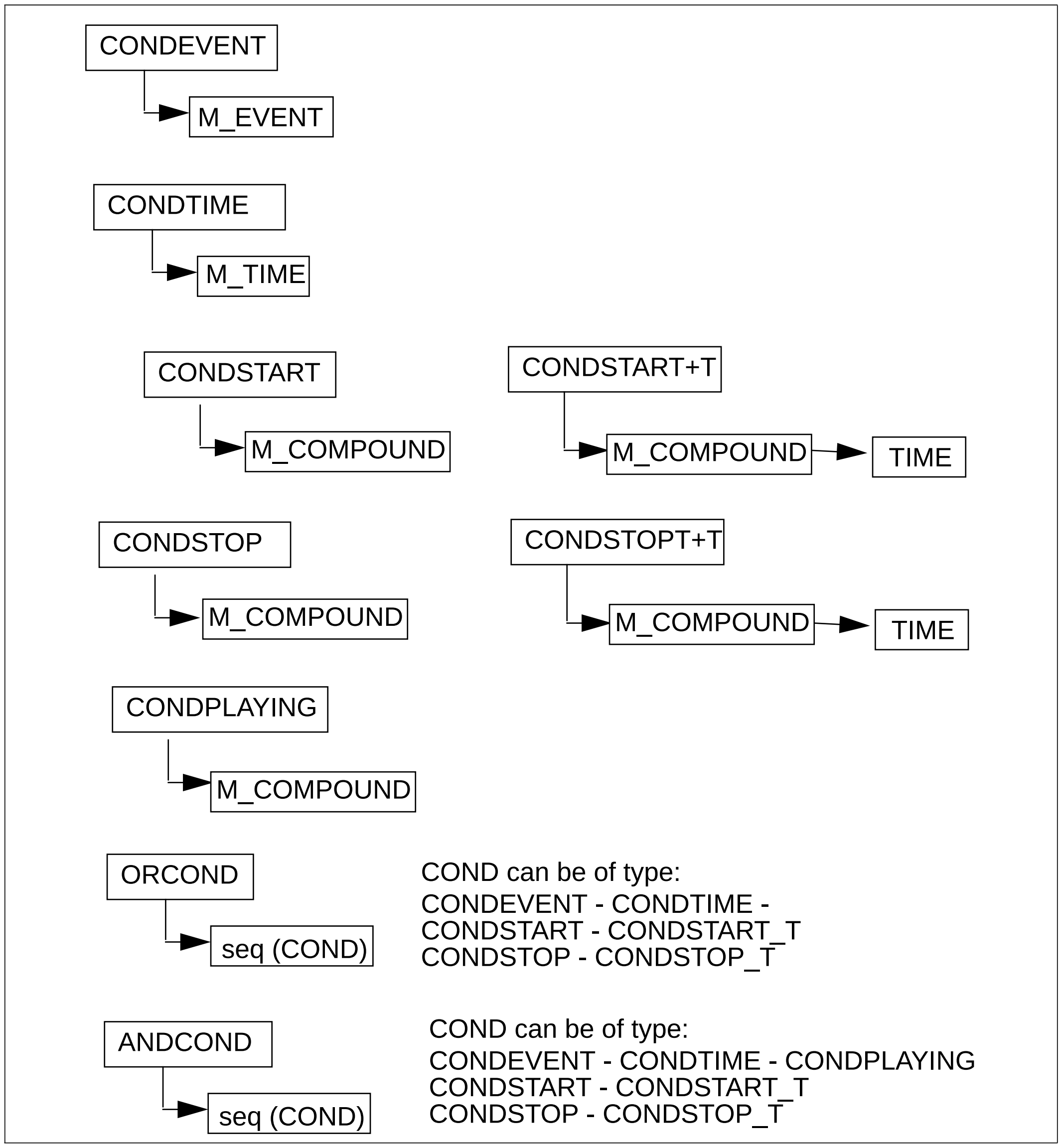


Figure 15. Conditional Types

Depending on the condition, it is possible to play or stop an object (M_COMPOUND object). The objects to do it are Play and Stop. See Figure 6, where the object COND is any type of condition.

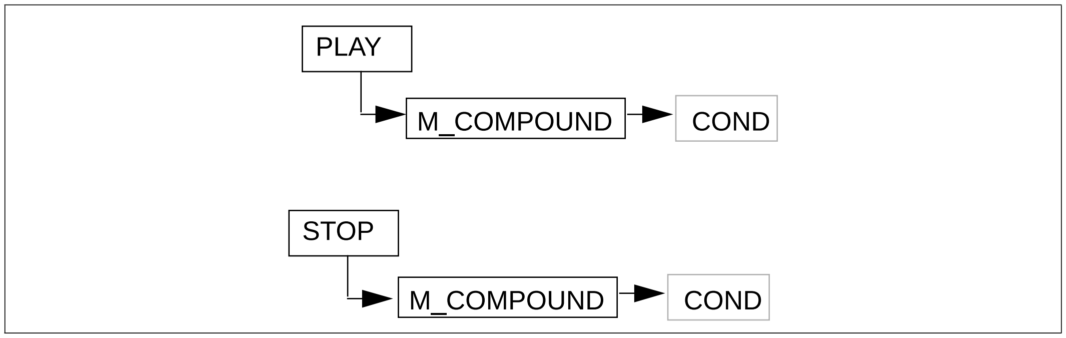
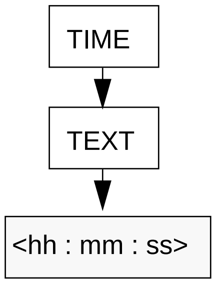


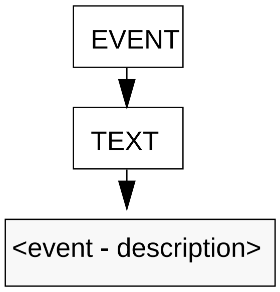
Figure 16. Play and Stop Types

12.5 Time and event types

The type Time contains an absolute value for the time.



The type Event contains the description of the external event the system is waiting for. External events may be "mouse button", "key", a signal from an application, ...



13 Playing data types

During the playing time, additional information are required, besides the one contained in the basic data types. The system creates new objects every time an object in the application is visited (or played). The object are linked by means of invariant functions. Objects requiring the concept of time are linked, by means of proper functions, to an object (Time Object) defining the system time.

In the following, these objects (named playing objects) are described.

13.1 Multi-media type

The object describing a media is now defined by the object V_OBJECT compound of a pointer to a multimedia object and two boolean objects: PlayBool and StopBool (Figure 7). The first says if the media is playing, the second if the media has finished playing. These booleans are set by other Views objects or by the external application, through the Views interface. For example, to play a media, PlayBool has to be set to true. When the media finishes playing, the StopBool is set to true.

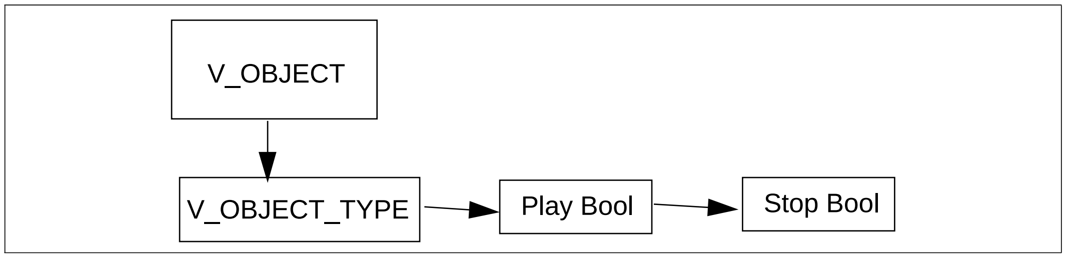


Figure 17. Multi-media type at playing time

13.2 Structuring types

The structuring types (parallel and sequential) used when the application plays, are V_SEQUENTIAL and V_PARALLEL.

The object V_SEQUENTIAL (Figure 8) contains an object of type V_LIST_OF_MM (which is a sequence of object of type V_OBJECT) and the booleans PlayBool and iStopBool whose meaning has been described above.

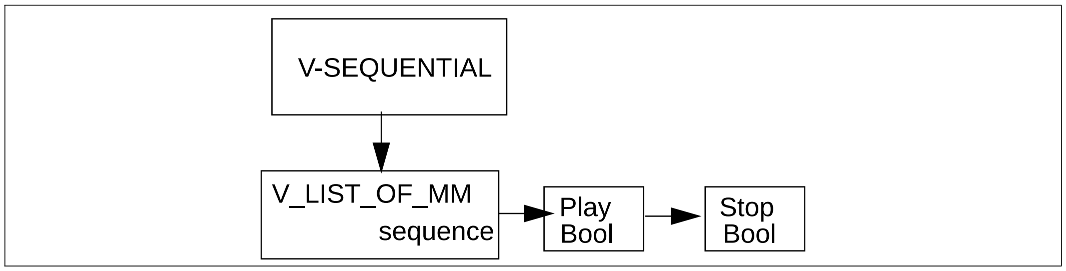


Figure 18. Sequential Type at playing time

The object V_PARALLEL (Figure 9) contains the same information of the object V_SEQUENTIAL and, furthermore, it contains two integer values: NumOfChildren (indicating the number of media constituting the object) and Counter (indicating the number of children that have been executed).

13.3 Conditional types

At the playing time, only three kinds of conditional objects are used: one to define a simple condition, one to define an OR condition and one for the AND condition.

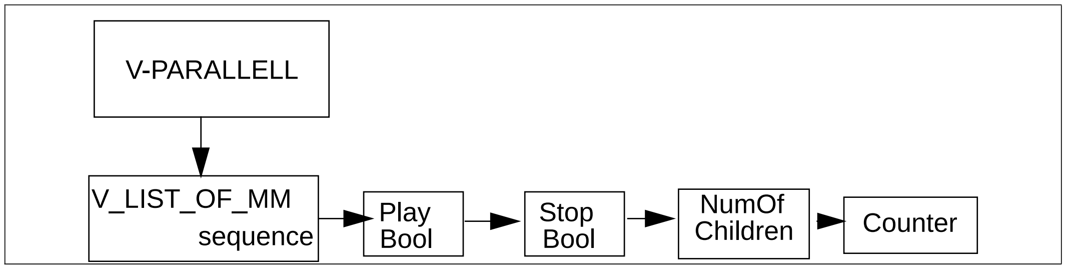


Figure 19. Parallel Type at playing time

13.3.1 Simple condition

A simple condition is defined by the object V_CONDITION, which contains a list of booleans:

- Enable Bool
  it states if the object is enable to receive events;
- Arrived Bool
  it is set to True if the expected event arrives;
- Playing Bool
  it is used to manage an AND Condition where the condition needs to check if a media (or compound) is playing.

13.3.2 OR condition

An OR condition is defined by the object V_ORCONDITION containing the Enable Bool and Arrived Bool described above.

13.3.3 AND condition

Besides the Enable Bool and Arrived Bool described above, the object V_ANDCONDITION contains the object Num Of Conditions that indicates the number of conditions linked to this object, and the object Counter indicating the number of conditions that have been satisfied.

13.4 Time and event types

The type V_TIME contains a pointer to a TIME object and a Time Bool which is set to true if the time is satisfied.

The type V_EVENT contains a pointer to an EVENT object and an Event Bool which is set to true when the expected event occurs.

14 Multi-media invariants

Invariants are used to create some relations among the objects. They are used to manage synchronization among objects, to manage conditional objects and to create links to external objects (external application).

In the following, the invariants linking the playing objects are listed.

14.1 Multi-media object: V_OBJECT

V_OBJECT ---> External

1. ToExtPlayMe:
   asks the external application to play the media when PlayBool becomes true.
2. ToExtStopMe:
   asks the external application to stop playing the media. when StopBool becomes true.

Each media object is linked to an object ticking, according to the system time. Each object can ask the Time object to start ticking and to be woken up when the period of time is elapsed.

Other invariants link a media object to parallel, sequential or conditional objects. They are described in the following.

14.2 Silence object: V_SILENCE

The Silence object is considered as a media object which does nothing for a certain time. It is connected to the system clock and receives a message of time over. It asks the clock to start counting seconds by means of the function PlaySilence.

V_SILENCE ---> External

1. PlaySilence:
   asks the clock to start counting seconds when PlayBool becomes true. When the seconds are passed, StopBool is set to false.

14.3 Time object: V_TIME

V_TIME ---> External

1. ExtTime:
   creates a link with the system clock. When the time occurs, TimeBool is set to true.

14.4 Event object: V_EVENT

V_EVENT ---> External

1. ExtEvent:
   creates a link with the system. When the expected event occurs, EventBool is set to true.

14.5 Sequential object: V_SEQUENTIAL

V_SEQUENTIAL ---> V_OBJECT

1. PlaySequence:
   it is an invariant from the object V_SEQUENTIAL to the first media child. When PlayBool of the object V_SEQUENTIAL gets true, this invariant sets PlayBool of the first media child to true.
2. ToChildStop:
   it is an invariant from the object V_SEQUENTIAL to each media children. When PlayBool of the object V_SEQUENTIAL gets false, this invariant stops all children by setting their StopBool to true.

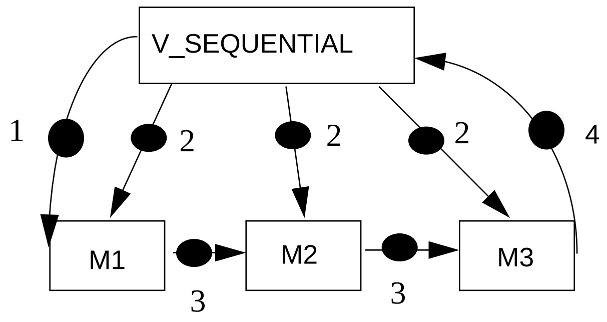
V_OBJECT ---> V_OBJECT

3. PlaySibling:
   creates a link with the system. When StopBool of a media gets true, this invariant sets PlayBool of the following media sibling to true.

V_OBJECT ---> V_SEQUENTIAL

4. FinishedSequence:
   when StopBool of the last media object gets true, this invariant sets StopBool of the object V_SEQUENTIAL to true.

An example is the following: S1 = Sequential (M1, M2, M3);



14.6 Parallel object: V_PARALLEL

V_PARALLEL ---> V_OBJECT

1. PlayParallel:
   it is an invariant from the object V_PARALLEL to each media child. When PlayBool of the object V_PARALLEL gets true, this invariant sets PlayBool of each media child to true.
2. ToChildStop:
   it is an invariant from the object V_PARALLEL to each media children. When PlayBool of the object V_PARALLEL gets false, this invariant stops all children by setting their StopBool to true.

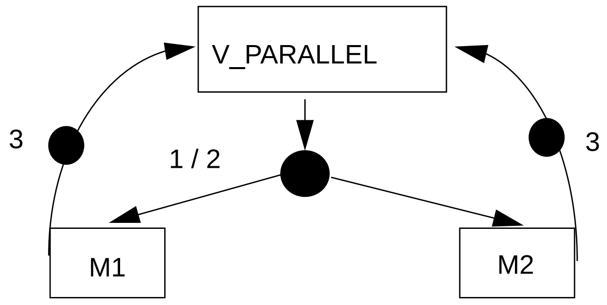
V_OBJECT ---> V_PARALLEL

3. OneParObjectFinished:
   when StopBool of one object gets true, this invariant increments the Counter variable of the object V_PARALLEl of one unit.

V_PARELLEL ---> V_PARALLEL

4. FinishedParallel:
   when Counter variable of the object V_PARALLEL is equal to the NumOfChildren value, this invariant sets StopBool of the object V_PARALLEL to true.

An example is the following: S1 = Parallel (M1, M2);



14.7 Conditions: V_CONDITION

V_CONDITION ---> V_OBJECT

1. ToObjStart:
   when ArrivedBool of the object V_CONDITION gets true, this invariant sets the PlayBool of a compound object to true.
2. ToObjStop :
   when ArrivedBool of the object V_CONDITION gets true, this invariant sets the StopBool of a compound object to true.

V_OBJECT ---> V_CONDITION

3. StartPlaying:
   when PlayBool of the media object gets true, this invariant sets the ArrivedBool of the object V_CONDITION to true.
4. StopPlaying:
   when StopBool of the media object gets true, this invariant sets the ArrivedBool of the object V_CONDITION to true.
5. StopPlaying2:
   when StopBool of the media object gets true, this invariant sets the PlayingBool of the object V_CONDITION to true. It is used with a V_ANDCONDITION.

V_SILENCE ---> V_CONDITION

3. StartPlaying:
   when PlayBool of the silence object gets true, this invariant sets the ArrivedBool of the object V_CONDITION to true.
4. StopPlaying:
   when StopBool of the silence object gets true, this invariant sets the ArrivedBool of the object V_CONDITION to true.

V_EVENT ---> V_CONDITION

6. EventCond :
   when ArrivedBool of the object V_EVENT gets true, this invariant sets the ArrivedBool of the object V_CONDITION to true.

V_TIME ---> V_CONDITION

7. TimeCond:
   when ArrivedBool of the object V_TIME gets true, this invariant sets the Arrived-Bool of the object V_CONDITION to true.

14.8 Conditions: V_ORCONDITION

V_CONDITION ---> V_ORCONDITION

8. LinkToOrCondition:
   when ArrivedBool of an object V_CONDITION gets true, this invariant sets the ArrivedBool of the object V_ORCONDITION to true an the EnableBool to false.

V_ORCONDITION ---> V_OBJECT

1. ToObjStart:
   when ArrivedBool of the object V_ORCONDITION gets true, this invariant sets the PlayBool of a compound object to true.
2. ToObjStop:
   when ArrivedBool of the object V_ORCONDITION gets true, this invariant sets the StopBool of a compound object to true.

14.9 Conditions: V_ANDCONDITION

V_CONDITION ---> V_ANDCONDITION

9. LinkToAndCondition:
   when ArrivedBool of an object V_CONDITION gets true, this invariant sets the Counter of the object V_ANDCONDITION to true.
10. ToAndFail:
    when PlayingBool of an object V_CONDITION gets true, this invariant sets the EnableBool of the object V_ANDCONDITION to false.

V_ANDCONDITION ---> V_OBJECT

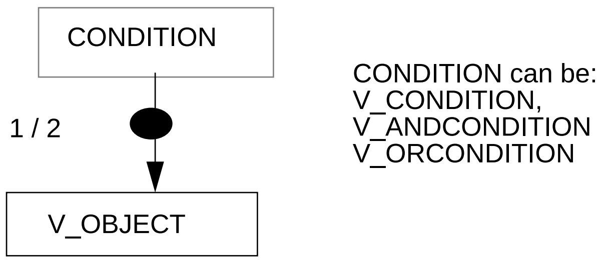
1. ToObjStart:
   when ArrivedBool of the object V_ANDCONDITION gets true, this invariant sets the PlayBool of a compound object to true.
2. ToObjStop:
   when ArrivedBool of the object V_ANDCONDITION gets true, this invariant sets the StopBool of a compound object to true.

15 From basic to playing data structure

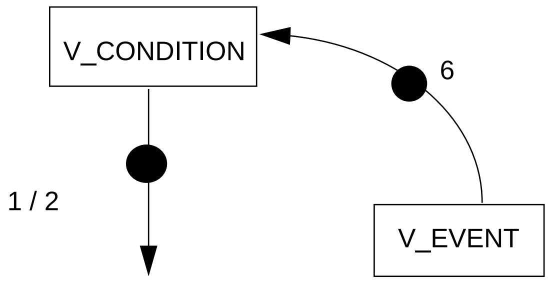
This section describes how some basic objects described in section 8.1 are converted into a playing data strcucture. The conversion consists of the creation of some objects and invariants (presented in section 1.4) to structure the application.

Some function have been implemented to convert basic objects to playing objects. The following pictures describe how basic Conditional types are rconerted into a playing structure. The numbers associated with invariants used in the following pictures are the ones adopted in the description of functions in the previous section 1.4-Conditions.

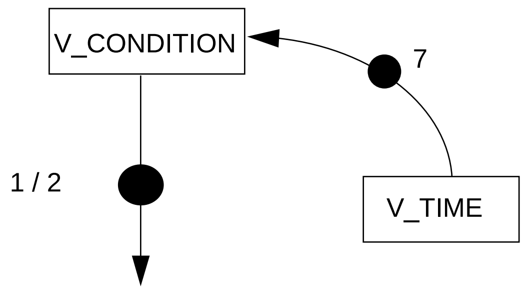
15.1 PLAY / STOP



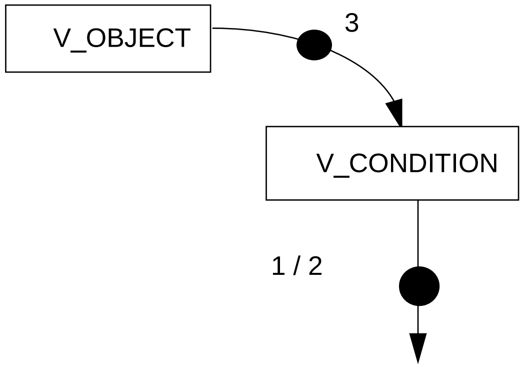
15.2 CONDEVENT --> V_CONDITION



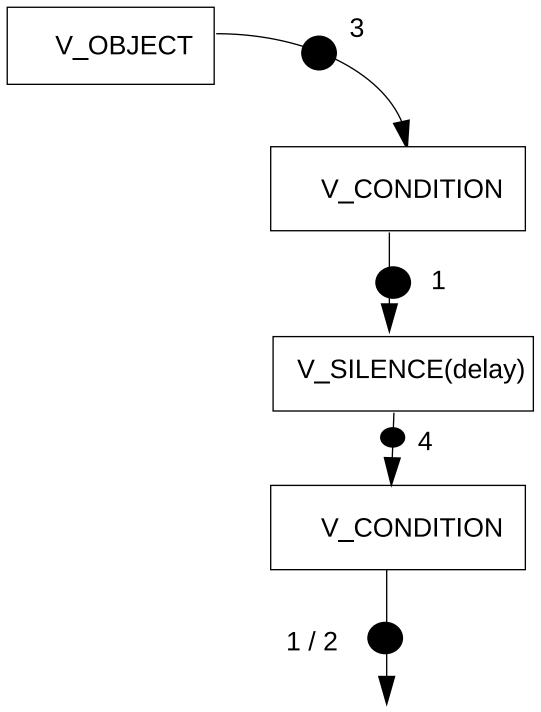
15.3 CONDTIME --> V_CONDITION



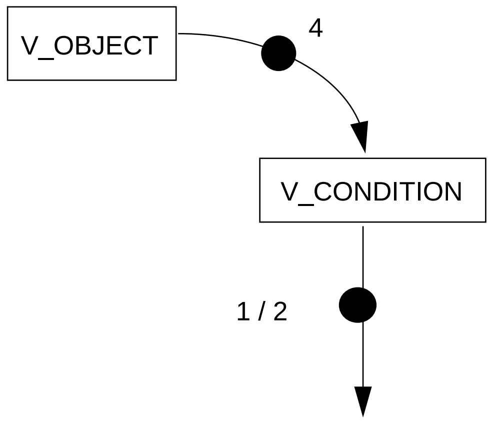
15.4 CONDSTART --> V_CONDITION



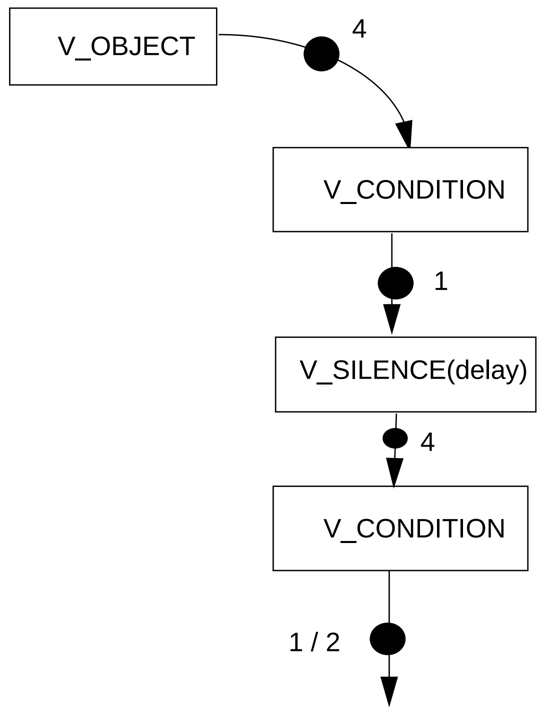
15.5 CONDSTART_T --> V_CONDITION



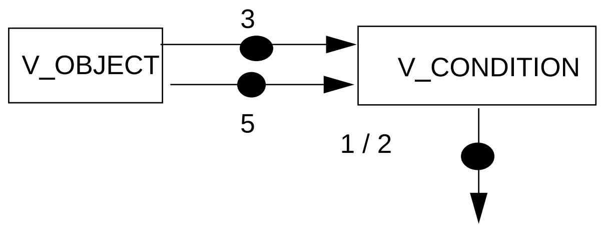
15.6 CONDSTOP --> V_CONDITION



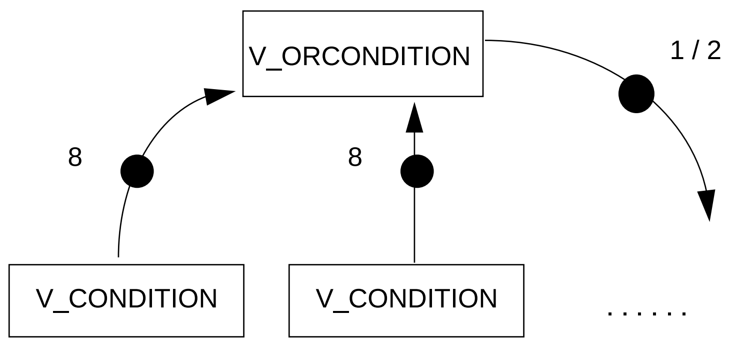
15.7 CONDSTOP_T --> V_CONDITION



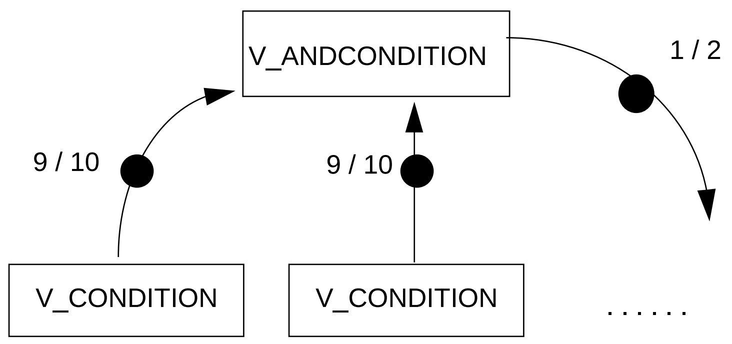
15.8 CONDPLAYING --> V_CONDITION



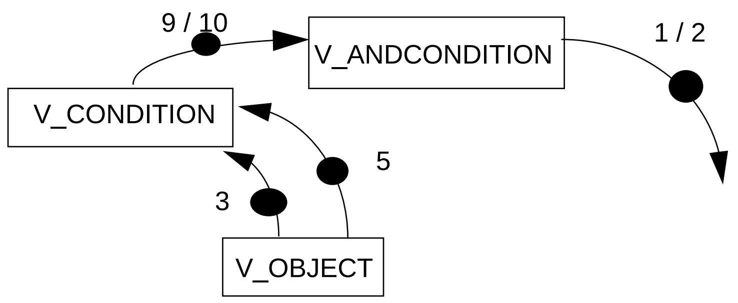
15.9 ORCONDITION --> V_ORCONDITION



15.10 ANDCONDITION --> V_ANDCONDITION



15.11 ANDCONDITION&CONDPLAYING->V_ANDCONDITION



16 Particular reductions

Some particular cases can be reduced as follows:

Case 1:

```
      C1 = ConditionStart ( M1);
      Play (M2, C1);->PlayParallel ( M1, M2 );
```

Case 2:

```
      C1 = ConditionStart ( M1, 30);
      Play (M2, C1);
         -> PlayParallel (M1, PlaySequential
            (Silence(30), M2) );
```

Case 3:

```
      C1 = ConditionStop (M1);
      Play (M2, C1);->PlaySequential ( M1, M2 );
```

Case 4:

```
      C1 = ConditionStop ( M1, 30);
      Play (M2, C1);
         ->  PlaySequential (M1,
            (Silence(30), M2);
```

17 Implementation of location of objects

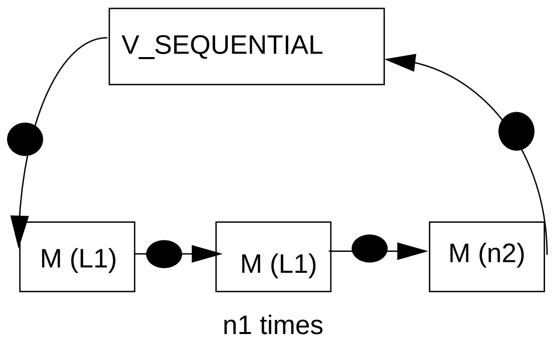
Implementation of Location of Objects within a Time Interval

17.1 STRETCH

Let L1 be the length of M, F1 its sampling rate, the new sampling rate will be (L1 / T) * F1

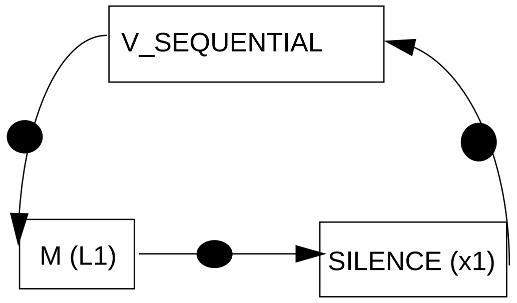
17.2 REPEAT AND CHOP

```
      n1 = T / L1 (times)
      n2 = T mod L1 (seconds)
      S1 = (M ; M ; M ; ... ) n1 times
      S2 = (S1; M | n2 )
```



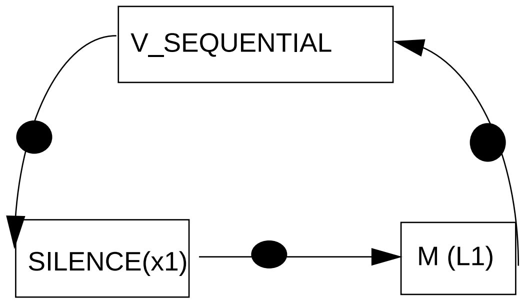
17.3 LOCATED AT THE BEGINNING

```
x1 = T - ( # M )
S1 = ( M ; x1secs )
```



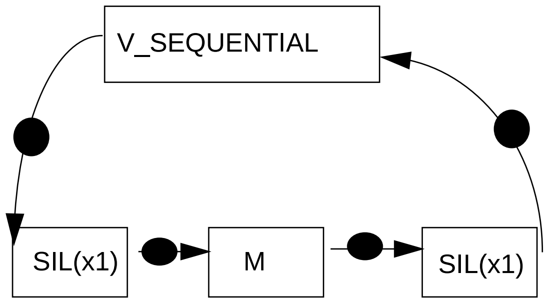
17.4 LOCATED AT THE END

```
x1 = T - ( # M )
S1 = ( x1secs ; M )
```



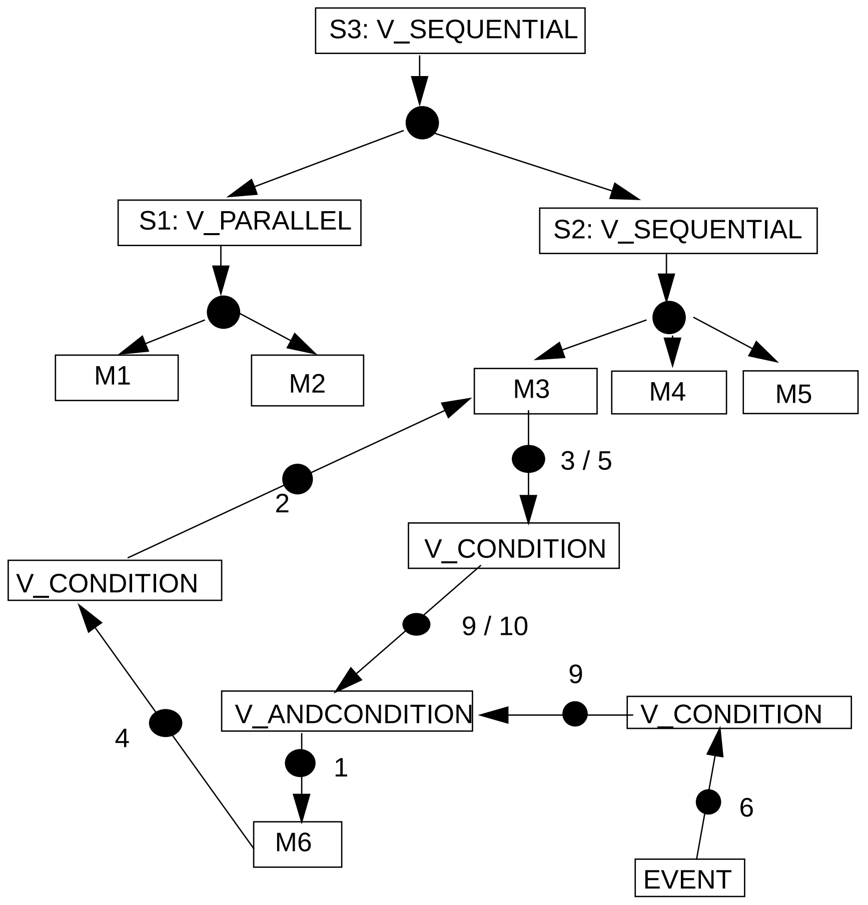
17.5 CENTRED

```
x1 = ( T - ( # M )) /2
S1 = ( x1secs ; M ; x1secs)
```



18 Example

In the following, a representation of the playing structure corresponding to the basic data objects of the following example is given: play M1 and M2 in parallel and then M3, M4 and M5 in series. A user's event can start the object M6 when M3 plays. When M6 finishes, M4 is shown.


S3: V_SEQUENTIAL
S1: V_PARALLEL
S2: V_SEQUENTIAL
M1
M2
M3
M4
M5
3 / 5
2
V_CONDITION
V_CONDITION
9 / 10
9
V_ANDCONDITION
V_CONDITION
4
1
6
M6
EVENT