MPGS: An Interactive Tool for the Specification and Generation of Multimedia Presentations

Elisa Bertino, Senior Member, IEEE, Elena Ferrari, and Marco Stolf

Abstract—Multimedia presentations are composed of objects belonging to different data types such as video, audio, text, and image. An important aspect is that, quite often, the user defining a presentation needs to express sophisticated temporal and spatial constraints among the objects composing the presentation. In this paper, we present a system (called MPGS—Multimedia Presentation Generator System) which supports the specification of constraints among multimedia objects and the generation of multimedia presentations according to the specified constraints. The constraint model provided by MPGS is very flexible and powerful in terms of the kinds of object constraints it can represent. A large number of innovative features are supported including: asynchronous and simultaneous spatial constraints; components of interest and priority levels; motion functions. Obviously, the flexibility provided to the users requires the development of nontrivial techniques to check constraint consistency and to generate a presentation satisfying the specified constraints. In this paper, we illustrate the solutions we have devised in the framework of MPGS.

Index Terms—Multimedia data, multimedia presentations, temporal/spatial synchronization, presentation consistency.

1 INTRODUCTION

The most important characteristic of a multimedia information system (MMIS) is the variety of data it must be able to support. Multimedia systems must have the capability to store, retrieve, transport, and present data with very heterogeneous characteristics and requirements, such as text, image, video, sound, and graphic. A major challenge of every multimedia system is how to synchronize data of various types, possibly coming from different sources, in order to compose multimedia presentations. This problem entails two main tasks. First, a set of powerful specification primitives must be provided, through the use of which the user can create a wide range of multimedia presentations. Second, a set of automatic tools should be developed to support the user in specifying a consistent presentation. Although several proposals exist for the specification of a multimedia presentation (which we survey in Section 7), most of them focus on the modeling aspect only and do not provide tools for checking presentation consistency and for generating the presentation according to the specified constraints. Other proposals, addressing the generation problem, provide a very simple set of specification primitives that do not fully address the requirements of applications.

In this paper, we present MPGS—Multimedia Presentation Generator System—having the goal of addressing both of the above tasks. The system, whose architecture is illustrated in Fig. 1, supports both the specification and the generation of a multimedia presentation. MPGS consists of two main environments: the presentation specification environment and the presentation generation environment. The presentation specification environment supports all functions concerning the specification of multimedia presentations. The user enters, by means of a graphical user interface, the objects he/she wishes to include in the presentation, as well as the constraints the objects must satisfy at presentation time. The constraint model provided by MPGS is very flexible and powerful in terms of the kinds of object constraints it can represent. MPGS supports the specification of both qualitative and quantitative spatio/temporal constraints among the objects composing the presentation. Qualitative constraints express the spatio/temporal relations that must occur between two objects without imposing any specific distance in time or space between the objects. By contrast, quantitative constraints support the specification of the temporal or spatial distance between two objects. To increase the expressive power of our constraint model, we have also included a new class of spatial constraints, called asynchronous constraints, that allows one to specify spatial constraints among objects whose presentation times do not necessarily overlap. For instance, an asynchronous constraint can be used to specify that two objects should be displayed in the same area even if they are not displayed at the same time. Another innovative feature is the possibility of specifying the motion of some objects during the presentation. This feature can be very useful to focus the attention of the audience on the content of a particular object.

Due to the large variety of constraints the user can specify, generating a consistent presentation is not a trivial task. To minimize the effort required to create a consistent presentation, MPGS provides an articulated approach for constraint consistency. This approach is based on a series of incremental steps, supporting the author in producing a consistent specification. The burden of checking constraint consistency is therefore moved from the author of the presentation to the system itself. A preliminary check on
constraint consistency is performed by the presentation specification environment as soon as the user enters a new constraint. If the check fails, an immediate feedback is returned to the user that can modify his/her specification. Such preliminary check is able to detect a large number of inconsistencies, thus avoiding, in most cases, the subsequent phases being executed on an incorrect specification.

After the specification process has been completed, the presentation generation environment is activated. The presentation generation environment consists of two components. The specification validator that makes a final check on the consistency of the temporal constraints (the final check for spatial constraints is performed when the presentation schedule is generated). The other component of the presentation generation environment is the presentation generator. The main feature of this component is the support for an innovative strategy to generate a presentation satisfying the specified constraints. The optimal solution to this problem, with respect to resource usage, would be to find the presentation with the lowest duration among those satisfying the specification. However, an exhaustive search among all possible presentations is expensive since it requires the examination of a huge number of temporal and spatial configurations. For this reason, we have developed a heuristic representing a compromise between precision and efficiency. The idea is to group the objects that should be included in the presentation according to the specified temporal constraints and to make the playout of objects belonging to different groups concurrent as much as possible.

MPGS provides a set of relaxation strategies to be used when no presentation exists satisfying the specified constraints. These relaxation strategies are used to automatically correct an inconsistent specification. The user can drive the relaxation process in two ways: by assigning a priority level to the objects and/or the constraints he/she includes in the specification and by identifying the components of interest, that is, the components with the highest informative content, within an object. The relaxation strategies try to generate a consistent presentation by dropping low priority objects and/or low priority constraints from the specification and by displaying only the components of interest within an object. The relaxation strategies are used by both the specification validator and the presentation generator when the specified constraints are found inconsistent.

The remainder of this paper is organized as follows: Section 2 introduces an example we will use throughout the paper. Section 3 illustrates the presentation model underlying MPGS. Sections 4 and 5 give a detailed description of the presentation specification and the presentation generation environments, respectively. Section 6 deals with implementation issues. Section 7 compares our work with related work. Finally, Section 8 concludes the paper and outlines future work. The Appendix reports the inference and compatibility rules used by MPGS to check constraint consistency.

2 AN ILLUSTRATIVE EXAMPLE

In the remainder of the paper, we refer to the following example dealing with the creation of a car company advertisement. Suppose that the author wishes to split the advertisement into two distinct parts, divided by a music break. In the first part, which should last 40 seconds, a jingle is played. Five seconds after the beginning of the jingle, the logo of the company is displayed. The jingle must be played for the entire duration of the logo playout. Eight seconds after the beginning of the presentation, a text object is displayed, which announces discounted rates. To capture the audience’s attention, the text moves on the monitor device during its playout. The playout of the text object ends at the same time as the logo playout. Then, a video of a car is shown, at a fixed position of the monitor device, while an audio explains the main features of the car. In the second part of the presentation, a graphic showing the increases in the company sales is shown at the same position where the video of the car was displayed (this requirement can be easily modeled using an asynchronous constraint). After a while, another graphic appears showing the reduction in the number of car accidents. Explanation of the graphics is given by associated audios. When the sales’ graphic disappears, an image of a crash text overlaps the accidents’ graphic. A possible scenario for this presentation is represented in Fig. 2.

In the remainder of the paper, we will show that, even for this simple example, a large number of spatial and

1. Asynchronous constraints will be described in the following section.
temporal constraints must be specified. Moreover, we will show that some of the requirements of the specification need new modeling primitives.

3 Multimedia Presentation Model

In this section, we illustrate the main notions underlying our multimedia presentation model. We first summarize the basic concepts of the multimedia object model we use and of our spatio/temporal constraint model. In particular, we present a complete taxonomy of the constraints that MPGS supports. We then introduce the notion of motion function allowing the specification of movements of objects during the presentation. Next, we introduce two features of MPGS, namely the components of interest and the priority levels, that are relevant in the constraint relaxation process.

3.1 Multimedia Objects

Multimedia objects (or objects for short) can be classified into two classes according to the output device on which their playout takes place:

- Display objects. These are multimedia objects that are displayed on a monitor device (in the remainder, DISPLAY_OBJ denotes the set of display objects). DISPLAY_OBJ is further specialized in the following subclasses: VIDEO_OBJ, representing the set of digital videos, IMAGE_OBJ, representing the set of still image objects, TEXT_OBJ, representing the set of text objects, GRAPHIC_OBJ, representing the set of graphical objects.

- Audio objects. These are multimedia objects that are played on an audio device. In the following, we denote with AUDIO_OBJ the class of audio objects.

Fig. 3 gives a graphical representation of the above classes. Edges represent the subclass/superclass relationship. We postulate the existence of a class MEDIA_OBJ representing the set of all multimedia objects.

Multimedia objects can be further classified into static and dynamic objects. A static object does not have a temporal dimension. This means that the duration of the object playout is not related to its informative content. All objects belonging to TEXT_OBJ, GRAPHIC_OBJ, and IMAGE_OBJ are static objects. By contrast, a dynamic object has an implicit temporal dimension. Instances of VIDEO_OBJ and AUDIO_OBJ are examples of dynamic objects.

Each object can be activated several times within a presentation. The overall duration of the playout of an object \( o \) is therefore represented by a set of intervals \( [st(o), et(o)] \), where \( st(o) \) denotes the starting time of the \( i \)th activation of object \( o \), and \( et(o) \) denotes the ending time of the \( i \)th activation of object \( o \), \( i = 1, \ldots, n \). Given an object \( o \), activated \( n \) times within a presentation, the term presentation time of \( o \) denotes the instants in \( \bigcup_{i=1}^{n} [st(o), et(o)] \). For simplicity, in the following, we assume each object is activated only once during the presentation.

MPGS assumes that each presentation specification contains two system-defined objects: monitor, which represents the monitor device, and pres, representing the entire presentation. The upper left corner of the object monitor is taken as the origin of the spatial reference system, whereas \( st(pres) \) is assumed as the origin of the temporal reference system.

If a user wishes to include a multimedia object in a specification, he/she must specify its spatial and/or temporal dimensions. For audio objects, only the temporal dimension is needed. For simplicity, we approximate an object \( o \) with the rectangle having the minimum area among those containing \( o \) and whose sides are parallel to the system reference axes. By abuse of notation, we refer to this rectangle as the minimum bounding rectangle (mbr) of the object. An mbr is completely specified by defining its height, width, and the spatial distance between its upper left corner and the upper left corner of the monitor object. Thus, a minimum bounding rectangle is represented by a quadruple \( (x, y, w, h) \), where \( x \) and \( y \) denote the coordinates of the upper left corner of the mbr, \( w \) denotes the mbr width, and \( h \) denotes the mbr height. In the following, we use the dot notation to indicate the elements of a tuple. For instance, given an mbr specification \( rect = (x, y, w, h) \), \( rect.x \) and \( rect.y \) denote the coordinates of the upper left corner of the mbr, whereas \( rect.w \) denotes the width of the mbr.

The temporal dimension of an object within the presentation is specified by defining its duration, that is, the distance in time between its starting and ending time.
3.2 Spatial and Temporal Constraints

MPGS supports both qualitative and quantitative spatio/temporal constraints. Qualitative constraints are used to specify the spatio/temporal relations that must occur between two objects at presentation time without imposing any specific distance in time or space between the objects (e.g., object a should be displayed before object b). By contrast, quantitative constraints specify a temporal or spatial distance between two objects (e.g., the presentation of object b should start one minute after the presentation of object a). In the following, we give an overview of the constraint model provided by our system.

3.2.1 Temporal Constraints

The qualitative temporal constraints supported by MPGS, reported in Table 1, are based on Allen’s temporal relations [2]. The prefix T in the constraints reported in Table 1, as in the other temporal constraints introduced in the following, distinguishes temporal constraints from the spatial constraints with the same name. Qualitative temporal constraints are further classified according to whether or not they require their argument objects to be simultaneous, that is, they require the presentation times of their arguments to overlap.

**Definition 1 (Simultaneous Objects).** Let \(o_i\) and \(o_j\) be two objects. Objects \(o_i\) and \(o_j\) are simultaneous if their presentation times overlap, that is: \(\exists i, j \text{ such that } st(o_i) \leq st(o_j) \leq et(o_i), i, j \in \{1, 2\}, i \neq j\).

Based on the previous definition, we can further classify temporal constraints into simultaneous temporal constraints, defined among simultaneous objects, and disjunctive temporal constraints, defined among nonsimultaneous objects. Simultaneous temporal constraints are: T_STARTS, T_DURING, T_OVERLAPS, T_EQUALS, T_MEETS, and T_FINISHES, whereas T_BEFORE is a disjunctive constraint.

As far as the quantitative aspect is concerned, MPGS supports the specification of the temporal interval between the starting times of two objects (using the temporal relation T_DELAY). The temporal relation T_DELAY \((o_1, o_2, t)\) imposes that the playout of object \(o_2\) starts \(t\) time instants after the starting time of object \(o_1\).

**Example 1.** With reference to the example in Section 2, a T_DELAY constraint can be used to express the relations between the starting time of the jingle and the starting time of the car company logo, and between the starting time of the presentation and the starting time of the moving object.

To further increase the expressive power of our constraint model, we have also included, among the possible temporal constraints, a number of additional primitives. These include: 1) the relation T_UN_PAIRED \((o_1, o_2)\), denoting that the presentation times of \(o_1\) and \(o_2\) must be disjoint; 2) the relation T_EMPTY\(^D\)(\(t_1, t_2\)), denoting that, in the interval \([t_1, t_2]\), no object should be displayed on the monitor device; 3) the relation T_EMPTY\(^A\)(\(t_1, t_2\)), denoting that, in the interval \([t_1, t_2]\), no audio object should be played. T_UN_PAIRED is a qualitative disjunctive constraint, whereas T_EMPTY\(^D\) and T_EMPTY\(^A\) are quantitative constraints.

T_EMPTY can be used, for instance, to split a presentation into several distinct parts, whereas T_UN_PAIRED can be used to require the presentation times of two objects to be disjoint without imposing any order on their presentations (by contrast, such order is imposed by T_BEFORE).

**Example 2.** With reference to the example of Section 2, the constraint T_EMPTY\(^D\) can be used to enforce the splitting of the presentation into two parts, divided by a music break.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_BEFORE((o_1, o_2))</td>
<td>(et(o_1) &lt; st(o_2))</td>
</tr>
<tr>
<td>T_MEETS((o_1, o_2))</td>
<td>(et(o_1) = st(o_2))</td>
</tr>
<tr>
<td>T_OVERLAPS((o_1, o_2))</td>
<td>(st(o_1) &lt; st(o_2) \land et(o_1) &lt; et(o_2))</td>
</tr>
<tr>
<td>T_FINISHES((o_1, o_2))</td>
<td>(st(o_2) &lt; st(o_1) \land et(o_1) = et(o_2))</td>
</tr>
<tr>
<td>T_DURING((o_1, o_2))</td>
<td>(st(o_2) &lt; st(o_1) \land et(o_1) &lt; et(o_2))</td>
</tr>
<tr>
<td>T_STARTS((o_1, o_2))</td>
<td>(st(o_1) = st(o_2) \land et(o_1) &lt; et(o_2))</td>
</tr>
<tr>
<td>T_EQUALS((o_1, o_2))</td>
<td>(st(o_1) = st(o_2) \land et(o_1) = et(o_2))</td>
</tr>
</tbody>
</table>
3.2.2 Spatial Constraints

Spatial constraints are meaningful only for objects instances of the class DISPLAY_OBJ. Qualitative spatial constraints supported by MPGS are based on the spatial relations defined in [10]. Table 2 lists the qualitative spatial constraints supported by MPGS and their semantics in terms of the temporal relations introduced in Table 1. In the table, we use \( o \times \) and \( o \wp \times \) to represent the projection on the \( x \) and \( y \) axis, respectively, of the mbr of an object \( o \). Moreover, given a temporal relation \( rel \), \( rel^{-1} \) indicates the inverse relation. Finally, \( o_1(\text{rel}_{1}, \ldots, \text{rel}_{n})o_2 \) is used as a compact notation for \( o_1\text{rel}_1o_2 \lor \ldots \lor o_{n}\text{rel}_{n}o_2 \), whereas temporal relations are represented by their first letter (e.g., \( B \) for \( \text{BEFORE} \), \( M \) for \( \text{MEETS} \), and so on). Spatial constraints are characterized by the prefix \( S \), to distinguish them from the analogous temporal constraints.

As far as quantitative spatial constraints are concerned, our model allows one to specify the distance between the upper left corners of the minimum bounding rectangles of two objects. This can be done by means of the spatial constraint \( S\_\text{DISTANCE} \). Given two objects \( o_1 \) and \( o_2 \), \( S\_\text{DISTANCE}(o_1, o_2, x_\text{dist}, y_\text{dist}) \) requires the distance along the \( x \)-axis between the upper left corner of the mbrs of \( o_1 \) and \( o_2 \) to be \( x_\text{dist} \) and the distance along the \( y \)-axis to be \( y_\text{dist} \). Formally, \( S\_\text{DISTANCE}(o_1, o_2, x_\text{dist}, y_\text{dist}) \) imposes the following constraints: \( o_2.x - o_1.x = x_\text{dist} \) and \( o_2.y - o_1.y = y_\text{dist} \).

For instance, an \( S\_\text{DISTANCE} \) constraint can be used in the car company advertisement example to specify that the video of the car must appear in a given position on the monitor device.

Sometimes, it would be useful to define spatial constraints among objects whose presentation times do not necessarily overlap (that is, nonsimultaneous objects). For this reason, our system supports the notion of asynchronous spatial constraints. Such spatial constraints do not impose any requirement on the presentation times of the two objects. The objects can also be displayed during disjoint temporal intervals. By contrast, constraints that impose the presentation times of their arguments to overlap are called simultaneous spatial constraints.

Example 3. In our example, a simultaneous spatial constraint can be used to specify that the image of the crash text should overlap the graphic of the car accidents at some point in the presentation, whereas an asynchronous constraint can be used to specify that the video of the car and the graphic of the car sales should be displayed in the same area, but not necessarily at the same time.

Spatial constraints supported by MPGS can therefore be classified into the following groups: 1) simultaneous spatial constraints, defined between simultaneous objects belonging to the class DISPLAY_OBJ; and 2) asynchronous spatial constraints, defined between any pair of objects belonging to DISPLAY_OBJ. Simultaneous spatial constraints are identified by the prefix \( SS \), whereas asynchronous spatial constraints are identified by the prefix \( SA \). Simultaneous spatial constraints supported by MPGS are: \( SS\_\text{DISTANCE} \), \( SS\_\text{MEETS} \), \( SS\_\text{OVERLAPS} \), \( SS\_\text{INCLUDES} \), \( SS\_\text{EQUALS} \), and \( SS\_\text{DISTANCE} \), whereas the asynchronous are: \( SA\_\text{DISTANCE} \), \( SA\_\text{MEETS} \), \( SA\_\text{OVERLAPS} \), \( SA\_\text{INCLUDES} \), \( SA\_\text{EQUALS} \), and \( SA\_\text{DISTANCE} \).

In the following, we use the prefix \( S \) to denote an asynchronous or a simultaneous spatial constraint when no distinction is needed. For instance, \( SA\_\text{EQUALS} \) denotes both \( SA\_\text{EQUALS} \) and \( SS\_\text{EQUALS} \).

### Table 2

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_\text{DISTANCE}(o_1, o_2) )</td>
<td>( o_1^x { T_T, T_B, B_B } \lor o_2^x { T_B, T_B, B_B } \lor o_1^y { T_T, T_B, B_B } \lor o_2^y { T_T, T_B, B_B } \lor o_1 \land o_2 \lor \neg o_1 \land o_2 )</td>
</tr>
<tr>
<td>( S_\text{MEETS}(o_1, o_2) )</td>
<td>( o_1^x { T_M, T_M } \lor o_1^y { T_M, T_M } \lor o_1 \land o_2 \lor \neg o_1 \land o_2 )</td>
</tr>
<tr>
<td>( S_\text{OVERLAPS}(o_1, o_2) )</td>
<td>( o_1^x { T_D, T_D } \lor o_1^y { T_D, T_D } \lor o_1 \land o_2 \lor \neg o_1 \land o_2 )</td>
</tr>
<tr>
<td>( S_\text{INCLUDES}(o_1, o_2) )</td>
<td>( o_1^x { T_D, T_D } \lor o_1^y { T_D, T_D } \lor o_1 \land o_2 \lor \neg o_1 \land o_2 )</td>
</tr>
<tr>
<td>( S_\text{EQUALS}(o_1, o_2) )</td>
<td>( o_1^x { T_T, T_T } \lor o_1^y { T_T, T_T } \lor o_1 \land o_2 \lor \neg o_1 \land o_2 )</td>
</tr>
</tbody>
</table>

3. Spatial distances are expressed in pixels.
4. By abuse of notation, here and in the remainder of the discussion, we denote as \( o \times \) the position along the \( x \)-axis of the upper left corner of the mbr of object \( o \). A similar notation is used to denote all the other components in the specification of the mbr associated with \( o \).
### 3.3 Motion Functions

Sometimes it would be useful to include in a presentation objects which do not always maintain the same position during their activations but which move on the monitor device according to some predefined functions. For instance, with reference to the example in Section 2, the movement of the object announcing discounts on sales is used to focus the attention of the viewer on the content of this object.

To support these requirements, MPGS allows the specification of object motion, formally defined as follows.

**Definition 2 (Motion Function Specification).** A motion function specification is a quadruple \((o, vel_x, vel_y, flag)\), where \(o\) is an object belonging to \(\text{DISPLAY}_{-}\text{OBJ}\), \(vel_x\) and \(vel_y\) denote the speed of object \(o\) along the \(x\) and \(y\) axis, respectively, and \(flag \in \{\text{yes}, \text{no}\}\) denotes whether the motion is periodic (\(flag = \text{"yes"}\)) or not (\(flag = \text{"no"}\)).

If \(flag = \text{"no"}\), then the object ends its motion when it reaches the monitor boundaries. By contrast, \(flag = \text{"yes"}\) denotes a periodic motion: Each time the object reaches the monitor boundaries, it reverses its movement until its presentation time expires.

**Example 4.** With reference to the example in Section 2, the movement of the object containing information on discounted rates can be specified by means of the following motion function specification: \((\text{Discounted} \_\text{Rates} \_\text{Txt}, 8, 0, \text{yes})\), specifying that the object \(\text{Discounted} \_\text{Rates} \_\text{Txt}\) moves periodically 8 pixels a second along the \(x\) axis.

Note, moreover, that, for simplicity, our system supports only vertical and horizontal motion. Therefore, we require that, for each motion function specification \((o, vel_x, vel_y, flag)\), one and only one between \(vel_x\) and \(vel_y\) be not null. Moreover, we assume that an object which dynamically changes its spatial position cannot be involved in any spatial constraint and that it can be involved in a temporal constraint only if this constraint imposes an absolute temporal distance between the starting time of the object and the starting time of the presentation. We do not consider other cases here since they complicate the implementation without rising any relevant issue. However, such assumptions could easily be relaxed in a future version of MPGS.

### 3.4 Components of Interest

MPGS supports a set of relaxation strategies to be used when the spatial and/or temporal constraints in the specification cannot be satisfied at run-time. The relaxation process can be driven by the user in two ways. First, the user can specify the components of interest within an object, that is, the elements that it is mandatory to include in the presentation.

**Example 5.** Consider Fig. 4, which shows the image of a country house. Suppose that the user specifies that the component of interest of this image is the one contained in the box. If the constraints in the specification make the entire image unrenderable, the system tries anyhow to present the image by displaying only the house without the park.

The specification of the components of interest within an object depends on its temporal dimension. For dynamic objects, a component of interest is specified by defining its

---

5. Speed is expressed in pixel/seconds.
6. We will elaborate on this in Section 5.2.
starting and ending time with respect to the starting time of the object in which the component is contained. For static objects, a component of interest is specified by defining the relative position of its mbr with respect to the mbr of the object in which the component is contained. The following definitions formalize these concepts.

Definition 3 (Component of Interest Specification for Dynamic Objects). Let \( o \) be a dynamic object. The specification of a component of interest within \( o \) is a triple \((o, t_{min}, t_{max})\), where \( t_{min} \) is the temporal distance between the starting time of \( o \) and the starting time of the component of interest and \( t_{max} \) is the temporal distance between the starting time of object \( o \) and the ending time of the component of interest.

Definition 4 (Component of Interest Specification for Static Objects). Let \( o \) be a static object. The specification of a component of interest within \( o \) is a pair \((o, mbr\_spec)\), where \( mbr\_spec \) is the specification of the minimum bounding rectangle of the component of interest with respect to the minimum bounding rectangle of object \( o \), that is, \( mbr\_spec = (x, y, w, h) \), where \( x \) and \( y \) are the distances along the \( x \) and \( y \) axis, respectively, between the upper left corner of the mbr of \( o \) and the upper left corner of the mbr of the component of interest, \( w \) and \( h \) are the width and height of the mbr of the component of interest, respectively.

3.5 Priority Levels
The second method by which the user can drive the relaxation process is by assigning a priority to each object, component of interest or constraint he/she includes in the specification. The priority measures the relevance of the object or constraint in the presentation. Low priority objects (resp. constraints) are objects (resp. constraints) that it would be better to include in the presentation, but that can also be dropped from the presentation if some constraints cannot be satisfied. If no presentation exists satisfying the requirements imposed by the user, the system tries to generate a consistent presentation by dropping low priority objects and low priority constraints from the presentation.

The priority assigned by the user is an integer number in the range \([1, ..., 100]\). To make the priority specification easier, the system supports a set of predefined priority levels, illustrated in Table 4, uniformly distributed in the priority range. If the user does not specify any priority level, the default value NORMAL is automatically assigned by the system. Moreover, when the user specifies a priority for a given object, this priority is automatically assigned to the components of interest within the object. However, the system allows the user to assign a different priority to the components of interest within an object by changing their default priority.

4 Presentation Specification Environment
In this section, we introduce the main features of the presentation specification environment (cf. Fig. 1). We start by introducing the notion of presentation specification. Then, we illustrate how MPGS supports the user in the specification task.

4.1 Presentation Specification
In this section, we formally introduce the notion of presentation specification. We start by introducing the definition of object and constraint specification.

Definition 5 (Object specification). An object specification is a tuple \((o, duration, mbr\_spec, ic\_spec, prio)\), where:

- \( o \) is the object name;
- \( duration \) is the duration of the playout of object \( o \), that is, \( duration = e\ell(o) - s\ell(o) \);
- \( mbr\_spec \) is a pair \((h, w)\), where \( h \) denotes the height of the minimum bounding rectangle of object \( o \) and \( w \) denotes its width;
- \( ic\_spec \) is a set which gives information on the components of interest within \( o \) and on their priority. Each item in this set is a pair \((ic\_def, ic\_prio)\), where \( ic\_def \) is the specification of a component of interest within \( o \), and \( ic\_prio \in \mathbb{N} \) is the corresponding priority.
- \( prio \in \mathbb{N} \) is the priority assigned to object \( o \).

Obviously, the specification of a component of interest must be consistent with the specification of the object in which the component is included. In case of a static object, we require that the mbr of a component of interest is contained into the mbr of the object to which the component of interest belongs to. In case of dynamic objects, we require that the presentation time of the component of interest is less than the presentation time of the object in which the component is included. Checks on the consistency of the interesting component specifications are automatically performed by the system upon the insertion of a new object specification. We will further discuss these checks in Section 4.2.
According to Definition 5, several components of interest can be defined for the same object. However, for the sake of simplicity, in the current implementation of MPGS, we assume that at most one component of interest is associated with a given object.

Note moreover that Definition 5 above implies that the user must specify the duration of each object he/she wishes to include in the presentation. Often, the playout of two related objects must start and end at the same instants. This is the case, for instance, of an image object and of the related audio object explaining its meaning. To make the specification of such requirements easier, the system allows the user to omit the duration of an object in the specification. In this case, the user must implicitly specify the object duration through the use of constraints. The following example illustrates the discussion.

Example 6. Consider the video of the car in our example and the associated audio object. The user can omit the duration of the audio object by specifying a T_EQUALS constraint, imposing that its duration is equal to the duration of the video. Using this approach, each time the duration of the video (resp. of the audio object) is modified, the duration of the audio object (resp. of the video) is automatically updated according to the duration of the video (resp. of the audio).

Table 5 reports a possible object specification for the objects in our example. For simplicity, we assume that each object has the default priority and that no components of interest have been specified. We therefore omit these two components from the table.

Table 5: Object Specification for Our Example

<table>
<thead>
<tr>
<th>Object</th>
<th>Duration</th>
<th>Mbr spec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>monitor</td>
<td>20</td>
<td>(23,20)</td>
</tr>
<tr>
<td>Jingie</td>
<td>15</td>
<td>(10,7)</td>
</tr>
<tr>
<td>Logo_Img</td>
<td>12</td>
<td>(3,10)</td>
</tr>
<tr>
<td>Discounted_Rates_Txt</td>
<td>20</td>
<td>(7,10)</td>
</tr>
<tr>
<td>Car_Vd_Audio</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Sales_Grp</td>
<td>22</td>
<td>(7,10)</td>
</tr>
<tr>
<td>Sales_Grp_Audio</td>
<td>10</td>
<td>(2,10)</td>
</tr>
<tr>
<td>Accidents_Grp</td>
<td>2</td>
<td>(7,10)</td>
</tr>
<tr>
<td>Accidents_Grp_Audio</td>
<td>19</td>
<td>(2,10)</td>
</tr>
<tr>
<td>Crash_Text_Img</td>
<td>19</td>
<td>(9,10)</td>
</tr>
</tbody>
</table>

A constraint specification is formally defined as follows:

Definition 6 (Constraint Specification). A constraint specification is a triple (cnstr_name;params;prio), where: 1) cnstr_name is the name of the constraint; 2) params = p1;p2 are the constraint parameters; and 3) prio ∈ IN is the priority assigned to the constraint.

Example 7. The temporal requirements of our example are expressed by the following constraint specifications:

(T_DELAY, Logo_Img,Jingle,5),
(T_DELAY,Discounted_Rates_Txt,pres,8),
(T_MEETS,Logo_Img,Car_Vd),
(T_EQUALS,Car_Vd,Car_Vd_Audio), (T_EMPTY,D,41,47),
(T_DELAY,Musıc_Break_Audio,pres,41),
(T_STARTS,Sales_Grp_Audio,Sales_Grp),
(T_EQUALS,Sales_Grp_Txt,Sales_Grp),
(T_EQUALS,Accidents_Grp_Audio,Accidents_Grp),
(T_EQUALS,Accidents_Grp_Txt,Accidents_Grp),
(T_BEFORE,Sales_Grp_Audio,Accidents_Grp_Audio),
(T_BEFORE,Sales_Grp,Car_Vd_Txt)
(T_OVERLAPS,Sales_Grp,Accidents_Grp).

The spatial constraints are as follows:

(S_Distance,monitor,Car_Vd,5,7),
(8_EQUALS,Car_Vd,Sales_Grp),
(SS_OVERLAPS,Crash_Text_Img,Accidents_Grp).

We are now ready to introduce the definition of presentation specification.

Definition 7 (Presentation Specification). A presentation specification is a quadruple (OS;TRS;SRS;MFS), where OS is a set of object specifications, TRS is a set of temporal constraint specifications, SRS is a set of spatial constraint specifications, and MFS is a set of motion function specifications.

Note that the OS component of each presentation specification cannot be empty since at least one object must be included in the specification. By contrast, all the other components of the specification can be empty. If the TRS component is null, it means that no constraints are imposed on the presentation times of the objects composing the presentation. Similarly, if the SRS component is empty, no spatial constraints are specified among the objects composing the presentation. In this case, MPGS assumes by default that the priority is IN.

We omit priority since we assume the default priority for all the specified constraints.
that the mbtrs of the specified objects are disjoint. Finally, if
\(MFS\) is empty, all the objects in the presentation maintain
the same spatial position for all the instants of their playout.

### 4.2 Entering the Specifications

When the user inserts or modifies an entity in a specification,
the system performs a preliminary check on the entity
correctness. Because such checks may require the revision
of the presentation, they are carried out through interactions
with the user. Such preliminary checks are crucial for
optimization since they greatly reduce the probability of
activating the subsequent phases on an inconsistent
specification. Preliminary checks differ depending whether
they deal with objects, motion functions, or constraints. We
brieﬂy discuss them in what follows.

When a user includes a new object in the speciﬁcation,
the system ﬁrst veriﬁes that its speciﬁcation is consistent
with the speciﬁcation of its components of interest (if any),
according to the criteria discussed in Section 4.1. Then, if the
object belongs to \(DISPLAY\_OBJ\), the system checks whether
the object is renderable, that is, whether its dimensions do
not exceed the monitor dimensions. If the check fails, the
system veriﬁes whether the dimensions of its components
of interest (if any) do not exceed the monitor dimension. If
this further check also fails, the system refuses the insertion
of the object.

Additionally, each time the user enters a motion
function, the system makes some preliminary checks on
its correctness. The checks, which are summarized in
Table 6, basically verify whether the speciﬁcation is consistent
with the formal deﬁnition of motion function and whether
the constraints involving the objects on which the motion
function is speciﬁed satisfy the restrictions we have imposed (cf. Section 3.3).

<table>
<thead>
<tr>
<th>Motion function</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>((o, vell_x, vell_y, flag))</td>
<td>(o \in DISPLAY_OBJ), (vell_x, vell_y \in \mathbb{N})</td>
</tr>
<tr>
<td></td>
<td>(vell_x \neq 0 \iff vell_y = 0)</td>
</tr>
<tr>
<td></td>
<td>(vell_y \neq 0 \iff vell_x = 0)</td>
</tr>
<tr>
<td></td>
<td>(\text{Vcnst}_{\text{spec}} \in S_RS) (cnst_\text{spec}_\text{params}) does not contain object (o)</td>
</tr>
<tr>
<td></td>
<td>(\text{Vcnst}_{\text{spec}} \in T_RS) such that (\text{cnst}_\text{spec}_\text{params}) contains object (o): (\text{cnst}_\text{spec}_\text{name} = \text{T_DELAY})</td>
</tr>
</tbody>
</table>

Further checks are performed when a \(T\_\text{EMPTY}\) constraint is specified. We recall that a \(T\_\text{EMPTY}\) constraint
takes as argument two time instants \(t_1\) and \(t_2\) and requires
that no audio or display objects, depending on the type of
the \(T\_\text{EMPTY}\) constraint, is played during the interval \([t_1, t_2]\).
Thus, a \(T\_\text{EMPTY}\) constraint is used to split the presentation
into distinct parts. When a \(T\_\text{EMPTY}\) constraint is entered,
the system asks the user to specify in which interval,
between those deﬁned by the \(T\_\text{EMPTY}\) relation, the objects
already included in the speciﬁcation must be placed. Then,
it makes a further check on the constraints in which such
objects are involved. The following example illustrates the
discussion.

**Example 8.** Let

\[DO = \{\text{Logo\_Img, Car\_Vd, Sales\_Grp, Sales\_Grp\_Txt,}\]

\[\text{Accidents\_Grp, Accidents\_Grp\_Txt, Crash\_Text\_Img}\]

be the set of display objects of our example. The presentation of the display objects is split into two parts by the constraint \(T\_\text{EMPTY}^{41,47}(\text{Sales\_Grp})\) (cf. Example 7); the
first part is from the starting time of the presentation to
40, whereas the second part is from instant 48 to the ending
time of the presentation. In the interval \([41, 47]\) no display object is displayed. Thus, the system
asks the author which ones among the objects in \(DO\)
should be displayed in the interval \([41, 47]\) (cf. Example 7); the
other elements in \(DO\) should be displayed in the interval \([48, et(\text{pres})]\).
According to our example, the author speciﬁes that \(\text{Logo\_Img}\) and
\(\text{Car\_Vd}\) should be displayed in the interval \([48, et(\text{pres})]\), whereas the other elements in \(DO\)
should be displayed in the interval \([40, 41]\).

When the user speciﬁes that an object \(o\) should be
displayed in the interval \([t_1, t_2]\) the system checks that: 1) the
duration of object \(o\) does not exceed \(t_2 - t_1\); 2) the temporal

### Table 7

**Preliminary Checks on User-Deﬁned Constraints**

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S_\text{DISJOINT}(o_1, o_2))</td>
<td>(o_1, h + o_2, h &lt; \text{monitor}_h \lor o_1, w + +o_2, w &lt; \text{monitor}_w)</td>
</tr>
<tr>
<td>(S_\text{MEETS}(o_1, o_2))</td>
<td>(o_1, h + o_2, h &lt; \text{monitor}_h \lor o_1, w + +o_2, w &lt; \text{monitor}_w)</td>
</tr>
<tr>
<td>(S_\text{INCLUDES}(o_1, o_2))</td>
<td>(o_1, h \geq o_2, h \land o_1, w \geq o_2, w \land -((o_1, h = \infty) \land (o_1, w = o_2, w)))</td>
</tr>
<tr>
<td>(S_\text{EQUALS}(o_1, o_2))</td>
<td>(o_1, h = o_2, h \land o_1, w = o_2, w)</td>
</tr>
<tr>
<td>(T_\text{FINISHES}(o_1, o_2))</td>
<td>(o_1, \text{duration} &gt; o_2, \text{duration})</td>
</tr>
<tr>
<td>(T_\text{DURING}(o_1, o_2))</td>
<td>(o_1, \text{duration} &lt; o_2, \text{duration})</td>
</tr>
<tr>
<td>(T_\text{STARTS}(o_1, o_2))</td>
<td>(o_1, \text{duration} &lt; o_2, \text{duration})</td>
</tr>
<tr>
<td>(T_\text{EQUALS}(o_1, o_2))</td>
<td>(o_1, \text{duration} = o_2, \text{duration})</td>
</tr>
</tbody>
</table>
constraints in which \( o \) is involved can be satisfied. The following example illustrates the discussion.

**Example 9.** Consider once again our example. According to Example 8 above, object Logo_Img should be displayed in the interval \([st(pres), 40]\). When the user enters this requirement, the system first checks whether Logo_Img\_duration \( \leq 40 \). In this case, the check succeeds since, according to Table 5, Logo_Img\_duration = 15. Then, the system considers each spatial and temporal constraint in which Logo_Img is involved to verify whether the display of object Logo_Img in the interval \([st(pres), 40]\) makes it unsatisfiable. Consider, for instance, the constraint \( T\_MEETS(\text{Logo}_\text{Img}, \text{Car}_\text{Vd}) \) (cf. Example 7). The system verifies that:

1) the author had specified that \( \text{Car}_\text{Vd} \) should also be displayed in the interval \([st(pres), 40]\);
2) \( \text{Car}_\text{Vd}\_duration + \text{Logo}_\text{Img}\_duration \leq 40 \).

The specification is accepted since both the above conditions are satisfied; otherwise, the author would be asked to revise the specification. A similar check is performed for all the other constraints involving Logo_Img.

By contrast, when a new constraint, except \( T\_EMPTY \), is specified, the system computes all the qualitative constraints that can be inferred from it and from the constraints specified till that point, and checks the consistency of this set. The inference and consistency check process is based on the rules reported in the Appendix. It is important to note that such rules exploit the relationships between temporal and spatial constraints. For instance, suppose a user specifies that the presentation time of two objects should be disjoint; if he/she subsequently specifies that there must exist at least an instant at which the two objects should be displayed in the same area (by means of a simultaneous spatial constraint), an inconsistency arises.

Deriving the qualitative constraints that can be inferred from the user-defined constraints and checking their consistency is an incremental process which involves several steps. Each time the user includes a new constraint in the specification, the system first verifies whether the constraint is qualitative or quantitative. Since, at this stage, we are interested in checking the consistency of qualitative constraints, each quantitative constraint specified by the user is temporarily mapped onto an equivalent qualitative constraint. The method for mapping a quantitative constraint into an equivalent qualitative constraint is as follows.

Given a quantitative spatial constraint \( S\_DISTANCE(o_1, o_2, x_{dist}, y_{dist}) \), we assume the coordinates of the upper left corner of the mbr of object \( o_1 \) as the origin of our reference system. Then, we determine the qualitative spatial constraint relating objects \( o_1 \) and \( o_2 \), by checking which one of the conditions in Table 2 is satisfied by the mbrs of \( o_1 \) and \( o_2 \).

Similarly, given a quantitative temporal constraint \( T\_DELAY(o_1, o_2, t) \), we assume, as our instant zero, the starting time of object \( o_1 \). Then, we determine the qualitative temporal constraint relating object \( o_1 \) and \( o_2 \) by checking which one of the conditions in Table 1 is satisfied by the presentation times of \( o_1 \) and \( o_2 \).

**Example 10.** Consider the constraint specifications of Example 7. The quantitative temporal constraint \( T\_DELAY(\text{Logo}_\text{Img}, \text{Jingle}, 5) \) is mapped onto the qualitative constraint \( T\_FINISHES(\text{Logo}_\text{Img}, \text{Jingle}) \) according to the data in Table 5.

Similarly, \( S\_DISTANCE(\text{monitor}, \text{Car}_\text{Vd}, 5, 7) \) is mapped onto \( S\_CONTAINS(\text{monitor}, \text{Car}_\text{Vd}) \).

Then, the qualitative constraint is compared with each user-defined and derived constraint \( c \) computed till the current point in the computation, according to the rules in the Appendix. Three cases can arise: 1) The comparison of the constraint with \( c \) reveals an inconsistency (such an inconsistency is detected by applying the compatibility rules in Tables 13, 14, and 15). In this case, an error message is returned to the user, listing the constraints causing the inconsistency. The user is asked to revise the specification; 2) the comparison generates a new constraint (such a new constraint is generated by applying the inference rules in Tables 11 and 12). In such a case, the constraint is added to the set of constraints generated till the current point in the computation and the process we are illustrating for the user-defined constraint is repeated for the derived constraint; 3) the comparison does not generate any inconsistency or new constraint. The user-defined constraints is added to the set of constraints generated till the current point in the computation.

This process is summarized by the algorithm reported in Fig. 5.

**Example 11.** Suppose the user enters the constraint according to the order of Example 7. Algorithm 1 generates the following additional constraints:

\[
\begin{align*}
T\_MEETS(\text{Jingle}, \text{Car}_\text{Vd}), \\
T\_MEETS(\text{Logo}_\text{Img}, \text{Car}_\text{Vd}\_\text{Audio}), \\
T\_MEETS(\text{Jingle}, \text{Car}_\text{Vd}\_\text{Audio}), \\
T\_OVERLAPS(\text{Sales}_\text{Grp}\_\text{Txt}, \text{Accidents}_\text{Grp}), \\
T\_OVERLAPS(\text{Sales}_\text{Grp}\_\text{Txt}, \text{Accidents}_\text{Grp}\_\text{Txt}), \\
T\_BEFORE(\text{Sales}_\text{Grp}\_\text{Audio}, \text{Crash}_\text{Text}_\text{Img}), \\
T\_OVERLAPS(\text{Sales}_\text{Grp}\_\text{Audio}, \text{Sales}_\text{Grp}\_\text{Grp}_\text{Audio}), \\
T\_STARTS(\text{Sales}_\text{Grp}\_\text{Audio}, \text{Sales}_\text{Grp}_\text{Grp}_\text{Txt}), \\
T\_EQUALS(\text{Accidents}_\text{Grp}\_\text{Audio}, \text{Accidents}_\text{Grp}_\text{Txt}), \\
T\_OVERLAPS(\text{Sales}_\text{Grp}\_\text{Txt}, \text{Accidents}_\text{Grp}_\text{Audio}), \\
T\_BEFORE(\text{Sales}_\text{Grp}_\text{Txt}, \text{Crash}_\text{Text}_\text{Img}).
\end{align*}
\]

The set of user-defined and inferred constraints is consistent and, therefore, no error message is returned to the user. By contrast, suppose that the user had specified the additional constraint:

\[
S\_OVERLAPS(\text{Sales}_\text{Grp}_\text{Txt}, \text{Crash}_\text{Text}_\text{Img}).
\]

This constraint conflicts with the constraint:

\[
T\_BEFORE(\text{Sales}_\text{Grp}_\text{Txt}, \text{Crash}_\text{Text}_\text{Img})
\]

and therefore its insertion would be rejected by the system.

An important question is whether the inference and consistency check process performed by Algorithm 1 is complete, that is, whether there could be other circumstances, besides the ones considered by the algorithm,
leading to the generation of a new constraint or of an inconsistency. It can be formally proven that the inferences and compatibility rules used by MPGS cover all the cases that can arise in comparing a qualitative constraint with all the qualitative constraints supported by the system. We refer the reader to [5] for the formal proof.

5 PRESENTATION GENERATION ENVIRONMENT

The presentation generation environment (cf. Fig. 1) receives as input a presentation specification, complemented with all the qualitative constraints that can be inferred from the user-defined constraints, and generates (if possible) a presentation schedule satisfying the specification. A presentation schedule is defined as follows:

Definition 8 (Presentation Schedule). Let PS be a presentation specification. A presentation schedule for PS is a set containing a schedulation item for each object specification in PS. A schedulation item is a tuple \((o, st, et, x, y)\), where \(o\) is the name of the object in the object specification, \(st\) and \(et\) are the starting and ending times of \(o\), and \(x\) and \(y\) are the coordinates of the upper left corner of its minimum bounding rectangle.

Each presentation schedule generated by the presentation generation environment must be correct, that is, it must satisfy the spatial and temporal constraints imposed by the specification. In Section 5.2, we formally prove that the presentation schedules generated by MPGS are always correct.

The presentation generation environment consists of two main components: the specification validator and the presentation generator. In the following, we describe both these components.

5.1 Specification Validator

The specification validator takes as input the overall set of temporal constraints generated by the specification environment and makes a final check on its consistency. To check consistency, the temporal constraints are translated into a set of difference constraints. Difference constraints are a special class of linear constraints [7] having the form: \(x - y \leq a\), where \(a\) is a rational number and \(x\) and \(y\) range over the real numbers.

For each object in the presentation specification and each user-defined or inferred temporal constraint, a set of difference constraints is generated according to the following rules:

- **Rule 1.** For each object \(o\) included in the presentation specification, the following difference constraints are generated:
  \[
  et(o) - st(o) \leq o\text{.duration} \\
  st(o) - et(o) \leq 0.
  \]

- **Rule 2.** For each user-defined or inferred temporal constraint, a corresponding set of difference constraints is generated according to the rules in Table 8. In the table, symbol \(\epsilon\) is used to denote a very small negative integer.
Example 12. Consider object Car_Vd of our example. The user-defined and inferred constraints on the object are: T_MEETS(Logo_Img, Car_Vd), T_MEETS(Jingle, Car_Vd), T_EQUALS(Car_Vd, Car_Vd_Audio) (cf. Examples 7 and 11), whereas Car_Vd duration is 20 (see Table 5). Thus, the following difference constraints are generated:

By Rule 1:

\[
\begin{align*}
\text{et}(\text{Car}_\text{Vd}) - \text{st}(\text{Car}_\text{Vd}) & \leq 20, \\
\text{st}(\text{Car}_\text{Vd}) - \text{et}(\text{Car}_\text{Vd}) & \leq 0,
\end{align*}
\]

and, by Rule 2:

\[
\begin{align*}
\text{et}(\text{Jingle}) - \text{st}(\text{Car}_\text{Vd}) & \leq 0, \\
\text{st}(\text{Car}_\text{Vd}) - \text{et}(\text{Jingle}) & \leq 0, \\
\text{et}(\text{Logo}_\text{Img}) - \text{st}(\text{Car}_\text{Vd}) & \leq 0, \\
\text{st}(\text{Car}_\text{Vd}) - \text{et}(\text{Logo}_\text{Img}) & \leq 0, \\
\text{st}(\text{Car}_\text{Vd}_\text{Audio}) - \text{st}(\text{Car}_\text{Vd}) & \leq 0, \\
\text{st}(\text{Car}_\text{Vd}) - \text{et}(\text{Car}_\text{Vd}_\text{Audio}) & \leq 0, \\
\text{et}(\text{Car}_\text{Vd}_\text{Audio}) - \text{et}(\text{Car}_\text{Vd}) & \leq 0, \\
\text{et}(\text{Car}_\text{Vd}) - \text{et}(\text{Car}_\text{Vd}_\text{Audio}) & \leq 0.
\end{align*}
\]

Due to the lack of space, we do not report here the algorithm for the generation of the difference constraints. We refer the interested reader to [5] for a detailed description of the algorithm.

The consistency of the set of difference constraints is then verified by means of the Bellman-Ford algorithm [7]. If the execution of the Bellman-Ford algorithm does not reveal any inconsistency, the presentation generator is activated. Otherwise, the predefined relaxation strategies are applied. The relaxation process works as follows: Given a specification, it first determines the minimum priority among the priorities specified for the constraints and the objects in the specification. Let \( p \) be such priority. Then, the system tries to obtain a consistent specification by dropping the constraints with priority \( p \). If the specification is still inconsistent, all the objects with priority \( p \) are replaced by their components of interest unless the substitution violates a specified constraint. If even this substitution does not solve the inconsistency, the relaxation process removes from the specification the objects with priority \( p \) which are no longer involved in any spatial or temporal constraint. After such deletion has been done, a new consistency check is performed on the updated specification and, if the check fails, a new iteration of the constraint relaxation process is performed. This process is iteratively executed till either a consistent presentation is obtained or all the objects have been removed from the specification. In the former case, the updated specification is returned to the user, who can decide whether he/she wishes to perform the subsequent phases on this specification or he/she prefers to manually correct the original specification. In the latter case, the user is asked to revise his/her specification.

The above steps are summarized by the algorithm in Fig. 6.

### 5.2 Presentation Generator

The core of the presentation generator is the MIN_TIME algorithm whose aim is twofold. On the one hand, the algorithm makes a final check on the consistency of the spatial constraints in the specification. If this check succeeds, MIN_TIME tries to generate the presentation with the lowest duration among those satisfying the presentation specification. Otherwise, the predefined relaxation strategies are applied and the algorithm is executed again.

#### 5.2.1 MIN_TIME Strategy

The algorithm strategy is as follows: First, objects appearing in the specification are grouped by MIN_TIME into blocks, according to the specified temporal constraints. The algorithm groups the objects in the specification into four distinct kinds of blocks: hard time blocks, initial fixed time blocks, motion blocks, and soft time blocks. The hard time blocks identify objects in the specification that are related by hard temporal constraints. Two objects belong to the same hard time block if they are related by a hard temporal constraint. Initial fixed time blocks are hard time blocks containing an absolute temporal reference to the starting time of the presentation. Motion blocks are hard time blocks containing an object for which a motion function is specified. Finally, soft time blocks are sets of hard time blocks. Two hard time blocks \( H_1 \) and \( H_2 \) belong to the same soft time block if there exists an object \( o_1 \) in \( H_1 \) and an object \( o_2 \) in \( H_2 \) such that \( o_1 \) and \( o_2 \) are related by a soft temporal constraint.

For each object belonging to a hard time block, we keep track of the relative temporal distance between its starting time and the starting time of the block. The starting time of a block is the starting time of the first object in the block which should be played. Thus, each hard time block is a set of pairs \((o, \text{disp})\), where \( o \) is an object and \( \text{disp} \) is the displacement between the starting time of object \( o \) and the starting time of the block. Moreover, for each initial fixed time block, we keep track of the temporal displacement between the starting time of the block and the starting time.

### Table 8

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Difference constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_BEFOREx1, x2</td>
<td>( ct(x_1) - st(x_2) &lt; \epsilon )</td>
</tr>
<tr>
<td>T_MEETsx1, x2</td>
<td>( ct(x_1) - st(x_2) \leq 0 )</td>
</tr>
<tr>
<td>T_OVRPLX(x1, x2)</td>
<td>( st(x_1) - ct(x_1) \leq \epsilon )</td>
</tr>
<tr>
<td>T_DURNGx1, x2</td>
<td>( st(x_2) - st(x_1) \leq \epsilon )</td>
</tr>
<tr>
<td>T_STARTsx1, x2</td>
<td>( st(x_1) - ct(x_1) \leq 0 )</td>
</tr>
<tr>
<td>T_FINISHsx1, x2</td>
<td>( st(x_2) - st(x_1) \leq \epsilon )</td>
</tr>
<tr>
<td>T_EQUALSx1, x2</td>
<td>( st(x_2) - st(x_1) = 0 )</td>
</tr>
<tr>
<td>T_DELAYx1, x2, t</td>
<td>( st(x_2) - st(x_1) \leq t )</td>
</tr>
</tbody>
</table>

---

8. Hard temporal constraints have been introduced in Section 3.2.3.
of the presentation. Thus, each initial fixed time block is a pair \(H;abs\;disp\)†, where \(H\) is a hard time block and \(abs\;disp\) is the temporal displacement between the starting time of \(H\) and the starting time of the presentation.

**Example 13.** The hard time blocks for our example are:

- \(H_1\): \((\text{Discounted Rates}_\text{Txt}, 0), 8\);
- \(H_2\): \([(\text{Logo}_\text{Img}, 5), (\text{Jingle}_0), (\text{Car}_\text{Vd}, 20), (\text{Car}_\text{Vd}_\text{Audio}, 20)]\);
- \(H_3\): \([(\text{Sales}_\text{Grp}_0), (\text{Sales}_\text{Grp}_\text{Audio}_0), (\text{Sales}_\text{Grp}_\text{Txt}_0)]\);
- \(H_4\): \([(\text{Accidents}_\text{Grp}_0), (\text{Accidents}_\text{Grp}_\text{Audio}_0), (\text{Accidents}_\text{Grp}_\text{Txt}_0)]\);
- \(H_5\): \([(\text{Crash}_\text{Text}_\text{Img}, 0)]\);
- \(H_6\): \([(\text{Music}_\text{Break}_\text{Audio}_0), 41]\).

\(H_1\) is both a motion and an initial fixed time block since it contains both a moving object and an absolute temporal reference to the starting time of the presentation. \(H_6\) is an initial fixed time block. The only soft time block is \(S_1 = \{H_5, H_4, H_5\}\).

In our system, the user can enter partial specifications in that it is not mandatory that the user specifies the temporal and spatial relations that each object in the presentation has with all the other components of the presentation. This implies that the system has some degree of flexibility in deciding the spatial and temporal positioning of an object. Generally, more than one temporal and/or spatial position exists which satisfies the constraint specification. In order to optimize resource usage, the strategy behind MIN_TIME is to position the objects in such a way that the lowest presentation time is obtained. Therefore, the algorithm tries to make the playout of objects belonging to different blocks as much as possible concurrent. The lowest duration is obtained when the playout of all the hard time blocks starts at the beginning of the presentation because, in this case, the highest degree of concurrency is obtained.

**MIN_TIME** iteratively considers each block in the input specification, trying to position each object within the block in such a way that the specified constraints are satisfied and the lowest presentation time is obtained. Motion blocks are considered first since the placement of their elements is more difficult than the placement of the other blocks, in that additional checks are needed. Then, the position of the initial fixed time blocks is decided since, for these blocks, we know the exact temporal position they must have at presentation time. For the remaining hard time blocks, the following selection criteria are applied:

1. The hard time blocks with the highest presentation time are considered first;

---

**Algorithm 2:**

**INPUT:** A presentation specification \(PS = (OS, TRS, SRS, MFS)\) where \(TRS\) and \(SRS\) contain both the user-defined and the inferred constraints

**OUTPUT:** \(\text{FALSE}, \text{if no consistent presentation can be found;}\)

\(\text{a consistent presentation, otherwise}\)

**METHOD:**

Repeat

\(\text{Let } p = \min\{p' \mid (o, d, ms, ics, p') \in OS \text{ or } (cn, par, p') \in SRS \cup TRS\}\)

For each \((cn, par, p') \in SRS \cup TRS:\)

- If \(p' = p\): Remove \((cn, par, p')\) from \(SRS \cup TRS\)

If \(PS\) is consistent: return \(PS\)

else

For each \((o, d, ms, ics, p') \in OS:\)

- If \(p' = p\): Replace \(o\) with its component of interest

If \(PS\) is consistent: return \(PS\)

else

For each \((o, d, ms, ics, p') \in OS:\)

- If \(p' = p\) and \(o\) is not involved in any constraint in \(SRS \cup TRS\): Remove \(o\) from \(OS\)

endif

endif

Until \(OS = \emptyset\)

return \(\text{FALSE}\)
2. If several hard time blocks with the same duration exist, the one with the minimum occupation area is considered first;
3. Let \( H_1 = \{(o_{11}, disp_{11}), \ldots, (o_{1k}, disp_{1k})\} \) and \( H_2 = \{(o_{21}, disp_{21}), \ldots, (o_{2m}, disp_{2m})\} \) be two hard time blocks belonging to the same soft time block \( S \). Let \( o_{11}, \ldots, o_{1n}, 1 \leq n \leq k, \) be the objects in \( H_1 \) related by a soft temporal constraint to an object in \( H_2 \). Let \( o_{21}, \ldots, o_{2m}, 1 \leq m \leq l, \) be the objects in \( H_2 \) related by a soft temporal constraint to an object in \( H_1 \). Let \( \tau_i \) be the object in \( \{o_{11}, \ldots, o_{1n}\} \) with the minimum displacement, and let \( \tau_j \) be the object in \( \{o_{21}, \ldots, o_{2m}\} \) with the minimum displacement. Let \( T_{\text{SOFT-CNSTR}} \) be the soft temporal constraint relating \( \tau_i \) and \( \tau_j \):

- if \( T_{\text{SOFT-CNSTR}} \in \{T_{\text{BEFORE}}(o_1, o_2), T_{\text{MEETS}}(o_1, o_2), T_{\text{OVERLAPS}}(o_1, o_2), T_{\text{DURING}}(o_1, o_2), T_{\text{STARTS}}(o_1, o_2), T_{\text{FINISHES}}(o_1, o_2)\} \), then block \( H_1 \) is selected first;
- if \( T_{\text{SOFT-CNSTR}} \in \{T_{\text{BEFORE}}(o_2, o_1), T_{\text{MEETS}}(o_2, o_1), T_{\text{OVERLAPS}}(o_2, o_1), T_{\text{DURING}}(o_2, o_1), T_{\text{STARTS}}(o_2, o_1), T_{\text{FINISHES}}(o_2, o_1)\} \), then block \( H_2 \) is selected first.

**Example 14.** Consider the blocks of Example 13. The only soft time block is \( S_1 = \{H_3, H_4, H_5\} \). We have \( \tau_1 = \text{Sales_Grp} \), \( \tau_2 = \text{Accidents_Grp} \), \( \tau_3 = \text{Crash_Text_Img} \). Since the only soft temporal constraint relating \( \text{Sales_Grp} \) and \( \text{Accidents_Grp} \) is \( T_{\text{OVERLAPS}}(\text{Sales_Grp}, \text{Accidents_Grp}) \), block \( H_3 \) should be considered before block \( H_4 \). Similarly, from \( T_{\text{BEFORE}}(\text{Sales_Grp}, \text{Crash_Text_Img}) \) we have that \( H_4 \) should be considered before \( H_5 \). Finally, since the duration of block \( H_4 \) is greater than the duration of \( H_3 \), block \( H_4 \) should be considered before \( H_5 \).

**MIN_TIME** incrementally builds two sets, namely \( \text{start_times} \) and \( \text{end_times} \), that respectively contain the starting and ending times of the objects during the presentation. Therefore, the set \( \text{start_times} \cup \text{end_times} \) contains all the instants, during the presentation, at which a variation in the spatial layout occurs. Information on the spatial positions of the objects belonging to the presentation are incrementally stored into the variable \( \text{spatial_positions} \), that contains an element for each instant \( t \) belonging to \( \text{start_times} \cup \text{end_times} \). This element contains the absolute positions of the minimum bounding rectangles of the objects which are displayed at time \( t \).

**5.2.2 Available Rectangles**

When deciding the spatial position of an object at a given instant \( t \), it is necessary to also have, besides the information on the spatial constraints in which the object is involved, information on the available space at time \( t \), that is, on the space on the monitor device in which no objects have been yet placed. Information on the available space at a given instant is maintained by \( \text{MIN_TIME} \), based on the concept of available rectangle, explained by means of the following example.

**Example 15.** Consider the spatial layout in Fig. 7a, consisting of two objects \( o_1 \) and \( o_2 \). The available rectangles corresponding to this spatial layout are computed as follows: First, the set of empty rectangles which can be obtained by prolonging the sides of the mbrs of \( o_1 \) and \( o_2 \) till they reach the monitor boundaries, is determined. With reference to our example, this set consists of six rectangles (cf. Fig. 7b): \( S = \{R_1, R_2, R_3, R_4, R_5, R_6\} \). Then, the available rectangles are computed by removing from \( S \) all the nonminimal rectangles, that is, all the rectangles \( R_j \) such that there exists \( R_j \in S \) and the constraint \( S_{\text{INCLUDES}}(R_j, R_i) \) holds. Therefore, the available rectangles for the spatial layout of Fig. 7a are the rectangles \( R_3, R_4, R_5, \) and \( R_6 \), illustrated in Fig. 7b.

For each instant \( t \in \text{start_times} \cup \text{end_times} \), \( \text{MIN_TIME} \) maintains into the variable \( \text{available_rectangles} \) the coordinates of the upper left corner and the height and width of each available rectangle which can be identified at time \( t \). When the system must determine the spatial position of an object at time \( t \), it first checks whether the object is related by a spatial constraint to an object which has been already positioned. In such a case, the spatial position of the object is chosen in such a way that the spatial constraint is satisfied. Otherwise, the system verifies whether there exists an available rectangle which can contain the object for each instant of its playout. If more than one rectangle exists satisfying the condition, the one with the minimum area is chosen to insert the object.
5.2.3 Overview of MIN_TIME

The MIN_TIME algorithm is reported in Fig. 8. The algorithm receives as input a presentation specification, complemented with all the inferred constraints, and returns an error message if it does not succeed in finding a presentation schedule satisfying the specified constraints; it returns a presentation schedule otherwise. Moreover, the algorithm receives as input the following set of blocks: initial_fixed_time_blocks, motion_blocks, unfixed_time_blocks, and soft_time_blocks.

We do not report here the algorithm for constructing the above blocks. We refer the interested reader to [5]. MIN_TIME iteratively considers each block received in input and tries to position the objects within the block in such a way that the constraints in the input specification are satisfied. Motion blocks are considered first. If their placement fails, an error message is returned. Then, the initial fixed time blocks are considered. Finally, the algorithm considers the remaining hard time blocks. The order in which these blocks are considered is determined by algorithm select_first_block, according to the criteria explained in Section 5.2.1. The placement in time of each object belonging to a given hard time block requires two steps. First, the temporal position of the starting time of the block must be determined. Such position determines the temporal position of each object belonging to the block. Indeed, given a hard time block \( H = \{(o_1, disp_1), \ldots, (o_n, disp_n)\} \), fixing the starting time of the block at time \( t \) is equivalent to fix the starting time of object \( o_i \) at time \( t + disp_i, \ i = 1, \ldots, n \). Then, the spatial position of each object belonging to the block must be determined in such a way that no spatial constraint is violated. The placement of a given block is performed by function place_block, presented in Fig. 9. Function place_block fixes the starting time of a block based on the block type. The placement criteria are summarized in Table 9. If the placement of the block fails, a false is returned.

---

**Algorithm 3:**

**INPUT:**
1) A presentation specification PS complemented with all the inferred constraints
2) The sets initial_fixed_time_blocks, motion_blocks, unfixed_time_blocks, and soft_time_blocks corresponding to PS

**OUTPUT:**
TRUE, if the algorithm succeeds; FALSE, otherwise

**METHOD:**

1. fixed_time_blocks, p_scheduled, start_times, end_times and spatial_positions are initialized to be empty

   For each block \( mb \in \text{motion_blocks} \):
   - \( res = \text{place_block}(mb, 1) \)
   - If \( res = false \) then return \( res \)
   - else:
     - Add \( mb \) to \( \text{fixed_time_blocks} \)
     - Remove \( mb \) from \( \text{motion_blocks} \)
   - endif

   endfor

2. For each block \( ib \in \text{initial_fixed_time_blocks} \):
   - \( res = \text{place_block}(ib, 0) \)
   - If \( res = false \) then return \( res \)
   - else:
     - Add \( ib \) to \( \text{fixed_time_blocks} \)
     - Remove \( ib \) from \( \text{initial_fixed_time_blocks} \)
   - endif

   endfor

3. For each block \( ub \in \text{unfixed_time_blocks} \):
   - \( ub := \text{select_first_block}(\text{unfixed_time_blocks}) \)
   - \( res = \text{place_block}(ub, 0) \)
   - If \( res = false \) then return \( res \)
   - else:
     - Add \( ub \) to \( \text{fixed_time_blocks} \)
     - Remove \( ub \) from \( \text{unfixed_time_blocks} \)
   - endif

   endfor

4. Generate p_scheduled from start_times, end_times and spatial_positions

---

**Fig. 8. The MIN_TIME algorithm.**
In deciding the minimum time at which a hard time block $H$ belonging to a soft time block can be placed, it is necessary to take into account which types of soft temporal constraints exist between the objects in $H$ and the objects in the soft blocks which have been already positioned. To reduce the overall presentation time, function $\text{find\_soft\_disp}$ (cf. Fig. 9), given a soft time block $S$ and a hard time block $H \in S$, proceeds as follows: It analyzes all the hard time blocks in $S$ whose position has been already fixed, to determine the lowest starting time of block $H$ which ensures the satisfaction of the soft temporal constraints between the objects in $H$ and the objects in $S$ which have been already placed. For each of these soft temporal constraints function $\text{find\_soft\_disp}$ keeps track of the minimum temporal interval which must occur between the starting and/or ending times of the constraint arguments in order to ensure the constraint satisfiability. Such an interval, which is computed according to the criteria listed in Table 10, depends on the type of the considered constraint.

The above criteria are used by function $\text{find\_soft\_disp}$ to determine the minimum temporal position of the hard time

---

**TABLE 9**

Placement Criteria for Function $\text{place\_block}$

<table>
<thead>
<tr>
<th>Block type</th>
<th>Additional conditions</th>
<th>Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion</td>
<td>The object belonging to the block is related by a</td>
<td>At the instant determined by the quantitative constraint</td>
</tr>
<tr>
<td></td>
<td>quantitative temporal constraint to the entire presentation</td>
<td></td>
</tr>
<tr>
<td>Motion</td>
<td>The object belonging to the block is not related by a</td>
<td>The starting time of the presentation. If this is not possible, at one of</td>
</tr>
<tr>
<td></td>
<td>quantitative temporal constraint to the entire presentation</td>
<td>one of the instants in $\text{end_times} \cup \text{start_times}$</td>
</tr>
<tr>
<td>Initial fixed time</td>
<td></td>
<td>The instant determined by the absolute temporal distance relating the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>starting time of one of the objects belonging to the block to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>starting time of the presentation</td>
</tr>
<tr>
<td>Unfixed time</td>
<td>The block belongs to a soft time block</td>
<td>The first instant such that the soft temporal constraints are satisfied</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(such instant is computed by function $\text{find_soft_disp}$ explained</td>
</tr>
<tr>
<td></td>
<td></td>
<td>below). If this is not possible, at one of the instants in $\text{end_times} \cup \text{start_times}$</td>
</tr>
<tr>
<td>Unfixed time</td>
<td>The block does not belong to a soft time block</td>
<td>The starting time of the presentation. If this is not possible, at one of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>one of the instants in $\text{end_times} \cup \text{start_times}$</td>
</tr>
</tbody>
</table>

---

**Fig. 9. Function place_block.**
block received as argument. The following example illustrates \( \text{find\_soft\_disp} \).

**Example 16.** Consider the blocks of Example 13 and suppose that the presentation consists only of blocks \( H_3, H_4, \) and \( H_5 \). During the execution of MIN\_TIME, block \( H_3 \) is considered first (cf. Example 14). The starting times of objects \( \text{Sales\_Grp}, \text{Sales\_Grp\_Txt}, \) and \( \text{Sales\_Grp\_Audio} \) are set equal to 0. Then, block \( H_4 \) is considered. Determining the starting time of block \( H_4 \) is equivalent to determining the starting time of object \( \text{Accidents\_Grp} \) (or, alternatively, of object \( \text{Accidents\_Grp\_Audio} \) or \( \text{Sales\_Grp\_Txt} \)) since all these objects have an associated displacement equal to 0. Thus, \( \text{find\_soft\_disp}(S, H_4) \) iteratively considers each soft temporal constraint relating \( \text{Accidents\_Grp} \) with an object in \( H_3 \) to determine which is the lowest starting time of object \( \text{Accidents\_Grp} \) such that the constraint is satisfied. The soft temporal constraints are:

\[
\text{T\_OVERLAPS}(\text{Sales\_Grp}, \text{Accidents\_Grp}), \\
\text{T\_BEFORE}(\text{Sales\_Grp\_Audio}, \text{Accidents\_Grp}), \\
\text{T\_OVERLAPS}(\text{Sales\_Grp\_Txt}, \text{Accidents\_Grp}).
\]

From \( \text{T\_OVERLAPS}(\text{Sales\_Grp}, \text{Accidents\_Grp}) \) and \( \text{T\_OVERLAPS}(\text{Sales\_Grp\_Txt}, \text{Accidents\_Grp}) \), we have:

\[
\text{st}(\text{Accidents\_Grp}) > 1, \text{st}(\text{Accidents\_Grp}) > 2, \\
\text{st}(\text{Accidents\_Grp}) > 10.
\]

Therefore, \( \text{find\_soft\_disp} \) returns 11. Instant 11 is the minimum instant at which \( H_4 \) can be placed without violating any temporal constraint.

The positioning of a block at a given instant is performed by function \( \text{place\_block\_at\_time} \), illustrated in Fig. 10, which is called by \( \text{place\_block} \) each time it tries to place a given block at a fixed instant. \( \text{place\_block\_at\_time} \) receives as input a time instant \( t \) and a block, and returns true if it succeeds in placing the block at time \( t \) in such a way that no constraint is violated by the objects belonging to the block; it returns false, otherwise.

To determine spatial positioning of an object \( o \) at time \( t \) function \( \text{place\_block\_at\_time} \) makes use of function \( \text{place\_obj\_at\_time} \), illustrated in Fig. 11. The function receives as input an object \( o \) and an instant \( t \) and returns true if it finds a spatial position for object \( o \) at time \( t \) such that no spatial constraint is violated; it returns false, otherwise. Function \( \text{place\_obj\_at\_time} \) first determines whether there exists an object, among those already positioned at the current point of the computation, which is related by a spatial constraint to object \( o \). In such a case, it tries to position the starting time of object \( o \) at time \( t \) in such a way that all the spatial constraints relating \( o \) with an object that has been already placed in the presentation are satisfied. This step is performed by \( \text{check\_spt\_cnstr\_satisfaction} \). Due to the lack of space, we do not give the details of \( \text{check\_spt\_cnstr\_satisfaction} \) (we refer the interested reader to [5]). However, it is important to point out that positioning an object \( o \) may require the execution of a considerably large number of iterations. For this reason, the number of different spatial layouts considered by \( \text{check\_spt\_cnstr\_satisfaction} \) is an input parameter of our algorithms and its setting depends on both the characteristics of the presentation specification and the user requirements. This solution allows us to reduce the computational cost when there are several spatial constraints involving the same object.

---

**TABLE 10**
Criteria Used by Function \( \text{find\_soft\_disp} \)

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{T_BEFORE}(o_1, o_2) )</td>
<td>At least one instant must occur between the ending time of ( o_1 ) and the starting time of ( o_2 )</td>
</tr>
<tr>
<td>( \text{T_OVERLAPS}(o_1, o_2) )</td>
<td>The ending time of ( o_2 ) must be at least one instant after the ending time of ( o_1 ). The starting time of ( o_2 ) must be at least one instant after the starting time of ( o_1 )</td>
</tr>
<tr>
<td>( \text{T_DURING}(o_1, o_2) )</td>
<td>The starting time of ( o_1 ) must be at least one instant after the starting time of ( o_2 ). The ending time of ( o_1 ) must be at least one instant before the ending time of ( o_2 )</td>
</tr>
<tr>
<td>( \text{T_UN_PAIRED}(o_1, o_2) )</td>
<td>Either the ending time of ( o_1 ) must be at least one instant before the starting time of ( o_2 ), or the ending time of ( o_2 ) must be at least one instant before the starting time of ( o_1 )</td>
</tr>
</tbody>
</table>

---

The positioning of a block at a given instant is performed by function \( \text{place\_block\_at\_time} \), illustrated in Fig. 10, which is called by \( \text{place\_block} \) each time it tries to place a given block at a fixed instant. \( \text{place\_block\_at\_time} \) receives as input a time instant \( t \) and a block, and returns true if it succeeds in placing the block at time \( t \) in such a way that no constraint is violated by the objects belonging to the block; it returns false, otherwise.

To determine spatial positioning of an object \( o \) at time \( t \) function \( \text{place\_block\_at\_time} \) makes use of function \( \text{place\_obj\_at\_time} \), illustrated in Fig. 11. The function receives as input an object \( o \) and an instant \( t \) and returns true if it finds a spatial position for object \( o \) at time \( t \) such that no spatial constraint is violated; it returns false, otherwise. Function \( \text{place\_obj\_at\_time} \) first determines whether there exists an object, among those already positioned at the current point of the computation, which is related by a spatial constraint to object \( o \). In such a case, it tries to position the starting time of object \( o \) at time \( t \) in such a way that all the spatial constraints relating \( o \) with an object that has been already placed in the presentation are satisfied. This step is performed by \( \text{check\_spt\_cnstr\_satisfaction} \). Due to the lack of space, we do not give the details of \( \text{check\_spt\_cnstr\_satisfaction} \) (we refer the interested reader to [5]). However, it is important to point out that positioning an object \( o \) may require the execution of a considerably large number of iterations. For this reason, the number of different spatial layouts considered by \( \text{check\_spt\_cnstr\_satisfaction} \) is an input parameter of our algorithms and its setting depends on both the characteristics of the presentation specification and the user requirements. This solution allows us to reduce the computational cost when there are several spatial constraints involving the same object.

---

Thus, according to Table 5, the above constraints are equivalent to:

\[
\text{st}(\text{Accidents\_Grp}) > 1, \text{st}(\text{Accidents\_Grp}) > 2, \\
\text{st}(\text{Accidents\_Grp}) > 10.
\]
By contrast, if object \( o \) is not related by any spatial constraint to the objects already positioned in the presentation, then \( o \) is included into the available rectangle with the minimum area, among those which can contain object \( o \) for all the instants of its playout (this step is executed by \( \text{place_obj_in_av_rect} \)). If no such rectangle exists, a \( \text{false} \) is returned. If one of the above two functions successfully ends, then \( \text{place_obj_at_time} \) successfully halts by updating the information on the spatial layout at time \( t \).

The correctness of the MIN_TIME algorithm is stated by the following theorem.

**Theorem 1.** 1) Algorithm 3 terminates and 2) it returns only correct presentation schedules with respect to the input presentation specification.

We refer the reader to [5] for the formal proof.

### 5.2.4 An Example of MIN_TIME Execution

In this section, we apply MIN_TIME to our example. The relevant blocks are reported in Example 13. The algorithm starts by positioning the objects belonging to \( H_1 \), which is the only motion block contained in the specification. The starting time of object \( \text{Discounted Rates Txt} \) is set equal to 8 and, thus, \( \text{start\_times} = \{8\} \), whereas \( \text{end\_times} = \{20\} \), since \( \text{Discounted Rates Txt.duration} = 12 \). Then, the algorithm considers the initial fixed time block \( H_6 \) and positions object \( \text{Music_Break_Audio} \) at time 41. During the third iteration of MIN_TIME, block \( H_2 \) is considered since it is the block with the highest duration among the remaining blocks. The algorithm checks whether it is possible to start the presentation of block \( H_2 \) at time 0. The check succeeds and, therefore, MIN_TIME places each object within the block according to its displacement: The starting time of \( \text{Jingle} \) is set equal to 0; the starting time of \( \text{Logo Img} \) is set equal to 5; the starting times of \( \text{Car Vd} \) and \( \text{Car_Vd Audio} \) are set equal to 20. The upper left corner of \( \text{Car Vd} \) is placed at coordinates (5,7) to satisfy the quantitative spatial constraint relating \( \text{Car Vd} \) to the object \( \text{monitor} \) (cf. Example 7). Thus, \( \text{start\_times} = \{0, 8, 20, 41\} \) and \( \text{end\_times} = \{20, 40, 47\} \).

During the fourth iteration of MIN_TIME, block \( H_3 \) is considered. According to the specification, object \( \text{Sales Grp} \) must be positioned in the interval \( [48, \text{et}(\text{pres})] \) (cf. Example 8). \( \text{Sales Grp} \) is positioned at time 48, in the same spatial position as \( \text{Car Vd} \), so that the constraint \( \text{SA_EQUALS(Car Vd, Sales Grp)} \) contained in the specification is satisfied. The starting times of \( \text{Sales Grp Audio} \) and \( \text{Sales Grp Txt} \) are set equal to 48. Then, block \( H_4 \) is considered. This block belongs to the soft time block \( S_1 \). \( \text{find\_soft\_disp} (S_1, H_4) \) is executed and returns 59 (cf. Example 16) as the minimum instant at which block \( H_4 \) can be placed so that no temporal constraints are violated. Thus, the starting times of \( \text{Accidents Grp Audio} \), \( \text{Accidents Grp Txt} \), and \( \text{Accidents Grp Audio} \) are set equal to 59. Therefore, at the end of this iteration \( \text{start\_times} = \{0, 5, 8, 20, 41, 48, 59\} \), and \( \text{end\_times} = \{20, 40, 47, 58, 70, 79\} \). The last iteration of MIN_TIME considers block \( H_5 \). The execution of \( \text{find\_soft\_disp} (S_1, H_5) \) returns 71 which is assigned to the starting time of \( \text{Crash Text Img} \). The mbr of

```plaintext
Function place_block_at_time(block, time, flag)
For each pair (obj, disp) ∈ block:
    try_time := time + disp
    res := place_obj_at_time(obj, disp, flag)
    If res = false: break
endfor
If res = true:
    For each pair (obj, disp) ∈ block:
        Add time + disp to start\_times
        Add time + disp + obj.duration to end\_times
    endfor
endif
return res
```

Fig. 10. Function place_block_at_time.

```plaintext
Function place_obj_at_time(obj, time, flag)
If there exists an object \( o_j \) in \( PS \) and an instant \( t \) such that \( o_j \) appears in \( \text{spatial\_positions}[t] \)
    ∧ \( o \) is related by a spatial constraint to \( o_j \):
    res := check\_sp\_cnstr\_satisfaction(t, obj, flag)
else:
    res := place_obj_in_av_rect(t, obj, flag)
If res = true: Update \( \text{spatial\_positions} \) and \( \text{available\_rectangles} \)
return res
```

Fig. 11. Function place_obj_at_time.
Crash_Text_Img overlaps the mbr of Accidents_Grp so that the specified constraint SS_OVERLAPS(Crash_Text_Img, Accidents_Grp) is satisfied. Fig. 12 reports the timeline of the entire presentation.

6 IMPLEMENTATION OF MPGS

A prototype of MPGS has been developed at the University of Milano on a Silicon Graphic Workstation under the IRIX operating system. The prototype has been built on top of the ObjectStore DBMS [23], using the C++ language. The graphical interface has been implemented using OSF/Motif [26].

The prototype implements all the algorithms described in this paper. The system allows the management of multiple presentations, each of which is an instance of the class presentation. Such class is characterized by several attributes including: attributes storing information on the presentation specification, such as the multimedia objects composing it, the user-defined constraints, the components of interest and the motion functions; attributes representing the inferred constraints; and attributes representing the generated presentation schedule. The methods of class presentation implement all functions for generating a presentation according to a given specification, including: consistency_check, which implements the interactive consistency check, diff_constraint, which, given a set of temporal constraints, generates the corresponding set of difference constraints; bellman_ford, which implements the Bellman-Ford algorithm; and min_time, which implements MIN_TIME.

MPGS supports the user with a graphical interface by which he/she can enter the specification and monitor the presentation generation process. MPGS provides a main window from which the user can control the specification and generation of a multimedia presentation. From the main window, the user can activate two additional windows for entering, respectively, objects and constraints. The object window, displayed in Fig. 13, shows all the information the user has to enter in order to specify a presentation object. By pressing the Constraints button from the object window, the user can visualize all the constraints in the specification having as argument the selected object. By contrast, by pressing the Constraints button in the main window, the user activates the constraint window shown in Fig. 14 by which he/she can add/remove or update a constraint in the specification.

Finally, the presentation generation process can be activated by pressing the Presentation button in the main window. By pressing this button, another window appears which contains the button to select a presentation specification, to activate the various steps of the presentation generation environment (e.g., difference constraint generation, Bellman-Ford and MIN_TIME algorithm) and to start the playout of the presentation according to the generated presentation schedule.

7 RELATED WORK

Issues related to multimedia presentations have been extensively investigated [4]. Several proposals, for instance the one by Li et al. [19], consider only qualitative constraints. The motivation is that, although quantitative constraints increase the expressive power of the model, they require a large amount of storage. In our model, we have chosen, by contrast, to support both qualitative and quantitative constraints so that users can better trade-off precision with storage requirements, depending on the specific environments and presentations.

Our work was greatly inspired by the work by Candan et al. [8], [9] which first proposed the use of difference constraints to model spatial and temporal constraints among multimedia objects and the use of the Bellman-Ford algorithm for checking constraint consistency. Our specification validator uses the same approach as [8], [9] to make the final check on the consistency of temporal constraints, whereas we use a different approach to check the consistency of spatial constraints. However, MPGS differs considerably with respect to the work reported in [8], [9]. First, MPGS provides a richer constraint model than the one presented in [8], [9]. We provide support for both qualitative and quantitative constraints, whereas, in [8], [9], only the quantitative aspect is considered. Moreover, asynchronous constraints are not provided in [8], [9]. Since we support both qualitative and quantitative constraints, our consistency checking mechanism is more articulated than the one presented in [8], [9]. We provide support for both qualitative and quantitative constraints, whereas, in [8], [9], only the quantitative aspect is considered. Moreover, asynchronous constraints are not provided in [8], [9]. Since we support both qualitative and quantitative constraints, our consistency checking mechanism is more articulated than the one presented in [8], [9]. With respect to [8], [9], we perform a first check on qualitative constraints by means of Algorithm 1 and, only if this check succeeds, we check the consistency of the difference constraints corresponding to the temporal constraints returned by the specification generation environment. This check is done using the same
approach as [8], [9]. The qualitative check detects a large number of inconsistencies, thus avoiding, in most cases, executing the subsequent phases on an incorrect specification. Another difference with respect to [8], [9] is that, in MPGS, the final consistency check for the spatial constraints is executed by the presentation generator. The reason is that a complete check on the consistency of the spatial constraints cannot be always executed without considering the temporal positions of the involved objects during the presentation. For this reason, we prefer not to make any further static check on spatial constraints. Another important difference between our work and the work reported in [8], [9] is that we exploit the relationships between spatial and temporal constraints when checking their consistency. Moreover, we provide an innovative strategy to select a presentation among the ones satisfying the specification (such strategy is implemented by MIN_TIME), whereas, in [8], [9], no criteria are provided for such a selection. Finally, neither the concept of component of interest nor the one of motion function are supported by [8], [9].

Other related approaches [6], [10], [18], [20], [21], mainly devoted to the specification task, suggest the use of graphical notations to model the constraints that multimedia objects must satisfy at presentation time. Day et al. [10] propose the Video Semantic Directed Graph (VSDG) for specifying spatio-temporal semantics of video data. However, their proposal only handles video data. Firefly [6] provides a graphical notation to express

Fig. 13. MPGS object window.

Fig. 14. MPGS constraint window.
temporal constraints in which nodes represent the starting and ending times of the objects composing the presentation, whereas arcs denote both qualitative and quantitative temporal constraints. Little and Ghafoor [20], [21] propose the Object Composition Petri Net (OCPN) model, an extension of timed Petri nets, for the specification of presentation constraints. However, both the above two proposals consider only the temporal aspect and they do not provide any support for the specification of spatial constraints.

The use of graphical notations and interactive visualization environments has also been investigated in other related fields. For instance, the MMVIS system proposed by Hibino and Rundenstainer [16] provides an interactive visualization environment for video analysis. The user can browse video data in search of temporal patterns, using ad hoc temporal visual query language. However, MMVIS is tailored toward data visualization, whereas our system deals with presentation specification and generation for multimedia data.

Other relevant related work include the work by Raghavan et al. [28], which proposes a context-free grammar-based approach for describing the synchronization requirements of an orchestrated presentation; the work by Erfle [11]; which shows how multimedia temporal constraints can be specified with the Hypermedia/Time-based Document Structuring Language (Hytime); the work by Ates et al. [3], which proposes the use of the Timed Communicating Sequential Process (TCSP) language as a specification language for temporal synchronization requirements; and the work by Steinmetz [29], which shows how conventional concurrent programming languages can be extended with temporal synchronization primitives.

Other papers [12], [13], [14], [15], [17], [22], [24], [25] present object-based approaches for modeling presentation

<table>
<thead>
<tr>
<th>Premise</th>
<th>Premise</th>
<th>Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{\text{INCLUDES}}(o_1, o_2)$</td>
<td>$S_{\text{INCLUDES}}(o_2, o_3)$</td>
<td>$S_{\text{INCLUDES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$\text{COMM}(S_{\text{EQUALS}}(o_1, o_2))$</td>
<td>$\text{COMM}(S_{\text{EQUALS}}(o_2, o_3))$</td>
<td>$S_{\text{EQUALS}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$\text{COMM}(S_{\text{EQUALS}}(o_1, o_2))$</td>
<td>$\text{COMM}(S_{\text{EQUALS}}(o_2, o_3)) \setminus {S_{\text{INCLUDES}}(o_3, o_2)}$</td>
<td>$S_{\text{QUAL}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$S_{\text{EQUALS}}(o_1, o_2)$</td>
<td>$S_{\text{INCLUDES}}(o_2, o_3)$</td>
<td>$S_{\text{DISJOINT}}(o_1, o_2)$</td>
</tr>
</tbody>
</table>

**TABLE 11**

Inference Rules for Spatial Constraints

<table>
<thead>
<tr>
<th>Premise</th>
<th>Premise</th>
<th>Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{BEFORE}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{BEFORE}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{FINISHES}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{DURING}}(o_1, o_2)$</td>
<td>$T_{\text{DURING}}(o_2, o_3)$</td>
<td>$T_{\text{DURING}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{STARTS}}(o_1, o_2)$</td>
<td>$T_{\text{STARTS}}(o_2, o_3)$</td>
<td>$T_{\text{STARTS}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$\text{COMM}(T_{\text{EQUALS}}(o_1, o_2))$</td>
<td>$\text{COMM}(T_{\text{EQUALS}}(o_2, o_3))$</td>
<td>$T_{\text{EQUALS}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$\text{COMM}(T_{\text{EQUALS}}(o_1, o_2))$</td>
<td>$\text{COMM}(T_{\text{EQUALS}}(o_2, o_3)) \setminus {T_{\text{UN_PAIRED}}(o_3, o_2)}$</td>
<td>$T_{\text{QUAL}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{MEETS}}(o_1, o_2)$</td>
<td>$T_{\text{MEETS}}(o_2, o_3)$</td>
<td>$T_{\text{MEETS}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{MEETS}}(o_1, o_2)$</td>
<td>$T_{\text{MEETS}}(o_2, o_3)$</td>
<td>$T_{\text{MEETS}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{MEETS}}(o_1, o_2)$</td>
<td>$\text{COMM}(T_{\text{STARTS}}(o_1, o_3))$</td>
<td>$T_{\text{MEETS}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$\text{COMM}(T_{\text{UN_PAIRED}}(o_1, o_2))$</td>
<td>$T_{\text{STARTS}}(o_2, o_3)$</td>
<td>$T_{\text{UN_PAIRED}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$\text{COMM}(T_{\text{UN_PAIRED}}(o_1, o_2))$</td>
<td>$T_{\text{FINISHES}}(o_2, o_3)$</td>
<td>$T_{\text{UN_PAIRED}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$\text{COMM}(T_{\text{UN_PAIRED}}(o_1, o_2))$</td>
<td>$T_{\text{DURING}}(o_2, o_3)$</td>
<td>$T_{\text{UN_PAIRED}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$\text{COMM}(T_{\text{UN_PAIRED}}(o_1, o_2))$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{UN_PAIRED}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{OVERLAPS}}(o_1, o_2)$</td>
<td>$T_{\text{MEETS}}(o_2, o_3)$</td>
<td>$T_{\text{OVERLAPS}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{OVERLAPS}}(o_1, o_2)$</td>
<td>$T_{\text{STARTS}}(o_2, o_3)$</td>
<td>$T_{\text{OVERLAPS}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
<tr>
<td>$T_{\text{FINISHES}}(o_1, o_2)$</td>
<td>$T_{\text{BEFORE}}(o_2, o_3)$</td>
<td>$T_{\text{FINISHES}}(o_1, o_3)$</td>
</tr>
</tbody>
</table>

**TABLE 12**

Inference Rules for Temporal Constraints
constraints. Among them, we recall the work by Escobar-Molano and Ghandeharizadeh [12], which provides an object-oriented framework for modeling spatial and temporal information of video data; the work by Hoepner [17] and Herrtwich and Delgrossi [15], whose aim is to extend the ODA architecture [1] with the capability of dealing with dynamic data, such as audio and video data; and the work by Gibbs [14], based on the notion of active object. However, all the above-mentioned proposals only focus on the modeling aspect and they do not provide mechanisms to check the consistency of the presentation nor to generate a presentation according to the specified constraints.

8 CONCLUSIONS

In this paper, we have presented MPGS, an interactive tool for the specification and generation of multimedia presentations. The main features of our system are: qualitative and quantitative spatial and temporal constraints; asynchronous and simultaneous spatial constraints; components of interest and priority levels;

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Constraint</th>
<th>Con./Inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.DISJOINT(c1,c2)</td>
<td>COMM(S.DISJOINT(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>S.DISJOINT(c1,c2)</td>
<td>COMM(S.DISJOINT(c1,c2))</td>
<td>Con.</td>
</tr>
<tr>
<td>S.MEETS(c1,c2)</td>
<td>COMM(S.MEETS(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>S.MEETS(c1,c2)</td>
<td>COMM(S.MEETS(c1,c2))</td>
<td>Con.</td>
</tr>
<tr>
<td>S.OVERLAPS(c1,c2)</td>
<td>COMM(S.OVERLAPS(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>S.OVERLAPS(c1,c2)</td>
<td>COMM(S.OVERLAPS(c1,c2))</td>
<td>Con.</td>
</tr>
<tr>
<td>S.INCLUDES(c1,c2)</td>
<td>COMM(S.INCLUDES(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>S.INCLUDES(c1,c2)</td>
<td>COMM(S.INCLUDES(c1,c2))</td>
<td>Con.</td>
</tr>
<tr>
<td>S.EQUALS(c1,c2)</td>
<td>COMM(S.EQUALS(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>S.EQUALS(c1,c2)</td>
<td>COMM(S.EQUALS(c2,c1))</td>
<td>Con.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Constraint</th>
<th>Con./Inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>T.UNPAIRED(c1,c2)</td>
<td>COMM(T.UNPAIRED(c1,c2))</td>
<td>Con.</td>
</tr>
<tr>
<td>T.UNPAIRED(c1,c2)</td>
<td>COMM(T.UNPAIRED(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>T.BEFORE(c1,c2)</td>
<td>COMM(T.BEFORE(c1,c2))</td>
<td>Con.</td>
</tr>
<tr>
<td>T.BEFORE(c1,c2)</td>
<td>COMM(T.BEFORE(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>T.MEETS(c1,c2)</td>
<td>COMM(T.MEETS(c1,c2))</td>
<td>Con.</td>
</tr>
<tr>
<td>T.MEETS(c1,c2)</td>
<td>COMM(T.MEETS(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>T.OVERLAPS(c1,c2)</td>
<td>COMM(T.OVERLAPS(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>T.OVERLAPS(c1,c2)</td>
<td>COMM(T.OVERLAPS(c1,c2))</td>
<td>Con.</td>
</tr>
<tr>
<td>T.Finishes(c1,c2)</td>
<td>COMM(T.Finishes(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>T.Finishes(c1,c2)</td>
<td>COMM(T.Finishes(c1,c2))</td>
<td>Con.</td>
</tr>
<tr>
<td>T.During(c1,c2)</td>
<td>COMM(T.During(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>T.During(c1,c2)</td>
<td>COMM(T.During(c1,c2))</td>
<td>Con.</td>
</tr>
<tr>
<td>T.Starts(c1,c2)</td>
<td>COMM(T.Starts(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>T.Starts(c1,c2)</td>
<td>COMM(T.Starts(c1,c2))</td>
<td>Con.</td>
</tr>
<tr>
<td>T.EQUALS(c1,c2)</td>
<td>COMM(T.EQUALS(c1,c2))</td>
<td>Inc.</td>
</tr>
<tr>
<td>T.EQUALS(c1,c2)</td>
<td>COMM(T.EQUALS(c2,c1))</td>
<td>Con.</td>
</tr>
</tbody>
</table>
motion functions; the development of a sophisticated method to check constraint consistency and an innovative technique to generate a presentation satisfying the requirements of the specification.

Future work includes an extensive performance analysis of both the constraint relaxation strategies and the MIN_TIME algorithm, the development of techniques for the automatic identification of the components of interest within an object, the management of user interactions and delays during the presentation, the support for transition effects (such as fade, wipe, etc.) of video and audio objects.

Another important issue is related to the development of more powerful tools for supporting the author in the specification task. We plan to complement our MPGS prototype with a more powerful user interface by which the user can visualize the specification and easily correct and update it.

APPENDIX A

RULES FOR CONSTRAINT GENERATION AND CONSISTENCY CHECK

A.1 Inference Rules

Tables 11 and 12 report the inference rules used by MPGS to generate the implicit qualitative constraints that can be inferred from the user-defined constraints. Table 11 deals with spatial constraints, whereas Table 12 reports the inference rules for temporal constraints.

In the tables, when the notation S_QUAL (resp. T_QUAL) appears in the second column, it denotes the set of all the qualitative spatial (resp. temporal) constraints, except the one which appears in the first column. Moreover, given a set of constraints CS, we denote with COMM(CS) the set containing all the constraints in CS and the constraints obtained by taking each element in CS and permuting its arguments.

A.2 Compatibility Rules

Tables 13, 14, and 15 report the rules by which MPGS checks the consistency of qualitative constraints. Table 13 deals with spatial constraints, Table 14 deals with temporal constraints, whereas rules in Table 15 exploit the relationships existing between spatial and temporal constraints.

In the tables, when T_QUAL_SIMULT appears in the second column of a table, it denotes the set of the simultaneous qualitative temporal constraints, whereas T_QUAL_DISJ denotes the set of disjunctive qualitative temporal constraints. Finally, with reference to the third column, we use “Con.” as an abbreviation for consistency, and “Inc.” as an abbreviation for inconsistency.

REFERENCES


Elisa Bertino received the doctor degree in computer sciences from the University of Pisa, Italy, in 1980. She is currently a professor of database systems in the Department of Computer Science of the University of Milano, where she heads the Database Systems Group. Since October 1997, she has been the chair of the Computer Science School of the University of Milano. She has also been on the faculty in the Department of Computer and Information Science of the University of Genova, Italy. Until 1990, she was a researcher for the Italian National Research Council in Pisa, Italy, where she headed the Object-Oriented Systems Group. She has been a visiting researcher at the IBM Research Laboratory (now Almaden) in San Jose, at the Microelectronics and Computer Technology Corporation in Austin, Texas, at George Mason University, at Rutgers University, and at Purdue University.

Her main research interests include object-oriented databases, distributed databases, deductive databases, multimedia databases, interoperability of heterogeneous systems, integration of artificial intelligence and database techniques, and database security. In those areas, Prof. Bertino has published several papers in refereed journals and in proceedings of international conferences and symposia. She is a coauthor of the books Object-Oriented Database Systems—Concepts and Architectures (Addison-Wesley International, 1993), Indexing Techniques for Advanced Database Systems (Kluwer Academic, 1997), and Intelligent Database Systems (Addison-Wesley International, forthcoming). She is or has been on the editorial boards of the following scientific journals: the IEEE Transactions on Knowledge and Data Engineering, the International Journal of Theory and Practice of Object Systems, the Very Large Database Systems (VLDB) Journal, the Parallel and Distributed Database Journal, the Journal of Computer Security, Data & Knowledge Engineering, and the International Journal of Information Technology.

Dr. Bertino is a senior member of the IEEE and a member of the ACM and AICFA and has been named a Golden Core Member for her service to the IEEE Computer Society. She has served as program chair of the 1996 European Symposium on Research in Computer Security (ESORICS’96), as general chair of the 1997 International Workshop on Multimedia Information Systems, and as program cochair of the 1998 IEEE International Conference on Data Engineering (ICDE).

Elena Ferrari received an MS degree in information sciences in 1992 from the University of Milano, Italy, and a PhD degree in computer science in 1998 from the same university. She is now an assistant professor in the Department of Computer Science of the University of Milano, Italy. She has also been a visiting researcher at George Mason University, Fairfax, Virginia, and at Rutgers University, Newark, New Jersey. Her main research interests include: database security, access control models, multimedia databases, and temporal object-oriented database models. On these topics, she has published several papers in refereed journals and conference proceedings.

She has served as an organizing committee member of the ECOOP ’99 Workshop on Object-Oriented Databases and as a program committee member of the 1997 International Workshop on Multimedia Information Systems, the 1999 International Workshop on Information Technologies for Electronic Commerce, and the 1999 International Workshop on Electronic Commerce and Security.

Marco Stolf received the MS degree in computer science from the University of Milano, Italy, in July 1997. From 1988 to 1994, he worked as programmer on business applications. Now, he works as a software department head and as a consultant in the area of business and industry solutions and applications with Office Automation. His research interests include multimedia and object-oriented databases.