VIRAGE : DESIGNING AN INTERACTIVE INTERMEDIA SEQUENCER FROM USERS REQUIREMENTS AND THEORETICAL BACKGROUND

ABSTRACT

We present the unrolling of the Virage project and its main achievement: a sequencer for authoring and performing interactive intermedia scenarii. This two years long project addressed the question of authoring and controlling interactive scenarii dealing with several heterogeneous digital media, in the context of the performing arts. The project involved multidisciplinary members including artists and developers and researchers in the fields of social studies and computer science. The main point was the design of a sequencer for scheduling and controlling the triggering of events embedded within the scenarii. We show here how we adapted a theoretical background, originally conceived for the interpretation of musical scores, to specific requirements in collaboration with field agents. We also describe how these exchanges about the features of the sequencer leaded the software development.

1. INTRODUCTION

The Virage project attempted to address the questions of authoring time and interaction in the context of media management (sound, lighting, image, machinery...) for the performing arts. For several years, media managers and creators have been using more and more digital contents and tools to control their media materials. For this purpose the field workers use either dedicated software and tools for each type of contents, or adapt softwares coming from other fields (e.g. computer music). However, more and more complex cases challenge the possibilities of authoring the time organization of heterogeneous contents, as some ways to interact with them. These new cases call for a tool that might allow mixing heterogeneous representations, and specifying temporal relations between them.

The Virage project proposed some answers to these questions, by designing an interactive sequencer for authoring time and interaction with heterogeneous digital contents. Since this tool is quite new, we based the design method on the needs and practices of the field agents, by involving some of them in the project and carrying out a field study. The results of the study and the participation of field workers fueled the discussion of the theoretical model. In the same manner, each step of the implementation was confirmed by every member of the project.

We first present the frame of the project. Then we focus on the theoretical background of the project and its implementation. At last, we examine how the field requirements influenced the design of the software and we propose some applications and perspectives.

2. CONTEXT AND CHALLENGES

This project can be considered as a practical reflection on time-based media scripting in digital artistic creation, particularly in relation to live performance. Previously to the project, a study has been carried out and published in order to analyze the limits of existing software systems for interactive performing arts situations. This preliminary study was then completed by a field survey conducted at the very beginning of the project, that leaded to write initial specifications that will be updated at the very term of the project (February 2010). The final objective of the project is not to produce effective software, but these specifications, in order to allow future development. Though, the key point of the research statement has been to make generate these specifications through the development of usable prototypes, and feedback from their confrontation to real-world cases

2.1. Interoperability

The insularity between the existing multi/inter-media software environments has been stated through the preliminary studies as a major bottleneck for creative workflow in the context of the performing arts.

The Virage sequencer has then been proposed as an experimental way to overcome this insularity by acting as a hub between these environments, dedicated to the design of temporal parameter management. This has been achieved by combining the Virage sequencer with the existing software and hardware environments that are actually used in production by the field agents, and that correspond to their habits. In order to do that, a plugin interface has been developed, that will be further explained in and allows the coexistence and simultaneous usage of several protocols, including:

- Open Sound Control (OSC)

• Minuit®: a custom query system developed in the project in order to allow actual experiment of the prototypes, by adding some features such as namespace discovery on top of OSC

• CopperLan®: a novel industrial protocol aiming to overcome both MIDI and OSC limitations through the development of a set of tools including a middleware, hardware chips, etc...

• potentially any network or device protocol can be added by developing a third-party dedicated plugin

In respect to this concern of interoperability, the Virage sequencer focuses on the scripting of behaviors of abstract parameters, and not of the media themselves, as in all existing sequencers. To our knowledge, the only application showing a similar statement is Iannix [10], which has been reviewed and didn’t show as relevant to the studied field, namely the performing arts. This design choice allows a certain independence from the medias’ intrinsic durations, thus allowing to continuously vary durations at duration, what we here coin as **flexible time**.

2.2. Flexible time

Existing creative multi/intermedia software environments generally address time-scripting by one of the next two paradigms:

• fixed timeline, such as in DAWs and Video editing softwares, that allow a very precise scripting of media behaviors, through parameter automations.

• cue lists, in theater cue managers such as Qlab® and also in Ableton Live®, which is very frequently used in performing arts projects

Graphical programming environments such as MaxMSP or Pd potentially allow any kind of time management, but they very hardly allow any representation of time structures, which makes their use difficult for non-programmers, in particular when accounting the specific production-constraints of the performing arts.

Time is then managed as a monolithic fixed flow (as in the timeline model), or as a set of unrelated discrete events (as in the cue list model). It is to note that Ableton Live proposes both of these paradigms in the software, but that they are completely unrelated temporally.

The requirements expressed by the field agents showed a need for a third strategy that might be a mix of these ones.

As soon as the comedians or the technical staff modify their behavior during a given performance, the fixed timeline representation appears to be too strict. On the other side, the cue representation is lacking of possibilities of designing complex time structures. An existing application mixing both of these paradigms for multimedia management is Medianon Manager® with multiple timelines triggered by cue lists and a strong concern on interoperability. Though this application is extremely expensive and technically-oriented which prevents any use in a low-economy and creative-centered field such as the performing arts.

It also appeared that a more innovative solution might be a system in which one can express complex and precise time organization: time relations between events of the scenario (“event A must appear before event B”), due to artistic choices or real-life constraints, and also some indicative values for the time intervals between the events. On the other side, it might allow to change some characteristics (such as order or duration) of the temporal organization during a performance, in order to follow the behavior of the agents. For this purpose, the system should adapt the scenario to respect written time relations and accept real-time modifications. Variable clocks have also has appeared as particularly useful for the field agents, on the global level of the whole scenario as well on the local level of each temporal object, in order to interactively and continuously adapt the time of media behaviors to those of human actors or dancers, similarly to what has now become a classical feature in any lighting console. We use the term “flexible time” to describe this type of behavior.

In such a system, a scenario is no more the representation of a single performance, but rather of a set of possible performances that share temporal proprieties. By this point of view, the system we tried to design can be compared to systems such as Harp [8] or DoubleTalk [13], which considered a musical score as the representation of several possible executions with shared proprieties.

3. THEORETICAL MODEL

The sequencer of the Virage project is based on an hybrid temporal paradigm that has been designed for musical composition and performance. This temporal model mixes two different temporal paradigms, one for authoring a musical piece and the other for performing, that is, for executing the piece. Modeling of interpretation is the first objective of this temporal paradigm, and it mostly draws its inspiration from the work of Jean Haury on the MetaPiano[11].

This model appeared to suit quite well the problem of authoring scenarii for live performances involving multimedia techniques. As a matter of fact, it provides the author a way to place multimedia events on a time-line, in sequence or synchronisation, some of them being eventually triggered in real time during live performance according to the scenario.

---

2 specifications available at:  
http://www.plateforme-virage.org/?p=1444  
http://www.copperlan.org/  
http://www.ableton.com/  
http://www.medialon.com/
Without fully exposing the theoretical model, we just recall its main aspects and the points we modified to fit with the stage management context. For a specific musical approach and a complete presentation, one can refer to [1] and [3].

An interactive scenario is the temporal organization of a set of events, in which the temporal characteristics are partly specified to allow some modifications during the performance. The author (the director and/or the stage manager) can specify some temporal proprieties that he wants to be respected during each performance, as what can be changed during a performance. This way of authoring the temporal organization is close to “constraints programming”.

We imagine the complete chain of use of the system as shown on the figure [1].

First, the author write the scenario by specifying the temporal organization of the events. The language used at this moment is a graphical programming language, designed to be used by non-programmers. During this step, the system must analyze the choices of the user in order to prevent him from over-constraining the scenario. Once the scenario is written, it can be executed on an abstract machine. Since this machine is designed as generic, the scenario must be compiled before being run on it.

3.1. The authoring model

We present here the graphical language used for authoring the interactive scenarii. An example of such a scenario is given on the figure [2].

Since the system is thought as a hub that can send abstract data to other applications, the authoring formalism is based on processes able to compute and produce abstract values. The user can define how the processes will run during the performance. For this purpose, he defines the temporal organization of his scenario by using temporal objects.

The temporal objects are the basic element of the temporal model which is inspired from the hierarchical models proposed by Mira Balaban [5]. A temporal object with no children, called a texture, represents the execution in time of a given process. A temporal object with children, a structure, represents the temporal organization of its children. Each structure has its own timeline scaled with its own time unit. A scenario is represented by a root structure.

On the figure [2] the objects “sound”, “red” and “green” are textures that represent the execution of processes, which read tables in order to send values to specialized applications; “lights” and “My scenario” are structures.

Formally, a temporal object is defined as a 11-uple:

\[ TO =< t, r, p, E, S, \mathcal{D}, Y, \mathcal{E}, \mathcal{B}, \mathcal{R}, \mathcal{C} > \]

- \( t \): a type.
- \( r \): a ratio between the TO’s time unit and the time unit of its parent structure; if TO is the root structure, \( r \) is the ratio between the time unit of TO and the clock of the machine which runs the scenario during the performance; by changing the value of \( r \), the performer can apply local speed changes.
- \( p \): a process associated to TO, only if TO is a texture, null otherwise.
- \( E \) (resp. \( S \)) : a set of inputs (resp. outputs); the in/outputs allow to route the data flow between the process, and to communicate with external applications through the inputs and outputs of the root structure; on the figure [2] the in/outputs are represented by points on the left and right borders of the objects.
- \( \mathcal{D} \) : a set of dated control points; the control points represent particular moment of the execution of TO, on the figure [2] they are represented by big circles on the up and down borders of the objects; if TO is a texture, its control points represent computation steps.
of its process $p$ (its beginning or end, or for example a particular value in the reading of a table such as the textures “red” and “sound” in the figure 2) by giving a date to each control point of a texture, the user can specify the temporal proprieties of the execution of the process $p$. If $TO$ is a texture a control point represents a control point of one of the $TO$'s children, in the purpose of defining temporal relations with objects in an upper level (see the “lights” structure on the figure 2); the dates of the control points are expressed in the time unit of the parent structure of $TO$.

- $\mathcal{V}$: a set of variables which describe the internal characteristics of $TO$, for example the current value of the ratio $r$, which is the execution speed of $TO$.
- $\mathcal{S}$: if $TO$ is a structure, the set of the children objects of $TO$, null otherwise
- $\mathcal{R}$: a set of routes between the in/ouputs of the $TO$'s children
- $\mathcal{R}_i$: a set of temporal relations between the events of $TO$ i.e. the set made up of the control points of the children of $TO$ and the own control points of $TO$; these relations are explained below.
- $\mathcal{G}_i$: a set of global constraints defined on the variables of $TO$ and the ones of its children; as an example a maximum value for the $TO$'s ratio $r$.

### 3.1.1. Temporal relations

The user can define the temporal proprieties of his scenario through temporal relations between the control points of the temporal objects. Thus these relations are taken from the point algebra and extended with quantitative constraints.

A relation $tr$ of the set $\mathcal{R}$ of a structure $S$ is defined by a 6-uple:

$$tr = \langle t, p_1, p_2, \Delta_{\min}, \Delta_{\max} \rangle$$

- $t$ is a type, precedence (Pre) or posteriority (Post)
- $p_1$ and $p_2$ are events of $S$
- $\Delta_{\min}$ and $\Delta_{\max}$ are real values in $[0, \infty)$

If $tr$ is a precedence relation, then it imposes the inequalities:

$$\Delta_{\min} \leq d(p_2) - d(p_1) \leq \Delta_{\max}$$

where $d(p_i)$ is the date of $p_i$. It has to be remembered that these dates are expressed in the time unit of $S$, so are expressed the values $\Delta_{\min}$ and $\Delta_{\max}$.

We define the following vocabulary for the time interval $I$ between 2 points bound by a relation $tr$:

- $\Delta_{\min} = \Delta_{\max}$. $I$ is said rigid; if these values are 0, $tr$ is a synchronization relation.
- $\Delta_{\min} = 0$ and $\Delta_{\max} = \infty$, $I$ is said flexible.

- otherwise, $I$ is said semi-flexible.

The system will maintain the proprieties imposed by the temporal relations, during the authoring process as well as during the performance. In the first case they can help the editing operations, in the second case, they define the limits for the modifications introduced by the performer through the interaction points.

### 3.1.2. Interaction Points

During the authoring process, the user can define some control points to be dynamically triggered by the performer during the execution. These control points are said dynamic (as the beginning of the “lights” structure on the figure 2), while the other control points, the static ones, will be triggered by the system. The written date of a dynamic control point is clearly indicative, since the performer can modify it. However, this modification possibility is limited the temporal relations. Then, on the figure 2, the system will refuse the dynamic triggering of the point $L_p$ before the duration $\Delta_{\min}$ has elapsed, in order to respect the relation between $M_{\alpha}$ and $L_{\alpha}$. In like manner, the system will automatically trigger $L_{\alpha}$ if the performer did not before the duration $\Delta_{\max}$ has elapsed.

The respect of the temporal relations can lead to indirect modifications of dates. For instance, modifying the date of one of the 2 points of a rigid interval leads to modify the date of the other one in order to respect the value of the time interval. If the duration of a structure $S$ is rigid, the modification of a time interval between some events of $S$ will lead to modify other intervals to respect the duration of $S$. As a consequence, the model provides some predefined strategies that the user can choose, to describe how the modifications of dates will be propagated through the temporal objects. The default one consists in delaying every static events following a delayed dynamic event. The field member of the project asked for a specific one which consists in trying to maintain the indicative value of an interval. These intervals with this strategy are said fixed.

### 3.2. The execution model

On the execution side of the system, we propose a generic abstract machine able to run any scenario written with the formalism. We call it the ECO Machine for Environment (the intermedia content of the scenario), Controls (the actions of the performer) and Outputs (the data broadcasted by the system). The intuitive idea is that the machine will produce outputs from the environment and the controls. The architecture of the machine has three elements, the Controller turns the messages from other applications into abstract external events, the Scheduler is in charge of triggering the control points of the scenario in accordance with the temporal relations and the external events, and of modifying the written dates according to the dynamic triggers and the propagation strategies, the Processor runs the processes under the temporal control of the Scheduler.
This architecture is shown on the figure 3. We chose to represent the temporal organization of the scenario with Petri nets structures (precisely hierchical time stream Petri nets [12]), which are used several times in computer music applications [9]. Other representation could be explored such as concurrent constraints [2].

The compilation process consists in turning the temporal proprieties of the scenario into a Petri net, by associating each control point to a transition and adding places and arcs in accordance of the temporal relations. The constraints store of the processor helps maintaining the global constraints of the set \( C \) of the structures.

### 4. IMPLEMENTATION

In this section we present the IScore library, an implementation of the formalism described in section 3. Thus, the library is the union of two engines, which match the two sides of the theoretical model:

- an editor engine to write the temporal constraints based scenarios and an execution engine to compile and to run the scenarios

This library, developed in C++ and currently working on Mac OSX and Linux, is open source.

#### 4.1. History

This development inside the Virage project is mainly based on two previous works at LaBRI:

- an update of the Boxes software [7]. Boxes is a graphical musical editor in C++, based on a temporal constraints model close from the Virage one. The update consisted in using a new constraints library (GeCode) for its graphical interface

The editor engine in Virage is inspired by this development, but with many adjustments.

- a Common Lisp implementation of the ECO machine in the OpenMusic software [4], written for a phd [1].

An important part of the execution engine in Virage is a migration of this work.

#### 4.2. Architecture

##### 4.2.1. Editor class

The editor class provides features to compose with constrained temporal objects. Every created box contains two control points, the first for the box start, and the second for the box end. Each point contains two variables, a start date and a length, added to the Gecode solver. After every modification, Gecode updates all the variables for providing a solution respecting the constraints. It is also possible to create boxes within boxes. Every object created with this class is referenced by an unique integer identifier. Trigger points can also be added thanks to this class.

##### 4.2.2. ECOMachine class

This class contains all the features for real-time performance. It implements a Petri network and a process manager. None of the real-time constraint solving features have been integrated yet. To implement the go-to functionality and the speed modification, we added a timer information on every token in the Petri network.

The process functionalities are not limited to the ones currently implemented. The process only needs to handle the information provided by the Petri network after each transition crossing. Any process which inherits from the ECOProcess class will respect this demand.

For now, we only have one type of process which sends discrete or continuous network messages, quantized at the millisecond. But we are thinking about other types of process, like for example data conditioning (or mapping) or scenic acquisition, to be integrated through a plugin system.

##### 4.2.3. Devices library

The Devices library is implemented to manage network communications. Indeed the Virage software has to control many different technologies by sending messages. Communication protocols quickly evolve so we had to think about an extendable solution, for which new communication plugins can be developed. We then decided to develop this library as a plugin manager because it allowed us to keep the IScore library independent from the communication protocol.

On the one hand, this library provides the DevicesManager class containing all methods needed by an application to communicate. It makes a link between the engine and applications (hardware and software) called ‘devices’ on the network. Once instantiated, the library scans the plugin

---

Figure 3. The ECO machine

---

source directory using the PluginFactory class. This class is based on a mechanism developed in the Tulip project[^9]. Then it uses each plugin to scan the network and get the devices. Currently the DevicesManager knows the network devices list and their specific communication protocol.

On the other hand, it provides the Plugin interface on which all communication plugins have to be implemented. For now three plugins have been developed: OSC, Minuit and CopperLan.

4.2.4. Engines class

The main goal of the Engines class is to simplify the library use, e.g. for a graphical interface developer. This class centralizes all the functions needed for composition and for performance, in order to hide the implementation detail. Just by instantiating an Engines object, the developer will have access to all implemented features.

4.3. Graphical interface

The key concept of the graphical interface design has been intuitivity in order to allow the non-scientific user to handle abstract models such as those proposed in section 3.

The main space of the software is dedicated to the edition of the temporal scenario, using the forthcoming elements: Temporal objects are represented by boxes on the timeline, that can be connected at each end by Relations, may they be fixed (bold lines), flexible (dashed lines) or semi-flexible with bounds (combination of bound and dashed lines representing the respective fixed and flexible durations). Trigger Points can be created and attached to the temporal objects, allowing the user to interactively trigger them. They are currently placed on an upper rail for more straightforward ordering and triggering by the operator, when executing the scenario. Though, it has been noticed that this representation was in some way restrictive to the creative temporal possibilities of the model. This placement of Trigger Points will be alternatively moved against the objects or on the timeline rail depending on the editing mode (authoring or execution) in a future version.

Apart from this main space, a couple of sub-windows allow content management to the user:

- a Namespace browser allows to explore and snapshot parts of the parameter spaces (namespaces) of the controlled remote environments.
- a Properties sub-window allows the user to set and fine-tune properties of the temporal objects, trigger points and relations.
- a Curve Editor allows to design and transform breakpoint functions for the parameter automations.

A transport bar allows the user to play, stop, pause the scenario, to trigger the next trigger point and to change the global speed of execution through a slider.

[^9]: http://www.tulip-software.org

Figure 4. Class diagrams
4.4. Users involvement

Users have been implied in the research process all along the project, and at all its stages: from the initial definition of features through directed experimentations to the conception stage by giving constant feedback and defining priorities of the development tasks. Then, once the first stable prototype had been produced, all successive steps of the development were the opportunity to confront the prototypes to a diversity of real-world situations generated by experimental workshops. These workshops were leaded by the 6 artistic partners of the research platform, ranging from the performing arts to digital interactive arts. These experiments of the prototypes have been done within the existing creative environments and setups of the field agents and developed around actual artistic questions. These experiments have been done in actual production situations, but out of the production time, in order to avoid stressing the research process with production deadlines and constraints.

Thus, the library implementation has been regularly provided to the users in combination with the graphical interface, in order to have constant feedbacks, and keep the development close to users needs. To enhance the communication between developers and users, we used some ideas from Agile Method, such as SCRUM.

4.4.1. Quarterly: tests and product backlog

A product backlog is a prioritized list of user features. Every feature is represented by a sentence describing a user need (not a programmer need): As [media-manager/creator/compositor/…], during [the edition/the performance], I want to …. For example: As a media-manager, during the performance, I want to modify the execution speed. A grade is then provided by the programmer for each feature, to describe the difficulty to implement this need.

A meeting with all Virage partners was organized every 3 months. The meeting started with the user tests of prototype (considered stable) and validation, or not, of the different user features in the backlog. It ended by a modification of the backlog: new classification, new features…

The programmers, after the meeting, work on the non validated features, and then develop new features matching the backlog order, until next meeting.

4.4.2. Weekly: audio conference

An audio conference was organized every week, with at least the developers and the project coordinator, representing the users. Any partner may assist this audio conference if he wished. This showed up as an effective way to follow the development process, while keeping the meeting short. Every participant would then concisely express what they did to the current day, what they are about to do next and the difficulties they met. If further information is needed, another audio conference is planned specifically for this topic.

4.4.3. Dayly: bug tracker and feature request

Every time a significant change has been done and stabilized, a work-in-progress version of the prototype is provided to users. They could test this version. We provide online bug and feature request trackers, in which users can create and comment items.

As for the backlog, they have to classify all these requests and bugs.

If a new request is important and quick to develop, a developer could decide to implement it. If not, the feature will be added to the product backlog, and discussed during the next meeting.

To respect the communication and interaction between developers and users, we decide that the person who opens a bug or a new feature request has to close it, and thus validates the fix or implementation.

In order to allow more use-cases and feedback, we made the software freely available for download and informed the professional community through public presentations and workshop all along the project and after its term.

5. PERSPECTIVES AND APPLICATIONS

A consolidation of the prototype as a usable cross-platform open source software is planned by a further structuration of the project’s consortium and associated partners as a cooperative in order to keep the collaborative spirit developed throughout the research project. In this frame, it is planned to fuse Virage with the successor of Boxes, namely the Acoutscribe, which will introduce the need to have different kinds of processes implemented in the way of plug-ins and to take into account musical problematics.

This consolidation phase will mostly focus on robustness (including porting the graphical interface to the Qt framework), ergonomy and usability.

A few features that were not developed in the project will be added, including routing and conditioning of data flows or events from/to remote addresses, in order to allow to script continuous interaction. The edition of hierarchical

10 [http://www.plateforme-virage.org/?p=1297]
scenarios, which is actually implemented in the engine also have to be added to the graphical interface.

Further research has been submitted to focus on the integration of conditional structures, which could allow to define several choices within a scenario. Thanks to this type of structures, a performer could choose between different options of execution. Another improvement asked by the field agents consists in multi-user facilities. Those could allow the edition and the execution of a scenario with different parts spread on different machines, and controlled by different users.

6. CONCLUSION

We presented a research project that consisted in gathering researchers and field agents to propose solutions to real problems. Both groups came with a background, theoretical models and results on one side, needs and experience on the other side. The all along communication allowed us to fuel the research process with new questions that led to adapt the theoretical background, but also to improve it for new developments. The scientific approach helped the field agents to clarify their point of view on their own constantly-evolving practices. It is clear that this partnership was successful because both parts were enough mature to contributed with existing models for authoring interaction on one hand, real cases and newly established practices on the other hand. Finally, the project led to an actual and usable piece of software, that can be considered as a basis for future consolidation and development.

7. ACKNOWLEDGMENTS

This research project has been funded by French National Agency, in the "Audiovisual and Multimedia program. The authors would also like to thank all of the partners of the Virage platform [11], that have been contributing by feedback, advising and workforce in order to achieve this project. Anthony Beurivé for initiating the Boxes project and Bruno Valèze for developing it.

8. REFERENCES


