

AN SQL-BASED CONTROL SYSTEM FOR LIVE ELECTRONICS

Francesco Canavese

Francesco Giomi

Damiano Meacci

Kilian Schwoon

Tempo Reale
Villa Strozzi – Via Pisana, 77
Florence (Italy)
[fc,fg,dm,ks]@centrotemporeale.it

ABSTRACT

Managing control data in live electronic systems can be very complex. Ideally, in a rehearsal situation the performers should be able to focus on the musically relevant problems and make significant changes to the structure of musical events in an immediate way. To obtain this kind of flexibility, a performance system has been developed at Tempo Reale which makes traditional live electronic instruments interact with a relational database, implemented using Structured Query Language (SQL).

1. EVENT-ORIENTED LIVE ELECTRONICS

There are many different compositional approaches for the use of live electronics in contemporary music. The system presented here is optimized for music in which there is no rigid timing, but where electronic events are triggered according to a score, following the natural flow of a human performance with all its irregularities.

An event is considered as a structured element, defined by an algorithm type (harmonizing, sampling, spatialization etc.) and references to score positions (cues). Additional information such as timed parameter sequences or digital mixer routings and operations can be associated to such an event. There are no limits for the overlapping of events, thus allowing a “polyphonic” organization of electronic layers. This is an important difference from systems based on concepts like program changes or scenes, which constrain by dividing a score into blocks.

2. A META-LEVEL FOR MAX

As a framework for the development of the performance environment, the MAX architecture turned out to be a good choice, especially because of the flexible event scheduling and the possibility of handling data flows involving all common formats and protocols used in computer music (a porting to other applications of the MAX family like pd and jMAX should not present any particular problems). A “cue manager” patch is at the core of the implementation. In this patch all control data are associated to a series of predefined reference points in the score. These cues are called back during the

performance one after the other by an external trigger, as, for example, a key on the computer keyboard or a MIDI message.

As its author Miller Puckette says, “MAX is oriented toward processes more than data” [3]. There are certainly objects developed for data handling (like [coll] or [qlist]), but they don’t support dynamic relationships between these data, as items (“messages”) are buffered in table-like structures without the means for a complex interaction between these tables. In order to handle data on a higher symbolic level, a strategy in which the data organization occurs outside MAX has been chosen. Afterwards, the generated data are sent to MAX and buffered in standard objects.

3. DATA STRUCTURES

In order to analyse the requirements for the design of a database, the dependencies between the most relevant data can be described in an entity relationship diagram as in figure 1. This diagram shows how the fundamental entity of an electronic *event* depends on various other entities. The most obvious link is the one to the entity *cue*, which defines the position of the event in the score. Furthermore, every event is related to a *command*, which defines the algorithm type, its parameters and certain configuration properties of this algorithm.

Each setup can be based on one or more digital mixers. The entity *mixer* defines settings such as the MIDI channel used for control. The definitions of the logical *inputs* and *outputs* of the global control system depend on these mixer settings. Each command can be linked to a static mixer output, whereas the inputs are referred to individually for each event.

Optionally events may also depend on data *sequences* which are typically used to store time/value pairs for parameter variations with a predefined timing. This timing is relative to the triggering of the event, so that timed sequences starting at different cues behave independently from each other.

The entity relationships described here can be translated into a relational database which is organized in tables linked by means of IDs. The attributes of the entity *cue*, namely a label and a score position (defined by page, measure and beat numbers) are for instance stored in a table structured as shown in table 1.

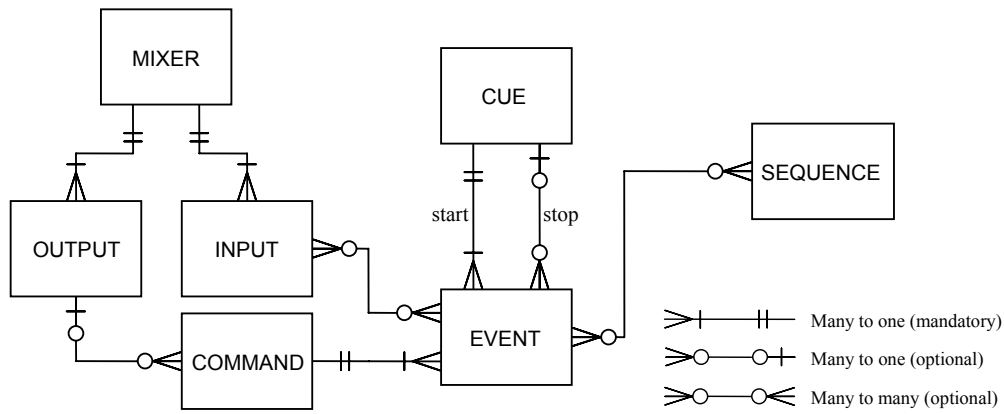


Figure 1. Relationships between the basic entities

ID	Label	Page	Measure	Beat
119	A.intro	1	1	4
120	A.1a	1	5	2
121	A.1b	5	1	1
126	A.2	5	3	1
123	A.stop	5	7	4
122	B.1	6	1	1
124	B.stop	9	3	1

Table 1. Table structure *cues*

The IDs are automatically created by simply following the order of their insertion. Another table in the core of the database structure defines the commands. This table actually contains a lot of configuration data, but for now only the command labels are considered.

ID	Label	...
26	Spatialization	
31	RecSample	
32	PlaySample	
60	Harmonizer	

Table 2. Table structure *commands*

By referring to the IDs of the rows of these two tables, *cues* and *commands*, it is possible to define some basic features of an event: the command type and the start and end positions (cues).

ID	Command	Start	Stop	...
122	31	126	123	
119	32	121	123	
118	31	120	121	
120	26	119	121	
116	60	119	123	
121	32	122	124	

Table 3. Table structure *events*

The dynamic linking of tables leads to a central table of *events* which actually contains no explicit values, but

only IDs of rows of other tables. Some advantages of this approach become evident at this point: for instance, the order in time of events only depends on the attributes page, measure, and beat of the cues assigned to them. A change in the positioning of a cue automatically affects all events linked to it. Inserting new cues and assigning events to them is also easy and straightforward. Furthermore, the cue labels are independent from the ordering mechanism and can be arbitrarily chosen. This makes it possible to define very intuitive and easily recognizable markers in the score. The user can modify these labels at any time without having to associate them again to every single corresponding event. The same is also true, for example, for the command labels.

Explaining the details of all table structures of the database is beyond the scope of this paper. However, in the current state of development, the database contains more than twenty tables, and this number will increase further with the implementation of additional features.

4. AUTOMATIC SUBCOMMAND GENERATION

For every command a list of parameters is defined that control the associated algorithm. Each parameter has a different label which describes the function within the algorithm. The list can have any length and may contain references to timed parameter sequences, which are stored in separate tables.

In addition to these user-defined parameters, there is the possibility to automatically generate special subcommands for each command. The format of these subcommands is predefined by the system, as they are used for specific purposes. The most evident example is the remote control of mixers, which are at the centre of the performance system. The mixer configurations are represented inside the database in several dynamically linked tables (declaring input/output labels, MIDI parameters, etc.). As mentioned before, certain events can be individually associated to the various mixer inputs, while each command can be globally associated to a mixer output.

For each event of this type, a series of subcommands is automatically generated in order to open and close the

corresponding output on all the input channels defined in the event. This type of approach, which is also applied to some other procedures, substantially reduces the redundancy of data and the number of instructions to be called explicitly. The strategy of handling subcommands using linked tables also increases the flexibility of the system. For instance, it is very easy to completely reconfigure the setup in order to work with different types of mixers, without changing anything in the command configurations or in the input assignments of the events. It is also possible, for instance, to change the configuration of the logical output (MIDI parameters, output label, etc.) without redefining the corresponding commands.

5. GOTO FEATURE AND BACKUP IMPLEMENTATION

Using the basic data from the three tables shown above, we can build a score-like representation of the events like the one shown in figure 2. Here, each “voice”

	A.intro	A.1a	A.1b	A.2	A.stop	B.1	B.stop
Harmonizer	▶				■		
PlaySample			▶		■	▶	■
RecSample		▶	■		▶	■	
Spatialization	▶		■				

Figure 2. Timeline display of a score

corresponds to a command, whereas the timeline shows the labels of the cues (ordered by their positions in the score).

This data representation not only provides the start and stop positions of an event, but at the same time it also shows all active events at a given cue. In fact, this information can be obtained by querying the *cues* and *events* tables, allowing the implementation of a GOTO mechanism for jumping to any cue and triggering all events that must be active at that instant. This feature is particularly helpful during the testing and the rehearsal phase of a live electronic performance.

The GOTO feature is also useful for the implementation of a backup system. Such a redundant system is relatively easy to build in a situation where all the processing is run on a single computer [2]. In a configuration with tasks distributed on various peripherals [1], a complete backup would mean duplicating all machines (computers, mixers, external signal processors). As this is obviously not a very practical solution, a partial backup strategy has been implemented which focuses on the most delicate tasks. The central MAX patch is run on two computers in parallel, connected through a local network. One of them actually processes data and the other one follows the cues in synchrony. In case of problems, the second computer can use the GOTO mechanism to immediately reconstruct the correct settings for all machines at that point of the score and take over the control of the performance.

6. DATABASE IMPLEMENTATION

For the implementation of the database using Structured Query Language (SQL), a robust database management system was necessary, as well as an environment which would allow the creation of a user-friendly interface. An open-source approach has been chosen to develop a system based on the integration of MySQL, Apache and PHP. In this configuration, MySQL provides the basic database management, including the storage of data and the processing of queries across the tables. Apache is a web server which allows the use of the scripting language PHP to communicate with MySQL via a standard Internet browser. This way of creating dynamic web pages gives the possibility of designing an intuitive interface with graphical elements like buttons, menus, etc. Figure 3 shows the browser window for the editing of an event.

The real database structure is hidden as much as possible from the user, who needs to organize data on a higher symbolic level. Commands and cues for start/stop

are thus not accessed by ID, but rather by pop-up menus with the labels defined in the corresponding tables. Besides some additional parameters (declared in the command table as well), a series of checkboxes show the labels of all available inputs according to the settings in the mixer configuration. All signal routing can be modified by clicking these checkboxes without worrying about the actual configuration of the mixers.

7. DUMPING DATA

The communication between MySQL and MAX is based on the Open Sound Control (OSC) network protocol [4]. Actually, PHP is used to extract data from MySQL and to format them as strings that correspond to the structure of messages in MAX. An OSC client written in PHP (based on the port by Andrew Schmeder, available at <http://www.cnmat.berkeley.edu/OpenSoundControl/>) allows the user to export these strings as messages to MAX via UDP.

The data representation in MAX is quite different from the original MySQL tables; whereas in MySQL the structure of the tables is highly normalized in order to minimize the data redundancy and avoid inconsistencies, the data structure in MAX is reorganized and optimized for an efficient use with standard objects.

The [coll] object stores messages as data associated to numerical indexes or symbols. For example, the information about cues is extracted in PHP is sorted by score position, and then sent to MAX as lists linked to

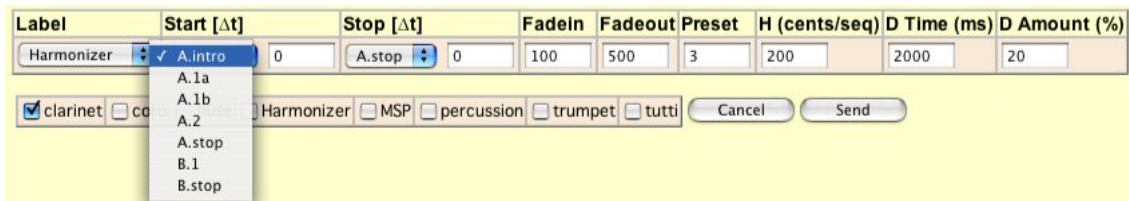


Figure 3. Event editing (snapshot)

symbols which correspond to the labels. The data of table 1 are translated into a [coll] structure like this:

```
A.intro, 119 1 1 4;
A.1a, 120 1 5 2;
A.1b, 121 5 1 1;
A.2, 126 5 3 1;
A.stop, 123 5 7 4;
B.1, 122 6 1 1;
B.stop, 124 9 3 1;
```

In this way the cues can be directly accessed by sending labels as symbols to the [coll]. Whereas the last three items represent the page, measure and beat numbers, the first item is the original table row ID. This ID is used in other [coll] objects as a numerical index with data defining the associated events and their behaviour.

Furthermore, the data stored in MAX can be dumped to a series of text files. This also allows the use of the patch without a running web server. This “off-line” mode is generally used in the final performance, when changes are no longer necessary.

Figure 4 shows the different methods of interaction between database, main patcher, and backup patcher. The patchers are always located on different machines, whereas the database can run on either of the two or on another computer of the local network.

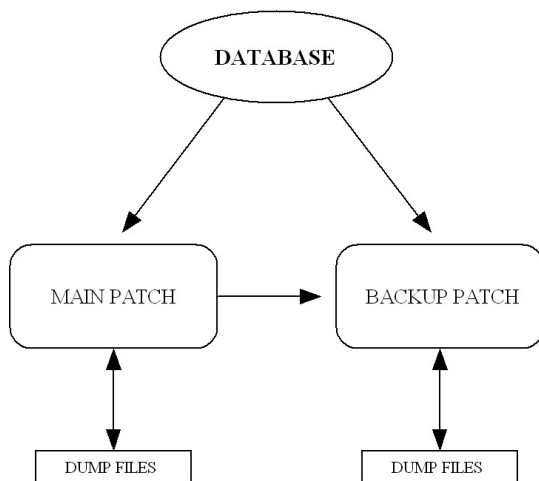


Figure 4. Data flow

8. CONCLUSIONS

The database system leads to a more generalized approach to live electronics. Different “electronic scores” can be efficiently described by the same system and only some parts of the control and processing structure have to be modified. In fact, the presented system has been used in several concert situations with various live electronic setups. Its last implementation was applied to recent Tempo Reale performances of Luciano Berio’s *Ofanim* (for two instrumental groups, two children’s choirs, female voice and live electronics) in London, Florence and Budapest in 2004.

In each of these occasions the system has indeed proved to be flexible and user-friendly. The consistency of data guaranteed by an SQL-based approach also improves the reliability of the performance. The speed of data management is helpful when preparing a performance in the studio, but it is essential during the interaction with musicians and in the phase of adapting parameters to the acoustic reality of concert halls, as the restricted amount of rehearsal time usually makes complex reconfigurations hard to achieve. It is a great advantage in these situations to have the opportunity to focus on musical problems rather than on data handling.

9. REFERENCES

- [1] Giomi, F., Meacci, D. and Schwoon, K., 2003. “Live electronics in Luciano Berio’s Music”, *Computer Music Journal* 27 (2), pp. 30-46.
- [2] Madden, T., Smith R. B., Wright M. and Wessel, D., 2001. “Preparation for Interactive Live Computer Performance in Collaboration with a Symphony Orchestra”, *Proceedings of the 2001 International Computer Music Conference*, La Habana, Cuba.
- [3] Puckette, M., 2002. “MAX at seventeen.” *Computer Music Journal* 26 (4), pp. 31-43.
- [4] Wright, M. and Freed, A., 1997. “Open Sound Control: A New Protocol for Communicating with Sound Synthesizers”. *Proceedings of the International Computer Music Conference*, Thessaloniki, Hellas, pp. 101-104.