Model Based Testing of an Interactive Music System

Clément Poncelet, Florent Jacquemard

Model Based Testing of an Interactive Music System [*]

Clément Poncelet
DGA/INRIA and Ircam (UMR SMTS - CNRS/UPMC)
clement.poncelet@ircam.fr

Florent Jacquemard
INRIA and Ircam (UMR SMTS - CNRS/UPMC)
florent.jacquemard@inria.fr

December 2014

Abstract

The role of an interactive music system (IMS) is to accompany musicians during live performances, like a real musician. It reacts in realtime to audio signals from musicians, according to a timed specification called mixed score, written in a domain specific language. Such goals imply strong requirements of temporal reliability and robustness to unforeseen errors in input, yet not so much studied in the computer music community.

We present the application of model-based testing techniques and tools to a state-of-the-art IMS, including the following tasks: generation of relevant input data for testing (including timing values) following coverage criteria, computation of the corresponding expected output, according to the semantics of a given mixed score, black-box execution of the test data and verdict. Our method is based on formal models compiled directly from mixed scores, and passed, after conversion to timed automata, to the model-checker Uppaal. This fully automatic approach has been applied to real mixed scores used in concerts and the results obtained have permitted to identify bugs in the target IMS.

1 Introduction

Score based interactive music systems (IMS) [13] are involved in live music performances and aim at acting as an electronic musician playing with other human musicians. Such a system requires a *mixed score* describing the parts of human musicians (input) together with electronic parts (output). During a performance, it aligns in real-time the performance of the human musicians to the score, handling possible errors, detects the current tempo, and plays the electronic part. Playing is done by passing messages to an external audio environment such as MAX [12]. A popular example of this scenario is automatic accompaniment [4].

An IMS is therefore a *reactive* system, interacting with the outside environment (the musicians), under strong timing constraints: the output messages must indeed be emitted at *the right moment*, not too late but also not too early. It is important to be able to assess the behavior of an IMS on a given score before its real use in a concert. A traditional approach is to rehearse with musicians, trying to detect potential problems manually, i.e. by audition. This tedious method offers no real guaranty since it is not precise, not complete (it covers only one or a few particular musician's performances), and error prone (it relies on a subjective view of the expected behavior instead of a formal specification).

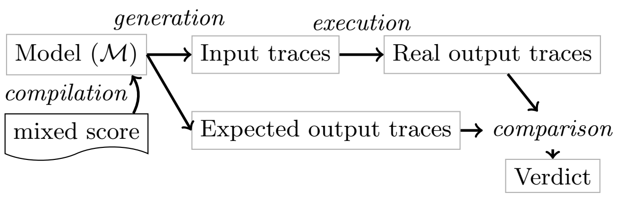

Figure 1: *Score-based IMS testing workflow*

In this paper, we present the application of model based testing (MBT) techniques to a score based IMS called Antescofo, used frequently in world class concerts in the contemporary repertoire. Roughly, our method proceeds with the following steps depicted in Figure 1. First, a given mixed score is compiled into an intermediate representation (IR). This formalism can be presented as a table of finite state machines extended with delays and asynchronous communications with the environment (using input and output symbols) and between machines. It is viewed as the model $\mathcal{M}$ of the Antescofo reactions according to the score. The IR is then transformed into a timed automata (TA) network [1], assuming some restrictions. TA's are processed by tools from the Uppaal suite [8] to generate covering suites of test cases: input timed traces together with the corresponding expected output; the input traces are then sent to the IMS (as a virtual musician's performance of the score) and the real outcome of the IMS is analyzed, resulting in a test verdict.

Several works like [6] and [8] implement different MBT techniques using Uppaal model checker features. The test problems are reduced into reachability or safety constraints delegated to Uppaal. Our case study presents important originalities compared to

[*]This work has been partly supported by a DGA-MRIS scholarship and the project Inedit (ANR-12-CORD-009).

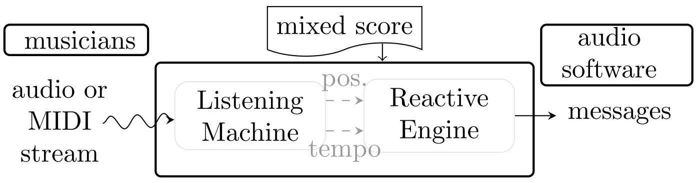

Figure 2: *Architecture of Antescofo*

other MBT applications to realtime systems. On the one hand, the time model supports several time units, including the wall clock time, measured in seconds, and the time of music scores, measured in number of beats relatively to a tempo. This situation raised several new problems for the generation of test suites and their execution. On the other hand, mixed scores specify completely the expected timed behavior (of the IMS), based on the DSL semantics implemented in the compiler described in Section 3.2. Hence, the formal specification of this behavior is produced automatically from the score (instead of being written manually). This enables a fully automatic test scenario fitting well in a music authoring workflow where scores in preparation are constantly evolving.

2 Preliminaries

We first introduce the IMS Antescofo, its domain specific language (DSL) for mixed scores and our MBT framework.

2.1 The score-based IMS Antescofo

Figure 2 describes roughly the architecture of Antescofo, which is made of two main modules. A *listening machine* (LM) decodes an audio or midi stream incoming from a musician and infers in realtime: (*i*) the musician's position in the given mixed score, (*ii*) the musician's instantaneous pace (*tempo*, in beats per minute) [3]. These values are sent to a *reactive engine* (RE) which schedules the electronic actions to be played, as specified in the mixed score. The actions are messages emitted *on time* to an audio environment. Therefore, the information exchanged between LM and RE as well as between RE and the output environment of the system is made of discrete events.

The mixed scores of Antescofo are written in a textual reactive synchronous language enabling the description of the electronic accompaniment in reaction to the detected instrumental events. We give here a simplified *abstract syntax* corresponding to a fragment of this language, in order to illustrate our test framework (see [7] for more complete descriptions). Let $\mathcal{O}$ be a set of *output messages* (also called *action symbols* and denoted a) which can be emitted by the system and let $\mathcal{I}$ be a set of *event symbols* (denoted e) to be detected by the LM (i.e. positions in score). An *action* is a term

BPM 144
$\mathsf{evt}(e_1, 1, s_1)$;
$\mathsf{evt}(e_2, 1, [])$;
$\mathsf{evt}(e_3, 1, [])$ where

$$
\begin{aligned}
s_1 = {} & \mathsf{act}(0, [a_0], [\,]);\\
& \mathsf{act}(0, s_2, [\mathsf{loose}; \mathsf{global}])\\
s_2 = {} & \mathsf{act}(0, [a_1], [\,]);\\
& \mathsf{act}(1/2, [a_2], [\,]);\\
& \mathsf{act}(1/2, [a_3], [\,]);\\
& \mathsf{act}(1/2, [a_4], [\,]);\\
& \mathsf{act}(1/2, [a_5], [\,]);\\
& \mathsf{act}(1/3, [a_6], [\,]);\\
& \mathsf{act}(1/3, [a_7], [\,])
\end{aligned}
$$

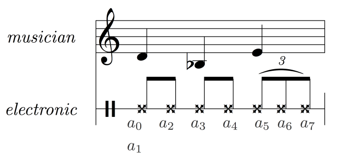

Figure 3: *A mixed score in abstract syntax.*

$\mathsf{act}(d, s, al)$ where d is the delay before starting the action, s is either an atom in $\mathcal{O}$ or a finite sequence of actions (such a sequence is called a *group*), and al is a list of attributes. A *mixed score* is a finite sequence of *input events* of the form $\mathsf{evt}(e, d, s)$ where $e \in \mathcal{I}$, d is a duration and s is the top-level group *triggered* by e. Sequences are denoted with square brackets [,] and the empty sequence is [ ].

We consider here two time units for expressing delays and durations d: (*i*) the number of beats (default unit): a logical time unit traditionally used in music scores that we call *relative time*, and (*ii*) milliseconds (ms), referred to as *physical time*. The reconciliation of the relative and physical times is done through the detected tempo values.

Example 1 *Figure 3 displays a small extract of a mixed score, in abstract syntax and traditional musical notation. The top stave is the musician part, it contains three quarter notes (whose duration is one beat): D4, B3♭ and E4. The bottom field is the electronic part with output messages $a_0, \ldots, a_7$. Note that actions are triggered only by the first event which fires the top group s1 when detected. s1 fires concurrently and simultaneously (0 delay) the atomic action a_0 and the loose global group s_2. A delay of $\frac{1}{2}$ (action a_2) corresponds to an eighth note i.e. half of a beat.*

The high-level attributes attached to an action $\mathsf{act}(d, s, al)$ are indications regarding musical expressiveness [4]. We consider here four attributes for illustration purpose (their interpretation will be defined formally in Section 3.2): two attributes are used to express the synchronization of the group s to the musician's part: loose (synchronization on tempo) and tight (synchronization on events) and two attributes describe strategies for handling errors in input: local (skip actions) and global (play actions immediately at the detection of an error). An error is in particular an event of the score missing during performance, either because the musician did not play it or because it was not detected by the LM.

Example 2 *The possible interpretations of the actions in our running example according to 4 combinations of strategies for err and sync is depicted in Figure 4.*

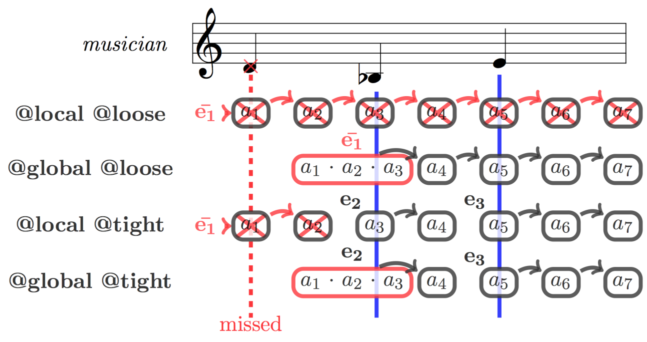

Figure 4: *Interpretation of the running example when the first note is missed, for various attributes.*

2.2 Model-Based Testing

We consider a black-box testing conformance approach for Antescofo, based on timed traces comparisons. We assume a given mixed score $\mathcal{M}$ with a default tempo value.

A *timed trace* is a sequence of pairs $\langle a, t \rangle$ made of a symbol $a \in \mathcal{I} \cup \mathcal{O}$, and a *timestamp* $t \in \mathbb{R}^+$, either in physical or relative time. A trace containing symbols exclusively in $\mathcal{I}$ (resp. $\mathcal{O}$) is called an *input trace* (resp. an *output trace*). We denote below $\mathcal{T}_{\mathsf{in}}$ (resp. $\mathcal{T}_{\mathsf{out}}$) the set of input (resp. output) traces with relative timestamps. The *ideal trace* is the input trace consisting in the projection of all events in $\mathcal{M}$ with their duration.

Example 3 *The ideal trace for the score in Figure 3 is the following:* $\langle e_1, 1 \rangle \cdot \langle e_2, 2 \rangle \cdot \langle e_3, 3 \rangle$.

By definition of music performance, traces of real executions can be arbitrarily far from ideal traces: the tempo and delays can diverge from the written values (the musician adding her/his own expressiveness values), and moreover their can be errors during a performance (missing notes).

We shall, in Section 3, associate to the model $\mathcal{M}$ two formal models: a specification $\mathcal{E}$ of the possible behaviors of the environment (i.e. the events of the musician playing $\mathcal{M}$, as detected by the LM) and a specification $\mathcal{S}$ of the behavior of the system on $\mathcal{M}$. In our case, $\mathcal{E}$ can be seen as a subset of $\mathcal{T}_{\mathsf{in}}$ and $\mathcal{S}$ as a function from $\mathcal{T}_{\mathsf{in}}$ into $\mathcal{T}_{\mathsf{out}}$. This disymmetry between $\mathcal{E}$ and $\mathcal{S}$ reflects our case study, with on one side the musician and the LM and on the other side the RE (see Figure 2).

A *test case* is a pair $\langle t_{\mathsf{in}}, t_{\mathsf{out}} \rangle \in \mathcal{T}_{\mathsf{in}} \times \mathcal{T}_{\mathsf{out}}$ where $t_{\mathsf{in}} \in \mathcal{E}$ and $t_{\mathsf{out}} = \mathcal{S}(t_{\mathsf{in}})$. Two complementary approaches for the offline generation of t_{in} are presented in Section 4.

The execution of a test case $\langle t_{\mathsf{in}}, t_{\mathsf{out}} \rangle$ consists in several tasks summarized in the following definition of conformance. First, we pass the events of t_{in} to the *implementation under test* (IUT), i.e. the IMS Antescofo, respecting the timestamps. Second, we monitor the outcome of the IUT in an output trace $t'_{\mathsf{out}} = \text{IUT}(t_{\mathsf{in}})$. Finally, we compare t'_{out} to $t_{\mathsf{out}} = \mathcal{S}(t_{\mathsf{in}})$. We define the *conformance* of the IUT to $\mathcal{S}$ *wrt* $\mathcal{E}$ as: $\forall t_{\mathsf{in}} \in \mathcal{E}$, $\text{IUT}(t_{\mathsf{in}}) = \mathcal{S}(t_{\mathsf{in}})$. This is a particular case of the relation rtioco considered in [6, 8].

This definition makes sense only if the timestamps of t_{in}, t_{out} and t'_{out} are in the same time unit. We will show how this important issue is addressed in practice in Section 4, with several options for the conversion of all traces into physical time (thanks to the addition of tempo values).

3 Models

We present a procedure for compiling a mixed score into an intermediate representation (IR) used as a model for test case generation in our MBT framework.

3.1 Intermediate Representation

The IR has the form of an executable code modeling the expected behavior of Antescofo on a given score. We present here a simplified version of Antescofo's IR suitable for our presentation, leaving features such as thread creation, conditional branching, and variable handling outside of the scope of this paper.

Syntax. An IR is a finite sequence (called *network*) of FSMs of the form $\mathcal{A} = \langle \Sigma, Q, \ell_0, \Delta_0, \Delta_1 \rangle$ where Σ is an alphabet partitioned into $\Sigma = \Sigma_{\mathsf{in}} \uplus \Sigma_{\mathsf{out}} \uplus \Sigma_{\mathsf{sig}}$ (respectively the sets of input, output symbols and internal signals), Q is a finite set of locations, partitioned into $Q = Q_0 \uplus Q_1$, $\ell_0 \in Q$ is the initial location, $\Delta_0 \subseteq Q_0 \times (\Sigma_{\mathsf{out}} \uplus \Sigma_{\mathsf{sig}}) \times Q$ is a finite set of *synchronous transitions* (denoted $\ell \xrightarrow{\sigma!} \ell'$), and $\Delta_1 \subset Q_1 \times ((\Sigma_{\mathsf{in}} \uplus \Sigma_{\mathsf{sig}} \uplus \mathbb{R}^+) \times Q)^+$ is a finite set of *asynchronous transitions*. We write $\ell \xrightarrow{\tau_1?|\ldots|\tau_p?} \ell_1|\ldots|\ell_p$, for $\langle \ell, \langle \tau_1, \ell_1 \rangle, \ldots, \langle \tau_p, \ell_p \rangle \rangle \in \Delta_1$ where $p \geq 1$, $\ell \in Q_1$, $\ell_1, \ldots, \ell_p \in Q$, and $\tau_1, \ldots, \tau_p \in \Sigma_{\mathsf{in}} \cup \Sigma_{\mathsf{sig}} \cup \mathbb{R}^+$. Moreover, there must be at most one delay $d \in \mathbb{R}^+$, and at most one input event in Σ_{in} amongst $\tau_1, \ldots, \tau_p$. Informally, every synchronous transition $\ell \xrightarrow{\sigma!} \ell'$ in Δ_0 describes the emission, while in *source* location ℓ, of the output message or signal σ, followed by the change of location to the *target* location ℓ', and every asynchronous transition $\ell \xrightarrow{\tau_1?|\ldots|\tau_p?} \ell_1|\ldots|\ell_p$ in Δ_1 describes the concurrent wait, in source location ℓ, of $\tau_1, \ldots, \tau_p$. The first event τ_i occurring provokes a change of location to the target ℓ_i. Moreover, every location can be the source of at most one transition in $\Delta_0 \cup \Delta_1$.

Semantics. In the above partition of Σ, Σ_{in} and Σ_{out} are sets of symbols used for communication with the environment, and Σ_{sig} is an auxiliary set of internal signals used for communication between FSMs of a network.

We consider a model of superdense time [10, 11] with timestamps in $(t, n) \in \mathbb{R}^+ \times \mathbb{N}$, where t is a date in relative time called *logical instant*. The logical time can flow only when in locations of Q_1 (sources of asynchronous transitions). In locations of Q_0, t is fixed and the execution of synchronous transitions will increase the second component $n \in \mathbb{N}$.

Let $\bar{\mathcal{A}} = \mathcal{A}_1 \parallel \ldots \parallel \mathcal{A}_k$ be a network of k FSMs composed in parallel with $\mathcal{A}_i = \langle \Sigma, Q^i, \ell_0^i, \Delta_0^i, \Delta_1^i \rangle$ for all $1 \leq i \leq k$. A *state* of $\bar{\mathcal{A}}$ is a tuple of the form $\langle t, n, [\ell_1, \ldots, \ell_k], \Theta, \omega \rangle$ where (t, n) is a timestamp, $[\ell_1, \ldots, \ell_k] \in Q^1 \times \ldots \times Q^k$ the array of current locations, $\Theta \subset \Sigma_{\mathsf{sig}}$ is a finite set of internal signals or messages and $\omega \in \Sigma_{\mathsf{out}} \uplus \{\bot\}$. The initial state of $\bar{\mathcal{A}}$ is $s_0 = \langle 0, 0, [\ell_0^1, \ldots, \ell_0^k], \emptyset, \bot \rangle$. An *external event* τ can be the arrival of an input in Σ_{in} or the expiration of a delay $d \in \mathbb{R}^+$ (we assume external timers are in charge of notifying the expiration with the necessary (re)-conversion of delays in physical time using (updated) tempo values, see [7]). We assume that a given input trace $t_{\mathsf{in}} \in \mathcal{T}_{\mathsf{in}}$ specifies the arrivals of inputs in Σ_{in}.

The *moves* of $\bar{\mathcal{A}}$ (between states) are defined as follows.

$$\langle t, n, [\ell_1, \ldots, \ell_k], \Theta, \omega \rangle \to \langle t, n{+}1, [\ell'_1, \ldots, \ell'_k], \Theta \cup \{\sigma\}, \bot \rangle \qquad (\mathsf{es})$$

$$\langle t, n, [\ell_1, \ldots, \ell_k], \Theta, \omega \rangle \to \langle t, n+1, [\ell'_1, \ldots, \ell'_k], \Theta, \sigma \rangle \qquad (\mathsf{em})$$

where the locations $\ell'_1, \ldots, \ell'_k$ are defined as follows. Let i be the smallest index (in the array of current locations) such that $\ell_i \in Q_0$, and there exists $\ell_i \xrightarrow{\sigma!} \ell''_i \in \Delta_0^i$, with $\sigma \in \Sigma_{\mathsf{sig}}$ for (es), and $\sigma \in \Sigma_{\mathsf{out}}$ for (em). Then, $\ell'_i = \ell''_i$ and $\ell'_j = \ell_j$ for all $j \neq i$. Here, a synchronous transition is executed and the signal (es) (resp. message (em)) emitted is added to Θ (resp. ω).

$$\langle t, n, [\ell_1, \ldots, \ell_k], \Theta, \omega \rangle \to \langle t, n+1, [\ell'_1, \ldots, \ell'_k], \emptyset, \bot \rangle \qquad (\mathsf{rs})$$

where $\ell_1, \ldots, \ell_k \in Q_1$, and for all i, $1 \leq i \leq k$, there exists $\ell_i \xrightarrow{\tau_{i,1}?|\ldots|\tau_{i,p_i}?} \ell_{i,1}|\ldots|\ell_{i,p_i} \in \Delta_1$. Moreover, let j' be the smallest j such that $1 \leq j \leq p_i$ and $\tau_{i,j} \in \Theta \cap \Sigma_{\mathsf{sig}}$. If j' exists, then $\ell'_i = \ell_{i,j'}$, and otherwise $\ell'_i = \ell_i$. Here, the signals expected and present in Θ are all received at once, and Θ and ω are flushed.

$$\langle t, n, [\ell_1, \ldots, \ell_k], \Theta, \omega \rangle \to \langle t', 0, [\ell'_1, \ldots, \ell'_k], \emptyset, \bot \rangle \quad (\mathsf{rx})$$

where $\ell_1, \ldots, \ell_k \in Q_1$, and for all i, $1 \leq i \leq k$, there exists $\ell_i \xrightarrow{\tau_{i,1}?|\ldots|\tau_{i,p_i}?} \ell_{i,1}|\ldots|\ell_{i,p_i} \in \Delta_1^i$, and moreover, letting $T^i = \{\tau_{1,1}, \ldots, \tau_{k,p_k}\}$, it holds that $T^i \cap \Theta = \emptyset$ (*i.e.* (rs) cannot be fired). Then $t' \geq t$ is the first date (after t) of occurrence of an external event $\tau \in T^i$, with a priority for delays ($\mathbb{R}^+$) over inputs (Σ_{in}). For all $1 \leq i \leq k$, if $\tau_{i,j} = \tau$ for some $1 \leq j \leq p_i$, then $\ell'_i = \ell_{i,j}$, otherwise $\ell'_i = \ell_i$.
Here, t' is the date of the first event τ after t in t_{in} or the expiration of a delay $d \in \mathbb{R}^+$ and in the latter case, $t' = t + d$.

Note that with the above definitions, the moves are mutually exclusive and performed in the given priority order. In particular, the steps (es) and (em) are repeated as long as there exists executable synchronous transitions (i.e. locations of Q_0 in the state). Note also that the steps (es) and (rs) may loop in the same logical instant.

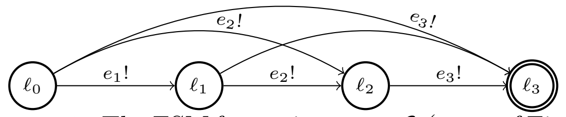

Figure 5: *The FSM for environment $\mathcal{E}$ (score of Fig. 3).*

A *run* ρ is a sequence of states $s_0, s_1, \ldots$ such that for all $i \geq 0$, there is a move between s_i and s_{i+1}. We associate to ρ the output trace t_{out} defined as the projection of ρ on its first and last components different from $\bot$. The above definitions ensure *determinism* of IRs, in the sense that there exists exactly one run ρ (hence one output trace t_{out}) for a given input trace $t_{\mathsf{in}} \in \mathcal{T}_{\mathsf{in}}$. This implies that $\mathcal{S}$ is functional when it is defined as an IR. By abuse of notations we shall keep writing $t_{\mathsf{in}} \in \mathcal{E}$ and $\mathcal{S}(t_{\mathsf{in}})$ when $\mathcal{E}$ and $\mathcal{S}$ are IRs.

The synchronous aspect and the set of instant logic signals of the IR model are inspired by the programming language Esterel for reactive systems [2].

3.2 Compiling mixed scores into IR

An IR is constructed directly from a given mixed score sc, during the parsing of the latter.

For the sake of conciseness, we do not give the inferences defining recursive constructions, but instead, a graphical representation of the IR obtained for our running example. The environment IR ($\mathcal{E}$) is built with a single pass through score events. There are several options for constructing $\mathcal{E}$, regarding missed notes (see Section 4). In Figure 5, the musician modeled by $\mathcal{E}$, when in ℓ_0, can miss the first note e_1 (upper edge to location ℓ_2) or e_1 and e_2 (ℓ_3). In ℓ_1, he can miss e_2 (going to ℓ_3).

The proxy IR ($\mathcal{P}$) in Figure 6 provides a definition of errors. We assume one internal signal $\overline{e_i} \in \Sigma_{\mathsf{sig}}$ for each input event in $e_i \in \mathcal{I}$. The proxy $\mathcal{P}$ emits this signal when e_i is detected as missing, because an event e_j with $j > i$ was received. The proxy IR will simplify the complex task of specifying error management in other IRs.

Figure 7 and 8 show the IR obtained from the running example, for two different synchronization strategies (resp. loose and tight). Those models of the IUT behavior are constructed by iteratively traversing the sequence of actions in a group. The parts build at each step are framed and annotated as $\mathcal{T}_g$, for starting the

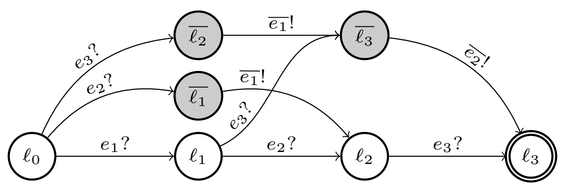

Figure 6: *The FSM for proxy $\mathcal{P}$ (score of Fig. 3).*

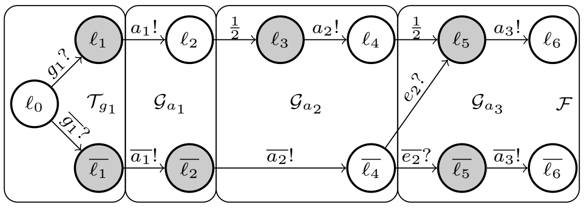

Figure 7: *The group s_2 FSM with al* = [loose; global]

group, $\mathcal{G}_a$, for handling one atomic action a, or $\mathcal{F}$, for ending the IR.
In Figure 7 (attributes loose, global), the top part of the IR (locations $\ell_1, ..., \ell_6$) corresponds to a mode for a normal behavior (in absence of errors), which consists in sending successively the actions with their respective delays. The bottom part of the IR (locations $\overline{\ell_1}, ..., \overline{\ell_6}$) corresponds the behavior in case of error: send instantaneously all actions (without delay) until the next detected event (location $\overline{\ell_4}$).
In Figure 8 (attributes tight, global), the top and bottom parts also correspond to normal and error modes. There is a possible switch (location ℓ_4) from the normal to the error mode if a missed event is detected. Also, in location ℓ_2, if the event e_2 is earlier than expected, either as detected (edge e_2?) or as missing (edge $\overline{e_2}$?) then there is a switch to an intermediate mode (respective locations ℓ_{e2} and $\overline{\ell_{e2}}$).

4 Implementation and Results

Let us now present the implementation and results of our MBT approach, applied to the score-based IMS Antescofo.

Compiling mixed score into IR

Compiling mixed scores into IR has been implemented as a command line tool, written in C++ on the top of the original Antescofo's parser. The parsing produces an abstract syntax tree which is traversed using a visitor pattern in order to build the IR following the approach presented in Section 3.2. Several options are offered for the production of the IR corresponding to the environment $\mathcal{E}$, in particular regarding the number of possible successive errors (missed notes). The most general case (any note can be missed) results in a model $\mathcal{E}$ with a quadratic (in score's size) number of transitions and an exponential number of input traces. The explosion can be controlled by choosing appropriate hypotheses on the environment $\mathcal{E}$.

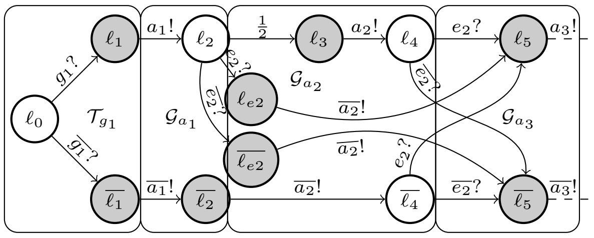

Figure 8: *The group s_2 FSM with al* = [tight; global]

Translation of IR into Timed Automata

The IR is then translated into a network of timed automata (TA), in a format that can be handled by tools of the Uppaal suite for MBT. Some graphical coordinates are computed during compiling, and used for a nice display of the score models under Uppaal, providing composers with useful visual feedbacks of the low level control-flow in their mixed score.

The translation of IR, in the simplified version presented in Section 3.1, into equivalent TA is possible whenever all the delays are expressed in *relative time*. Indeed, in TA, all the clock values are expressed in a unique abstract *model time unit* (mtu). For the TA associated to the environment IR model $\mathcal{E}$, several options are offered for adding lower and upper bounds on the duration of each event, in order to limit the state exploration for the generation of t_{in}.

Some care has to be taken for the simulation of the move rule (rs) in the semantics of IR. In fact, in the states of IR runs, signals are stored in an unordered set at each logical instant, and hence can be received in an arbitrary order. This is not the case with TA models. In order to sort the the interleaving between signals and external input events, an auxiliary step is performed, possibly modifying the IR structure (*e.g.* with urgent state in proxy, and early signals for group's triggers).

We have chosen to use an IR instead of directly translating mixed scores into TA [7] because there is a clear correspondence between this ad'hoc model and the semantics of Antescofo's DSL, and the general IR format is used for other purposes and includes features not supported by TA (such as variables and dynamic thread creation).

Model-based generation of covering suites of test cases

We use the Uppaal extension called CoVer [8] to generate automatically suites of test cases, under a certain $\mathcal{E}$, that cover the possible behavior of the specification $\mathcal{S}$ according to some coverage criteria. These criteria are defined by a finite state automaton $\mathcal{O}bs$ called *observer* monitoring the parallel execution of $\mathcal{A}_\mathcal{E}$ and $\mathcal{A}_\mathcal{S}$, the TA associated to the IR $\mathcal{E}$ and $\mathcal{S}$. Every transition of $\mathcal{O}bs$ is labeled by a predicate checking whether a transition of $\mathcal{A}_\mathcal{E} \| \mathcal{A}_\mathcal{S}$ is fired. The model checker Uppaal is used by CoVer to generate the set of input traces $t_{\mathsf{in}} \in \mathcal{T}_{\mathsf{in}}$ resulting from an execution of the Cartesian product of $\mathcal{A}_\mathcal{E} \| \mathcal{A}_\mathcal{S}$ with $\mathcal{O}bs$ reaching a final state of $\mathcal{O}bs$.

For loop-free IR $\mathcal{S}$ and $\mathcal{E}$, with an observer checking that all transitions of $\mathcal{A}_\mathcal{E}$ and $\mathcal{A}_\mathcal{S}$ are fired, CoVer will return a test suite $\mathcal{T}$ complete for non-conformance: if there exists an input trace $t_{\mathsf{in}} \in \mathcal{E}$ such that $\mathrm{IUT}(t_{\mathsf{in}})$ and $\mathcal{S}(t_{\mathsf{in}})$ differ, then $\mathcal{T}$ will contain such an input trace. Note that the IR produced by the fragment of the DSL of Section 2.1, using the procedure of Section 3.2, are loop-free. However this is not true for the general DSL which allows e.g. jump to label instructions.

In practice, we avoid state explosion with appropriate restrictions on $\mathcal{E}$ (number of missed events, see above) and the associated TA $\mathcal{A}_{\mathcal{E}}$ (bounds on event's durations).

Test cases generation by fuzzing ideal trace

An alternative method for the generation of relevant test cases is to start with the ideal trace associated to a mixed score and add deformations of several kinds. *Time-warps* [5] and variants like *Time-Maps* (Jaffe 1985), *Time-deformations* (Anderson and Kuivila 1990), are continuous and monotonically increasing functions used to define either *variations of tempo* or variations of the duration of individual notes wrt the written score events (*time-shift*). Some models of performance [9, 5] are defined by combination of these two transformations, defined independently. We consider a discrete version of such models, with extended input traces t_{in} made of triples $\langle a, t, p \rangle$, where a, t are like in Section 2.2 and p is a tempo value in beats per minute (BPM). The time-shifts are applied to the timestamps t (they are expressed in relative time), and the tempi b are values on a tempo curve. An important difference with [9, 5] is the possibility to include missed notes in input traces.

Tests execution and verdicts

We have developed several scenarios for the execution of a test case $\langle t_{\mathsf{in}}, t_{\mathsf{out}} \rangle$, corresponding to several boundaries for the black box tested inside the whole system – see Figure 2.

In a first scenario, t_{in} contains triples like in the above paragraph. The tempo values are either values in a curve, in the case of traces generated by fuzzing, or a fixed value in the case of trace generated by CoVer. This scenario is performed on a standalone version of Antescofo equipped with an internal *test adapter* module. The adapter iteratively reads one element $\langle e, d, p \rangle$ of a file containing t_{in}, converts d into a physical time value d' (remember that delays are expressed in relative time in t_{in}), and waits d' seconds before sending e and p to the RE. More precisely, it does not physically wait, but instead notifies a *virtual clock* in the RE that the time has flown of d' seconds. This way the test needs not to be executed in realtime but can be done in fast-forward mode. This is very important for batch execution of huge suites of test cases. The timestamps in the expected trace t_{out} are converted from relative to physical time using the tempo values in t_{in}, in order to be compared to the monitored trace t'_{out}. Here, the blackbox is the RE (the LM is idle).

In a second scenario, the tempo values are not read in t_{in} but detected by the LM. The rest of the scenario follows the first case. Here, the blackbox is the RE plus the part of the LM in charge of tempo inference.

A third scenario is executed in a version of Antescofo embedded into MAX (as a MAX patch). In this case, the blackbox is the whole IMS Antescofo, and instead of seeing discrete events to the IUT (like in scenarios 1 and 2), we generate an audio stream with a MIDI synthesizer (in MAX), using the events in t_{in} as MIDI events.

The verdicts are produced offline by a tool comparing the expected and monitored traces t_{out} and t'_{out} with an acceptable latency (about 0.1 ms). The comparison is not totally obvious since we have no clue *a priori* about missed or added actions/events in the traces and about the order of items.

Experiments

Two case studies will be reported: B, a benchmark made of hundreds of little mixed scores, covering many features of the IUT's DSL and *EIN*, a real mixed score of the piece of *Einspielung* by *Emmanuel Nunes* [1]. The first benchmark can be useful for the development (debugging and regression tests) of further versions of the system Antescofo. It aims at covering the IUT's DSL functionality and checking the reactions of the system. The second is a long real test case, for evaluating the scalability of our test method. It is composed of two extracts: the first 4 bars (22 events and 112 actions) and 14 bars (72 events and 389 messages).

Each case study is processed three times, with different numbers of possible consecutive missing events (0, 3 and 6 events) and a bound of 5% for the variation of the duration in the interpretation on each event. A script creates the IR and TA models, generates test suites (using CoVer), executes them according to *the first* scenario presented above and compares the outcome to test cases. Table 1 summarizes the results for the Benchmark B, reporting the number of traces generated by CoVer and the time taken by the whole test. Note that the increase of the number of traces with

[1] `http://brahms.ircam.fr/works/work/32409/`

case	c. miss	nb score	nb trace	time (s)
B	0	582	1843	140
	3	582	5718	334
	6	582	6387	405

case	c. miss	locations	nb trace	time (s)
EIN	0	400/1394	7/35	1/24
	3	518/1812	36/50	3/198
	6	771/2815	67/NA	97/400

Table 1: *Results of experiments*

missing events (the length of the tested scores is generally between 3 to 6 events).The second table summarizes the results for Einspielung, with the number of IR locations, traces and testing time for each extract. CoVer did not succeed to generate the input traces for the 14 bars extract in the case of 6 possible missed events.

Despite CoVer scalability (that can be bypassed with other scenarios), generated suites of traces are relevant and test the IUT for an exhaustive set of possible performances.

A problem encountered with CoVer is that it generates only time optimal test suites, i.e. input trace with minimum time-delay satisfying a given reachability property. This is not well suited to our case study. Indeed, since the trace t_{in} is stamped in relative time, a time optimal t_{in} will result in a geometric progression of the tempo.

5 Conclusion and further work

Thanks to an ad'hoc intermediate representation for mixed scores, and conversion into timed automata, we have developed a fully automatic offline model-based testing procedure dedicated to an interactive music system. An advantage of this case study for MBT is the possibility to generate the formal specifications (as IR) automatically from the given scores. A drawback is the necessity to deal with different time units, in particular relative time. This latter problem prevented us from using the online testing tool Tron [8] (roughly, Tron can deal with several clocks but they must all be defined as a factor of the wall clock).

Our method is designed to test the behaviour of the IMS on one given score, by generation of a covering set of input traces describing a range of musical performance of the score. This approach is advantageous both for IMS debugging, thanks to coverage criteria, and for user assistance to authors of mixed scores, using fuzzing based on models of musical performance. A more general perspective could be to test the behavior of the IMS on *any* score. This would require a complete specification of the IMS (written manually) as *e.g.* an hybrid system, and the automatic generation, as test input, of a covering set of "*extreme*" scores *and* covering sets of performance traces for these scores.

Acknowledgments

The authors wish to thank the members of the teams developing Uppaal and Antescofo for their help.

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126:183–235, 1994.

[2] G. Berry and G. Gonthier. The Esterel Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, 19(2):87-152, 1992.

[3] A. Cont. A coupled duration-focused architecture for realtime music to score alignment. *IEEE TPAMI*, 32(6):974–987, 2010.

[4] A. Cont, J. Echeveste, J.-L. Giavitto, and F. Jacquemard. Correct Automatic Accompaniment Despite Machine Listening or Human Errors in Antescofo. In *proceedings of ICMC*, 2012.

[5] R. B. Dannenberg. Abstract time warping of compound events and signals. *Computer Music Journal*, 21(3):61–70, 1997.

[6] A. David, K. G. Larsen, S. Li, M. Mikucionis, and B. Nielsen. Testing real-time systems under uncertainty. *In proceedings of FMCO*, Springer LNCS 6957:352–371, 2010.

[7] J. Echeveste and all. Operational semantics of a domain specific language for real time musician–computer interaction. *JDEDS*, 23(4):343–383, 2011.

[8] A. Hessel and all. Testing real-time systems using Uppaal. In *Formal methods and testing* 77–117, Springer, 2008.

[9] H. Honing. Structure and interpretation of rhythm and timing. *Dutch Journal of Music Theory*, 7(3):227–232, 2002.

[10] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer, 1992.

[11] C. Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II.* Ptolemy.org, 2014.

[12] M. Puckette. Combining event and signal processing in the max graphical programming environment. *Computer Music Journal*, 15:68–77, 1991.

[13] R. Rowe. *Interactive Music Systems: Machine Listening and Composing.* AAAI Press, 1993.