Model-Based Testing for Building Reliable Realtime Interactive Music Systems

Clement Poncelet, Florent Jacquemard

Model-Based Testing for Building Reliable Realtime Interactive Music Systems *

Clement Poncelet[†1,2,3] and Florent Jacquemard[‡1,2]

[1]IRCAM, 1 place Igor-Stravinsky, 75004 Paris
[2]Sorbonne Universités, Inria, UPMC Univ Paris 06, IRCAM – CNRS UMR SMTS, Paris, France.
[3]DGA Direction Generale de l'Armement (French national defense).

Abstract

The role of an Interactive Music System (IMS) is to accompany musicians during live performances, acting like a real musician. It must react in realtime to audio signals from musicians, according to a timed high-level requirement 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 much addressed by the computer music community.

We present the application of Model-Based Testing techniques and tools to a state-of-the-art IMS, including in particular: offline and on-the-fly approaches for the generation of relevant input data for testing (including timing values), with coverage criteria, the computation of the corresponding expected output, according to the semantics of a given mixed score, the black-box execution of the test data on the System Under Test and the production of a verdict. Our method is based on formal models in a dedicated intermediate representation, compiled directly from mixed scores (high-level requirements), and either passed, to the model-checker Uppaal (after conversion to Timed Automata) in the offline approach, or executed by a virtual machine in the online approach. Our fully automatic framework 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

Interactive Music Systems (IMS) [33] are involved in live music performances and aim at acting as an electronic musician playing with other human musicians. We consider such systems that work with a *mixed score*, written in a

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

[†]clement.poncelet@ircam.fr
[‡]florent.jacquemard@inria.fr

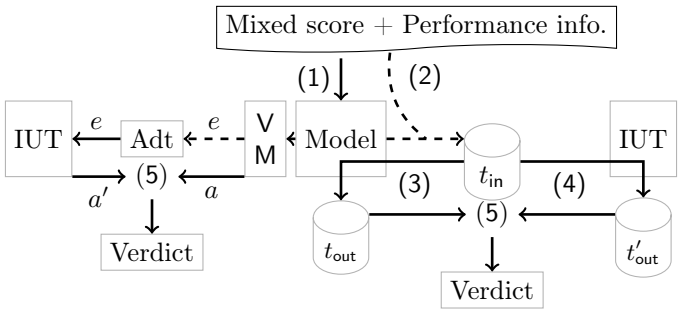

Figure 1: Highlights of our Score-based IMS testing workflows. Left: Online - Right: Offline.

Domain Specific Language (DSL), which describes the input expected from human musicians, together with the electronic output to be played in response. During a performance, a score-based IMS 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. A popular example of this scenario is automatic accompaniment [14].

A score-based IMS is therefore a *reactive* system, interacting with the outside environment (the musicians) under strong timing constraints: the output (generally messages passed to an external audio application such as MAX [32]) must indeed be emitted at *the right moment*, not too late but also not too early. This may be a difficult task since audio calculations often have an important impact on the resource consumptions. In this context, 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).

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 steps depicted in Figure 1. First, (1) a given mixed score is compiled into an Intermediate Representation (IR). This formalism is an executable medium level code modeling the behavior expected from the Implementation Under Test (IUT), the IMS Antescofo, when playing the given mixed score. It has the form of a finite state machine with delays, asynchronous communications and alternations.

Based on this IR, we follow two main approaches for testing: In an *offline* approach (the right side of the figure), the IR is transformed into a Timed Automata (TA) network [3], for generation and simulation purposes. The input test data (2) is generated either by tools from the Uppaal suite [23] for the

generation of covering test suites (under restrictions), or either by adding to an *ideal* trace of musician some fuzz of different kinds, or also by translation from a musical performance. Once some timed input traces t_{in} have been generated (representing a musician's performance on the score), the IR is used to compute offline, by simulation (3), the corresponding output traces t_{out}, expected from the system in response to the input. Finally, every input trace t_{in} is sent to the IMS (4) and the real outcome of the IMS, t'_{out} is compared (5) to the expected output t_{out} in order to produce a test verdict.

In an *online* approach (left side of the figure), the IR is executed by a Virtual Machine, and the intput and output test data are generated on the fly, using an adapter (Adt). In this approach, the generated input test data is also sent on the fly to the IUT, as the input of an artificial musician, and the real outcome of the IUT is compared online to the expected output (5), event by event. The use of *virtual clocks* permits to execute the tests (in both approaches) in a fast forward fashion, without having to wait for the real duration of the score.

Our case study presents important originalities compared to other MBT applications to realtime systems such as [16, 23]. On the one hand, the model supports several time units, including the *physical time* (wall clock), measured in seconds, and the *musical time*, measured in number of beats relatively to a tempo. This situation raises several new problems for the generation of test suites and their execution. On the other hand, the formal model on which our test procedure is based is constructed automatically from a given mixed score, instead of being written manually by an expert – see the discussion on that last point in the related work below. This enables a fully automatic test scenario fitting well in a music authoring workflow where scores in preparation are constantly evolving.

Our main contributions are the design of an appropriate IR, the implementation of a compiler of Antescofo mixed score into IR, and the design of the two above offline and online test procedures based on IR models. Our MBT framework permits the test of the timing behaviors in addition to the output correctness of a system. Applied to Antescofo, it allowed to detect tempo computation errors and synchronization mistakes occurring during non-trivial performances. It is also used to apply regression tests in order to ensure the stability of the system during the development of new versions.

The paper is organized as follows: Section 2 introduces the system under test, Antescofo, and the principles of our test method. The models and their construction from mixed scores are formally defined in Section 3. The implemented model-based testing framework is then described in Section 4. Interesting results, following several options are presented Section 5, and finally perspectives in Section 6.

Related works

This paper extends an earlier version published in the proceedings of ACM-SAC 2015, track SVT. The main additions compared to the former version are

complete semantics of the IR, more details on the compilation of mixed scores into IR and translation of IR into TA, and the presentation of a new online MBT framework, in addition to the offline framework (based on Uppaal) which was presented at ACM-SAC.

Some tools exist for automating the test of IMS, like for instance the MAX-test package [28] for testing MAX patches through assertions. These systems conveniently provide sophisticated tools for automating execution of test data and reporting. But they generally do not offer procedures for generating test data, hence the user must compute some input test data and the expected corresponding output by other means. Our approach ([29, 30]) in contrast focus on the generation of test data, based on formal models, and in this respect the two approaches can be seen as complementary.

Other works have addressed the formal verification of multimedia systems based on TA models, like for instance the verification of a lip-synchronisation protocol (synchronization of audio and video streams) in [11]. Model checking procedures have also been used for music improvisation [6]. TA, Uppaal, as well as timed Petri nets, are used in i-Score [5], a framework for composition, verification and real-time performance of Multimedia Interactive Scenarios. To our knowledge, no other work has applied such formal models to the test of IMS.

One drawback of many MBT methods is that they generally require a manual expert intervention for the construction of models. Some specification-based testing procedures also involve the automatic construction of models from high-level user specification of test case scenarios. For instance, in [12] the specifications are written in the quasi-natural language Gherkin and the models are used with the model-based testing tool QuickCheck. In our case, the mixed scores can be considered as a complete specification of all the possible timed scenarios rather than some test case scenarios. Moreover, these specifications are not written for test purpose but prior to the execution of the system during an interactive performance. Hence, they are defined by users before the time of testing and no more intervention is needed during the test workflow. Therefore, our test procedure involves temporal values that is not common in MBT with an automated construction of timed models.

GUITAR [27] proposes to use static analysis and semi-automatically reverse-engineer methods for constructing an event-flow model from an implementation of the GUI functions of JAVA programs. Notice that we are producing our IR models by analysis of mixed scores too, however we are not testing the written mixed score but the interpreter (i.e. the system) executing this score. More precisely, an IR model is produced by parsing a mixed score and traversing its abstract syntax tree, using sequential and concurrent composition operators for the IR, similar to the glue operators in [8, 9]. This approach is modular in the sense that the model of several scores can be combined into a larger model using these operators.

Moreover, our IR models are also executable, similar to the E-code of [22, 20], which is obtained by compilation of programs in the time-triggered programming Giotto language and is used for static analysis of properties such as time-safety

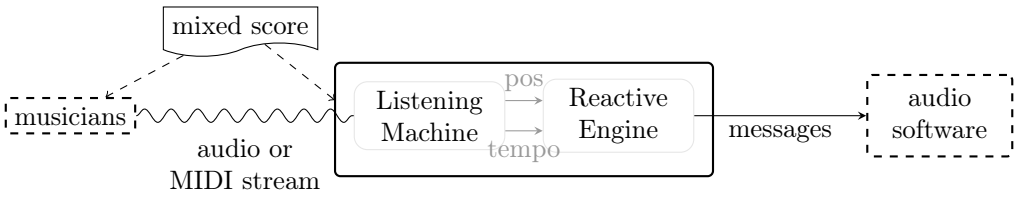

Figure 2: Architecture of Antescofo

or schedulability. An earlier version of the IR presented in this paper has also been used for analysis of the robustness of Antescofo mixed scores [19].

Our approach for the offline generation of test input by fuzzing ideal performances is inspired by fuzz testing. In [10] the fuzzing method is used in a white box fashion in a large scale testing framework. Although we follow a black-box testing approach, we use the same strategy which consist in starting from a perfect input and mutating it. Note that, in contrast to most fuzz testing approaches, we needed to deal with time values in this context, and applied for this purpose models of music performance from the literature.

2 Principle of IMS Testing

This section introduces the main notions needed across the paper. We first present the IMS Antescofo and a fragment of its companion Domain Specific Language (DSL), through a running example. We recall next the principles of Model-Based Testing (MBT), and their adaptation to our realtime IMS case study.

2.1 The score-based IMS Antescofo

Antescofo, which stands for *Anticipatory Score Following*, is a software created by Arshia Cont in 2007 for automatic accompaniment of human musician by electronic instruments during live concerts. The software is constantly evolving due to the feedbacks and requirements from a wide use in concert (by famous orchestra and composers). The reader is invited to consult the Antescofo's web page[1] for videos and examples of applications.

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, and (*ii*) the musician's instantaneous pace (*tempo*, in beats per minute) [13]. These values are sent to a *reactive engine* (RE) which schedules the electronic actions to be played, as specified in a *mixed score*. For Antescofo the actions are messages emitted *on time* to an audio environment. It

[1] Antescofo's videos: `http://repmus.ircam.fr/antescofo`

is important to note that the information exchanged between the LM and RE as well as between the RE and the output environment of the system is composed of discrete events.

2.2 DSL for Mixed Scores

A mixed score is required and parsed before Antescofo can start playing with musician(s). The mixed scores used by Antescofo are written in a textual reactive synchronous language enabling the description of the electronic accompaniment in reaction to the detected instrumental events.

Example 1 Figure 3 displays our running example in common western music notation. The part of the musician contains three notes: a quarter note e_1 of pitch $D^\sharp 5$ (its duration is one beat), and two eighths notes e_2 and e_3 of respective pitches A4 and $C^\sharp 4$ (both of duration half of a beat). The intention in this example is to launch three pairs of actions (called t_1, t_2 and t_3) controlling external systems via two on/off messages. The first electronic part launches on^{t_1} immediately at the detection of the first event e_1, and on^{t_3} is launched 1.5 beats later (it is time-triggered). The messages off^{t_1} and off^{t_3} are launched respectively 0.5 beat after on^{t_1} and 0.25 beat after on^{t_3}. The second electronic part launches independently on^{t_2} at the detection of the second event e_2, and launches off^{t_2} 0.25 beats after the detection of the last event e_3. ♢

Abstract syntax We use here a simplified *abstract syntax* corresponding to a fragment of Antescofo's DSL in order to illustrate our test framework (see [17] for a more complete description). This representation of a mixed score, defined by the formal grammar in Figure 4, corresponds to the Abstract Syntax Tree (AST) used in our test tools.

Let *OM* be a set of *output messages* (also called *action symbols* and denoted a) which can be emitted by the system and let *IS* be a set of input *event*

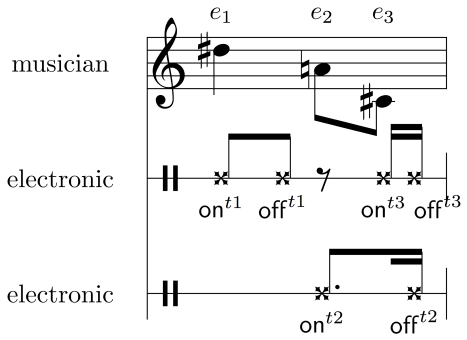

Figure 3: Running example in common western music notation

$$
\begin{array}{lcll}
\mathsf{score} & ::= & \varepsilon \mid \mathsf{event\ score} & \\
\mathsf{event} & ::= & \mathsf{evt}(e, d, \mathsf{group}) & e \in IS \\
\mathsf{group} & ::= & \varepsilon \mid \mathsf{action\ group} & \\
\mathsf{action} & ::= & \mathsf{act}(d, \mathsf{group}, \mathsf{al}) \mid \mathsf{act}(d, a, \mathsf{al}) & a \in OM \\
\mathsf{al} & ::= & \mathsf{sync?\ err?} & \\
\mathsf{sync} & ::= & \mathsf{loose} \mid \mathsf{tight} & \\
\mathsf{err} & ::= & \mathsf{local} \mid \mathsf{global} &
\end{array}
$$

Figure 4: The grammar of the handled AST of the mixed score

symbols (denoted e) to be detected by the LM (*i.e.* positions in the score). An *action* is a term $\mathsf{act}(d, s, al)$ where d is the delay to wait before starting the action (see the next section regarding time units), s is either an atom in OM 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 $e \in IS$, d is the duration of e and s is the top-level group *triggered* by e. Sequences are denoted with square brackets. An important feature of the semantics Antescofo's DSL is that a priority is defined over actions, following their order in the mixed score: if two actions can be started at the same time, Antescofo gives the priority to the first action in the mixed score.

Example 2 Figure 5 presents the abstract syntax for our running example. The curious readers can find the mixed score in Antescofo's original concrete syntax in A. The three events are listed first in the top-left side of Figure 5, after the specification of the indicated tempo (in bpm). Each event e_i is associated with a top-level group of triggered actions, denoted by s_{e_i}. The two groups s_{e_1} and s_{e_2} are only in charge of launching the two subgroups, respectively s_1 and s_2, immediately at the detection of their related event. The group s_1, specifying the first electronic part, sends the messages for the pair t_1 and a second subgroup s_3, which in turns sends the messages for the pair t_3. The simultaneity of the detection of e_1 and the start of group s_1 is specified by a null delay. The detection of events is performed in concurrence with the execution of actions

$$
\begin{array}{ll}
\text{bpm } 120 & s_1 = \mathsf{act}(0, [\mathsf{on}^{t_1}], [\,]); \\
\mathsf{evt}(e_1, 1, s_{e1}); & \quad\ \mathsf{act}(1/2, [\mathsf{off}^{t_1}], [\,]); \\
\mathsf{evt}(e_2, 1/2, s_{e2}); & \quad\ \mathsf{act}(1, [s_3], [\,]) \\
\mathsf{evt}(e_3, 1/2, []) \text{ where} & s_2 = \mathsf{act}(0, [\mathsf{on}^{t_2}], [\,]); \\
s_{e1} = \mathsf{act}(0, [s_1], [\mathsf{loose}; \mathsf{local}]) & \quad\ \mathsf{act}(3/4, [\mathsf{off}^{t_2}], [\,]) \\
s_{e2} = \mathsf{act}(0, [s_2], [\mathsf{tight}; \mathsf{global}]) & s_3 = \mathsf{act}(0, [\mathsf{on}^{t_3}], [\,]); \\
 & \quad\ \mathsf{act}(1/4, [\mathsf{off}^{t_3}], [\,])
\end{array}
$$

Figure 5: Running example: The mixed score in abstract syntax.

groups. More precisely, after the detection of e_1, the group s_1 is started and, concurrently, the system waits for e_2 meanwhile. At the detection of e_2, the group s_2 is started concurrently to s_1 (if the latter group has not terminated its execution). ◊

Group Attributes The high-level attributes in the list al attached to an action $\mathsf{act}(d, s, al)$ are indications regarding musical expressiveness [14]. We consider here four attributes for illustration purpose (their interpretation will be defined formally in Section 3.3): two attributes are used to express the synchronization of the actions to the events in 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 in our case is an event of the score missing during the performance, either because the musician did not play it or because it was not detected by the LM.

Example 3 Figure 6 depicts the possible executions of the groups s_1 and s_3 in our running example for the 4 combinations of error handling and synchronization strategies. Here, the first event e_1 is missing. Event-triggers are depicted by vertical dashed lines and stars and time-triggers (the delays to wait after a previous action following the musician pace) are depicted by horizontal arrows. The actions striken through are not sent.

In the running example, the attributes of s_1 are loose and local, hence the actions on^{t_1} and off^{t_1} are just skipped (not sent) when it is detected that the event e_1 is missed (*i.e.* when e_2 is detected instead of e_1). Note that with the attibute global, the actions would be sent without delay. Then, the actions on^{t_3} and off^{t_3} are also skipped (since s_3 inherits the attributes of s_1). ◊

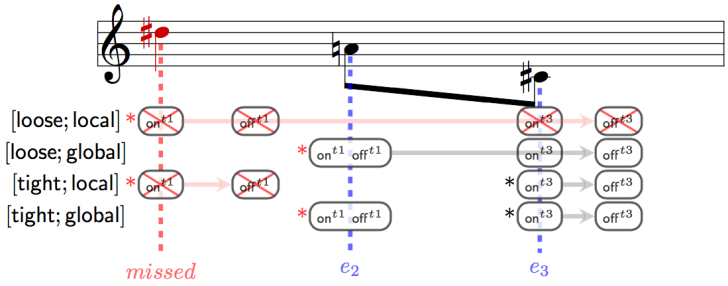

Figure 6: Interpretation of the running example when the first note is missed, for various attributes of the group s_1.

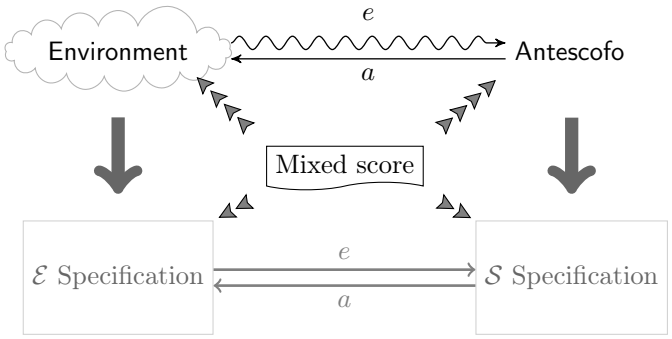

Figure 7: Model-Based Test vision for the Score-Based IMS Antescofo

2.3 Model-based Testing Embedded Systems

Model-Based Testing (MBT) is a general technique for testing an embedded system called Implementation Under Test (IUT) with respect to a specification of the good behavior of this system. It is a black box technique since the source code of the IUT is not known and only its inputs and outputs are observed.

Following the Figure 7, the IUT receives input events (e) from an environment and reacts by sending output actions (a) to this environment. In our case, Antescofo analyses the events received from the musicians and sends in reaction atomic actions (messages to audio software). A first step in MBT consists in the creation of a model $\mathcal{M}$ composed of a specification $\mathcal{S}$ of the IUT and a model of the environment $\mathcal{E}$. Specifying the environment of the system is convenient in order to delimit the test sessions. For instance, $\mathcal{E}$ can be used to describe a specific protocol or to prevent from impossible sequences of input events.

The conformance of the IUT to the specification $\mathcal{S}$ *wrt* $\mathcal{E}$ means informally that every exchange of inputs and outputs that can be observed between the IUT and its environment can be also simulated between their counterparts $\mathcal{S}$ and $\mathcal{E}$.

A big difference of our approach with the standard applications of MBT to critical systems, is the fully automatic construction of the model $\mathcal{M}$. Indeed, in general $\mathcal{M}$ has to be written manually by a formal method's expert, which is both a tedious and error prone process. In Section 3, we describe a procedure for constructing automatically the model $\mathcal{M}$ from a given mixed score. In this settings, the sub-model $\mathcal{E}$ represents the possible behaviors of the environment (i.e. the events of the musician playing the mixed score, as detected by the LM) and the specification $\mathcal{S}$ the behavior of the system according that same mixed-score.

What makes possible the automatic construction of $\mathcal{M}$ in our case is the existence of a mixed score of the form described in the above sections. Indeed, the purpose of a music score is traditionally to specify the behavior expected from the players, hence we assume that the information it contains is sufficient

in order to built the model $\mathcal{M}$.

2.4 Time Units

For realtime embedded systems, time is a critical semantic value, and not just a measure of efficiency, as some output may be expected at a precise moment, neither later nor earlier. This is in particular the case of Interactive Music Systems, where the electronic accompaniment produced by the system must follow timely the human musicians. The mixed score given to the system contains precise timing information on the reception of inputs and the emission of outputs in answer. This timing information is used during performances both by the musicians and the system for coordination. Therefore this information must also be taken into consideration in a testing procedure.

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

Let us assume a *tempo curve* τ associating an instant tempo value, in beats per minute (bpm), to each timestamp t (in physical time). The conversion of a duration d from musical time into physical time is obtained by integration over $[0, d]$ of the inverse of τ.

In Antescofo, a new tempo value is inferred by the LM at the arrival of every event, and the tempo stays constant between two events. In this sense, we shall restrict ourselves to piecewise constant tempo curves here, and assume that they can be defined in traces of events as described in the next section.

2.5 Timed Traces and Test Cases

The black-box testing conformance approach presented in this paper is based on timed traces comparison. Let us describe in this section the format used for traces and test suites. We assume a given mixed score with a default tempo value. A *timed trace* is a sequence of triples $\langle \alpha_i, t_i, p_i \rangle$ made of:

a symbol $\alpha_i \in IS \cup OM$,

a *timestamp* $t_i \in \mathbb{R}^+$, expressed in musical time, and

a tempo value p_i in beat per minute (bpm).

Such that for all i, $t_i \leq t_{i+1}$ and if $t_i = t_{i+1}$ then $p_i = p_{i+1}$. The tempo curve τ associated to a trace t_{in} as above is defined by $\tau(t) = p_i$ iff $t_i \leq t < t_{i+1}$. A trace containing symbols exclusively in IS (resp. OM) 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 and the *ideal trace* ($t_{\mathsf{in}}^{ideal} \in \mathcal{T}_{\mathsf{in}}$) the projection of all events in the mixed score such that $t_i = t_{i-1} + d_{i-1}$ with $t_0 = 0$, d_{i-1} the last event duration and p_i the default tempo specified in the score.

Example 4 The ideal trace for our running example is the following:

$$t_{\mathsf{in}}^{ideal} = \langle e_1, 0, 120\rangle \cdot \langle e_2, 1, 120\rangle \cdot \langle e_3, 1.5, 120\rangle.$$

$\Diamond$

Remember that the second components t_i are timestamps and not durations as in the score. The durations, in musical time, from the score can be obtained by $d_i = t_{i+1} - t_i$. By definition of music performance, traces of real executions can be arbitrarily far from ideal traces: the tempo and durations can diverge from the written values (the musician adding her/his own expressiveness values), and moreover there can be errors during a performance (missing notes). In our case, the model of environment $\mathcal{E}$ can be seen as a subset of $\mathcal{T}_{\mathsf{in}}$ specifying all the possible performances. The model of the system $\mathcal{S}$ can be seen as a function from $\mathcal{T}_{\mathsf{in}}$ into $\mathcal{T}_{\mathsf{out}}$. This asymmetry between $\mathcal{E}$ and $\mathcal{S}$ reflects our case study (detailed Section 2.1), with on the one side the musician/LM and on the other side the RE. Some additional descriptions of models of music performance from the literature are given in Section 3.5.

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 $t_{\mathsf{out}} = \mathcal{S}(t_{\mathsf{in}})$. The input trace t_{in} is a fake performance and the output trace t_{out} is the expected behavior obtained by the simulation of t_{in} on the specification $\mathcal{S}$, denoted $\mathcal{S}(t_{\mathsf{in}})$. Some complementary approaches for the offline or "on-the-fly" generation of t_{in} are presented below in Sections 4.2 and 4.3.

2.6 Conformance

The execution of a test case $\langle t_{\mathsf{in}}, t_{\mathsf{out}}\rangle$ consists of several tasks summarized in the following definition of conformance. First, we stimule the events of t_{in} to the IUT, *i.e.* the IMS Antescofo, respecting the timestamps. Second, we monitor the outcome of the IUT in an output trace t'_{out}, this monitored execution is denoted as $t'_{\mathsf{out}} = \mathrm{IUT}(t_{\mathsf{in}})$. 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:

$$\forall t_{\mathsf{in}} \in \mathcal{E},\ \mathrm{IUT}(t_{\mathsf{in}}) = \mathcal{S}(t_{\mathsf{in}}).$$

This is a particular case of the relation rtioco considered in [16, 23].

However, this definition only makes sense if the timestamps of t_{out} and t'_{out} are in the same time unit during the comparison. We will show how this important issue is addressed in practice in Section 4.2, with different options for the conversion of all traces into physical time (thanks to the addition of tempo values).

3 Abstract Model

In this section we present the formalism we designed for writing the models $\mathcal{M}$ ($\mathcal{E}$ and $\mathcal{S}$) used in our MBT framework. It is an intermediate representation (IR) in the form of finite state machines with message passing, dynamic thread

creation (alternations) and durations (Section 3.1). We describe in Sections 3.2 and 3.3 a procedure for compiling a mixed score into an IR, and a translation of IR into Timed Automata in Section 3.4, in order to use (in Section 4) tools of the Uppaal suite for MBT. Related models of musical performance are discussed in Section 3.5.

3.1 Intermediate Representation

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

Syntax An IR is a Finite State Machine (FSM) of the form $\mathcal{A} = \langle \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, L, \ell_0, \Delta \rangle$ where Σ_{in} (resp. Σ_{out}) is the input (resp. output) local alphabet, L is a finite set of locations, $\ell_0 \in L$ is the initial location, Δ is a finite set of transitions, partitioned into $\Delta = \Delta_0 \uplus \Delta_1$, where Δ_0 is the subset of *urgent* transitions, that must be fired without delay, and Δ_1 is the subset of *suspending* transitions, whose execution may require some time to flow.

The alphabets Σ_{in} and Σ_{out} are not assumed to be disjoint. We assume moreover a total ordering $\prec$ over $\Sigma_{\mathsf{in}} \cup \Sigma_{\mathsf{out}}$. This will be used for defining a priority for the emission and reception of symbols in the FSM. We also assume a partition of output symbols into: $\Sigma_{\mathsf{out}} = \Sigma^{\mathsf{sig}}_{\mathsf{out}} \uplus \Sigma^{\mathsf{ext}}_{\mathsf{out}}$. The symbols of $\Sigma^{\mathsf{sig}}_{\mathsf{out}}$ represent internal signals, emitted and captured by the FSM (*i.e.* they can be also in Σ_{in}) whereas the symbols of $\Sigma^{\mathsf{ext}}_{\mathsf{out}}$ are destinated to the external environment: they are emitted but not captured by the FSM (*i.e.* they are symbols of the output test traces t_{out}). Moreover, $\Sigma^{\mathsf{ext}}_{\mathsf{out}} \cap \Sigma_{\mathsf{in}} = \emptyset$. The symbols of $\Sigma^{\mathsf{sig}}_{\mathsf{out}}$ and $\Sigma^{\mathsf{ext}}_{\mathsf{out}}$ will be emitted with different priorities by the FSM, to reflect the semantics of Antescofo's DSL.

There are two kinds of *urgent transitions* in Δ_0 called *emit*-, and *and*-transitions:
1. an *emit*-transition, in $L \times \Sigma_{\mathsf{out}} \times L$, is denoted by $\ell \xrightarrow{\sigma!} \ell'$ with $\ell, \ell' \in L$ (respectively called *source* and *target* of the transition), and $\sigma \in \Sigma_{\mathsf{out}}$. It provokes the sending of an output symbol, followed by the change of current control point from location ℓ to ℓ'.
We say that a location ℓ *emits* a symbol $\sigma \in \Sigma_{\mathsf{out}}$ if there exists a transition of the form $\ell \xrightarrow{\sigma!} \ell'$ for some location ℓ'.

2. an *and*-transition, or *alternation*, in $L \times L^2$, is denoted by $\ell \xrightarrow{\mathsf{and}} \ell_1 \| \ell_2$, with $\ell \in L$ (called *source* of the transition), and $\ell_1, \ell_2 \in L$ (called *targets* of the transition). It creates dynamically a new control point. Intuitively, the current control point, initially in the source location ℓ, is transferred to the first target location ℓ_1 while a new concurrent control point is created at the second target location ℓ_2. The source of this transition cannot have a non-singleton branch.

There are two kinds of *suspending transitions* in Δ_1, called *recv*- and *wait*-transitions:

1. a *recv*-transition, in $L \times \Sigma_{\mathsf{in}} \times L$ is denoted by $\ell \xrightarrow{\tau?} \ell'$, with ℓ, $\ell' \in L$ (respectively called *source* and *target* of the transition), and $\tau \in \Sigma_{\mathsf{in}}$. It waits for the reception of an input symbol, and then changes the current control point from location ℓ to ℓ'.
2. a *wait*-transition, in $L \times \mathbb{R}_*^+ \times \mathbb{R}_*^+ \times L$, is denoted by $\ell \xrightarrow{[d,d']} \ell'$, where ℓ, $\ell' \in L$ (respectively called *source* and *target* of the transition), and $d, d' \in \mathbb{R}_*^+$ are durations expressed in the same time unit (musical or physical) with $0 < d \leq d'$. It waits for the expiration of a delay before changing the current control point from location ℓ to ℓ'. Such a transition can only be fired when the control point has spent at least d time units in ℓ. Moreover, it is required that when d' time units have been spent in ℓ, then this transition, or another transition outgoing from ℓ must be fired (it is the analogous of invariant in Time Automata).

We call *branch* of a location $\ell \in L$ the set of all transitions with source location ℓ and *exit* locations the locations without outgoing transitions. A branch is the dual of an *and*-transition, representing the passing of the control point from the source location to one and only one of the target locations of the branch (a branch could be called *or*-transition to this respect).

A FSM is called *deterministic* iff it contains no branch with more than one emit transition, and for every wait transition labeled $[d, d']$, it holds that $d = d'$ (in this case, we simply write d instead of $[d, d]$), and moreover it must not contain two waits in a branch labeled with the same delay (*i.e.* there must be no transitions $\ell \xrightarrow{d} \ell'$ and $\ell \xrightarrow{d} \ell''$ with $\ell' \neq \ell''$).

Semantics Let us now define formally the runs of the above FSMs. We consider a model of superdense time [26, 31] with superdense timestamps of the form $\langle t, n \rangle \in \mathbb{R}^+ \times \mathbb{N}$, where $t \in \mathbb{R}^+$ is a timestamp in physical time called *logical instant*, and $n \in \mathbb{N}$ is a step number inside this logical instant. Intuitively, several transitions may be executed during the same logical instant, at different step numbers. The logical time will flow only when the control is in sources of suspending transitions of Δ_1.

A *state* of a FSM $\mathcal{A} = \langle \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, L, \ell_0, \Delta \rangle$ is a tuple of the form $\langle t, n, \Gamma, cp, \Theta \rangle$ where

- $\langle t, n \rangle$ is a superdense timestamp: $t \in \mathbb{R}^+$ and $n \in \mathbb{N}$,
- Γ is a vector of *running locations* of the form $\langle \ell, \gamma, \beta \rangle$ where ℓ is a location, $\gamma \in \mathbb{R}^+$ is the time spent in ℓ, and $\beta \in \{\mathsf{T}, \mathsf{F}\}$ is a flag. When $\beta = \mathsf{T}$, then the running locations is called *suspended*,
- cp is a natural number in $[1...|\Gamma|]$, pointing the current running location in Γ,
- Θ is a set of symbols of Σ_{out}.

The initial state of $\mathcal{A}$ is $s_0 = \langle 0, 0, \langle \langle \ell_0, 0, \mathsf{F} \rangle \rangle, 1, \emptyset \rangle$. In the following definition of *moves* between states of $\mathcal{A}$, we denote by :: the operator of concatenation of an element to the vector Γ (at the beginning or the end). By abuse of notation, we also denote by :: the operator of addition of an element to the set Θ. Moreover,

to simplify notations, we underline the current running location in Γ (i.e. the cp^{th} element of Γ), and consequently the component cp will be omitted in states below.

$$\langle t, n, \Gamma :: \underline{\langle \ell, \gamma, \mathsf{F} \rangle} :: \Gamma', \Theta \rangle \xrightarrow[\mathsf{and}]{} \langle t, n+1, \Gamma :: \underline{\langle \ell_1, 0, \mathsf{F} \rangle} :: \Gamma' :: \langle \ell_2, 0, \mathsf{F} \rangle, \Theta \rangle \quad (\mathsf{and})$$

if there exists $\ell \xrightarrow{\mathsf{and}} \ell_1 \| \ell_2 \in \Delta_0$ (creation of a new thread, with starting location ℓ_2).

$$\begin{array}{ll} \langle t, n, \Gamma :: \underline{\langle \ell, \gamma, \mathsf{F} \rangle} :: \langle \ell', \gamma', \beta' \rangle :: \Gamma', \Theta \rangle & \xrightarrow[\mathsf{exit}]{} \quad \langle t, n+1, \Gamma :: \underline{\langle \ell', \gamma', \beta' \rangle} :: \Gamma', \Theta \rangle \\ \langle t, n, \langle \ell', \gamma', \beta' \rangle :: \Gamma :: \underline{\langle \ell, \gamma, \mathsf{F} \rangle}, \Theta \rangle & \xrightarrow[\mathsf{exit}]{} \quad \langle t, n+1, \underline{\langle \ell', \gamma', \beta' \rangle} :: \Gamma, \Theta \rangle \end{array} \quad (\mathsf{exit})$$

if ℓ has no outgoing transition (ℓ is popped from the vector of running locations).

$$\left\langle t, n, \Gamma :: \underline{\langle \ell, \gamma, \mathsf{F} \rangle} :: \Gamma', \Theta \right\rangle \xrightarrow[\mathsf{emit}]{} \langle t, n+1, \Gamma :: \underline{\langle \ell', 0, \mathsf{F} \rangle} :: \Gamma', \sigma :: \Theta \rangle \quad (\mathsf{emit})$$

if there exists $\ell \xrightarrow{\sigma!} \ell' \in \Delta_0$ with $\sigma \in \Sigma^{\mathsf{sig}}_{\mathsf{out}}$ (emission of an internal signal).

$$\left\langle t, n, \Gamma :: \underline{\langle \ell, \gamma, \mathsf{T} \rangle} :: \Gamma', \Theta \right\rangle \xrightarrow[\mathsf{send}]{} \langle t, n+1, \Gamma :: \underline{\langle \ell', 0, \mathsf{F} \rangle} :: \Gamma', \sigma :: \Theta \rangle \quad (\mathsf{send})$$

if all elements of Γ are suspended and there exists $\ell \xrightarrow{\sigma!} \ell' \in \Delta_0$, with $\sigma \in \Sigma^{\mathsf{ext}}_{\mathsf{out}}$ and σ is the smallest symbol of $\Sigma^{\mathsf{ext}}_{\mathsf{out}}$ emitted by a location of Γ (emission of an external symbol in a predefined order).

$$\langle t, n, \Gamma :: \underline{\langle \ell, \gamma, \mathsf{F} \rangle} :: \Gamma', \Theta \rangle \xrightarrow[\mathsf{expir}]{} \langle t, n+1, \Gamma :: \underline{\langle \ell', 0, \mathsf{F} \rangle} :: \Gamma', \Theta \rangle \quad (\mathsf{expir})$$

if (emit) is not applicable and there exists $\ell \xrightarrow{[d,d']} \ell' \in \Delta_1$ such that $d \leq \gamma \leq d'$ (expiration of delay for wait transition).

$$\langle t, n, \Gamma :: \underline{\langle \ell, \gamma, \beta \rangle} :: \Gamma', \Theta \rangle \xrightarrow[\mathsf{recv}]{} \langle t, n+1, \Gamma :: \underline{\langle \ell', 0, \mathsf{F} \rangle} :: \Gamma', \Theta \rangle \quad (\mathsf{recv})$$

if none of (emit) or (expir) can be applied and there exists $\ell \xrightarrow{\tau?} \ell' \in \Delta_1$ such that τ is minimal (wrt $\prec$) in $\Theta \cap \{\tau' \mid \exists \ell \xrightarrow{\tau'?} \ell'' \in \Delta_0\}$ (reception of an expected symbol or internal signal).

$$\begin{array}{ll} \langle t, n, \Gamma :: \underline{\langle \ell, \gamma, \beta \rangle} :: \langle \ell', \gamma', \beta' \rangle :: \Gamma', \Theta \rangle & \xrightarrow[\mathsf{susp}]{} \quad \langle t, n, \Gamma :: \langle \ell, \gamma, \mathsf{T} \rangle :: \underline{\langle \ell', \gamma', \beta' \rangle} :: \Gamma', \Theta \rangle \\ \langle t, n, \langle \ell', \gamma', \beta' \rangle :: \Gamma :: \underline{\langle \ell, \gamma, \beta \rangle}, \Theta \rangle & \xrightarrow[\mathsf{susp}]{} \quad \langle t, n, \underline{\langle \ell', \gamma', \beta' \rangle} :: \Gamma :: \langle \ell, \gamma, \mathsf{T} \rangle, \Theta \rangle \end{array} \quad (\mathsf{suspend})$$

if none of (and), (exit), (emit), (send), (expir), (recv) can be applied and there exists at least one runing location in Γ which is not suspended or emitting a symbol of $\Sigma^{\mathsf{ext}}_{\mathsf{out}}$.

$$\langle t, n, \Gamma, \Theta\rangle \xrightarrow[\mathsf{delay}]{} \langle t+\delta, 0, \Gamma+\delta, \emptyset\rangle \qquad (\mathsf{delay})$$

if no other move can be applied, where $\Gamma+\delta$ stands for $\{\langle \ell, \gamma+\delta, \mathsf{F}\rangle \mid \langle \ell, \gamma, \beta\rangle \in \Gamma\}$ and δ is such that

1. $\delta > 0$,
2. for all $\langle \ell, \gamma, \beta\rangle$ in Γ such that there exists $\ell \xrightarrow{[d,d']} \ell' \in \Delta_1$, it holds that $\gamma + \delta \leq d'$,
3. there exists at least one $\langle \ell, \gamma, \beta\rangle$ in Γ and one $\ell \xrightarrow{[d,d']} \ell' \in \Delta_1$ such that $d \leq \gamma + \delta$.

The principle of moves is the following. Every element of Γ represents a thread. Threads run in cooperative scheduling: every thread executes until it gets *suspended*, and then it hands over to the next thread in Γ – with a move (suspend).

The urgent transitions are applied immediately – moves (and), (exit), (emit), except for emit transitions which may be suspended when the symbol to emit is in $\Sigma^{\mathsf{ext}}_{\mathsf{out}}$.

The suspending transitions may be applied immediately when conditions allow: a wait-transition is applied if the time already spent in the source location (second component of the current element of Γ) is within the bounds defined for the guard of the transition – move (expir), and a recv-transition is applied if the expected symbol is present in Θ, because it was sent during the same logical instant – move (recv). Failure of these conditions causes the suspension of the thread, by the move (suspend) which changes the value of the flag to F. Note that the steps (emit) and (recv) may loop in the same logical instant. When all threads are suspended, the emit-transitions with symbols of $\Sigma^{\mathsf{ext}}_{\mathsf{out}}$ can be executed, following the ordering $\prec$ – move (send). This strategy corresponds to the semantics of Antescofo's DSL which requires a predefined ordering for the messages sent to the external environment.

The move (delay) lets a positive amount δ of time flow (in adequacy with the upper bounds d' in guards of active wait transitions) when all threads are suspended and no symbol of $\Sigma^{\mathsf{ext}}_{\mathsf{out}}$ can be emitted. A new logical instant is then started at the date timestamped at $t+\delta$ (where t is the former logical instant), the step counter is reset to zero, the threads of Γ are unsuspended and the list Θ of sent symbols is flushed. Note that we impose that δ unlocks at least one wait transition, in order to prevent consecutive applications of (delay). Moreover for branches, the execution priority of transitions is, in decreasing order: emit then expir then recv. These priorities follow the actual semantics of Antescofo [17].

Notice that the only non-deterministic choices are: the choice of the duration δ in (delay) when $d < d'$, and the choice of a local input symbol to emit in a set of $\Sigma^{\mathsf{sig}}_{\mathsf{out}}$, in (emit), when there are several emit-transitions in the same branch. All the other moves are deterministic.

Runs A *run* ρ of an IR $\mathcal{A}$ is a sequence of the form $s_0 \xrightarrow[m_1]{} s_1 \ldots s_{k-1} \xrightarrow[m_k]{} s_k$ where $s_0, \ldots, s_k$ are states, s_0 is the initial state, and for all $0 \leq i < k$, s_{i+1} is obtained from s_i by the move m_{i+1}. We associate to the run ρ the output trace t_{out} defined as follows. For all $0 \leq i \leq k$, let t_i and Θ_i be respectively the first and the last components of s_i (*i.e.* the timestamp of the logical instant of s_i and the set of accumulated symbols sent during that instant). Let $1 \leq i_1 < \ldots < i_p \leq k$ be the subsequence of all steps in ρ such that for all $1 \leq j \leq p$, $m_{i_j} = (\mathsf{send})$ and $\Theta_{i_j} \setminus \Theta_{i_j - 1} = \{a_j\} \subset \Sigma^{\mathsf{ext}}_{\mathsf{out}}$ (at the move i_j, one output message $a_j \in \Sigma^{\mathsf{ext}}_{\mathsf{out}}$ was emitted). The trace t_{out} contains the sequence of triples of the form $\langle a_j, t_{i_j}, 60\rangle$ for all $1 \leq j \leq p$. Given an IR $\mathcal{A}$, we denote by $\mathcal{L}(\mathcal{A})$ the set of traces t_{out} such that there exists a run ρ of $\mathcal{A}$ and t_{out} is associated to ρ.

The timed transitions in this IR model are close to Timed Automata transitions [3] (see Section 3.4 below), whereas the accumulation of symbols during a logical instant is inspired by the programming language Esterel for reactive systems [7].

3.2 Operators for the Composition of IR

From a mixed score given in the abstract syntax format, we construct an IR as shown below. This construction is recursive, following the nested items defined by the grammar in Section 2.2. In order to specify it, we define first two typed operators for the composition of the above FSMs: a parallel composition $\|$ and a sequential concatenation $+$. For this purpose, we distinguish in every FSM two sequences of locations, respectively called *providers* and *seekers*. The *type* of a FSM $\mathcal{A}$ is the pair $\langle n, m\rangle$ where n is its number of providers and m is its number of seekers. We shall sometimes write the type of a FSM in exponent, as in $\mathcal{A}^{\langle n,m\rangle}$, in order to make it explicit when needed. A FSM of type $\langle 0, 0\rangle$ is called *complete*. In the graphical representation of FSMs, the provider and seeker locations are marked respectively with $i\circ$ and $\prec i$, where i is an index in the sequence of providers, resp. seekers. Moreover, remark that the sources of urgent transitions are filled in gray and the sources of suspending ones are in white.

The binary operator $\| : \langle n, m\rangle \to \langle n', l\rangle \to \langle p, s\rangle$ with $p = max(n, n')$ and $s = m + l$ is a parallel composition of FSMs defined as follows. Let $\mathcal{A} = \langle \Sigma^{\mathcal{A}}_{\mathsf{in}}, \Sigma^{\mathcal{A}}_{\mathsf{out}}, L^{\mathcal{A}}, \ell^{\mathcal{A}}_0, \Delta^{\mathcal{A}}\rangle$ and $\mathcal{B} = \langle \Sigma^{\mathcal{B}}_{\mathsf{in}}, \Sigma^{\mathcal{B}}_{\mathsf{out}}, L^{\mathcal{B}}, \ell^{\mathcal{B}}_0, \Delta^{\mathcal{B}}\rangle$, be two FSMs of respective types $\langle n, m\rangle$ and $\langle n', l\rangle$ and with respective sequences of providers and seekers: $p^{\mathcal{A}}_1, \ldots, p^{\mathcal{A}}_n$, and $s^{\mathcal{A}}_1, \ldots, s^{\mathcal{A}}_m \in L^{\mathcal{A}}$, $p^{\mathcal{B}}_1, \ldots, p^{\mathcal{B}}_{n'}$, and $s^{\mathcal{B}}_1, \ldots, s^{\mathcal{B}}_l \in L^{\mathcal{B}}$ (we assume $L^{\mathcal{A}}$ and $L^{\mathcal{B}}$ disjoint). Their parallel composition is defined by

$$\langle \Sigma^{\mathcal{A}}_{\mathsf{in}} \cup \Sigma^{\mathcal{B}}_{\mathsf{in}}, \Sigma^{\mathcal{A}}_{\mathsf{out}} \cup \Sigma^{\mathcal{B}}_{\mathsf{out}}, L^{\mathcal{A}} \uplus L^{\mathcal{B}} \uplus \{\ell_0, \ldots, \ell_p\}, \ell_0, \Delta^{\mathcal{A}} \uplus \Delta^{\mathcal{B}} \uplus \Delta\rangle$$

where $\ell_0, \ldots, \ell_p$ are new locations, the sequences of providers and seekers of $\mathcal{A}\|\mathcal{B}$ are respectively $\ell_0, \ldots, \ell_p$, and $s^{\mathcal{A}}_1, \ldots, s^{\mathcal{A}}_m, s^{\mathcal{B}}_1, \ldots, s^{\mathcal{B}}_l$, and Δ contains the set of transitions of the form $\ell_i \xrightarrow{and} \ell^{\mathcal{A}}_{i_1} \| \ell^{\mathcal{B}}_{i_2}$, with $1 \leq i \leq p$, such that if $i \leq n$ then $i_1 = i$, and otherwise $i_1 = n$, and if $i \leq n'$ then $i_2 = i$, and otherwise $i_2 = n'$.

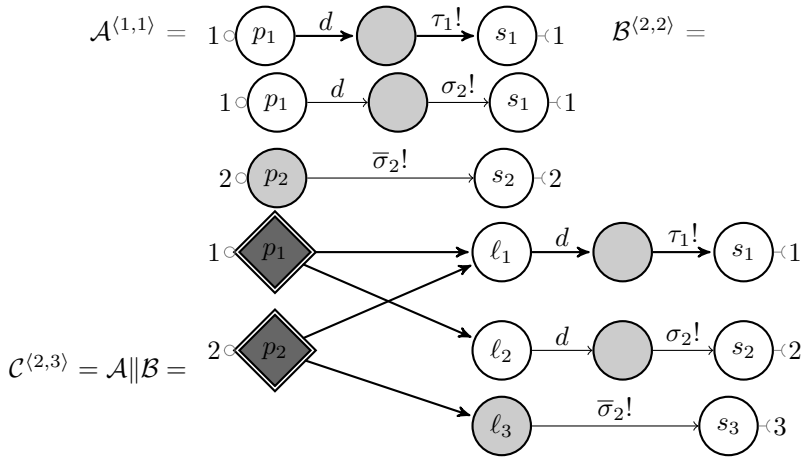

Figure 8: Example of parallel composition of two FSMs.

Note that the set of input and output symbols of $\mathcal{A}$ and $\mathcal{B}$ are not required to be disjoint, since these symbols are used for communication between the two FSMs after composition. The Figure 8 exemplifies this composition, puting in parallel two FSMs with different types.

We also define a binary operator $+ : \langle k, n\rangle \to \langle n', m\rangle \to \langle k, m\rangle$ for the sequential composition of FSMs. Let $\mathcal{A} = \langle \Sigma^{\mathcal{A}}_{\mathsf{in}}, \Sigma^{\mathcal{A}}_{\mathsf{out}}, L^{\mathcal{A}}, \ell^{\mathcal{A}}_0, \Delta^{\mathcal{A}}\rangle$ and $\mathcal{B} = \langle \Sigma^{\mathcal{B}}_{\mathsf{in}}, \Sigma^{\mathcal{B}}_{\mathsf{out}}, L^{\mathcal{B}}, \ell^{\mathcal{B}}_0, \Delta^{\mathcal{B}}\rangle$, be two FSMs with respective types $\langle k, n\rangle$ and $\langle n', m\rangle$ and with respective sequences of providers and seekers: $p^{\mathcal{A}}_1, \ldots, p^{\mathcal{A}}_k$, and $s^{\mathcal{A}}_1, \ldots, s^{\mathcal{A}}_n \in L^{\mathcal{A}}$, $p^{\mathcal{B}}_1, \ldots, p^{\mathcal{B}}_{n'}$, and $s^{\mathcal{B}}_1, \ldots, s^{\mathcal{B}}_m \in L^{\mathcal{B}}$. Their sequential composition is defined by

$$\langle \Sigma^{\mathcal{A}}_{\mathsf{in}} \cup \Sigma^{\mathcal{B}}_{\mathsf{in}}, \Sigma^{\mathcal{A}}_{\mathsf{out}} \cup \Sigma^{\mathcal{B}}_{\mathsf{out}}, (L^{\mathcal{A}} \setminus \{s^{\mathcal{A}}_1, \ldots, s^{\mathcal{A}}_n\}) \uplus (L^{\mathcal{B}} \setminus \{p^{\mathcal{B}}_1, \ldots, p^{\mathcal{B}}_{n'}\}) \uplus \{\ell_1, \ldots, \ell_{n''}\}, \ell'_0, \Delta\rangle$$

where $\ell_1, \ldots, \ell_{n''}$ are new locations, not in $L^{\mathcal{A}} \cup L^{\mathcal{B}}$ and $n'' = min(n, n')$. The sequences of providers and seekers of $\mathcal{A} + \mathcal{B}$ are respectively $p^{\mathcal{A}}_1, \ldots, p^{\mathcal{A}}_k$ and $s^{\mathcal{B}}_1, \ldots, s^{\mathcal{B}}_m$. $\ell'_0 = \ell_i$ if there exists $i \leq n$ such that $\ell^{\mathcal{A}}_0 = s^{\mathcal{A}}_i$, otherwise $\ell'_0 = \ell^{\mathcal{A}}_0$. Finally the set of transitions Δ is obtained by replacing in $\Delta^{\mathcal{A}} \cup \Delta^{\mathcal{B}}$ every location $s^{\mathcal{A}}_i$ or $p^{\mathcal{B}}_i$ by ℓ_i for $1 \leq i \leq n''$.

Intuitively, every seeker of $\mathcal{A}$ is merged with the provider of $\mathcal{B}$ with the same index. Note that if $n > n'$, then the seekers $s^{\mathcal{A}}_{n'+1}, \ldots, s^{\mathcal{A}}_n$ of $\mathcal{A}$ without matching providers in $\mathcal{B}$ become *exit* locations in $\mathcal{A} + \mathcal{B}$. If $n < n'$, then the providers $p^{\mathcal{B}}_{n+1}, \ldots, p^{\mathcal{B}}_{n'}$ of $\mathcal{B}$ without matching seekers in $\mathcal{A}$ are deleted and become standard locations in $\mathcal{A} + \mathcal{B}$ (not providers nor seekers).

An example of sequential composition is depicted in Figure 9.

3.3 Compiling mixed scores into IR

We recall that we assume two sets *OM* and *IS* respectively of output messages and input symbols (see Section 2.2). We moreover assume a set $\mathcal{S}igs$ of internal

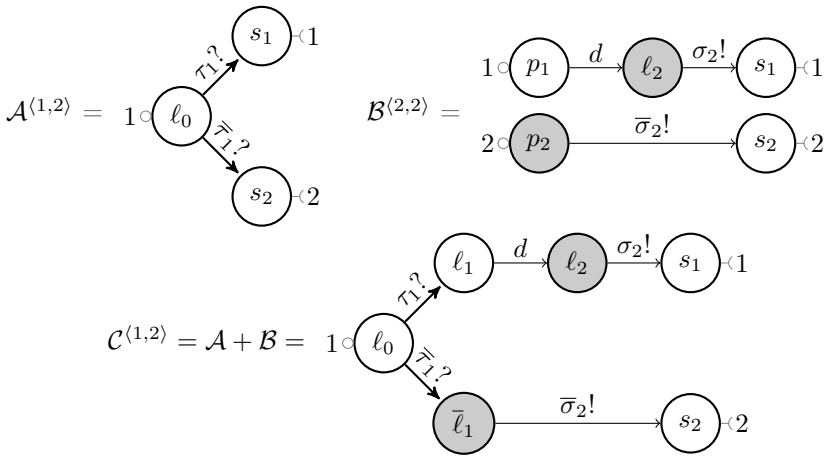

Figure 9: Example of sequential composition of two FSMs.

signals, containing in particular one internal signal $\overline{e} \in \mathcal{S}igs$ for each input event $e \in IS$. The latter represents the fact that event e was detected as missing (that defines an error following Section 2.2).

Inference Rules The construction is defined using rules of the following form:

$$\langle aux \rangle : \langle \text{AST} \rangle \vdash_{rule} \langle \text{FSM} \rangle$$

where $\langle aux \rangle$ is a sequence of auxiliary arguments, $\langle \text{AST} \rangle$ is the element of the mixed score currently parsed (in abstract syntax) and $\langle \text{FSM} \rangle$ is the corresponding part of FSM constructed and returned.

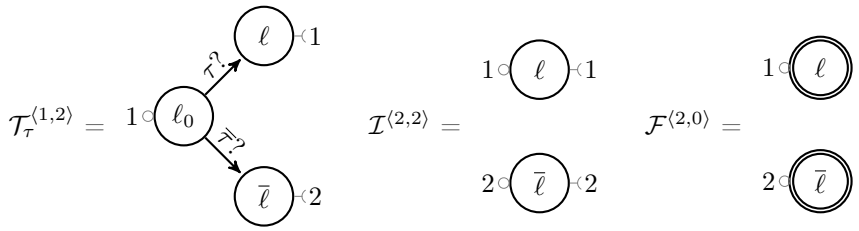

Figure 10: An example of the FSM $\mathcal{T}$, $\mathcal{I}$ and $\mathcal{F}$.

Starting and Ending FSM We define three kinds of FSM commonly used during the construction:

triggers $\mathcal{T}_\tau^{\langle 1,2\rangle}$ with $\tau \in IS \cup \mathcal{S}igs$, for starting a FSM at the detection of some input symbol τ. Every trigger FSM has one provider and two seekers, corresponding to a start of the FSM in a normal or an error mode.

idlers $\mathcal{I}^{\langle i,i\rangle}$ where each location is both a provider and a seeker,

enders $\mathcal{F}^{\langle i,0\rangle}$ with an empty list of seekers.

Some generic examples of these parts of FSM are depicted Figure 10.

Score processing The rule $\vdash_{\mathsf{all}}$ constructs the FSM associated to the parsed mixed score.

$$\frac{}{: \emptyset \vdash_{\mathsf{all}} \mathcal{A}_\emptyset} \qquad \frac{: ms \vdash_{\mathsf{env}} \mathcal{E} \qquad : ms \vdash_{\mathsf{proxy}} \mathcal{P} \qquad : ms \vdash_{\mathsf{sys}} \mathcal{A}}{: ms \vdash_{\mathsf{all}} \mathcal{E} \| \mathcal{P} \| \mathcal{A}}$$

If the input score is empty the rule $\vdash_{\mathsf{all}}$ returns an empty FSM $\mathcal{A}_\emptyset$ (FSM with an empty set of locations). Otherwise three rules are applied to the score ms:

$\vdash_{\mathsf{env}}$ to construct the environment model $\mathcal{E}$,

$\vdash_{\mathsf{sys}}$ in charge of creating $\mathcal{A}$ the model of the system's behavior when playing the mixed score ms given in argument.

$\vdash_{\mathsf{proxy}}$ which constructs the proxy FSM $\mathcal{P}$ (interface between $\mathcal{E}$ and $\mathcal{A}$).

Hence we define $\mathcal{M} = \mathcal{E} \| \mathcal{S}$ with $\mathcal{S} = \mathcal{P} \| \mathcal{A}$: the model $\mathcal{M}$ is the result of the parallelization of the environment and the specification $\mathcal{S}$, which is itself the composition of the proxy $\mathcal{P}$ and the FSM $\mathcal{A}$. All the FSMs $\mathcal{E}$, $\mathcal{P}$ and $\mathcal{A}$ have the same type $\langle 1, 0\rangle$.

Environment FSM The environment FSM $\mathcal{E}$ is built with a single pass through the score events. There are several options for constructing $\mathcal{E}$: the number n_{err} of consecutive events possibly missing, and κ, the percentage of variation tolerated on the event's durations. This second option permits the creation of bounds of the form $[d(1-\kappa), d(1+\kappa)]$ centered around the ideal duration d specified in the score. It is used for the non deterministic choices of durations in input traces by $\mathcal{E}$ (see Section 3.5).

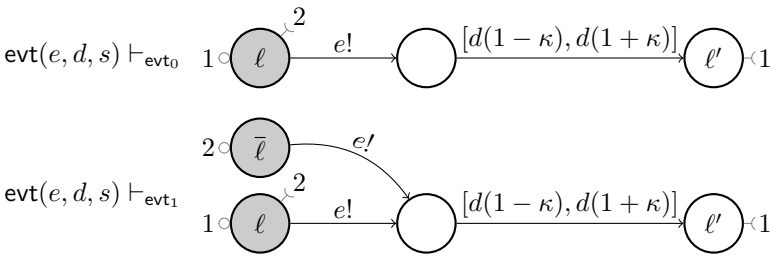

Figure 11: The parts of the environment FSM corresponding to the first and next events.

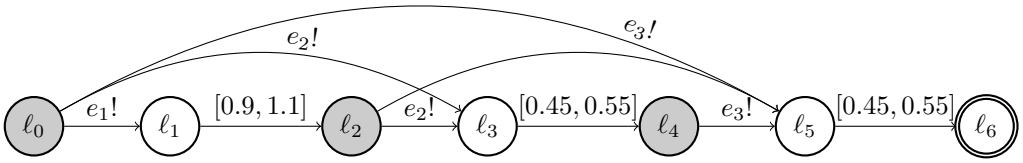

Figure 12: Running example: A FSM for the environment $\mathcal{E}$, with $n_{err} = 2$ and $\kappa = 0.10$.

Formally, $\mathcal{E}$ is a non-deterministic FSM of the form $\langle OM, IS, L^{\mathcal{E}}, \ell_0^{\mathcal{E}}, \Delta^{\mathcal{E}} \rangle$. We present the construction of $\mathcal{E}$ for the value of $n_{err} = 1$. Moreover, notice that the partition of the output alphabet $\Sigma_{\mathsf{out}} = IS$ is $\Sigma_{\mathsf{out}}^{\mathsf{sig}} = IS$ and $\Sigma_{\mathsf{out}}^{\mathsf{ext}} = \emptyset$.

$$\frac{: \mathsf{evt}(e,d,s) \vdash_{\mathsf{evt}_0} \mathcal{E}_0^{\langle 1,2 \rangle} \qquad : ms' \vdash_{\mathsf{env}_1} \mathcal{E}^{\langle 2,0 \rangle}}{: \mathsf{evt}(e,d,s)::ms' \vdash_{\mathsf{env}} \mathcal{E}_0^{\langle 1,2 \rangle} + \mathcal{E}^{\langle 2,0 \rangle}}$$

$$\frac{: \mathsf{evt}(e,d,s) \vdash_{\mathsf{evt}_1} \mathcal{E}_1^{\langle 2,2 \rangle} \qquad : ms' \vdash_{\mathsf{env}_1} \mathcal{E}^{\langle 2,0 \rangle}}{: \mathsf{evt}(e,d,s)::ms' \vdash_{\mathsf{env}_1} \mathcal{E}_1^{\langle 2,2 \rangle} + \mathcal{E}^{\langle 2,0 \rangle}} \qquad \frac{}{: \emptyset \vdash_{\mathsf{env}_1} \mathcal{F}^{\langle 2,0 \rangle}}$$

The rule $\vdash_{\mathsf{evt}_0}$ initializes the FSM $\mathcal{E}$, by playing the first event $\mathsf{evt}(e,d,s)$ and waiting for a duration in the interval $[d(1-\kappa), d(1+\kappa)]$. The rule $\vdash_{\mathsf{evt}}$ treats each following event of the score, with the possibility to play the current event e from the previous step (provider 1) or the step before (provider 2). The FSMs constructed by these two rules are depicted in Figure 11. The rules $\vdash_{\mathsf{env}}$ and $\vdash_{\mathsf{env}_1}$, assemble the FSMs for the first, respectively next, events. At the end (the case of empty event sequence), an ender FSM terminates the construction of the environment FSM.

Example 5 We present in Figure 12 the FSM $\mathcal{E}$ constructed for our running

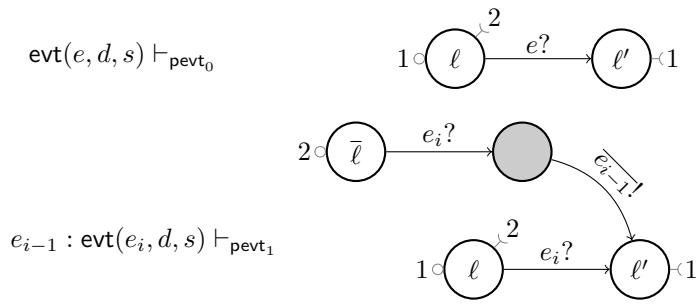

Figure 13: The parts of the proxy FSM corresponding to the first and next events.

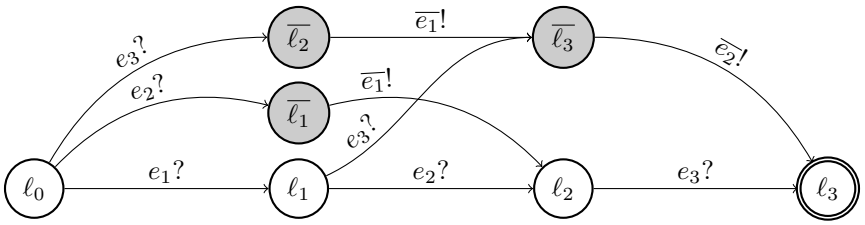

Figure 14: Running example: The proxy FSM $\mathcal{P}$ with $n_{err} = 2$.

example with $n_{err} = 2$ and $\kappa = 0.10$. ◊

Proxy FSM A FSM of the form $\mathcal{P} = \langle IS, \mathcal{S}igs, L^{\mathcal{P}}, \ell_0^{\mathcal{P}}, \Delta^{\mathcal{P}} \rangle$ called proxy is in charge of receiving detected events and signaling to other FSMs the missing events, using signals of the form $\overline{e}$. Recall (Section 2.2) that we define an error as a missing event e_i, detected at the arrival of a next event e_{i+k}, with $k > 0$. This modular approach with a proxy FSM permits to easily replace this definition of error with alternative definitions, without changing the rest of the FSM. We present the construction of $\mathcal{P}$ for $n_{err} = 1$.

$$\frac{: \mathsf{evt}(e,d,s) \vdash_{\mathsf{pevt}_0} \mathcal{P}_0^{\langle 1,2 \rangle} \qquad e : ms' \vdash_{\mathsf{proxy}_1} \mathcal{P}^{\langle 2,0 \rangle}}{: \mathsf{evt}(e,d,s)::ms' \vdash_{\mathsf{proxy}} \mathcal{P}_0^{\langle 1,2 \rangle} + \mathcal{P}^{\langle 2,0 \rangle}}$$

$$\frac{e_{i-1} : \mathsf{evt}(e_i,d,s) \vdash_{\mathsf{pevt}_1} \mathcal{P}_1^{\langle 2,2 \rangle} \qquad e_i : ms' \vdash_{\mathsf{proxy}_1} \mathcal{P}^{\langle 2,0 \rangle}}{e_{i-1} : \mathsf{evt}(e_i,d,s)::ms' \vdash_{\mathsf{proxy}_1} \mathcal{P}_1^{\langle 2,2 \rangle} + \mathcal{P}^{\langle 2,0 \rangle}} \qquad \frac{}{e : \emptyset \vdash_{\mathsf{proxy}_1} \mathcal{F}^{\langle 2,0 \rangle}}$$

The rule $\vdash_{\mathsf{pevt}_0}$ initializes the FSM $\mathcal{P}$, by waiting for the first event $\mathsf{evt}(e,d,s)$. The rule $\vdash_{\mathsf{pevt}_1}$ treats each following event e_i of the score. Provider and seeker 1 correspond to the case when this event e_i is received after the previous event e_{i-1} while provider and seeker 2 correspond to the case when e_i is received whereas e_{i-1} was not received. In the latter case, the signal $\overline{e_{i-1}}$ is emitted to notify that the last event e_{i-1} is missing. The FSMs constructed by these rules $\vdash_{\mathsf{pevt}_0}$ and $\vdash_{\mathsf{pevt}_1}$ are depicted in Figure 13.

Example 6 Figure 14 depicts the result for our running example with $n_{err} = 2$. ◊

FSM for the score reactions The rule $\vdash_{\mathsf{sys}}$ constructs a FSM of the form $\mathcal{A} = \langle IS \cup \mathcal{S}igs, OM, L^{\mathcal{A}}, \ell_0^{\mathcal{A}}, \Delta^{\mathcal{A}} \rangle$ specifying the behavior of the system in reaction to the events of the environment, *i.e.* the automatic accompaniment.

$$\frac{\begin{array}{c}0, \mathsf{evt}(e', d', \_)::ms', \mathsf{evt}(e, d, s), \mathsf{F} : s \vdash_{\mathsf{seq}}^{\mathsf{loose,global}} \mathcal{A}_s^{\langle 2,0\rangle} \\ : \mathsf{evt}(e', d', \_)::ms' \vdash_{\mathsf{sys}} \mathcal{A}^{\langle 1,0\rangle}\end{array}}{: \mathsf{evt}(e, d, s)::\mathsf{evt}(e', d', \_)::ms' \vdash_{\mathsf{sys}} (\mathcal{T}_e^{\langle 1,2\rangle} + \mathcal{A}_s^{\langle 2,0\rangle}) \| \mathcal{A}^{\langle 1,0\rangle}}$$

$$\frac{: ms' \vdash_{\mathsf{sys}} \mathcal{A}^{\langle 1,0\rangle}}{: \mathsf{evt}(e, d, \varepsilon)::ms' \vdash_{\mathsf{sys}} \mathcal{A}^{\langle 1,0\rangle}} \qquad \frac{}{: \emptyset \vdash_{\mathsf{sys}} \mathcal{F}^{\langle 1,0\rangle}}$$

The first part of the returned FSM is associated to $\mathsf{evt}(e, d, s)$, and describes the behavior of a top level group containing the sequence of actions s, and triggered by an event e. It is the sequential composition of $\mathcal{T}_e$, a trigger FSM labeled by e and the FSM associated to s, by a call to the rule $\vdash_{\mathsf{seq}}^{\mathsf{loose,global}}$, presented below, where s is treated as a group with attributes loose, global. This part is composed in parallel with the FSM $\mathcal{A}$ built by a recursive call of $\vdash_{\mathsf{sys}}$ on $\mathsf{evt}(e', d', \_)::ms'$, the rest of the score. When the end of score is reached, the final FSM is constructed by adding an ender $\mathcal{F}^{\langle 1,0\rangle}$ with 1 provider.

FSM for actions sequences We now define the rule $\vdash_{\mathsf{seq}}^{al}$ for building the FSM associated to an action sequence s, under the attributes in al. The rule will parse the sequence s and build a FSM that will send these actions according to the strategies in al. This rule will also traverses the list of events occurring in the score, after the event e. Indeed, synchronization with these events is required in some strategies.

Every call to this rule will have the form

$$\delta, ms, evt, x : s \vdash_{\mathsf{seq}}^{al} \mathcal{A}$$

where the second auxiliary argument ms is the list of events that remain to be processed, the third auxiliary argument evt (called *closest event*) is the last event before the action currently parsed (*i.e.* the first action of s), in the timeline defined by the score. The first auxiliary argument δ is an accumulator: it is the sum of delays of actions parsed so far in the given sequence, minus the durations of the processed events. In other term, it is the duration between the *closest event* and the action itself. Finally, the fourth auxiliary argument x is a flag whose role is explained later.

The base case, when the list of actions is empty, simply returns an ender.

$$\frac{}{\delta, ms, evt, x : \varepsilon \vdash_{\mathsf{seq}}^{al} \mathcal{F}^{\langle i,0\rangle}}$$

where i depends on the attribute sequence al.

When the list of actions is not empty, a call to $\vdash_{\mathsf{seq}}^{al}$ will first update the accumulator δ by adding the delay d of the currently parsed action, and carry on with a call to a second rule $\vdash_{\mathsf{seq}_1}^{al}$.

$$\frac{\delta + d, ms, evt, \mathsf{F} : \mathsf{act}(d, a, al')::s' \vdash_{\mathsf{seq}_1}^{al} \mathcal{A}}{\delta, ms, evt, \mathsf{F} : \mathsf{act}(d, a, al')::s' \vdash_{\mathsf{seq}}^{al} \mathcal{A}}$$

Note that the flag x must be F and keeps this value. The rule $\vdash^{al}_{\mathsf{seq}_1}$ will look for the *closest event* before the action currently parsed, in order to update the third auxiliary argument.

$$\frac{\delta - d_e, ms', \mathsf{evt}(d', e', s'), \mathsf{T} : s \vdash^{al}_{\mathsf{seq}_1} \mathcal{A}}{\delta, \mathsf{evt}(d', e', s')::ms', \mathsf{evt}(d_e, e, s_e), x : s \vdash^{al}_{\mathsf{seq}_1} \mathcal{A}}$$

if $\delta \geq d_e$.

If the *closest event* $\mathsf{evt}(d_e, e, s_e)$ finishes before the action currently parsed, then the third auxiliary argument is updated to $\mathsf{evt}(d', e', s')$ (the head of the secondary argument) which is removed from the list and the flag (fourth auxiliary argument) is set to T. Moreover, the duration d_e of che closest event e is subtracted from the accumulator δ.

Otherwise, when the *closest event* e finishes after the action currently parsed, it means that we have found the *closest event* before the currently parsed action. Then we proceed by sending the current action.
We consider two cases. The first case is for an *atomic action* $\mathsf{act}(d, a, al')$, with $a \in OM$,

$$\frac{\begin{array}{c} d, \delta, e', e, x : \vdash^{al}_{\mathsf{delay}} \mathcal{A}_d^{\langle n,m\rangle} \qquad \mathsf{act}(d, a, al') \vdash^{al}_{\mathsf{atom}} \mathcal{A}_a^{\langle m,m\rangle} \\ \delta, \mathsf{evt}(d', e', s')::ms', \mathsf{evt}(d_e, e, s_e), \mathsf{F} : s \vdash^{al}_{\mathsf{seq}} \mathcal{A}^{\langle m,0\rangle} \end{array}}{\delta, \mathsf{evt}(d', e', s')::ms', \mathsf{evt}(d_e, e, s_e), x : \mathsf{act}(d, a, al')::s \vdash^{al}_{\mathsf{seq}_1} \mathcal{A}_d^{\langle n,m\rangle} + \mathcal{A}_a^{\langle m,m\rangle} + \mathcal{A}^{\langle m,0\rangle}}$$

if $\delta < d_e$.

In order to treat $\vdash^{al}_{\mathsf{seq}}$ with an atomic action, we call first $\vdash^{al}_{\mathsf{delay}}$ to specify the management of the delay (when $d > 0$), according to the attribute list al. The FSM $\mathcal{A}_d$, returned by $\vdash^{al}_{\mathsf{delay}}$, is concatenated with $\mathcal{A}_a$, a FSM in charge of sending the action a. Both $\mathcal{A}_d$ and $\mathcal{A}_a$ will be defined below according to the attribute list al. Finally we call $\vdash^{al}_{\mathsf{seq}}$ to iterate on the rest of the action sequence s and concatenate the result to the FSM already computed. Note that the flag is set to F in this recursive call.

The case of a *compound action* $\mathsf{act}(d, sa, al')$ is as follows.

$$\frac{\begin{array}{c} d, \delta, e', e, x : \vdash^{al}_{\mathsf{delay}} \mathcal{A}_d^{\langle n,m\rangle} \\ \delta, \mathsf{evt}(d', e', s')::ms', \mathsf{evt}(d_e, e, s_e), \mathsf{F} : sa \vdash^{al'}_{\mathsf{seq}} \mathcal{A}_{sa}^{\langle m',0\rangle} \\ \delta, \mathsf{evt}(d', e', s')::ms', \mathsf{evt}(d_e, e, s_e), \mathsf{F} : s \vdash^{al}_{\mathsf{seq}} \mathcal{A}^{\langle m,0\rangle} \end{array}}{\delta, \mathsf{evt}(d', e', s')::ms', \mathsf{evt}(d_e, e, s_e), x : \mathsf{act}(d, sa, al')::s \vdash^{al}_{\mathsf{seq}_1} \mathcal{A}}$$

if $\delta < d_e$,
where $\mathcal{A} = \mathcal{A}_d^{\langle n,m\rangle} + (\mathcal{I}^{\langle m,m\rangle} \| \mathcal{A}_{sa}^{\langle m',0\rangle}) + \mathcal{A}^{\langle m,0\rangle}$ if $m \geq m'$,
and $\mathcal{A} = \mathcal{A}_d^{\langle n,m\rangle} + (\mathcal{I}^{\langle m,m\rangle} \| \mathcal{I}^{\langle 2,2\rangle} + \mathcal{A}_{sa}^{\langle m',0\rangle}) + \mathcal{A}^{\langle m,0\rangle}$ otherwise.

The only difference with the case of an atomic action is the treatment of the action itself, which is processed with a recursive call to $\vdash^{al'}_{\mathsf{seq}}$, applied to the sequence of actions sa (the content of the compound action), and following the attributes al'.

It remains to consider the case where the second auxiliary argument is empty, because we have reached the end of the event list on the score. In this case, the sequence of actions is treated with the loose strategy (whatever the strategy specified in the score). The case of an *atomic action* is then:

$$\frac{\mathsf{act}(d,a,al') \vdash^{\mathsf{loose},err}_{\mathsf{atom}} \mathcal{A}_a^{\langle m,m\rangle} \qquad d,\delta,e,e,\mathsf{F} : \vdash^{\mathsf{loose},err}_{\mathsf{delay}} \mathcal{A}_d^{\langle n,m\rangle} \qquad \delta,\varepsilon,evt,\mathsf{F} : s \vdash^{sync,err}_{\mathsf{seq}} \mathcal{A}^{\langle m,0\rangle}}{\delta,\varepsilon,evt,x : \mathsf{act}(d,a,al')::s \vdash^{sync,err}_{\mathsf{seq}_1} \mathcal{A}_d^{\langle n,m\rangle} + \mathcal{A}_a^{\langle m,m\rangle} + \mathcal{A}^{\langle m,0\rangle}}$$

And the case of a compound action is treated similarly as above.

FSM for action's delays and atomic actions The attribute list will determine how to manage a delay d in the call of the rule $\vdash^{al}_{\mathsf{delay}}$ and how to treat an atomic action $a \in OM$ in the call of the rule $\vdash^{al}_{\mathsf{atom}}$. We detail in the following the FSMs constructed for every combination of attributes al.

Case $al = \mathsf{loose}, \mathsf{local}$ In this case, displayed Figure 15, the FSM build by $\vdash_{\mathsf{delay}}$ waits for the delay d when in provider 1 (normal mode) and the FSM of $\vdash_{\mathsf{atom}}$ sends the action a, with no care of event detections. In the error mode (provider 2), both $\vdash_{\mathsf{delay}}$ and $\vdash_{\mathsf{atom}}$ do nothing (the delay and action are skipped).

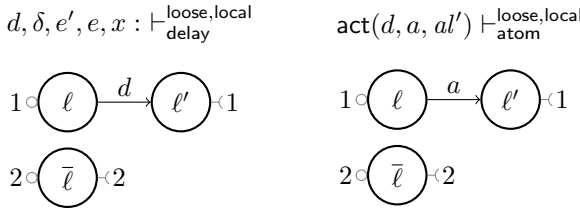

Figure 15: FSM managing the delay d (left) and the atomic action a (right) for the attributes loose, local.

Example 7 Figure 16 depicts the FSM constructed for the group s_1 of our running example which has the attributes loose and local. The parts built at each application of rule $\vdash^{al}_{\mathsf{seq}}$ are framed and annotated with $\mathcal{I}^{\langle 2,2\rangle}$, for starting the group, $\mathcal{A}_a$ (resp. $\mathcal{A}_s$), for handling one atomic action a (resp. one subsequence s) including the delay and the action emission, or $\mathcal{F}^{\langle 2,0\rangle}$, for ending the FSM. ◊

Case $al = \mathsf{loose}, \mathsf{global}$ Figure 17 depicts the FSM constructed by $\vdash_{\mathsf{delay}}$ and $\vdash_{\mathsf{atom}}$ for the combination of attributes loose and global, with different cases according to the flag and the value of the accumulator δ.

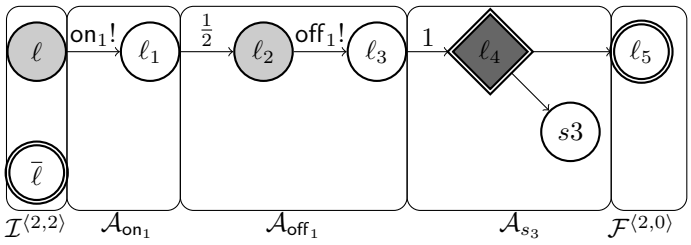

Figure 16: Running example: The FSM for the group s_1.

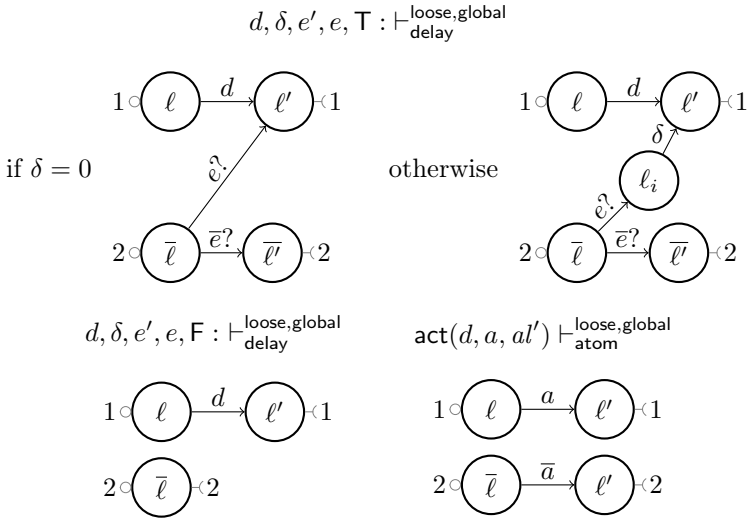

Figure 17: FSMs managing a delay d and atomic action for the attributes loose, global.

In the left part of the Figure 17 the FSM waits for d time units in normal mode (provider 1) and does not wait in error mode (provider 2). In the top right part, the expected *closest event* before the current action is detected, and causes a transition from the error mode into the normal mode. The duration δ is the delay between e and the action (computed in the accumulator of $\vdash_{\mathsf{seq}_1}$). For the treatment of an atomic action (bottom and right part), the action is sent into the normal mode, however for the error mode the emission depends on its proper attribute (al'), depicted with the notation $\overline{a}$: it is sent for the attribute global and not otherwise.

Example 8 Figure 18 shows the FSM for the group s_{e1} of our running example. This group waits for the detection of e_1 or its erroneous signal and launches the group s_1 (without delay since $d = 0$). ◊

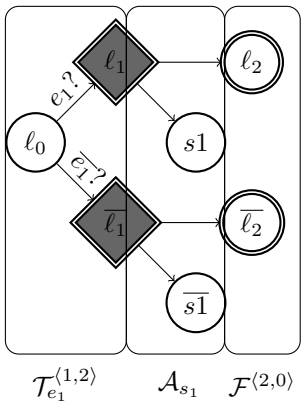

Figure 18: Running example: The FSM of the group s_{e1}.

Case $al = \mathsf{tight}, \mathsf{local}$ The case of the attribute tight is depicted in Figure 19 for the rule $\vdash_{\mathsf{delay}}$. In the first case (left), when the flag is F, the FSM waits (when in normal mode – provider 1) for the delay d before the current action. Recall that the third auxiliary argument e' is the event, next to the *closest event* e (fourth auxiliary argument) before the current action. If this event e' arrives earlier than expected, *i.e.* before d, then the FSM switches from normal mode (provider 1) to another mode called *early mode* (provider 3). If e' is notified missing (signal $\overline{e'}$) before e' was expected, then the FSM switches from normal mode (provider 1) to a fourth mode called *early error mode* (provider 4). The provider 2 corresponds to the error mode, as above.

In the second case (middle), when the flag is T, the FSM synchronizes the current action a to the *closest event* e, *i.e.* it waits first for the event e (transition from the provider 1 – normal mode), and then waits for δ, the delay (computed by $\vdash_{\mathsf{seq}_1}$) between e and the current action. If, instead of receiving e, the FSM receives a notification that e is missing (signal $\overline{e}$) then it moves to the error mode (provider 2). Moreover, if the next event e' arrives (resp. is detected missing) earlier than expected, then there is a move to the early mode – provider 3 (resp. the early error mode – provider 4).

In the third case (right), the delay δ is null, hence it is just skipped.

Note that the composition of such FSMs with 4 providers and 4 seekers is managed properly by the above rules, using appropriate idlers $\mathcal{I}^{\langle m,m\rangle}$ and enders $\mathcal{F}^{\langle m,0\rangle}$, with $m = 2$ or 4, for a correct type inference.

The case of the combination of the attributes tight and local for the rule $\vdash_{\mathsf{atom}}$ is depicted in Figure 20 (left). The local strategy simply skips the action a when in early, early error or error modes (provider 3, 4, 2 respectively).

Example 9 Figure 21 depicts the group s_2 of our running example for the attributes tight and local. The first action is sent directly in case of normal mode, and not if s_2 is started in an error mode. The second action implies a

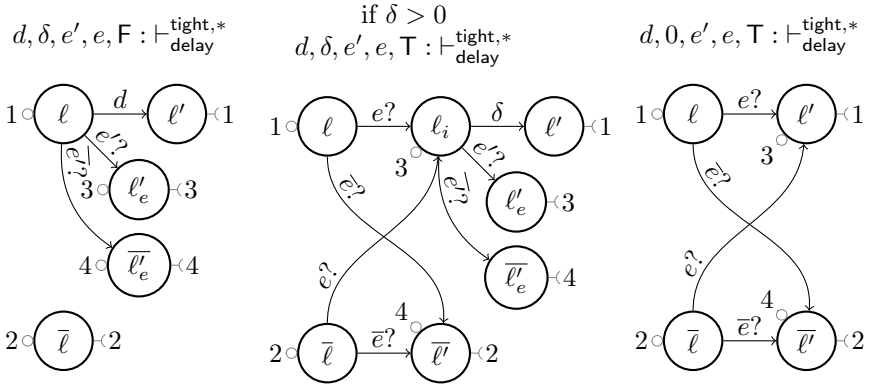

Figure 19: FSM managing the delay for the attribute tight.

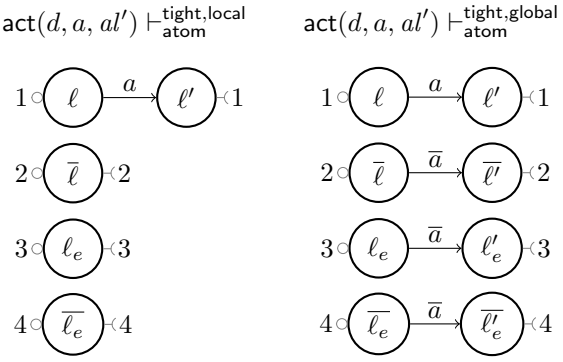

Figure 20: The part of a sequence FSM managing atomic action for the attributes tight, local (left) and tight, global (right).

new closest event (e_3) since the delay $\frac{3}{4}$ is longer than e_2 duration ($\frac{1}{2}$). It applies so the second rule for managing the delay according to the attribute tight with $e' = \epsilon$ and $\delta = \frac{3}{4} - \frac{1}{2} = \frac{1}{4}$. $\Diamond$

Case $al =$ tight, global The case of the combination of the tight and global attributes is depicted in Figure 19 for the rule $\vdash_{\mathsf{delay}}$ and Figure 20 for the rule $\vdash_{\mathsf{atom}}$.

The only difference with the case tight and local is for the missed mode and the early detection of next events. If the second happens, all the not yet handled actions are sent directly (until this next event) and not skipped as in the previous case.

Example 10 The FSM for the group s_2 of the running example is depicted

Figure 22. It holds that $e' = \epsilon$ and $\delta = \frac{1}{4}$, as for the previous case for the management of the atomic action off_2. ◊

Example 11 To complete the IR of our running example, the FSM corresponding to s_{e3} and s_3 are depicted Figures 23. ◊

Example 12 We present a simulation in order to show how a run of the complete model can be performed. For this simulation the input trace $t_{\mathsf{in}}^{sim} = \langle e_1, 0, 60\rangle \cdot \langle e_3, 1.4, 60\rangle$ is played (note that we have chosen a tempo of $60bpm$ to simplify the conversions (a beat played with a tempo of $60bpm$ lasts 1 second)). The initial state is $\langle 0, 0, \{\underline{\ell_0^{\mathcal{M}}}\}, \emptyset\rangle$ and we denote a control point on the location i of the FSM $\mathcal{A}$ as $\ell_i^{\mathcal{A}}$ which is colored in:

red when he can apply the moves (and), (exit), (emit),

green if the time spent in this location is sufficient to move via an (expir),

blue if the elements of Θ allow to run with the move (recv),

pink if it is suspended but can move thanks to the move (send),

black otherwise.

Note that $\mathcal{E}$ is not the environment given for the compilation example but a simpler one following the input trace t_{in}^{sim}:

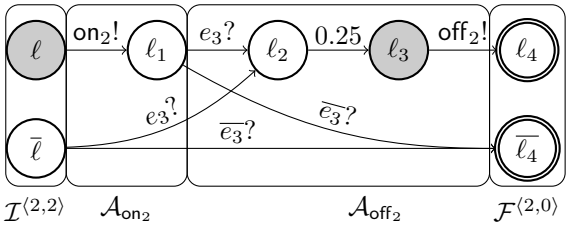

Figure 21: Running example: The FSM for the group s_2 for the attributes tight and local.

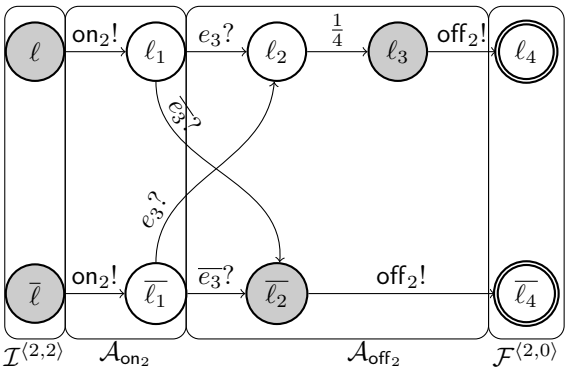

Figure 22: Running example: The FSM for the group s_2.

$\langle 0,0,\{\ell_{\mathbf{0}}^{\mathcal{M}}\},\emptyset\rangle$ —*and*→ $\langle 0,1,\{\ell_{\mathbf{1}}^{\mathcal{E}},\ell_{\mathbf{0}}^{\mathcal{S}}\},\emptyset\rangle$ —*emit* e_1→
$\langle 0,2,\{\ell_1^{\mathcal{E}},\ell_{\mathbf{0}}^{\mathcal{S}}\},\{e_1\}\rangle$ —*and*→ $\langle 0,3,\{\ell_1^{\mathcal{E}},\ell_{\mathbf{0}}^{\mathcal{P}},\ell_{\mathbf{0}}^{\mathcal{A}}\},\{e_1\}\rangle$ —*recv* e_1→
$\langle 0,4,\{\ell_1^{\mathcal{E}},\ell_1^{\mathcal{P}},\ell_{\mathbf{0}}^{\mathcal{A}}\},\{e_1\}\rangle$ —*and*→ $\langle 0,5,\{\ell_1^{\mathcal{E}},\ell_1^{\mathcal{P}},\ell_{\mathbf{0}}^{\mathcal{A}_{se1}},\ell_0^{\mathcal{A}_{se2}}\},\{e_1\}\rangle$ —*recv* e_1→
$\langle 0,6,\{\ell_1^{\mathcal{E}},\ell_1^{\mathcal{P}},\ell_{\mathbf{1}}^{\mathcal{A}_{\mathbf{se1}}},\ell_0^{\mathcal{A}_{se2}}\},\{e_1\}\rangle$ —*and*→ $\langle 0,7,\{\ell_1^{\mathcal{E}},\ell_1^{\mathcal{P}},\ell_{\mathbf{2}}^{\mathcal{A}_{\mathbf{se1}}},\ell_0^{\mathcal{A}_{se2}},\ell_0^{\mathcal{A}_{s1}}\},\{e_1\}\rangle$ —*exit* $\mathcal{A}_{se1}$→
$\langle 0,8,\{\ell_1^{\mathcal{E}},\ell_1^{\mathcal{P}},\ell_0^{\mathcal{A}_{se2}},\ell_{\mathbf{0}}^{\mathcal{A}_{s1}}\},\{e_1\}\rangle$ —*send* on1→ $\langle 0,9,\{\ell_1^{\mathcal{E}},\ell_1^{\mathcal{P}},\ell_0^{\mathcal{A}_{se2}},\ell_{\mathbf{1}}^{\mathcal{A}_{\mathbf{s1}}}\},\{e_1,\mathsf{on}_1\}\rangle$ —*delay* $\delta{=}\frac{1}{2}$→
$\langle 0.5,0,\{\ell_1^{\mathcal{E}},\ell_1^{\mathcal{P}},\ell_0^{\mathcal{A}_{se2}},\ell_{\mathbf{1}}^{\mathcal{A}_{\mathbf{s1}}}\},\emptyset\rangle$ —*expir*→ $\langle 0.5,1,\{\ell_1^{\mathcal{E}},\ell_1^{\mathcal{P}},\ell_0^{\mathcal{A}_{se2}},\ell_{\mathbf{2}}^{\mathcal{A}_{\mathbf{s1}}}\},\emptyset\rangle$ —*send* off_1→
$\langle 0.5,2,\{\ell_1^{\mathcal{E}},\ell_1^{\mathcal{P}},\ell_0^{\mathcal{A}_{se2}},\ell_{\mathbf{3}}^{\mathcal{A}_{\mathbf{s1}}}\},\{\mathsf{off}_1\}\rangle$ —*delay* $\delta{=}0.9$→ $\langle 1.4,0,\{\ell_{\mathbf{1}}^{\mathcal{E}},\ell_1^{\mathcal{P}},\ell_0^{\mathcal{A}_{se2}},\ell_3^{\mathcal{A}_{s1}}\},\emptyset\rangle$ —*expir*→
$\langle 1.4,1,\{\ell_{\mathbf{2}}^{\mathcal{E}},\ell_1^{\mathcal{P}},\ell_0^{\mathcal{A}_{se2}},\ell_3^{\mathcal{A}_{s1}}\},\emptyset\rangle$ —*emit* e_3→ $\langle 1.4,2,\{\ell_{\mathbf{3}}^{\mathcal{E}},\ell_1^{\mathcal{P}},\ell_0^{\mathcal{A}_{se2}},\ell_3^{\mathcal{A}_{s1}}\},\{e_3\}\rangle$ —*exit* $\mathcal{E}$→
$\langle 1.4,3,\{\ell_1^{\mathcal{P}},\ell_0^{\mathcal{A}_{se2}},\ell_3^{\mathcal{A}_{s1}}\},\{e_3\}\rangle$ —*recv* e_3→ $\langle 1.4,4,\{\ell_{\mathbf{3}}^{\mathcal{P}},\ell_0^{\mathcal{A}_{se2}},\ell_3^{\mathcal{A}_{s1}}\},\{e_3\}\rangle$ —*emit* $\overline{e_2}$→
$\langle 1.4,5,\{\ell_{\mathbf{3}}^{\mathcal{P}},\ell_{\mathbf{0}}^{\mathcal{A}_{\mathbf{se2}}},\ell_3^{\mathcal{A}_{s1}}\},\{\overline{e_2},e_3\}\rangle$ —*exit* $\mathcal{P}$→ $\langle 1.4,6,\{\ell_{\mathbf{0}}^{\mathcal{A}_{\mathbf{se2}}},\ell_3^{\mathcal{A}_{s1}}\},\{\overline{e_2},e_3\}\rangle$ —*recv* $\overline{e_2}$→
$\langle 1.4,7,\{\ell_{\overline{\mathbf{1}}}^{\mathcal{A}_{\mathbf{se2}}},\ell_3^{\mathcal{A}_{s1}}\},\{\overline{e_2},e_3\}\rangle$ —*and*→ $\langle 1.4,8,\{\ell_{\overline{\mathbf{2}}}^{\mathcal{A}_{\mathbf{se2}}},\ell_3^{\mathcal{A}_{s1}},\ell_{\overline{\mathbf{0}}}^{\mathcal{A}_{s2}}\},\{\overline{e_2},e_3\}\rangle$ —*exit* $\mathcal{A}_{se2}$→
$\langle 1.4,9,\{\ell_3^{\mathcal{A}_{s1}},\ell_{\overline{\mathbf{0}}}^{\mathcal{A}_{\mathbf{s2}}}\},\{\overline{e_2},e_3\}\rangle$ —*send* on_2→ $\langle 1.4,10,\{\ell_3^{\mathcal{A}_{s1}},\ell_{\overline{\mathbf{1}}}^{\mathcal{A}_{\mathbf{s2}}}\},\{\mathsf{on}_2,\overline{e_2},e_3\}\rangle$ —*recv* e_3→
$\langle 1.4,11,\{\ell_3^{\mathcal{A}_{s1}},\ell_{\overline{\mathbf{2}}}^{\mathcal{A}_{\mathbf{s2}}}\},\{\mathsf{on}_2,\overline{e_2},e_3\}\rangle$ —*delay* $\delta{=}0.1$→ $\langle 1.5,0,\{\ell_{\mathbf{3}}^{\mathcal{A}_{\mathbf{s1}}},\ell_2^{\mathcal{A}_{s2}}\},\emptyset\rangle$ —*expir*→
$\langle 1.5,1,\{\ell_{\mathbf{4}}^{\mathcal{A}_{\mathbf{s1}}},\ell_2^{\mathcal{A}_{s2}}\},\emptyset\rangle$ —*and*→ $\langle 1.5,2,\{\ell_{\mathbf{5}}^{\mathcal{A}_{\mathbf{s1}}},\ell_2^{\mathcal{A}_{s2}},\ell_0^{\mathcal{A}_{s3}}\},\emptyset\rangle$ —*exit* $\mathcal{A}_{s1}$→
$\langle 1.5,3,\{\ell_2^{\mathcal{A}_{s2}},\ell_{\mathbf{0}}^{\mathcal{A}_{\mathbf{s3}}}\},\emptyset\rangle$ —*send* on_3→ $\langle 1.5,4,\{\ell_2^{\mathcal{A}_{s2}},\ell_{\mathbf{1}}^{\mathcal{A}_{\mathbf{s3}}}\},\{\mathsf{on}_3\}\rangle$ —*delay* $\delta{=}0.15$→
$\langle 1.65,0,\{\ell_{\mathbf{2}}^{\mathcal{A}_{\mathbf{s2}}},\ell_1^{\mathcal{A}_{s3}}\},\emptyset\rangle$ —*expir*→ $\langle 1.65,1,\{\ell_{\mathbf{3}}^{\mathcal{A}_{\mathbf{s2}}},\ell_1^{\mathcal{A}_{s3}}\},\emptyset\rangle$ —*send* off_2→
$\langle 1.65,2,\{\ell_{\mathbf{3}}^{\mathcal{A}_{\mathbf{s2}}},\ell_1^{\mathcal{A}_{s3}}\},\{\mathsf{off}_2\}\rangle$ —*exit* $\mathcal{A}_{s2}$→ $\langle 1.65,3,\{\ell_{\mathbf{1}}^{\mathcal{A}_{\mathbf{s3}}}\},\{\mathsf{off}_2\}\rangle$ —*delay* $\delta{=}0.10$→
$\langle 1.75,0,\{\ell_{\mathbf{1}}^{\mathcal{A}_{\mathbf{s3}}}\},\emptyset\rangle$ —*expir*→ $\langle 1.75,1,\{\ell_{\mathbf{2}}^{\mathcal{A}_{\mathbf{s3}}}\},\emptyset\rangle$ —*emit* off_3→
$\langle 1.75,2,\{\ell_{\mathbf{3}}^{\mathcal{A}_{\mathbf{s3}}}\},\{\mathsf{off}_3\}\rangle$ —*exit* $\mathcal{A}_{s3}$→ $\langle 1.75,2,\{\},\{\mathsf{off}_3\}\rangle$

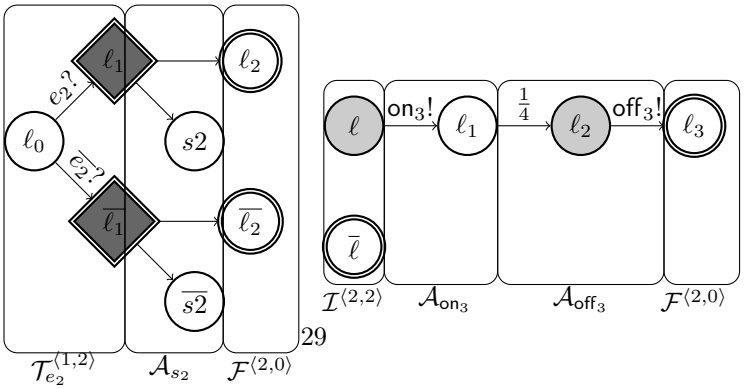

Figure 23: Running example: The FSM for the group s_{e_2} (left) and the FSM for the group s_3 (right).

The expected output trace *tout* created from this simulation is: $t^{sim}_{\mathsf{out}} = \langle \mathsf{on}_1, 0, 60\rangle \cdot \langle \mathsf{off}_1, 0.5, 60\rangle \cdot \langle \mathsf{on}_2, 1.4, 60\rangle \cdot \langle \mathsf{on}_3, 1.5, 60\rangle \cdot \langle \mathsf{off}_2, 1.65, 60\rangle \cdot \langle \mathsf{off}_3, 1.75, 60\rangle$
$\Diamond$

3.4 Translating IR into TA

Once constructed, the IR model can be translated into a network of Timed Automata (TA) [3] in a format that can be handled by tools of the Uppaal suite for MBT [16]. Some graphical coordinates are computed during the translation and used for a nice display of the models under Uppaal, providing composers with useful visual feedbacks of the low level control-flow in their mixed score.

In this section we present a procedure of translation of IR into an equivalent TA. We shall first briefly present the TA variant used in Uppaal. Then, we introduce an alternative semantics for the IR, called broadcast semantics. We denote $\mathcal{L}_{\mathsf{bst}}(\mathcal{A})$ the set of output timed traces of the IR $\mathcal{A}$, following the broadcast semantics. Every IR $\mathcal{A}$ can be straightforwardly converted into an equivalent TA $\mathcal{A}'$. Roughly, equivalence means that the sets of output timed traces of $\mathcal{A}'$ coincide with $\mathcal{L}_{\mathsf{bst}}(\mathcal{A})$ (modulo some restrictions described below). However, it may happen that $\mathcal{L}_{\mathsf{bst}}(\mathcal{A}) \neq \mathcal{L}(\mathcal{A})$ (recall that the latter is the language of output timed traces of the IR $\mathcal{A}$ following the semantics defined in Section 3.1), but this problem can be avoided in the IR obtained by compilation of Antescofo mixed scores, following the procedure presented in Section 3.3. Altogether, this gives the wanted translation of IR obtained from mixed scores into equivalent TA.

Uppaal Timed Automata Let X be a set of variables in $\mathbb{R}_{\geq 0}$ called *clocks*, $\mathcal{G}(X)$ be a set of *guards* on clocks, of the form $x \bowtie c$ with $x \in X$, $c \in \mathbb{N}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$, and $\mathcal{U}(X)$ be a set of *updates* of clocks of the form $x := c$. Moreover let Σ a set of actions and $\Sigma_\tau = \Sigma \cup \{\tau\}$, where τ is an unobservable action. A *timed automaton* [3] over Σ_τ and clocks X is a tuple $\langle L, \ell_0, I, E\rangle$ where:

- L is a set of locations and $\ell_0 \in L$ an initial location,
- $I : L \to \mathcal{G}(X)$ assigns invariants to locations and,
- E is a set of transitions such that $E \subseteq L \times \mathcal{G}(X) \times \Sigma_\tau \times \mathcal{U}(X) \times L$ denoted $\ell \xrightarrow{g,\alpha,u} \ell'$.

We distinguish two kinds of location in L, urgent and committed. The *urgent* locations must be left without letting time pass, the *committed* location forces the network to fire one of its outgoing transitions for the next move. The semantics is defined over states $s = \langle \ell, \bar{v}\rangle$, where $\ell \in L$ is a location and $\bar{v} \in \mathbb{R}^X_{\geq 0}$ is a clock valuation satisfying the invariant $I(\ell)$. A timed automaton can do moves of the two following kinds.

$$\langle \ell, \bar{v}\rangle \xrightarrow[\mathsf{discrete}]{\alpha} \langle \ell', \bar{v}'\rangle \qquad (\mathsf{discrete})$$

if there exists a transition $\ell \xrightarrow{g,\alpha,u} \ell'$, $\bar{v} \models g$, $\bar{v}' \models I(\ell')$ and for each $x := c \in u$, x is updated with the value c in $\bar{v}'$.

$$\langle \ell, \bar{v} \rangle \xrightarrow[\mathsf{delay}]{d} \langle \ell, \bar{v} + d \rangle \qquad (\mathsf{delay})$$

where ℓ is not *urgent* nor *committed*, $\bar{v} + d$ is obtained from $\bar{v}$ by incrementing all clock values by the amount of time d and for all $0 \leq \delta \leq d$, $\bar{v} + \delta \models I(\ell)$.

The move (delay) is the analogous of the IR move of the same name, whereas (discrete) corresponds to all the other IR moves. Having many more discrete moves in IR than in TA was required for modeling priorities in Antescofo synchronous semantics. We have introduced a dedicated IR for this reason, instead of translating Antescofo code directly into TA [18].

A *run* of the automaton $\mathcal{A}$ is a finite sequence of the form

$$r = \langle \ell_0, \bar{v}_0 \rangle \xrightarrow[\mathsf{delay}]{d_0} \langle \ell_0, \bar{v}_0 + d_0 \rangle \xrightarrow[\mathsf{discrete}]{\alpha_0} \langle \ell_1, \bar{v}_1 \rangle \xrightarrow[\mathsf{delay}]{d_1} \dots \xrightarrow[\mathsf{discrete}]{\alpha_n} \langle \ell_{n+1}, \bar{v}_{n+1} \rangle$$

The *trace* associated to the run r is the sequence of triples $\langle \alpha_i, \sum_{j=0}^{i} d_j, 60 \rangle$ for $0 \leq i \leq n$. Here, we assume that $\Sigma = IS \cup OM \cup \mathcal{S}igs$ and consider the *language* $\mathcal{L}(\mathcal{A})$ of output traces of $\mathcal{A}$ defined as the set of all projections on OM of traces associated to a run of $\mathcal{A}$.

The synchronized product TAs $\mathcal{A}_1, \dots, \mathcal{A}_m$ over respectively Σ_τ^i and X_i, denoted $sync(\mathcal{A}_1 \| \dots \| \mathcal{A}_m)$, is the timed automaton: $\mathcal{A} = \langle L, \ell_0, I, E \rangle$ over Σ_τ and X, where:

- $\Sigma_\tau = \cup_{i=1}^{m} \Sigma_\tau^i$
- $L = L_1 \times \dots \times L_m$, $\ell_0 = \ell_0^1 \times \dots \times \ell_0^m$
- for every $\ell = \langle \ell_1, \dots, \ell_m \rangle \in L$, $I(\ell) = \bigwedge_{i=1}^{m} I_i(\ell_i)$
- $X = \cup_{i=1}^{m} X_i$
- E is the set of all the edges $\langle \ell_1, \dots, \ell_m \rangle \xrightarrow{g,\alpha,u} \langle \ell'_1, \dots, \ell'_m \rangle$ such that (where for $\alpha \in \Sigma_\tau, S_\alpha = \{i | 1 \leq i \leq m, \alpha \in \Sigma_\tau^i\}$)
  - for all $1 \leq i \leq m$, if $i \notin \mathcal{S}_\alpha$, then $\ell'_i = \ell_i$, if $i \in S_\alpha$, then there exists g_i and u_i such that $(\ell_i, g_i, \alpha, u_i, \ell'_i) \in E_i$,
  - $g = \bigwedge_{i \in \mathcal{S}_\alpha} g_i$,
  - $u = \cup_{i \in \mathcal{S}_\alpha} u_i$.

Hypotheses and equivalence The conversion of IR into TA, and hence its application in offline MBT workflows involving Uppaal and described in Section 4.2 works under the following restrictions for the IR:

(R_1) Only one time unit is supported for the specification of delays: the musical time.

(R_2) There is no loop including an *and*-transition.

Moreover, let us define precisely the notion of equivalence for the translation. Two output traces t_{out} and t'_{out} are equivalent, written $t_{\mathsf{out}} \cong t'_{\mathsf{out}}$ if the output traces contain the same sets of actions at each logical instant (*i.e.* at the same timestamps). It means that the equivalence is modulo logical instant and doesn't care of the order of actions with the same timestamp. This relation is extended to sets of output traces as expected. This leads to the following third restriction for the output traces:

(R_3) The output traces of $\mathcal{T}_{\mathsf{out}}$ are considered modulo permutations of actions with the same timestamp. The total ordering $\prec$ over $\Sigma_{\mathsf{in}} \cup \Sigma_{\mathsf{out}}$ is ignored during the comparison of output traces.

The purpose of the restriction (R_3) is to fill the gap between the semantics of TA, where there is only one kind of discrete move, and the IR semantics, with several kind of discrete moves for dealing with symbol priorities.

Note that the above restrictions apply only to the offline test procedures involving Uppaal, not to the online procedure.

Broadcast semantics of IR Let us consider first an *alternative semantics* for IR, defined like the IR semantics of Section 3.1, except for the following changes.

The move (emit) is replaced by the following move (broadcast):

$$\langle t, n, \Gamma :: \underline{\langle \ell, \gamma, \mathsf{T} \rangle} :: \Gamma', \Theta \rangle \xrightarrow[\mathsf{broadcast}]{} \langle t, n+k, \tilde{\Gamma} :: \underline{\langle \ell', 0, \mathsf{F} \rangle} :: \tilde{\Gamma}', \sigma :: \Theta \rangle \qquad (\mathsf{broadcast})$$

if all elements of Γ are suspended and there exists $\ell \xrightarrow{\sigma!} \ell' \in \Delta_0$ with $\sigma \in \Sigma^{\mathsf{sig}}_{\mathsf{out}}$, there are $k-1$ running locations $\langle \ell_i, \gamma_i, \beta_i \rangle$ in $\Gamma \cup \Gamma'$ such that there exists $\ell_i \xrightarrow{\sigma?} \ell'_i \in \Delta_1$ and each of them is replaced by $\langle \ell'_i, 0, \mathsf{F} \rangle$, giving $\tilde{\Gamma} \cup \tilde{\Gamma}'$.

The move (recv) is replaced by the following move (deadlock):

$$\langle t, n, \Gamma :: \underline{\langle \ell, \gamma, \beta \rangle} :: \Gamma', \Theta \rangle \xrightarrow[\mathsf{deadlock}]{} \langle t, n+1, \Gamma :: \underline{\langle \ell', 0, \mathsf{F} \rangle} :: \Gamma', \Theta \rangle \quad (\mathsf{deadlock})$$

if none of (broadcast) or (expir) can be applied and there exists $\ell \xrightarrow{\sigma?} \ell' \in \Delta_1$ with $\sigma \in \Theta$.

The move (send) is replaced by the following move also called (send):

$$\big\langle t, n, \Gamma :: \underline{\langle \ell, \gamma, \mathsf{F} \rangle} :: \Gamma', \Theta \big\rangle \xrightarrow[\mathsf{send}]{} \langle t, n+1, \Gamma :: \underline{\langle \ell', 0, \mathsf{F} \rangle} :: \Gamma', \sigma :: \Theta \rangle \qquad (\mathsf{send})$$

if there exists $\ell \xrightarrow{\sigma!} \ell' \in \Delta_0$ with $\sigma \in \Sigma^{\mathsf{ext}}_{\mathsf{out}}$.

A (broadcast) move is an asynchronous communication by rendez-vous, similar to the communication in the synchronized product of TA. Roughly, it gathers one (emit) move of Section 3.1, with none, one or several (recv) moves of Section 3.1. However, an important difference is that the running location $\langle \ell, \gamma, \mathsf{T} \rangle$ enabling the (emit) and the (recv) must all be present in the current vector $\Gamma :: \langle \ell, \gamma, \mathsf{T} \rangle :: \Gamma'$. At the opposite, in the synchronous semantics of Section 3.1, the (recv) could occur later thanks to the use of the set Θ for storing all the symbols sent during one logical instant.

A (deadlock) move is the reception of a symbol or internal signal σ that cannot be received in a (broadcast), because this symbol was sent by an earlier (broadcast) in the logical instant. Moreover a second (broadcast) would not be able to check that this symbol is present in the set Θ. Intuitively, our goal in this semantics is to avoid the (deadlock) as much as possible. For this purpose, we give to the (broadcast) move the lowest priority (using the suspend flag), in order to delay the use of (broadcast) as much as possible.

Finally, the new (send) move is the same as the (send) of Section 3.1, except that it does not care of the order of symbols (restriction R_3) and moreover, it has priority over (broadcast) because it cannot be suspended.

The *broadcast semantics* of IR is the same as the alternative semantics, without the (deadlock) move. We consider the same definition of runs as in Section 3.1, and given an IR $\mathcal{A}$, we denote by $\mathcal{L}_{\mathsf{alt}}(\mathcal{A})$ (resp. $\mathcal{L}_{\mathsf{bst}}(\mathcal{A})$) the set of traces t_{out} such that there exists a run ρ of $\mathcal{A}$ following the alternative semantics (resp. broadcast semantics) and t_{out} is associated to ρ.

The alternative semantics is equivalent (wrt $\cong$, *i.e.* under the restriction R_3) to the IR semantics. Indeed, the (broadcast) move is a sequence of successive (emit) and (recv) moves for the same signal or symbol. In other terms, for all IR $\mathcal{A}'$ $\mathcal{L}_{\mathsf{alt}}(\mathcal{A}') \cong \mathcal{L}(\mathcal{A}')$. However, this does not hold for the broadcast semantics: there exists some IR $\mathcal{A}$ such that $\mathcal{L}_{\mathsf{bst}}(\mathcal{A}) \ncong \mathcal{L}(\mathcal{A})$.

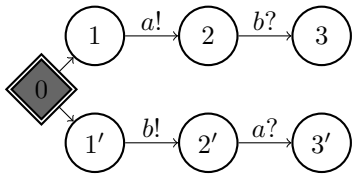

Figure 24: An IR such that $\mathcal{L}_{\mathsf{bst}}(\mathcal{A}) \ncong \mathcal{L}(\mathcal{A})$.

Example 13 Figure 24 displays an example of IR $\mathcal{A}$ such that $\mathcal{L}_{\mathsf{bst}}(\mathcal{A}) \ncong \mathcal{L}(\mathcal{A})$. Indeed, when the running locations are 1 and $1'$, the IR can send a, with a (broadcast), and then send b (in $1'$) and receive it (in 2), again with a (broadcast). But then, the IR is stuck in $2'$. In order to capture the already sent a, a move (deadlock) would be required, looking in the set Θ of symbols sent in the same logical instant. $\Diamond$

The above problem can also occur with the IR obtained by compilation of Antescofo mixed scores, following the procedure of Section 3.3.

Example 14 Let us go back to our running example and consider the input trace t_{in}^{sim} of Example 12. The event e_2 is missing in t_{in}^{sim}, where e_3 is emitted by $\mathcal{E}$ directly after e_1. The proxy $\mathcal{P}$, after the reception of e_3, will emit $\overline{e_2}$ (see Figure 14) to signal the missing event e_2. The group s_{e_2} starts the sub-group

s_2 in error mode when receiving $\overline{e_2}$ and then s_2 sends on_2 and waits for e_3 or $\overline{e_3}$ (see Figures 23 and 22).

In the IR semantics of Section 3.1, the event e_3 is captured by (recv) because it had been sent during the same logical instant (and hence is present in Θ). However, in the broadcast semantics, e_3 is not captured and off_2 is never sent (the FSM for the group s_2 is stuck). The output trace under the broadcast semantics is indeed

$$\langle \mathsf{on}_1, 0, 60\rangle \cdot \langle \mathsf{off}_1, 0.5, 60\rangle \cdot \langle \mathsf{on}_2, 1.4, 60\rangle \cdot \langle \mathsf{on}_3, 1.5, 60\rangle \cdot \langle \mathsf{off}_3, 1.75, 60\rangle$$

It differs from t^{sim}_{out} of Example 12. ◊

This problem can be solved for the IR obtained from mixed score compilation, with an IR transformation procedure which roughly works as follows. We dissociate the communications between $\mathcal{E}$ and $\mathcal{P}$ and between $\mathcal{P}$ and the rest of the model. We introduce for this purpose a new and fresh signal $s_{e_i} \in \mathcal{S}igs$ for each $e_i \in IS$, signaling the detection of the event e_i. We rename the signals $\overline{e_i}$ into $\overline{s_{e_i}}$ since it is an error detection for e_i. After this transformation, all the symbols received in the IR model, with the exception for $\mathcal{E}$ and $\mathcal{P}$, are in $\mathcal{S}igs$. Moreover the proxy is modified in order to echo the reception of an event e_{i+1} that causes the emission of an error signal $\overline{e_i}$. The echo has the form of a signal $s_{e_{i+1}}$ emitted right after $\overline{e_i}$. This is illustrated in Figure 25 for the running example.

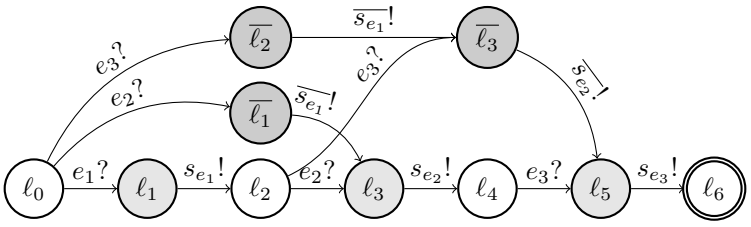

Figure 25: Running example: The proxy FSM transformed to prevent from (deadlock).

This transformation has been implemented in the offline verification framework presented in Section 4.2. It ensures the following property: for every IR $\mathcal{A}'$ obtained by compilation of Antescofo mixed scores, following the procedure presented in Section 3.3, there exists an IR $\mathcal{A}''$ such that $\mathcal{L}(\mathcal{A}'') = \mathcal{L}(\mathcal{A}')$ and $\mathcal{L}_{\mathsf{bst}}(\mathcal{A}'') \cong \mathcal{L}(\mathcal{A}'')$.

Translation of IR into TA Let $\mathcal{A}'$ be an IR obtained by compilation of an Antescofo mixed score and let $\mathcal{A}''$ be the IR obtained from $\mathcal{A}$ following the above procedure. We show how to construct a corresponding TA $\mathcal{A} = \langle L, \ell_0, I, E\rangle$ over Σ_τ and X such that $\mathcal{L}(\mathcal{A}) \cong \mathcal{L}_{\mathsf{bst}}(\mathcal{A}'')$. Altogether, it holds that $\mathcal{L}(\mathcal{A}) \cong \mathcal{L}(\mathcal{A}')$, which is the wanted property.

The TA is constructed as the synchronized product of several TAs $\mathcal{A}_i$ built from the IR. Each TA $\mathcal{A}_i$ will be over a single clock x_i and over the alphabet $\Sigma_\tau = IS \cup OM \cup \mathcal{S}igs$, with $\tau \in \mathcal{S}igs$. The locations of the TAs $\mathcal{A}_i$ are location of the IR $\mathcal{A}''$, plus some fresh locations added below. The transitions and invariants of the TAs $\mathcal{A}_i$ are built during a traversal of the IR $\mathcal{A}''$. We consider that the construction works on a current TA (one of the $\mathcal{A}_i$'s), and present below each step of the traversal.

For a transition *recv*-transition $\ell \xrightarrow{\sigma?} \ell'$ or *emit*-transition $\ell \xrightarrow{\sigma!} \ell'$ of the IR $\mathcal{A}''$, we add to the current TA $\mathcal{A}_i$ a transition $\ell \xrightarrow{\mathsf{T},\sigma,\{x_i:=0\}} \ell'$, with the action α the label σ emitted or waited in the IR, without guard and with a reset of the local clock x_i.

For every *wait*-transition $\ell \xrightarrow{[d,d']} \ell'$ of $\mathcal{A}''$, we add to the current TA $\mathcal{A}_i$ a transition $\ell \xrightarrow{x_i \geq d,\tau,\{x_i:=0\}} \ell'$ in E, and an invariant $I(\ell) = x_i \leq d'$. In this translation, d' is set as the maximum bound of the local clock x_i in the invariant of its previous location $I(\ell)$. This prevents the automaton from staying on the location more than d'. Then, d is set as the minimum bound of x_i into the guards of the transition (note that when $d = d'$ this combination forces the wait for strictly d time unit).

We unfold the *and*-transitions of the IR model into several transitions of concurrent TAs. An *and*-transition $\ell \xrightarrow{\mathsf{and}} \ell_1 \| \ell_2$ contains two branches. For the branch $\ell \to \ell_1$, we add to the current TA $\mathcal{A}_i$ a transition of the form $\ell \xrightarrow{\mathsf{T},\lambda_{i+1},\{x_i:=0\}} \ell_1$, where λ_{i+1} is a fresh internal signal of Σ_τ, used to trigger another TA. Then we continue the construction of $\mathcal{A}_i$ starting with the location ℓ_1. When the construction of $\mathcal{A}_i$ is terminated, we start with a new current TA $\mathcal{A}_{i+1}$ which contains initially a transition $\ell \xrightarrow{\mathsf{T},\lambda_{i+1},\{x_i:=0\}} \ell_2$ (associated to the second branch $\ell \to \ell_2$) and we continue the construction of $\mathcal{A}_{i+1}$ starting with the location ℓ_2.

The unfolding terminates for all IR satisfying restriction (R_2). It can be observed that this is the case of every IR obtained by compilation of an Antescofo mixed score, and that the property is preserved by the above transformation of IR.

Finally for each urgent transition in $\mathcal{A}''$, we declare its source location ℓ as an *urgent* location in the TA, if ℓ belongs to $\mathcal{E}$ or $\mathcal{P}$, or as *committed* location otherwise.

Example 15 The TA obtained by unfolding the IR of the running example is depicted in Figure 26 for the group s_2. ◊

Note that the above transformation of IR into equivalent TA is correct only for the particular case of IR obtained from mixed scores by the compilation procedure presented in Section 3.3. However, we do not propose a generic translation from arbitrary IR into equivalent TA.

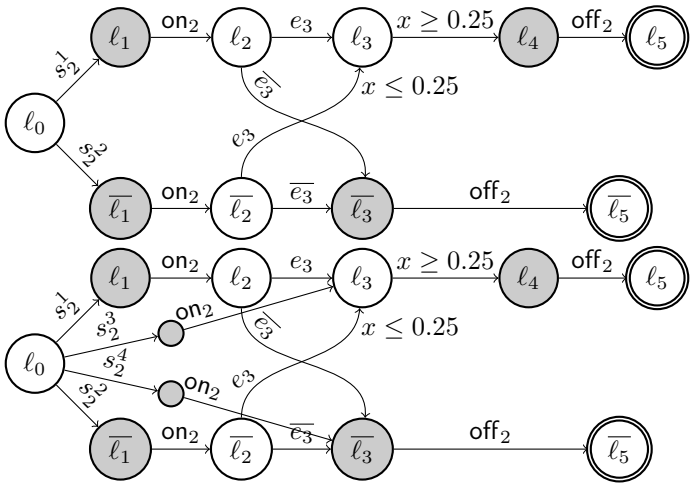

Figure 26: The translated TA for the group s_2 in case of 2 or 4 providers. Each transition resets x to 0.

3.5 Model of Performances

A timed input trace t_{in} is commonly seen as a sequence of input symbols $e \in IS$ timely separated by physical durations. In our case (as described Section 2.3) it is a suite of triples $\langle e_i, t_i, p_i \rangle$ of events with timestamps in musical time. In this section, we present some related models of musical perfomance, which will be useful for test generation in Section 4.

We recall that the tempo curve τ associated to a trace t_{in} as above is defined by $\tau(t) = p_i$ for all t with $t_i \leq t < t_{i+1}$. We can convert an input trace t_{in} into a musical performance, as follows. First, we extract from the third components of the triples in t_{in} a tempo curve τ. Second, we convert the durations, computed from the musical timestamps in the second component of triples in t_{in}, into real durations, in physical time, using τ.

Some works in musical cognitive research have proposed more accurate representation of musical performances. *Time-maps* by Jaffe [1], *time-warps* by Dannenberg [2], or *time-deformations* by Anderson and Kuivila [4] are monotonically non-decreasing functions mapping score durations (in musical time) into performed durations (in physical time). In [15], Dannenberg gives two special cases of such functions called *shift* and *stretch* operators. The first express operations such as delay, rest or pause and the second deals with the tempo variations. In [24, 25], Honing proposes *Timing Functions* (TIF) which combine two time-warps: a tempo curve $f^{\times}$ and a time-shift function f^{+}, defining variations of events' durations, independently of the tempo changes. Although tempo variations induce changes of durations and reciprocally, Honing outlines the interest of considering independently tempo curves and time-shifts for defining musical performances. They have indeed two well distinct musical significance. Roughly,

the first describes global continuous changes of durations, and the second local changes (like *swing notes*).

Our work can take advantage of these functions and can explicitly describe some interpretations of the input durations. A direct application can be done to the ideal input trace using interesting time-functions for the generation phase. In reverse, interesting input traces can be defined in this formalization to make well understandable a specific composer or musician's performance.

4 Test Framework

This section presents the implementation of our MBT framework, and its application to the score-based IMS Antescofo. We have developed different approaches of MBT: (a) an offline approach (Section 4.2), where test data is generated using the Uppaal suite, and then stored into files before being executed, and (b) an online approach (Section 4.3), based on a VM interpreting IR, with an on-the-fly generation and execution of test cases. Both approaches include the preliminary phase of the compilation from mixed-scores into IR (Section 4.1). We have developed testing solutions based on existing tools, and also developed our own tools better suited to our case study.

4.1 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 3.3. Several options are offered for the construction of the IR related to the environment $\mathcal{E}$, in particular to fix the values of n_{err} and κ from the Section 3.3. The most general case (any note can be missed) results in a model $\mathcal{E}$ with a quadratic number (in score's size) of transitions and an exponential number of possible input traces. The explosion can be controlled by choosing appropriate hypotheses on the environment $\mathcal{E}$.

4.2 Offline Testing

Figure 27 outlines our implementation of the MBT framework with an offline generation of input traces. The workflow of Figure 27, following the principles announced in Section 2, proceeds in several steps described below.

In a first step, after the construction of the IR models $\mathcal{E}$ and $\mathcal{S}$ from the given mixed score, using the techniques and tools presented in Sections 3.3 and 4.1, these IR are translated into TA networks, respectively $\mathcal{A}_\mathcal{E}$ and $\mathcal{A}_\mathcal{S}$, as described in Section 3.4. Consequently, this approach works under the restrictions $\mathsf{R}_1 - \mathsf{R}_3$ needed in Section 3.4.

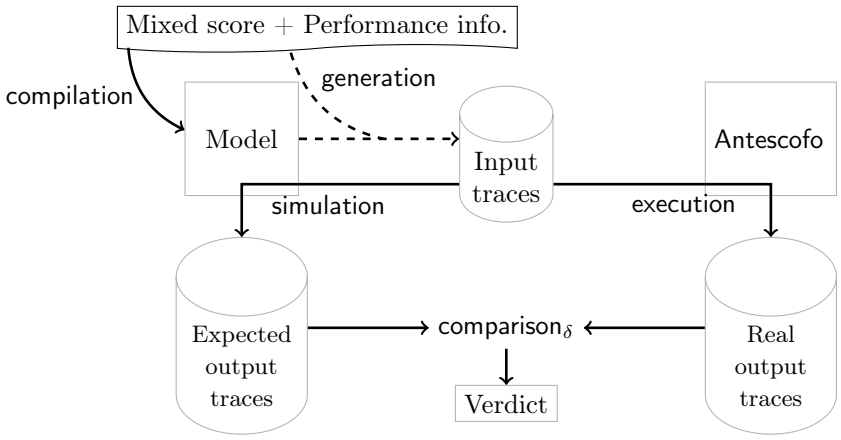

Figure 27: Offline Score-based IMS testing workflow

Covering Generation The model checker Uppaal [16] is a framework to visualize, create, simulate and verify Networks of Timed Automata. We use the Uppaal extension called CoVer [23] in order to generate automatically suites of input traces t_{in}, in $\mathcal{E}$, 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 $\mathcal{A}_{\mathcal{E}}$ or $\mathcal{A}_{\mathcal{S}}$ 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 synchronized product of $\mathcal{A}_{\mathcal{E}}$ and $\mathcal{A}_{\mathcal{S}}$ 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 IUT(t_{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 3.3, 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 $\mathcal{E}$, using the parameters n_{err} (the number of successive missable events) and κ (the allowed variation of events' durations) presented in Section 3.3 for the construction of $\mathcal{E}$.

The main limitations of the offline approach are that (a) it does not scale well for testing large real mixed scores and (b) the input traces are not musically relevant (because of CoVer which strictly follows the model constraints). However, this approach is well suited for debugging the system Antescofo, using small ad-hoc scores (see Section 5).

Test cases generation by fuzzing the ideal trace Trying to answer of these limitations, an alternative method for the generation of relevant test data is to start with the ideal trace associated to a mixed score (as defined in Section 2.5) and add deformations of several kinds. For this purpose, we use the models from the music cognition described in Section 3.5 in order to create musically relevant performances for test purpose.

The implemented fuzzing function takes in input an ideal trace and parameters for bounding the deviations on the time-shifts, the tempo values and the number of missing notes. It generates some random values within theses limits and applies them to return an input trace t_{in} as a mutation of the ideal trace. An interesting open question in this context is the definition of TIFs in [24, 25] for the generation of covering test suites following criteria similar to those of the above paragraph.

Generation from an audio file We have considered a third alternative for the generation of test input traces, based on an audio recording. The developers of the IMS Antescofo use to work with sound files in order *e.g.* to analyse a specific performance that causes errors. Such sound files can be translated into input traces simply by marking the timestamps of their event's onsets. We can do that manually or with a software, *e.g.* Antescofo itself, which can trace the events triggered when the listening machine detects them from the audio file.

Simulation We describe now the procedure based on Uppaal that we follow in order to compute the output trace t_{out} corresponding to a given input trace t_{in}. It can be applied whatever the method was used to generate t_{in} (with CoVer, the fuzzing or the audio file processing fashion).

In the compilation procedure described in Section 3.3, we decorate the IR generated with coordinates used for their visualization in Uppaal. It is useful for manual exploration of the models and also for graphical simulation of their execution in Uppaal.

For the computation of t_{out} nevertheless, we use the command line tool Verifyta in order to execute a given input trace t_{in} and compute the corresponding output trace $t_{\mathsf{out}} = \mathcal{S}(t_{\mathsf{in}})$, according to the model $\mathcal{S}$.

More precisely, given t_{in} we first generate a deterministic IR $\mathcal{E}_{t_{\mathsf{in}}}$ modeling an environment which will strictly follow the input trace. This IR is converted into a deterministic TA $\mathcal{A}_{t_{\mathsf{in}}}$. The simulation of the TA network is then performed by traversing $\mathcal{A}_{t_{\mathsf{in}}}$, which will send event symbols to the rest of the model $\mathcal{A}_{\mathcal{S}}$. Uppaal offers options to trace the result, which is then translated in a t_{out} in our format of output traces with triples $\langle a, t, p\rangle$. We note here that the traces (t_{in} and t_{out}) are in musical time (*i.e.* in Uppaal model time unit).

Tests case execution The execution computes the real output t'_{out} by timely sending to the IUT the suite of inputs present in the input trace t_{in}. 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

Section 2.1.

Scenario 1 The first scenario is performed using a standalone version of Antescofo equipped with an internal *test adapter* module. The adapter iteratively reads elements $\langle a_i, t_i, p_i \rangle$ of t_{in} in a file. The duration $d_i^{\mathsf{mu}} = t_{i+1} - t_i$ of the event e_i (in musical time) is converted into physical time by:

$$d_i^{\mathsf{ph}} = \frac{d_i^{\mathsf{mu}}.60}{p_i} \tag{1}$$

The adapter then waits for d_i^{ph} seconds before sending the event e_{i+1} and the tempo value p_{i+1} to the RE. More precisely, it does not physically wait, but instead notifies a *virtual clock* in the reactive engine RE that the time has flown of d_i^{ph} 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 messages sent by the RE are logged in t'_{out}, with timestamps in physical time (*i.e.* with a tempo of 60bpm).
Here, the blackbox is the RE (the LM is idle).

Scenario 2 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. Notice that the CoVer input traces are not well suited for this scenario because of the second limitation (not musically relevant). It conduces to exponential tempo fluctuations that can be controlled in scenario 1 with an added tempo curve.

Scenario 3 A third scenario is executed in a version of Antescofo embedded into the visual programming language MAX [32] (as a MAX patch). In this case, the blackbox is the whole IMS Antescofo, and instead of sending discrete events of t_{in} to the IUT (like in scenarios 1 and 2), we convert them into MIDI data and generate an audio stream with a MIDI synthesizer (in MAX).

Comparison and verdicts The verdicts are produced offline by a tool comparing the expected and monitored traces t_{out} and t'_{out} with an acceptable latency δ (generally 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. Moreover we have to convert the expected output trace t_{out}, created in musical time from Uppaal, into physical time by applying the tempo updated from the execution (with the equation (1)). This operation is executed locally, using the tempo values associated to each event.

A verdict is pretty printed to inform the testers on the conformance of Antescofo to the models. We mark as errors unexpected or missed atomic actions sent or not by the IUT and a delay more than 0.1 ms between the model and the system time-stamps. The document is split in logical instants in order to visualize clearly the sequence of actions related to an external event reception. The

verdicts also detail the variations between the input trace and the ideal trace in order to outline early or late events, which are not always easy to detect.

Example 16 We depict, as an example, Figure 28 the verdict of our running example tested on the input trace t_{in}^{sim}: $\langle e_1, 0, 60\rangle \cdot \langle e_3, 1.4, 60\rangle$. The verdict follows the real trace (on the left) and details the expected values (on the right) depicting respectively the label, the physical timestamp (called *now* for Antescofo) and the relative score timestamp (called *rnow*) for each item. Notice that the labels on the right change to be compatible with Uppaal model checker, moreover the score tempo is initially set to 120_{bpm} that has an impact on the relative durations. Each time flowing is compared to detect any time differences, moreover the interpretation information is depicted with the line *ideal*, event e_3 in our case which is detected late ($1.4s$ rather than $0.75s$). This verdict fails, raising an error in Antescofo. Indeed a cross can be seen at the action off_2 which is timestamped to 1.4 seconds for Antescofo but was expected at 1.525 seconds (resulting to a delay of 0.125 seconds). It is typically an error of synchronization management (the action seems to be related to the previous event e_2 rather than e_3) and was reported to the developers. The event e_4 is an "end" event, added in the input suite to stop explicitly the test. ◊

```
-------------------------------------------------------------------------------
|-        Antescofo Trace                |          Expected Trace          -|
|-    label      now      [rnow]         |label  comp. timestamp [ref beat] -|
|-----------------------------------------------------------------------------|
|        on1       0        [      0]   |         a0       0       [      0]|
|    e1_0.00       0        [      0]   |         e1       0       [      0]|* 120BPM
 +   0.25 (    0.5 *     120)           ==   0.25 (    0.5 *     120)
|       off1    0.25        [    0.5]   |         a1    0.25       [    0.5]|
 +    0.5 (    1 *     120)              ==   0.5 (      1 *     120)
|        on3    0.75        [      1]   |         a2    0.75       [    1.5]|
 +  0.125 (      0.25 *     120)         ==   0.125 (   0.25 *     120)
|      offt3   0.875        [      1]   |         a3   0.875       [   1.75]|
 +  0.525 (    1.05 *     120)           ==   0.525 (   1.05 *     120)
|        on2     1.4        [    1.5]   |         a4     1.4       [    2.8]|
x       off2     1.4        [    1.5]   x         a5   1.525[   3.05]       x   delta:-0.125
|    e3_1.50     1.4        [    1.5]   |         e3     1.4       [    2.8]|* 60BPM
                                 >  ideal 0.75s
 +    0.5 (    0.5 *      60)           ==     0.5 (      1 *      120)
|        END     1.9        [      2]   |         e4     1.9       [    3.8]|* 60BPM
|-----------------------------------------------------------------------------|
Error :: Test KO
```

Figure 28: The verdict for the trace t_{in}^{sim}. The real output trace is depicted on left, the expected on the right, showing the label, the physical timestamp and the relative timestamp for each item. The mark '+' indicates a new logical instant and details the time elapsed. The differences between the ideal trace and the input trace are shown with '<', '>' and '=='. The mark 'x' indicates an error.

4.3 Online Testing

Online MBT of realtime systems is a complementary method allowing non-determinism and "on-the-fly" generation that prevents state explosions (but at the price of loosing the exhaustiveness of the generated test suites). This testing method is an important research topic, in particular for the Uppaal team [16] which designed an online MBT tool called TRON [23]. We have tried to apply this tool to our case study, without success. The reason is that we need to deal with multiple time units, in particular the musical time relative to a tempo together with physical time. Roughly, TRON can manage several clocks rates only when they are all defined as a constant factor of the wall clock. This restriction does not comply with the notion of an evolving tempo (following tempo curves), which is crucial in our case.

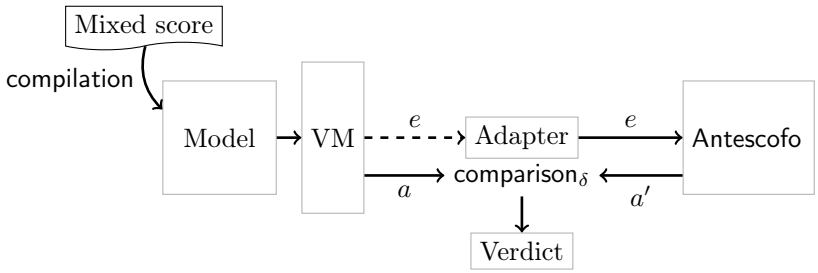

Figure 29: Online Score-based IMS testing workflow

We have developed an online MBT framework dedicated to our case study, based on a Virtual Machine (VM) implemented on purpose, for executing the IR described Section 3, and managing the IR multi-rate clocks. Thanks to this machine the online test framework runs directly on the IR obtained by compilation of the mixed score (see Figure 29). The VM is linked via an adapter to an instance of our IUT Antescofo. The test procedure works in several steps performed online: the generation of input test data (dashed arrow in Figure 29), the generation of the expected and real output test data corresponding to the input and the comparison.

Unlike the offline approach of Section 4.2, the online procedure does not need the restrictions of Section 3.4.

Generation Depending on the environment $\mathcal{E}$, the IR may be non-deterministic. Indeed we recall that $\mathcal{E}$ may contain *wait*-transitions of the form $\ell \xrightarrow{[d(1-\kappa),d(1+\kappa)]} \ell'$ for some $0 \leq \kappa \leq 1$, and branches with several *emit*-transitions representing the possibility to miss events. The VM will choose appropriate values of durations or symbols to emit for resolving this non-determinism. These choices correspond to the "on-the-fly" generation of an input trace t_{in}. As an example we present the first strategy of choices implemented in our framework.

Assume that the VM is in state $s = \langle t, n, \Gamma, pc, \Theta \rangle$ and that an (emit) move

can be applied. More precisely, assume that the running location $\Gamma[pc] = \langle \ell, \gamma, \beta \rangle$, where ℓ is a location of $\mathcal{E}$, and that the branch at ℓ contains several *emit*-transitions for symbols $e_1, \ldots, e_n \in IS$ (remember that $\Sigma^{\mathsf{sig}}_{\mathsf{out}} = IS$ in the construction of $\mathcal{E}$ – see Section 3.3). Then the VM chooses randomly one of $e_1, \ldots, e_n$ and applies an (emit) move with it.

Assume that a (delay) move can be applied to the state $s = \langle t, n, \Gamma, pc, \Theta \rangle$ and the running location $\Gamma_{pc} = \langle \ell, \gamma, \beta \rangle$. The VM will choose randomly a duration δ following the conditions of the (delay) move. More precisely we compute the bounds δ_{min} and δ_{max} using the bounds in the transition $\ell \xrightarrow{[d_{min}, d_{max}]} \ell'$ as follows: $\delta_{min} = d_{min} - \gamma$ and $\delta_{max} = d_{max} - \gamma$. Then it chooses randomly δ in $[\delta_{min}, \delta_{max}]$ satisfying the conditions (2) and (3) of the (delay) move. To prevent from the choice of a systematic δ_{max} value, we randomly choose the expiration of the *wait*-transition when $\delta_{min} \leq 0$.

Note the two above cases are the only two possible cases of non-determinism, according the definition of IR moves in Section 3.1.

Adapter The adapter catches all the output messages sent by the Virtual Machine and Antescofo. More precisely, for the VM, the adapter catches the moves (emit), (send) and (delay). When an event $\sigma \in IS$ is emitted by the VM, a corresponding stimulus is sent to Antescofo. For the emission of an action $\sigma \in OM$, the adapter stores σ with the value of the current logical instant t. When the time flows with a (delay) move, the corresponding duration is spent in Antescofo. Note that, like in the offline framework, a *virtual clock* is used in order to perform the tests in a fast forward mode and not in real time.

Comparison The comparison is done on-the-fly, when an action emitted by the VM or the IUT Antescofo is received and stored in the adapter. In order to detect missed and unexpected actions, a check is done after each logical instant (when the move (delay) is executed by the VM). After each comparison, the actions concerned are deleted from the store. During the on-the-fly generation, the test procedure continues if no error is detected and stops at the first error found. Moreover a third choice is possible during the generation consisting in ending the test with a verdict: "pass".

5 Experiments/Result

We present in this section some experiments with our framework whose purpose are to evaluate its effectiveness and report the pros and cons of the different approaches. We first recall the context before specifying the goal and the organization we chose to present this section.

We want to measure a black-box testing framework and thus assume that we have no feedbacks in the coverage of the Implementation Under Test's line codes and specially don't know the erroneous lines (even if we worked with the developer team for each error our framework raised). To measure the effectiveness of our framework, we take as a metric of effectiveness:

- the size of the input score, to test if our framework is scalable for real cases,
- the coverage (in the model IR) of the input traces set.

The first criteria ensures a large use via the possibility to test real mixed-scores and the second criteria allows us to fulfill the rtioco conformance [16, 23] which is the most important goal for a Model-Based-Testing method.

Finally, because it is a young framework and is applied to a constantly improving system, the accuracy of raised errors is sometimes difficult to assess. However the errors raised by the framework are time-errors since they come from a wrong output or a wrong timing of an output. Actually we reported errors which came from a concurrency problem which disturbed the scheduling of the outputs, a communication/synchronization problem regarding the inputs/outputs, a wrong time computation (from the tempo updating function) and a wrong management of the specified group's attributes. Moreover several of these errors happened for non trivial input cases, making them hard to find by other means.

Following our goal to measure the coverage and the scalability of the framework, we assess the methods implemented in a chronologic manner. We test the CoVer generation method and the fuzzing generation to compare their pros and cons *wrt* the scalability and coverage measures. We finally present our early online testing framework and compare its first outcome to the offline results.

The results were obtained with a MacBook Pro Retina with a 2.3 GHz Intel Core i7 and 16Go 1600 MHz DDR3 of Memory. The laptop ran on the Yosemite version of MAC OS X (10.10.4).

We have considered two case studies in out experiments:

1. a benchmark made of hundreds of little mixed scores, covering many features of the IUT's DSL
2. a real mixed score of the piece of *Sonata in F major, HWV 369 third movement: Alla Siciliana* by *Georg Friedrich Händel* [2].

The first benchmark is useful for the development (debugging and regression tests) of the system Antescofo. It aims at covering the functionality of the system's DSL and checking the reactions of the IMS. The second is a long real test case, for evaluating the scalability of our test method. Its total size is 1018 events and 3237 actions gathered in a big group in order to do automatic accompaniment by sending MIDI notes. This second case study is split into five extracts: the first 5^{th} bars (25 events and 84 actions), 8^{th} bars (48-185), 10^{th} bars (74-264), 15^{th} (122-444) and 40^{th} bars (360-1218).

Each case study is processed with various values for n_{err} and κ (the numbers of possible consecutive missed events and the bound on the variation of event's

[2] You can have a quick representation of the piece (with a description (in French) of Antescofo) here:
https://interstices.info/jcms/c_17524/interaction-musicale-en-temps-reel-entre-musiciens-et-ordinateur

durations). During the next results, we used the VM developed for online testing in order to evaluate the covertness of the sets of input traces generated for each experiment.

κ (%) \ n_{err}	0	1	3	5	7
0	18,231 - 051	19,402 - 107	23,377 - 164	26,424 - 313	26,443 - 307
1	18,231 - 025	19,402 - 076	23,377 - 137	26,424 - 319	26,443 - 318
3	18,111 - 024	19,402 - 077	23,377 - 137	26,424 - 318	26,443 - 314
5	18,231 - 028	19,402 - 077	23,377 - 132	26,424 - 312	26,443 - 309
10	18,231 - 035	19,402 - 082	23,377 - 140	26,424 - 320	26,443 - 323
25	18,231 - 059	19,402 - 086	23,377 - 135	26,424 - 334	26,443 - 328
50	18,231 - 103	19,402 - 132	23,377 - 160	26,424 - 354	26,443 - 352

Table 1: CoVer on the Benchmark: The table depicts the total size in number of state of the IR model and the time in seconds to perform the whole script.

κ (%) \ n_{err}	0	1	3	5	7
0	754 - 67.78%	1674 - 82.33%	2781 - 87.95%	3271 - 87.60%	3262 - 87.28%
1	612 - 67.79%	1471 - 81.56%	2615 - 87.75%	3183 - 88.16%	3152 - 87.44%
3	609 - 67.75%	1456 - 80.42%	2635 - 87.94%	3160 - 87.05%	3162 - 87.41%
5	613 - 67.97%	1468 - 81.54%	2633 - 87.81%	3140 - 87.07%	3124 - 86.06%
10	715 - 68.01%	1513 - 81.51%	2681 - 87.90%	3191 - 87.37%	3201 - 87.56%
25	994 - 68.18%	1720 - 82.36%	2691 - 87.60%	3243 - 88.29%	3277 - 88.02%
50	1623 - 69.22%	2301 - 83.41%	3006 - 88.82%	3577 - 88.34%	3553 - 88.54%

Table 2: CoVer on the Benchmark: The table depicts the number of t_{in} generated and the rate of coverage.

n_{err}-κ (%) \ bars	5	8	10	15	40
0-00	1 - 43.59%	1 - 38.92%	1 - 39.01%	1 - 38.72%	1 - 38.72%
0-10	38 - 43.59%	74 - 38.92%	117 - 39.01%	262 - 38.72%	×
0-25	95 - 43.59%	201 - 38.92%	427 - 39.01%	×	×
3-00	84 - 66.66%	130 - 86.48%	×	×	×
3-10	85 - 66.66%	148 - 86.48%	×	×	×
3-25	94 - 66.66%	159 - 86.48%	×	×	×
7-00	113 - 96.94%	×	×	×	×
7-10	133 - 96.94%	×	×	×	×
7-25	147 - 96.94%	×	×	×	×

Table 3: CoVer on the Real-Case: Number of t_{in} generated and their coverage.

5.1 Covering generation

We evaluated the generation of test data with Uppaal/CoVer following the offline method presented in Section 4.2, *i.e.* with a script which creates the IR models, translates them into networks of TA, generates test suites using CoVer, executes them according to *the first* scenario presented Section 4.2 and compares the outcome to test cases.

Tables 1 and 2 report the results with different environment options for all the scores in the benchmark. The first table details the total size of the

model part $\mathcal{S}$ in number of IR-states and the total time to execute the whole benchmark. The second presents the number of input traces generated by CoVer and the rate of coverage on $\mathcal{S}$ for the same scenarios as the first table.

The same script was ran for the extracts of the real mixed-score and the results are reported Table 3. The table depicts the number of input traces generated with their total coverage for each extract (characterized by its number of bars). The size of the IR-model $\mathcal{S}$ is 328, 697, 1005, 1678 and 4668 states for respectively the extract of 5, 8, 10, 15 and 40 bars of the mixed-score. In Table 3, the crosses depict a failure in the generation of input traces, because no output was returned or because a crash happened during one of the script steps.

Feedbacks The advantages of the covering test suites generation of CoVer is obvious for the first case study which contains a lot of small-sized mixed-scores that is perfect in such a case. Moreover the time is correct since the scripts spent 352 seconds to generate and test 3553 input traces (an average of 10 seconds per input trace) with a good coverage on 88.5% of the model. However the inconvenient are also multiples. We have not a clear control on the coverage according to the environment parameters. Here, the possibility of missing one more event improves more the coverage than allowing more interpretation on the durations. The real case shows clearly the lack of scalability where a score of more than 10 bars (74 events and 264 actions) cannot be tested with missed events which is unfortunately the parameter the most covering.

The CoVer generation is good for toy-examples where the mixed scores are written in a purpose of debugging but cannot be satisfying for real cases. Moreover (and not shown here) the input traces are commonly generated with the lower values in their durations (because of the guards in the model). That is not musically relevant since when converting musical time into physical time, having an input trace with shortest delays may result in a geometric progression of the tempo inferred by Antescofo, leading to exponential accelerations and unrealistic tempo values (like more than 300 *bpm*). These weaknesses encouraged us to explore other approaches for test data generation and execution as the scenario 1 of the Section 4.2 (tempo generated) which can be used to circumvent this problem.

5.2 Fuzzing generation

In the case of the fuzzing generation, the script creates the IR models without environment model $\mathcal{E}$, fuzzes the ideal trace to create a set of input traces, translates the IR models, with a specific $\mathcal{E}$ for each input trace, into networks of TA, simulates the TA using Verifyta (Uppaal model checker command tool) to compute the expected traces, executes the input traces according to the first scenario presented in Section 4.2 and compares the outcome to test cases.

The same values are depicted Table 4 for the real mixed-score. We do not report time since we can't compare a script doing an input generation against a random fuzzing one (to have an idea the last case (7-25) lasted 750 seconds for

CoVer on the five bars extract, against 97 seconds for a test of one input trace in the same extract using Verifyta).

The advantage of the fuzzing generation is the little deformations of the ideal trace, that keeps the input traces musically relevant. Moreover in a musical point of view, we think that a little interpretation is sufficient to consider latter, earlier or missed cases (*i.e.* cover all the performance cases). The method is fast and can manage huge mixed-scores that is good for real cases. However since the fuzz is done randomly we have no control on the coverage which is low for a set of input traces.

n_{err}-κ (%) \ bars	5	8	10	15	40
0-00	1 - 38.81%	1 - 14.67%	1 - 38.27%	1 - 38.05%	1 - 38.02%
0-10	10 - 38.81%	10 - 14.67%	10 - 38.27%	10 - 38.05%	10 - 38.04%
0-25	10 - 38.81%	10 - 14.72%	10 - 38.27%	10 - 38.11%	10 - 38.02%
3-00	10 - 37.65%	10 - 21.15%	10 - 34.21%	10 - 28.72%	10 - 23.21%
3-10	10 - 41.91%	10 - 34.69%	10 - 32.75%	10 - 32.59%	10 - 27.91%
3-25	10 - 28.05%	10 - 28.83%	10 - 28.48%	10 - 27.85%	10 - 25.89%
7-00	10 - 17.03%	10 - 17.21%	10 - 17.26%	10 - 16.37%	10 - 15.60%
7-10	10 - 17.68%	10 - 17.36%	10 - 16.76%	10 - 16.21%	10 - 15.90%
7-25	10 - 18.01%	10 - 18.09%	10 - 16.55%	10 - 16.53%	10 - 15.66%

Table 4: Fuzz on the Real-Case: Number of t_{in} generated - coverage according to each extract and different environment restrictions ($n_{err} - \kappa$).

number of t_{in}	1	10	100	1000
percent of coverage	12.72%	15.78%	21.95%	32.35%

Table 5: Fuzz: Number of t_{in} generated for 40 bars of the real mixed score with 7-25 values.

5.3 Evaluation of the coverage of fuzzing generation

Random test is an important strategy for the test input generations and is widely used in the state-of-the-art. However, it lacks of precision since no control is possible in the randomized values. We add an experiment to evaluate the covertness of our fuzzing script according to the number of input traces generated with 7-25 values for the parameters n_{err} and κ respectively.

We present on Table 5 the results on the first 40^{th} bars of the mixed score of the Sonate used as the second case studies (see page 44). The rise of the generated trace number improves as expected the coverage, but it is still very low even for a thousand of traces (that lasts as long as a CoVer generation). This experiment confirms that this second generation cannot be covering and motivates for another strategy or targets addition for guiding the fuzz algorithm.

number of t_{in}	10	50	100
percent of coverage	59.32%	62.09%	62.09%
time in seconds	24	114	249

Table 6: Online: Number of t_{in} generated for the all mixed score (18,641 model's states) and with the generation described Section 4.3.

5.4 Online testing with VM

Finally we report Table 6 an evaluation of our online testing approach. For the online experiment, we deployed a script running the model simulation and the IUT Antescofo at the same time on the machine, the two softwares communicating via the protocol Open Sound Control (OSC). The simulation constructs the model and simulates it via the method and the algorithm detailed Section 4.3. The IUT is run with an online adaptor which waits an input stimulation (an event or a duration) from the model. Remark that although the method is online, we execute it in a fast-forward fashion, preventing from waiting for the real durations.

The online framework is promising. Our first experiments succeeded in managing the entire real mixed score (study 2) and performed a hundred of input traces for 4 minutes. We are working on improving the online algorithm *wrt* covertness of the test suite generation. It appears after evaluation that with the first version algorithm, a hundred of inputs covers no more locations than fifty inputs. We are working on improving the distribution of input events, using constraint solving techniques like in SAGE [10, 21].

5.5 Discussion

Finally we have two complementary test data generation methods, the former, offline, ensuring a good quality of the test data via covering features, the latter, online, scaling to big scores and the musical context of the generated traces. Comparing these two methods is tedious since the advantage of the first is the weakness of the second, (*i.e.* its covering feature). However the fuzzing generation permits to manage real cases and is faster than the first generation. Merging the two techniques is an interesting future work that can add a covering feature in the fuzzing generation in, for example, targeting the fuzz in the action-trigger-events.

Notice that the use of the standard TA model forces us to convert durations with multiple time units (in particular the musical time unit in input traces t_{in}) into durations with a unique time unit (the Model Time Unit (*mtu*) used in Uppaal). In our framework, we consider that the mtu for TA is the musical time. It is the reason of our problem with the fact that CoVer tries to generate systematically optimal test suites. This translation implies drastic restrictions ($\mathsf{R}_1 - \mathsf{R}_3$ in Section 3.4) on the model and so restrict the possible mixed scores the framework can test.

The development of a VM made possible our online framework for MBT. At the opposite of the offline method, this framework is based directly on the executable IR model (without translation into TA), and a maximum of coverage by the generated input traces can be ensured. The first experiment is promising both on the scalability and the speed of the testing method. However as for the fuzzing generation, the algorithm has to improve its covering abilities. An interesting idea in the online generation method would be to compute an input trace with duration intervals ensuring the same behavior in $\mathcal{S}$. From this, relevant input suites could be generated with an objective of exhaustiveness.

6 Conclusion

We have developed a fully automatic Model-Based Testing framework dedicated to a Realtime Interactive Music System for automatic accompaniment, with offline and online procedures for the generation of covering test cases. This framework is based on a dedicated Intermediate Representation for modeling the implementation under test and its environment (*i.e.* the human musicians accompanied). This IR was designed to model easily the semantics of Antescofo. It borrows both from the Timed Automata model and the logic time semantics of the synchronous programming languages for reactive systems Esterel [7].

One originality of the case study of IMS for MBT is that the models are constructed automatically from the mixed scores (high-level requirements) followed by the IMS, instead of being written manually by an expert. Moreover, the models in IR can be converted (under restrictions) into Timed Automata for using tools of the Uppaal suite. One technical difficulty is the necessity to deal with the constraint of real time with different time units, in particular the musical time relative to a tempo. This prevented us from using the online testing tool Tron out of the box for our case study. Hence, we implemented our own online MBT framework using a Virtual Machine interpreting directly the IR without any restrictions. This newly method is yet promising since an entire real mixed score passed successfully a first experiment using a non trivial "on-the-fly" generation algorithm.

Besides the case of IMS, our approach could be applied to the test of other realtime reactive systems involving pre-specified temporal scenarios, feedbacks and timed interaction with humans. That includes in particular video games and other entertainment systems, robotics, smart buildings or smart cities... and more generally, cyber-physical systems coupling computing devices with physical components and humans in the loop.

The offline test generation approach based on CoVer is a good first step into applying some existing MBT tools to our case study, with a purpose of exhaustiveness. It has however some limitations and cannot be considered as the best way to generate our input traces (e.g restriction to shortest delays t_{in} that causes exponential accelerations, not scalable) presented in Section 5. A way to bypass these problems could be to re-implement (in Uppaal) the algorithm

of CoVer with random choices of delays insides regions, instead of the systematic choice of the shortest delays.

We are planing to extend the techniques of test cases generation by fuzzing an ideal trace presented in Section 4.2 with a notion of coverage of the IR model $\mathcal{S}$ similar to the approach of CoVer. The Virtual Machine can indeed compute the coverage of the IR model $\mathcal{S}$ for a given set of t_{in} (in the case of offline generation). This information could be very useful in order to assess the quality of a set of input traces independently generated from the offline generation (*e.g.* using the model of performances).

Our method is designed to test the behavior of the IMS on one given score, by generating 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 the fuzz generation 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 reviewers at ACM-SAC for their useful comments, and Antoine Rollet, Alexandre David and the Uppaal team for their help.

References

[1] Jaffe, D. A. Ensemble Timing in Computer Music In *proc. International Computer Music Conference (ICMC)*. 1983.

[2] Dannenberg, R. Music Representation: A Position Paper. 1989.

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

[4] Anderson, D. P., and Kuivila, R. A system for computer music performance. *ACM Transactions on Computer Systems*, 8(1):56–82, 1990.

[5] Arias, J., Desainte-Catherine, M., and Rueda, C. A Framework for Composition, Verification and Real-Time Performance of Multimedia Interactive Scenarios. In *15th International Conference on Application of Concurrency to System Design*. 2015.

[6] Bazin, T. and Dubnov, S. Constrained Music Generation Using Model-Checking, In *Proceedings of Journées d'Informatique Musicale*. 2016

[7] Berry, G. The constructive semantics of pure Esterel, 1996.

[8] Bliudze, S., and Sifakis, J. A notion of glue expressiveness for component-based systems. In *proc. of the 19th International Conference on Concurrency Theory* (CONCUR) 508–522. Springer, 2008.

[9] Bliudze, S., and Sifakis, J. Synthesizing Glue Operators from Glue Constraints for the Construction of Component-Based Systems. In *proc. 10th International Conference on Software Composition* (SC) 51–67. Springer 2011,

[10] Bounimova, E., Godefroid, P., and Molnar, D. Billions and billions of constraints: Whitebox fuzz testing in production. Tech. Rep. MSR-TR-2012-55, Microsoft Research, 2012.

[11] Bowman, H., Faconti, G., Katoen, J.-P., Latella, D., and Massink, M. Automatic verification of a lip synchronisation algorithm using Uppaal. In *proc. 3rd International Workshop on Formal Methods for Industrial Critical Systems*, 97–124. 1998.

[12] Colombo, C., Micallef, M., and Scerri, M. Verifying web applications: From business level specifications to automated model-based testing. In *proc. 9th Workshop on Model-Based Testing*, 14–28. 2014.

[13] Cont, A. A coupled duration-focused architecture for realtime music to score alignment. *IEEE Transaction on Pattern Analysis and Machine Intelligence* 32(6):974–987. 2010.

[14] Cont, A., Echeveste, J., Giavitto, J.-L., and Jacquemard, F. Correct Automatic Accompaniment Despite Machine Listening or Human Errors in Antescofo. In *proc. International Computer Music Conference* (ICMC) 2012.

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

[16] David, A., Larsen, K. G., Li, S., Mikucionis, M., and Nielsen, B. Testing real-time systems under uncertainty. In *9th International Symposium on Formal Methods for Components and Objects (FMCO)* vol. 6957 of LNCS, 352–371, Springer, 2010.

[17] Echeveste, J. Un language de programmation pour composer l'interaction musicale. PhD thesis, UPMC, 2015.

[18] Echeveste, J., Cont, A., Giavitto, J.-L., and Jacquemard, F. Operational semantics of a domain specific language for real time musician–computer interaction. *Discrete Event Dynamic Systems*, 23(4)343–383. 2011.

[19] Fanchon, L. and Jacquemard, F. Formal Timing Analysis Of Mixed Music Scores In *proc. International Computer Music Conference* (ICMC), 2013.

[20] Ghosal, A., Henzinger, T. A., Kirsh, C. M., and Sanvido, M. A. A. Event-Driven Programming with Logical Execution Times. in *Hybrid Systems: Computation and Control*. vol. 2993 of Springer LNCS 357-371. 2004.

[21] Godefroid, P., Levin, M. Y., and Molnar, D. A. Automated whitebox fuzz testing. In *Network Distributed Security Symposium (NDSS)*. 2008.

[22] Henzinger, T., Horowitz, B., and Kirsch, C. Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84– 99. 2003.

[23] Hessel, A., Larsen, K. G., Mikucionis, M., Nielsen, B., Pettersson, P., and Skou, A. Testing real-time systems using Uppaal. In *proc. Formal methods and testing*, 77–117. Springer, 2008.

[24] Honing, H. From time to time: The representation of timing and tempo. *Computer Music Journal*, 25(3):50–61. 2001.

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

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

[27] Memon, A. M. An event-flow model of gui-based applications for testing. *Softw. Test., Verif. Reliab. 17*, 3:137–157. 2007.

[28] Peters, N., Lossius, T., and Place, T. An automated testing suite for computer music environments. In *9th Sound and Music Computing Conference (SMC)*. 2012

[29] Poncelet, C., and Jacquemard, F. Model Based Testing of an Interactive Music System. In *proc. 30th ACM/SIGAPP Symposium On Applied Computing* (ACM SAC) 2015.

[30] Poncelet Sanchez, C., and Jacquemard, F. Test Methods for Score-Based Interactive Music Systems. In *ICMC/SMC* 2014.

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

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

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

A Antescofo's score

We present here the actual Antescofo mixed score for the running example. The textual Antescofo's DSL has keywords like bpm to give the expected tempo, **note** to specify an event detection (with its pitch, its duration and its label) and group to specify a sequence of actions (with its delay, its name and its attributes).

```
bpm     120
note    D♯5   1       e1_0.00
        0.0   group   s1          @loose   @local
        {
              0.0     on_t1       @name    on1
              0.5     off_t1      @name    off1
              1.0     group s3
              {
                                  0.0      on_t3    @name on3
                                  0.25     off_t3   @name off3
              }
        }
note    A4    0.5     e2_1.00
        0.0   group   s2          @tight   @global
        {
              0.0     on_t2       @name    on2
              0.75    off_t2      @name    off2
        }
note    C♯4   0.5     e3_1.50
```