Timed Automata for Video Games and Interaction

Jaime Arias, Raphael Marczak, Myriam Desainte-Catherine

Timed Automata for Video Games and Interaction

Jaime Arias, Raphaël Marczak and Myriam Desainte-Catherine

Synonyms

Formal Methods, Interactive Multimedia Scenarios, Player Experience, Serious Games

Definition

Timed Automata is a formalism for modelling and verification of time-critical systems. It has been proven to be a formalism that is well adapted to the expression of the timing constraints appearing in interactive scores and video games because it is a powerful model for describing both the logical ordering of the events in such scenario and also the duration of events and the timing between them.

Introduction

As coined by Espen Aarseth in his very first editorial launching the "game studies" journal (considered as one of the core events establishing "game studies" as an aca-

Jaime Arias
Université Paris 13, Sorbonne Paris Cité, CNRS, LIPN, UMR7030, F-93430 Villetaneuse, France, e-mail: arias@lipn.univ-paris13.fr

Raphaël Marczak
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France, e-mail: raphael.marczak@u-bordeaux.fr

Myriam Desainte-Catherine
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France, e-mail: myriam.desainte-catherine@labri.fr

demic field), games are "*both object and process [which] must be played. Playing is integral, not coincidental [. . .]. The complex nature of simulations is such that a result cannot be predicted beforehand; it can vary greatly depending on the players luck, skill and creativity.*" (Aarseth, 2001). Activating a video game requires a processing of player(s) inputs, directly impacting the game system evolution, both **temporally** (unfolding the game narrative and ludic progression) and **conditionally** (tailoring the game accordingly to the player behavior and the gameplay elements reactions).

Segmenting play Since early ("*vintage*") arcade games, temporal and conditional (*i.e.,* challenge) segmentation of gameplay have been considered as core perspectives to simultaneously design and analyze video games (Zagal et al., 2008). In games, time is employed as a resource (divisions, temporal limit, etc.) and/or a coordination mechanism (rounds, turn-taking, narrative key moments, etc.) (Zagal et al., 2008, p. 180). Challenges, on their sides, condition the way player(s) can interact with a game system (a change in challenge is, in general, accompanied by a change of allowed activations). Another segmentation identified by Zagal et al. is the *spatial* one, which is the "*division of the gameworld into different spaces that also partition the gameplay*" (Zagal et al., 2008, p. 182). The spatial segmentation is tied to the creative process of level designing, itself tied to the notion of **hierarchy**. Indeed, a *level* can be seen as a spatial segment containing specific *gameplay elements*, themselves interactable regarding the current *degree of freedom* enabled to the player. It is noteworthy that other game studies research, such as the six dimensions of implication identified by Gordon Calleja (Calleja, 2011), also regularly encompass time (narration, emotion), condition (ludic, kinesthesic, social) and space/hierarchy (strategy).

Hierarchy in games Segmenting a video game hierarchically has also been suggested by Raphaël Marczak as part of the *Gameplay Performance Segmentation model* (Marczak, 2014, chapter 4), which defines five hierarchical layers of analysis, toward the understanding of how a specific player actually engaged with a specific game system, contextualizing each interaction from within the whole game system: the game system layer represents to top level, then including different game instances, themselves separated in different time and space segments, containing different phases conditioning the level of freedom offered to a player at a given time, to the bottom level being the actual interaction performed by the player (see Fig. 1).

Gameplay elements themselves are hierarchically related, as emphasized by the *Game Ontology Project* (Zagal et al., 2005), classifying gameplay concepts by affiliations and groups of dependencies. For example, a *head up display* in a first person shooter game is part of *visual output*, itself part of *sensory output*, part of *presentation*, and finally part of the top theme *interface*.

Example An example illustrating the three segmentation axis emphasized above (time, condition and hierarchy) can be seen on Fig. 2, displaying a boss fight in the game *The Legend of Zelda: The Windwaker HD* (Nintendo, 2013). In terms of

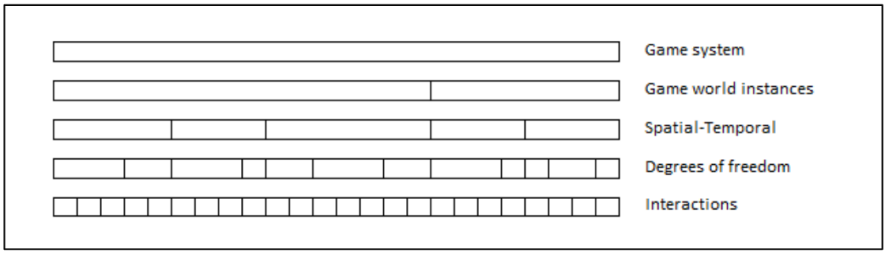

Fig. 1 The five hierarchical layers of the Gameplay Performance Segmentation model (Marczak, 2014, chapter 4)

hierarchy, this boss fight happens in an enclosed arena, in which the player's avatar (Link) can find specific items in relation with the fight (mostly life-regenerating items), and the actual boss (itself subdivided into three main components: two hands and a head). *Time-wise*, the boss behavior follows a pattern during which it opens and closes its hands, mouth and eyes, and moves within the arena. This pattern has to be learned by the player in order to correctly defeat the boss (shooting its eyes and mouth). In terms of *conditions*, the player must switch between weapons (arrow, bomb, etc.) in order to react accordingly to the boss current pattern. Each weapon has its own control and impacts the game space differently. This simple example illustrates how these three axis work concurrently.

Another example of time, condition and hierarchy segmentation can be read in Marlène Dulaurans and Raphaël Marczak paper about the game *Clash Royal* (Dulaurans and Marczak, 2018).

Game engines, such as *Unreal Engine*, *Unity3D*, *Godot*, *Game Make Studio* (to name a few) are now widely available to game designers, developers and the general public, for supporting game making whether in companies or for independent creatives. Time, condition and hierarchy are all encompassed in these engines, but as separated elements, and rarely displayed all-together in the same interface and representation. In *Unity3D* for example, hierarchy is linked with graphical entities (a gameobject can be a child of another gameobject); condition is part of the C#/javascript scripting feature, or for animators; and time is managed by the game engine scheduler (calling update functions each frame) or by the scripting ability to change level/scene. Other engines, like *Unreal Engine*, *Godot* or *Game Make Studio* also have a blue-print feature, making visual the conditional possibilities.

These three main features constituting a gameplay are included in all game engines software, **but they are not part of a general scenario/representation structure, able to, at the same time, model time, condition and hierarchy**. For that reason, during the last few years, a promising interactive score system called *i-score* (`https://ossia.io/`) has been emerging with the main objective of blending together the above three features for performing art and video games. This software can be seen as a (1) *conductor* (in the musical sense), scheduling media with the flexible time concept, and as a (2) *Master Hand* (in the role playing sense), structuring a game design both temporally, conditionally and hierarchically.

Fig. 2 A boss in Zelda The Windwaker HD (Nintendo, 2013), illustrating the three main segmentation axis: time, condition and hierarchy

Interactive Scores with i-score

i-score provides a theoretical framework and a graphical application for the authoring and execution of interactive scores. Interactive scores consist of a temporal and hierarchical organization of events that allow users to describe interactions of the performer during the performance of the score. These interactions allow the possibility for a score to have more than one different possible performances.

The graphical model for i-score was introduced in (Celerier et al., 2015). This model is based on hierarchy of graphical elements whose semantics allows to describe the flow of time. Moreover, it is adapted for the authoring of automation and processes which are happening over a duration of time, either fixed or dependent on interactive events (see Fig. 3). i-score is closer in mind to sequencers like Reaper®, Ableton Live® or Avid Pro Tools®, than music notation software such as Finale®, Sibelius® or Bach®. For instance, Fig. 3 shows an i-score score controlling over time the brightness, the intensity and the colors of a light projected on a surface in order to play with the spectators' perception of time and lead them to a more thoughtful mental state.

In general, interactive scores are described by powerful expressive models similar to full-fledged programming languages, *e.g.,* the programming language for real-time and interactive computer music *Chuck* (Wang et al., 2015). The above has the main consequence of putting the users in front of the common problems that occur during the activity of programming, leading to undesired behaviors during

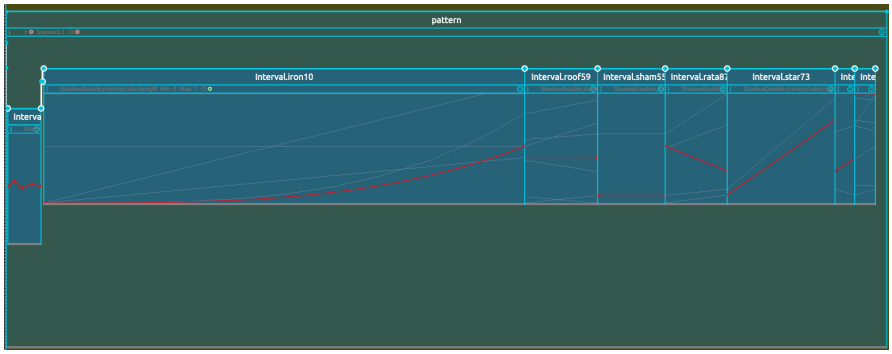

Fig. 3 Interactive score controlling over time a light projected on a surface in order to play with the spectators' perception of time and lead them to a more thoughtful mental state. Score created by The Baltazars (`http://www.baltazars.org/`)

the performance: bugs, data races and specification problems. Hence, the need for advance verification techniques for interactive score software becomes necessary.

Multiple families of formal methods, such as Petri nets (Desainte-Catherine et al., 2013; Arias et al., 2014), process calculi (Toro et al., 2014; Olarte and Rueda, 2009) and linear logic (Arias et al., 2015a) have been proposed to give formal semantics to i-score, with the goal of performing static verification on the score in order to prevent unwanted situations during the performance, However, the proposed models do not support (1) flexible control structures such as *conditionals*; and (2) practical mechanisms for the *automatic verification* of scores.

In 2015, a novel model of i-score using Timed Automata was defined (Arias et al., 2015b). This model expresses, in the same framework, the main features of interactive scores (*i.e.,* time, conditions and hierarchy). Moreover, it allows to verify the written scores automatically. Timed Automata has been proven to be a formalism that is well adapted to the expression of the timing constraints appearing in an interactive score following system (Sanchez and Jacquemard, 2016; Echeveste et al., 2013; Jacquemard and Poncelet, 2016) because it is a powerful model for describing both the logical ordering of the events in such scenario and also the duration of events and the timing between them.

In the next, a formal definition of Timed Automata model is presented and also some important properties of interactive scores that can now be proven.

Formal Verification of Interactive Scores in i-score

Timed Automata (Alur and Dill, 1994) is a formalism for modelling and verification of time-critical systems. A timed automaton (see Fig. 4) is a finite-state machine with a finite set of real-valued variables modelling logical *clocks*. A *transition* in a timed automaton (represented by an edge) is labelled with a *guard* (*i.e.,* when is it

allowed to take an edge?), an *action* (*i.e.,* what is performed when taking the edge) and a set of clocks (*i.e.,* which clocks must be reset). A *location* (represented by a node) is equipped with an *invariant* that constrains the amount of time that may be spent in that location. In that sense, invariants ensure the progress of the model while guards restrict the behavior of the timed automaton. Both invariants and guards are *clock constraints* that are formally defined in Definition 1.

Definition 1. A *clock constraint* δ over a set C of clocks is defined inductively by

$$\delta ::= x \sim n \mid \delta_1 \wedge \delta_2 \mid \text{true}$$

where $n \in \mathbb{N}_0$, $\sim \in \{=, <, >, \leq, \geq\}$, and $x \in C$. Let $\Phi(C)$ denote the set of clock constraints over C, and $\Phi_{\leqslant}(C)$ the set of *downward closed* clock constraints of the form $x \leq n$ and $x < n$.

The formal definition of a timed automaton is as follows.

Definition 2. A *timed automaton* is a tuple $\langle L, l_0, \Sigma, C, E, I \rangle$, where

- L is a finite set of *locations*,
- $l_0 \in L$ is the *initial* location,
- Σ is a finite alphabet denoting *actions*,
- C is a finite set of *clocks*,
- $E \subseteq L \times \Phi(C) \times \Sigma \times 2^C \times L$ is the set of transitions between locations, and
- $I : L \to \Phi_{\leqslant}(C)$ assigns invariants to locations.

Fig. 4 shows the timed automaton model of the intelligent light switch defined in (Fahrenberg et al., 2010). The initial state of the model is `off` that listens for the pressing of the button in order to turn the light on (*i.e.,* state `light`). Once the light is on, it waits for 100 time units (*i.e.,* until clock $x = 100$). If an additional press is done during this time, the clock x is reset and prolongs the time in the state by 100 time units, otherwise the light turns off again. Pressing the button twice, with at most three time units between the presses, triggers a special bright light.

The Timed Automata model for i-score scores was formalized in (Arias et al., 2017, 2015b). Currently, i-score is equipped with an automatic tool for translating scores into their equivalent Timed Automata model. It covers a strict subset of i-score because the latter is based on a plug-in architecture which is easily extensible. For instance, a plug-in allowing to run arbitrary Javascript code.

The verification of scores is carried out using the UPPAAL (David et al., 2015) model-checker, which allows for the verification of networks of Timed Automata using the method of model-checking (Clarke et al., 2018). Roughly speaking, model-checking is a computer-assisted method for the analysis of systems that explores exhaustively all possible system states in a systematic manner in order to check whether a model of the system satisfies a given property.

Once the Timed Automata model of the score is generated by i-score, it is possible to check the following useful properties with UPPAAL:

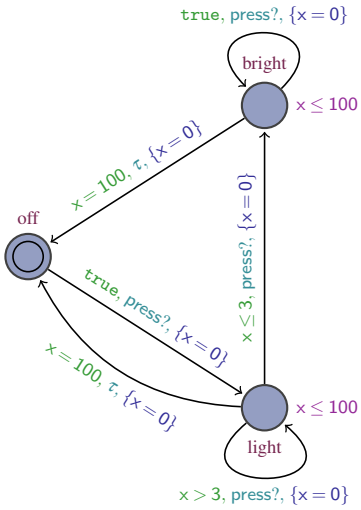

Fig. 4 A light switch modelled as a timed automaton (Fahrenberg et al., 2010).

- The temporal exclusivity of two multimedia processes P_1 and P_2. That means that in all possible executions of a score, always both P_1 and P_2 are not executed at the same time. This is important, for instance, if a multimedia device can only handle one automation at the same time.
- The unreachability of an *error* state. That is, in all possible executions of the score, all the multimedia processes never reach an error state (*i.e.,* all the temporal constraints of the system are always satisfied).
- It is possible to assert that given an annotation of the multimedia processes relative to the part they pertain to (*e.g.,* Intro, Chorus, Verse), the score's structure satisfies a specific order. For instance, the Intro is always followed by a Chorus.

Applications to Games and Interaction

In the following, two special applications of i-score in the domain of games and interaction are presented.

SEGMent

SEGMent (*Study and Education Game Maker and entertainment*) is a diagram-based game making language allowing anyone willing to create (digital) escape games to compose multimedia scenarios without any programming skill pre-required.

From the three main phases of game making process, being conception (brainstorming, story-boarding, proof of concept, etc.), implementation (technical programming of the game) and experience (gameplay metrics gathering (Drachen and Canossa, 2009) (Marczak et al., 2015) for analysis and understanding); SEGMent is handling the two intimidating ones from non computer scientist perspective (implementation and experience), so that anyone, from professors, students to game designer, can fully focus on the conception phase. The main objective of SEGMent is to help teachers create ludo-pedagogical experiences in relation to their discipline, in the form of an escape game (including still images, sounds, animations, videos, digicode/keypad behavior for user input, object states, etc.), by the simple drawing of a diagram. The main features of SEGMent are:

- Scenes background creation ;
- Objects addition in scenes (hierarchically) ;
- Click areas to go from one scene to another (hierarchically) ;
- Background atmospheric sound for scenes ;
- Possibility to interact with objects and change their states (condition) ;
- Object animation when changing state (time) ;
- Self-reflexive dialog the first time a user enters a scene (time) ;
- Possibility to interact with areas on screen to generate input (like digicode or keyboard) (condition)
- Possibility to move elements as puzzle pieces (condition)
- Possibility to go from one scene to another by entering an answer (manually or with generated input), or by having several objects in specific states (condition)
- Possibility to go directly to another scene after a specified amount of time (time)
- Possibility to go back to the previous scene (time)
- etc.

From SEGMent to SEGMent^2

SEGMent was originally designed as a proof of concept. The SEGMent-engine is implemented in Unity3D, and can be launched on MacOSX, Windows, Linux, Android and online (WebGL). The diagram editor is, for its part, based on *DIA Diagram Editor* (`http://dia-installer.de/`), where flow chart has been adapted for escape game making properties. DIA is a perfect diagram drawing tool, however it lacks the ability to easily add custom properties on state/transition objects, and hierarchy is not implemented for including an image (object) into another (scene). Editing a SEGMent scenario in DIA can be really cumbersome for a neophyte. Therefore, for a proof of concept, DIA has perfectly full-filled its job, but the need to have a fine management of time, condition and hierarchy forced to switch to another editor environment: i-score. This new engine is the main goal of SEGMent^2.

By using the plug-in management system included in i-score, it becomes possible to port all the above mention features of SEGMent within the i-score environment, thus taking advantage of the time, condition and hierarchy possibilities offered by

the system. As illustrated in Fig. 5, objects in scenes are now fully enclosed into them (hierarchically) and each element (scenes, objects, transitions, etc.) can have time and condition tied to it, as custom property. Furthermore, SEGMent being now in i-score means that it can be linked with other i-score plug-ins, and notably the ones about flexible time (useful in game scenario, where the player inputs condition the narrative unfolding) and about parameters automation. Finally, a SEGMent scenario can now be separated into several sub-level, easing the game and level designers tasks.

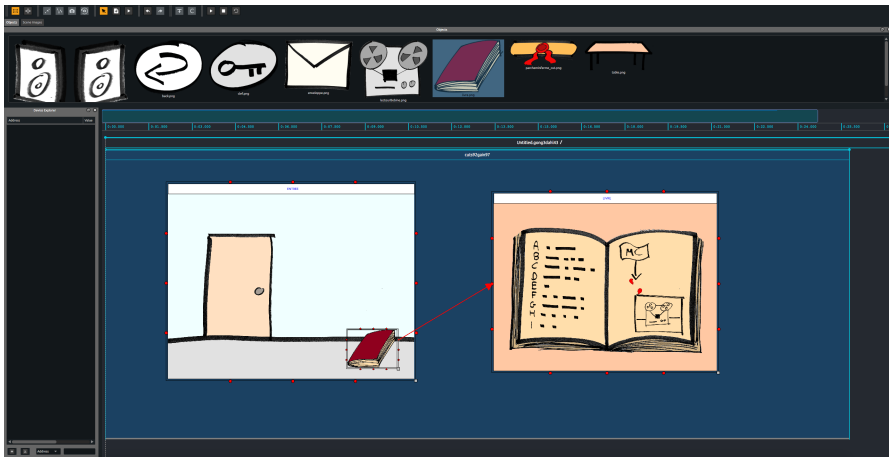

Fig. 5 Screen capture displaying the SEGMent^2 plugin within score

With SEGMent^2, it is now possible to say that i-score is not only a multimedia **conductor**, but also a **master hand**, able to guide the scenarization of video games.

VMO-Score

Machine improvisation refers to artificially augmented musical performances where machines contribute creatively to the musical outcome. Musicians use machine analysis tools to generate a computer accompaniment from musical examples. This allows the computer to generate music in ways that preserve the style of the musical examples given to it without a need for programming. During performance, these generative mechanisms are either controlled in real-time by another operator or are programmed to interact according to readily prepared scenarios.

Variable Markov Oracle (VMO) (Wang and Dubnov, 2015) is a machine improvisation method based on automatic analysis of the musical structure of a recording. A VMO is a data structure capable of identifying repeated subsequences within a

multivariate time series. After symbolizing the time series, repetitive structures can be extracted and used for music improvisation.

Currently, interactive music pieces require separate phases of constructing generative models and structuring them into a larger compositional plan. In (Arias et al., 2016), a new system is proposed in which the machine improvisation tools based on VMO is combined with i-score scores in order to control the improvisation according to larger structures found directly from a musical recording and real-time interactions defined by the user. This allows construction of improvisation scenarios in ways that (1) are organic with the musical materials used for generating the music, and (2) interact with the performer/environment during the improvisation.

Conclusion

We showed that the interactive score system *i-score* is able to specify scenarios featuring time, conditions and hierarchy, which are essentials for exhaustive interactive systems such as performing art and video-games. Moreover, we presented a formal model for i-score, showing that Timed Automata is a well-suited formalism for the modelling and verification of interactive systems where both logical and temporal constraints are needed for a rigorous interactive creation.

References

E. Aarseth. Computer game studies, year one. *Game Studies*, 1(1), 2001. URL `http://gamestudies.org/0101/editorial.html`.

R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. doi: 10.1016/0304-3975(94)90010-8.

J. Arias, M. Desainte-Catherine, and C. Rueda. Modelling data processing for interactive scores using coloured petri nets. In *14th International Conference on Application of Concurrency to System Design, ACSD 2014, Tunis La Marsa, Tunisia, June 23-27, 2014*, pages 186–195. IEEE Computer Society, 2014. doi: 10.1109/ACSD.2014.23.

J. Arias, M. Desainte-Catherine, C. Olarte, and C. Rueda. Foundations for reliable and flexible interactive multimedia scores. In T. Collins, D. Meredith, and A. Volk, editors, *Mathematics and Computation in Music - 5th International Conference, MCM 2015, London, UK, June 22-25, 2015, Proceedings*, volume 9110 of *Lecture Notes in Computer Science*, pages 29–41. Springer, 2015a. URL `https://doi.org/10.1007/978-3-319-20603-5_3`.

J. Arias, M. Desainte-Catherine, and C. Rueda. A framework for composition, verification and real-time performance of multimedia interactive scenarios. pages 140–151. IEEE, June 2015b. doi: 10.1109/ACSD.2015.8.

J. Arias, M. Desainte-Catherine, and S. Dubnov. Automatic construction of interactive machine improvisation scenarios from audio recordings. In *4th International Workshop on Musical Metacreation, MUME 2016*, Paris, France, 2016. ISBN 978-0-86491-397-5. URL `http://musicalmetacreation.org/buddydrive/file/arias_automatic_construction/`.

J. Arias, J.-M. Celerier, and M. Desainte-Catherine. Authoring and automatic verification of interactive multimedia scores. *Journal of New Music Research*, 46(1):15–33, Jan. 2017. doi: 10.1080/09298215.2016.1248444.

G. Calleja. *In-Game: From Immersion to Incorporation*. The MIT Press, 1st edition, 2011. ISBN 0262015463, 9780262015462.

J.-M. Celerier, P. Baltazar, C. Bossut, N. Vuaille, J.-M. Couturier, and M. Desainte-Catherine. OSSIA: towards a unified interface for scoring time and interaction. In *Proceedings of the First International Conference on Technologies for Music Notation and Representation, TENOR 2015*, pages 81–90, Paris, France, 2015. ISBN 978-2-9552905-0-7. URL `http://tenor2015.tenor-conference.org/papers/13-Celerier-OSSIA.pdf`.

E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018. doi: 10.1007/978-3-319-10575-8.

A. David, K. G. Larsen, A. Legay, M. Mikucionis, and D. B. Poulsen. Uppaal SMC tutorial. *STTT*, 17(4):397–415, 2015. doi: 10.1007/s10009-014-0361-y.

M. Desainte-Catherine, A. Allombert, and G. Assayag. Towards a hybrid temporal paradigm for musical composition and performance: The case of musical interpretation. *Computer Music Journal*, 37(2):61–72, 2013. doi: 10.1162/COMJ_a_00179.

A. Drachen and A. Canossa. Towards gameplay analysis via gameplay metrics. In *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*, MindTrek '09, pages 202–209, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-633-5. doi: 10.1145/1621841.1621878.

M. Dulaurans and R. Marczak. Quand le jeu vido devient affaire de clan : un clash royale entre incitation et inhibition. *Revue Franaise des Sciences de l'Information et de la Communication*, 13, 2018. URL `https://journals.openedition.org/rfsic/3610`.

J. Echeveste, A. Cont, J.-L. Giavitto, and F. Jacquemard. Operational semantics of a domain specific language for real time musiciancomputer interaction. *Discrete Event Dynamic Systems*, 23(4):343–383, Dec. 2013. doi: 10.1007/s10626-013-0166-2.

U. Fahrenberg, K. G. Larsen, and C. R. Thrane. Verification, performance analysis and controller synthesis for real-time systems. In *Fundamentals of Software Engineering*, volume 5961, pages 34–61. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi: 10.1007/978-3-642-11623-0_2.

F. Jacquemard and C. Poncelet. An automatic test framework for interactive music systems. *Journal of New Music Research*, 45(2):87–100, 2016. doi: 10.1080/09298215.2016.1173707.

R. Marczak. *Feedback-Based Gameplay Metrics and Gameplay Performance Segmentation: An audio-visual approach for assessing player experience.* PhD thesis, University of Waikato, Faculty of Arts and Social Sciences, 2014.

R. Marczak, G. Schott, and P. Hanna. Postprocessing gameplay metrics for gameplay performance segmentation based on audiovisual analysis. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(3):279–291, Sept 2015. doi: 10.1109/TCIAIG.2014.2382718.

C. Olarte and C. Rueda. A declarative language for dynamic multimedia interaction systems. In E. Chew, A. Childs, and C.-H. Chuan, editors, *Mathematics and Computation in Music*, pages 218–227, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-02394-1.

C. P. Sanchez and F. Jacquemard. Model-based testing for building reliable realtime interactive music systems. *Sci. Comput. Program.*, 132:143–172, 2016. doi: 10.1016/j.scico.2016.08.002.

M. Toro, M. Desainte-Catherine, and C. Rueda. Formal semantics for interactive music scores: a framework to design, specify properties and execute interactive scenarios. *Journal of Mathematics and Music*, 8(1):93–112, 2014. doi: 10.1080/17459737.2013.870610.

C.-i. Wang and S. Dubnov. The variable markov oracle: Algorithms for human gesture applications. *IEEE MultiMedia*, 22(4):52–67, Oct. 2015. doi: 10.1109/MMUL.2015.76.

G. Wang, P. R. Cook, and S. Salazar. ChucK: A strongly timed computer music language. *Computer Music Journal*, 39(4):10–29, Dec. 2015. doi: 10.1162/COMJ_a_00324.

J. Zagal, M. Mateas, C. Fernandez-Vara, B. Hochhalter, and N. Lichti. Towards an ontological language for game analysis. In *DiGRA '05 - Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play*, Vancouver, Canada,

2005. URL http://www.digra.org/digital-library/publications/towards-an-ontological-language-for-game-analysis/?doing_wp_cron=1531064778.6737051010131835937500.

J. Zagal, C. Fernandez-Vara, and M. Mateas. Rounds, levels, and waves : The early evolution of gameplay segmentation. *Games and Culture*, 3(2):175–198, 2008. URL https://pubweb.eng.utah.edu/~zagal/Papers/Zagal_et_al_Gameplaysegmentation.pdf.