ZenStates: Easy-to-Understand Yet Expressive Specifications for Creative Interactive Environments

Jeronimo Barbosa, Marcelo M. Wanderley, Stéphane Huot

ZenStates: Easy-to-Understand Yet Expressive Specifications for Creative Interactive Environments

Jeronimo Barbosa
IDMIL, CIRMMT,
McGill University
Montreal, Canada
jeronimo.costa@mail.mcgill.ca

Marcelo M. Wanderley
IDMIL, CIRMMT
McGill University & Inria
Montreal, Canada
marcelo.wanderley@mcgill.ca

Stéphane Huot
Inria
Univ. Lille, UMR 9189 - CRIStAL
Lille, France
stephane.huot@inria.fr

***Abstract*—Much progress has been made on interactive behavior development tools for expert programmers. However, little effort has been made in investigating how these tools support creative communities who typically struggle with technical development. This is the case, for instance, of media artists and composers working with interactive environments. To address this problem, we introduce ZenStates: a new specification model for creative interactive environments that combines Hierarchical Finite-States Machines, expressions, off-the-shelf components called *Tasks*, and a global communication system called the *Blackboard*. Our evaluation is three-folded: (a) implementing our model in a direct manipulation-based software interface; (b) probing ZenStates' expressive power through 90 exploratory scenarios; and (c) performing a user study to investigate the understandability of ZenStates' model. Results support ZenStates viability, its expressivity, and suggest that ZenStates is easier to understand–in terms of decision time and decision accuracy– compared to two popular alternatives.**

***Index Terms*—Human-computer interaction; hierarchical finite state machines; creativity-support tools; interactive environments; end-user programmers.**

I. Introduction

Over the last years, several advanced programming tools have been proposed to support the development of rich interactive behaviors: The HsmTk [1] and SwingStates [2] toolkits replace the low-level event-based paradigm with finite-states machines; ConstraintJS [3] introduces constraints declaration and their implicit maintenance as a way to describe interactive behaviors; InterState [4] and Gneiss [5] are dedicated programming languages and environments. These advances are primarily focused on programmers and are important because:

- They can make interactive behavior *easier to understand*– sometimes even by non-experts programmers. This is the case, for instance, of the SwingStates toolkit [2]. SwingStates was successfully used by graduate-level HCI students to implement advanced interaction techniques despite of their limited training, when other students with similar skills were unsuccessful in implementing the same techniques with standard toolkits. In addition, better understanding of the implementation of interactive behaviors can make it easier to reuse and modify [1];
- These tools do not sacrifice *expressiveness* to make programing faster or understanding easier. Examples such as [3] and [5] illustrate how such tools can potentially implement a wide variety of complex interactive behaviors, which would have been more costly in existing alternatives.

Despite these advantages, little effort has been made to investigate how these advances could support end-user programmers (EUP) [6], [7] from creative communities who typically struggle with coding such as artists and designers [8], [9]. Making these behaviors *easier to understand* could make them easier to learn, reuse, and extend. At the same time, addressing *expressiveness* (i.e., going beyond standard and well-defined behaviors) could improve the diversity of creative outcomes, helping in the exploration of alternatives. All in all, these characteristics have the potential to foster one's creative process [10].

One example of such community is composers and media artists working with *creative interactive environments* [11]. These environments are immersive pieces that react to the presence of visitors, based on a wide diversity of input sensors (eg. video cameras) and actuators (eg. sound, lightning systems, video projection, haptic devices). Creating easy to understand and expressive development tools for these environments is relevant and challenging for two reasons. First, interactive environments' setups are often more complex than standard interactive systems (such as desktop or mobile computers), as they (i) need to deal with multiple advanced input/output devices and multiple users that both can connect/disconnect/appear/disappear dynamically; (ii) need to be dynamic, flexible and robust. Therefore, by tackling more complex setups, some of its unique features could potentially transfer to more standard setups (e.g. desktop). Secondly, because programming directly impacts on the artistic outcomes, increasing programming abilities and possibilities can likely yield finer artistic results, by reducing the gap between artistic and technical knowledge.

These are the context and the challenge we address in this paper. Here, we investigate innovations brought by powerful development tools for expert programmers [4], [5], [12],

This work was partially supported by the NSERC Discovery Grant and the *Fonds de recherche du Québec–Société et culture* (FRQSC) research grant. **Authors' version. Final version published in 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'18).**

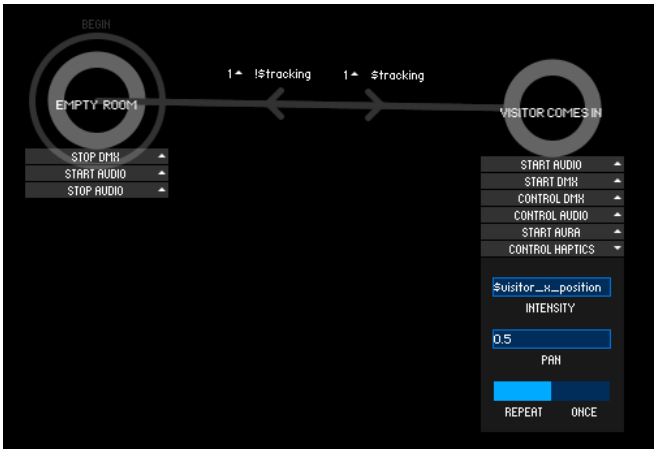

Fig. 1. ZenStates allows users to quickly prototype interactive environments by directly manipulating *States* connected together via logical *Transitions*. *Tasks* add concrete actions to states and can be fine-tuned via high-level parameters (e.g. 'intensity' and 'pan'). *Blackboard* global variables (prefixed with the '$' sign) can enrich tasks and transitions.

aiming at bringing them to the context of creative interactive environments. The result is ZenStates: an easy to understand yet expressive specification model that allows creative EUPs to explore these environments using Hierarchical Finite-States Machines, expressions, off-the-shelf components called tasks, and a global communication system called the blackboard. This model has been implemented in a visual direct manipulation-based interface that validates our model (Fig. 1). Finally, as an evaluation, we also report: (a) 90 exploratory scenarios to probe ZenStates' expressive power; and (b) a user study comparing ZenStates' ease of understanding against the specification model used by two popular interactive environments development tools: Puredata and Processing.

II. Related work

One of the most basic approaches for interactive environments development is trigger-action programming [13]. This approach–used, for instance, in the context of smart homes–describes simple conditional "if-then-else" behaviors that can be easily customized by users. Despite being intuitive and effective, this approach limits interactive environments to simple one-level behaviors. For instance, any example involving temporal actions cannot be implemented.

Other approaches can be found within the context of creative communities. This is the case of the visual "Max paradigm" [14], found in musical tools such as Max/MSP, and Puredata. These dataflow-inspired software environments have become widely adopted over the years by creative EUPs. They allow users to build multimedia prototypes by visually combining elementary building blocks, each performing a specific real-time operation. However, because everything is designed for real-time, little support is provided for managing the flow of control of application (e.g. conditionals, loops, routines) [14].

Another approach is the case of textual programming environments and libraries [15], such as Processing, openFrameworks, and Cinder. However, although these tools lower the floor required to get started in programming, they still require users to deal with abstract concepts whose meaning is not often transparent, especially to novices [16].

To overcome this understandability issue, direct manipulation [17] has been applied in this context to make programming less abstract and therefore more tangible to creative users. Examples include Sketch-N-Sketch [18]–where direct manipulation is combined with textual programming language–and Para [19]–direct manipulation combined with constraint-based models. While the strategy is helpful, the solution is kept on the interface level (i.e., that could potentially be applied to every specification model), whereas the specificion model itself remains unexplored. Furthermore, both examples are limited to static artistic results (i.e., not capable of reacting to inputs over time).

A. Non-programmers expressing interactive behaviors

Given our interest in exploring more accessible specification models, we need to understand first how non-programmers deal with expressing interactive behaviors. A few works have dealt with this topic.

In [20] for instance, two studies focused on understanding how non-programmers express themselves in solving programming-related problems. In one study, 10-11 years old children were asked to describe how they would instruct a computer to behave in interactive scenarios of the Pac-man game. In another, university-level students were asked to do the same in the context of financial & business analytical problems. Results suggest that participants tended to use an event-based style (e.g., if then else) to solve their problems, with little attention to the global flow of the control (typical in structured imperative programming languages, for example). This approach to problem-solving arguably has similarities to the approach that state machines have in problem-solving.

Some work also presented practices and challenges faced by professional multimedia designers in designing and exploring rich interactive behaviors derived from series of interviews and a survey [8]. The results suggest that in the case of designing interactive behaviors: a) current tools do not seem to fulfill the needs of designers, especially in the early stages of the design process (i.e., prototyping); b) texts, sketches, and several visual "arrow-centered" techniques (e.g. mind map, flowcharts, storyboards) are among the most used tools to communicate ideas; and c) designers prefer to focus on content rather than "spending time" on programming or learning new tools. These findings resulted in a new system, DEMAIS, which empowers designers to communicate and do lo-fi sketch-based prototypes of animations via interactive storyboards.

Similar findings were reported by [9]. In a survey with 259 UI designers, the authors found that a significant part of participants considered that prototyping interactive behaviors was harder than prototyping appearance (i.e., the "look and feel"). Participants reported struggling with communicating interactive behaviors to developers–although this communication was necessary in most of the projects–when compared to

communicating appearance. In addition, participants reported necessity for iteration and exploration in defining these behaviors, which generally involved dealing with changing states. The authors conclude by reporting the need for new tools that would allow quicker and easier communication, design and prototyping of interactive behaviors.

These works suggest that state machines could be suited to the development of interactive behavior by non-programmers, and, we believe, by creative end-user programmers who struggle with technical development.

B. Experts programming interactive behaviors

Several tools have been developed to support expert programming of interactive behaviors [21]. Examples include Gneiss [5], ICon [22], and OpenInterface [23]. Among these, a significant number of tools use state machines to specify such interactive behaviors. A complete survey is beyond the scope of this research. Here, we focus on works we consider more closely related to our research.

Perhaps one of the most well-known example is [12], which introduced StateCharts: a simple yet powerful visual-based formalism for specifying reactive systems using enriched hierarchical state machines. StateCharts' formalism was later implemented and supported via a software tool called StateMate [24], and is still used (more than 30 years later) as part of IBM Rational Rhapsody systems for software engineers[1].

Examples are also notable in the context of GUIs, for instance: HsmTk [1], a C++ toolkit that incorporates state machines to the context of rich interaction with Scalable Vector Graphics (SVG); SwingStates [2], that extends the Java's Swing toolkit with state machines; and FlowStates [25], a prototyping toolkit for advanced interaction that combines state machines for describing the discrete aspects of interaction, and data-flow for its continuous parts.

More recently, research has been done on how to adapt state machines to the context of web applications. ConstraintJS [3] proposed an enhanced model of state machines that could be used to control temporal constraints, affording more straightforward web development. This idea is further developed by the authors in [4], resulting in InterState, a new full programming language and environment that supports live coding, editing and debugging.

These works introduce advances that make interactive behaviors programming faster and easier to understand. However, as a drawback, these tools are hardly accessible for creative EUPs who would still need to develop strong programming skills before benefiting from them. To address this problem, we have built upon these works, aiming at making them accessible to creative EUPs. The result, a new specification model called *ZenStates*, is introduced in the next section.

III. Introducing ZenStates

ZenStates is a specification model centered on the idea of *State machines*. A *State Machine* is defined as a set of abstract *States* to which users might add *Tasks* that describe (in terms of *high-level parameters*) what is going to happen when that particular state is being executed. These states are connected to one another via a set of logical *Transitions*. Execution always starts at the "Begin" state, following these transitions as they happen until the end. At any moment, inside a Task or a Transition, users can use *Blackboard* variables to enrich behaviors (e.g., use the mouse x position to control the volume, or to trigger certain transitions).

A. Enriched state machines as specification model

ZenStates borrows features from the enriched state machines model proposed by StateChart [12] and StateMate [24]. These works overcome typical problems of state machines–namely, the exponential growth of number of states, and its chaotic organization–by introducing:

- **Nested state machines (clustering):** One state could potentially contain other state machines;
- **Orthogonality (parallelism):** Nested state machines need to be able to execute independently and in parallel whenever their parent state is executed;
- **Zooming-in and zooming-out:** A mechanism athat allows users to navigate between the different levels of abstraction introduced by the nesting of state machines;
- **Broadcast communication:** That allows simple event messages to be broadcasted to all states, having the potential to trigger other states inside the machine.

When combined, these features result in a powerful hierarchical model of state machines. This model provides improved *organization* (i.e., state machines are potentially easier to understand), *modularity* (i.e., subparts of a system could be independently designed and later merged into a larger structure), and *expressiveness* (broadcast communication allows enhanced description of complex behaviors) when compared to the traditional states machine approach.

B. Key features

Building upon this model, ZenStates proposes five key features described in the following subsections.

1) *Extending communication: the Blackboard*

The blackboard–in the top-right of Figure 2–is a global repository of variables that can be defined and reused anywhere inside a state machine (i.e., inside states, nested states, tasks or transitions). Therefore, we extend the notion of broadcast communication introduced by [12] because variables can be used to other functionalities, and not only triggering transitions. In addition, because the blackboard is always visible on the interface, users can easily infer what inputs are available on the environment, as well as their updated values.

Users can interact with the blackboard using *pull* and *push* operations. Pull operations are accessed by using the variable's name (as in Gneiss [5]). Push operations can be performed by tasks (see next subsection).

Some useful context-dependent variables can be automatically added to the blackboard and become directly accessible to users. In interactive environments, examples include mouse

[1]https://www.ibm.com/us-en/marketplace/rational-rhapsody

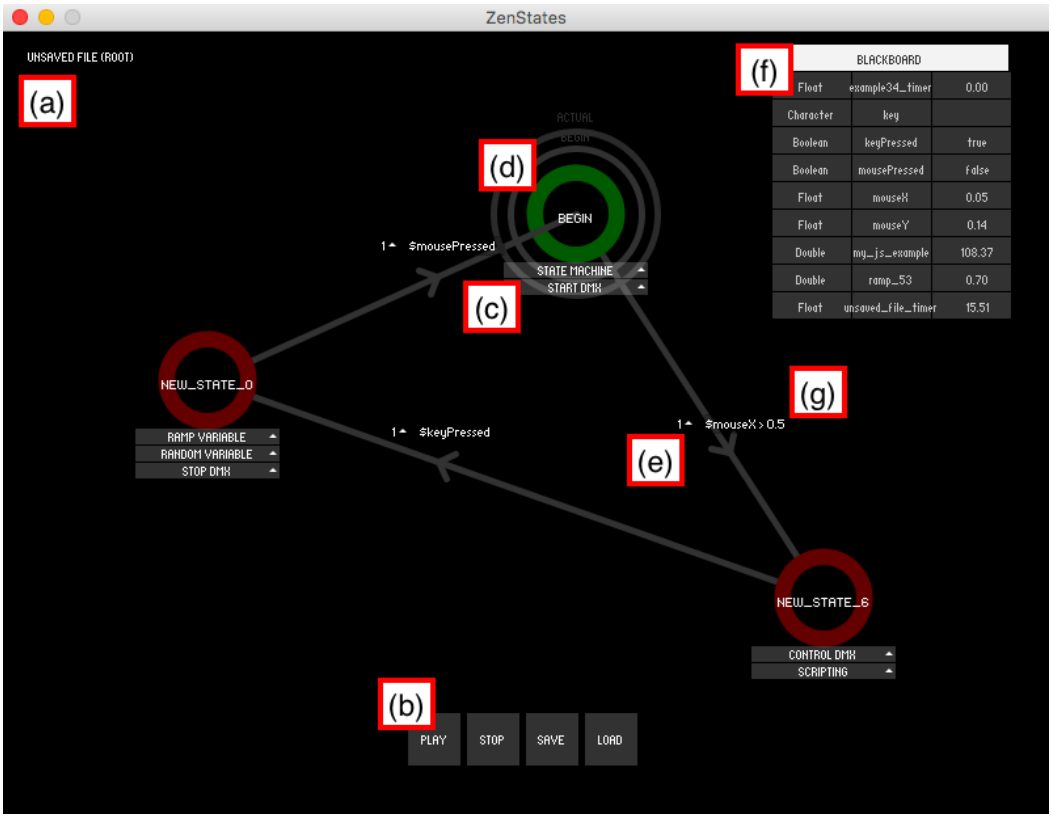

Fig. 2. ZenStates' components: (a) the state machine name; (b) control buttons; (c) tasks associated with a state; (d) the initial state (in this case, also the one which is currently running, represented in green); (f) the blackboard; and transitions, composed by a guard condition (g) and its execution priority (e).

coordinates, key presses, various sensor inputs, incoming Open Sound Control (OSC[2]) messages, etc.

*2) **Making behaviors concrete: the Tasks***

One challenge we needed to address concerns specifying concrete behaviors to happen inside the states. In the case of tools for developers, this behavior could be easily specified via programming language. But how could we make this easier for creative EUPs?

We solve the issue by introducing the notion of tasks: simple atomic behaviors representing actions (if it happens only once) or activities (if it keeps occurring over time) and that can be attached to states as off-the-shelf components [23].

Tasks work in the following way: Once a certain state is executed, all associated tasks are run in parallel. Each individual task also has its own set of idiosyncratic high-level parameters, that allow users to fine-tune its behaviors. In Figure 1, for instance, these high-level parameters are the intensity of the vibration, and the panning of a stereo audio-driven bass shaker. The UI controls to edit these parameters can be easily hidden/shown by clicking on the task, allowing users to focus their attention on the tasks they care the most.

Tasks are normally context-dependent, and need to be changed according to the domain of application. In creative interactive environments, for example, potential tasks could be: sound-related tasks (e.g., start sound, control sound, stop sound), light-related tasks (start, control, and stop light), and haptics related tasks (start, control, and stop haptics)–as shown in Figure 3.

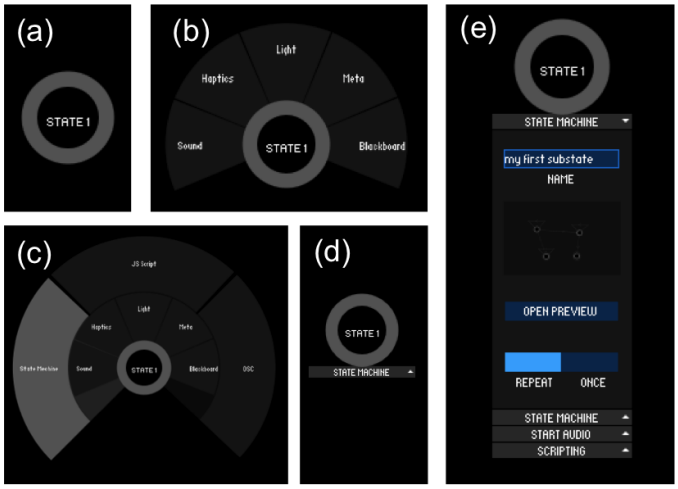

Fig. 3. Tasks inside states are reified with a contextual pie menu, as shown in the sequence above–(a), (b), (c), and (d). High-level parameters related to each task can be found by clicking on the task–as shown in (e).

There are however two categories of tasks which are not context dependent: the blackboard-tasks, and the meta-tasks.

[2]http://opensoundcontrol.org/

Blackboard-tasks relate to creating new variables within the blackboard so that they can be later used anywhere inside the state machine. Our motivation is to increase reuse by providing users with a set of recurrent functionalities often found in interactive environments. Examples include oscillators, ramps, and random numbers.

Meta-tasks relate to extending ZenStates in situations where currently-supported functionalities are not enough. For example, OSC tasks allow communication with external media tools via the OSC protocol. JavaScript tasks allow custom JavaScript code to be incorporated to states. Finally, we have State Machine tasks, that allows nesting as shown in Figure 3.

3) ***Enriching transitions***

We enrich transitions by incorporating two functionalities.

First, transitions make use of priorities that define the order they are going to be checked. This means that one state can have multiple transitions evaluated one after the other according to their priority (see Fig. 4). This allows users to write complex logical sentences similar to cascades of "if-then-else-if" in imperative languages. It also avoids potential logical incoherences raised by concurrent transitions.

Second, transitions support any Javascript expression as *guard conditions*, expressed as *transition and constraint events* as in [4]. In practice, this functionality combines logical operators (e.g. '`&&`', '`||`', '`!`'), mathematical expressions, and blackboard variables used either as events (e.g. '`$mousePressed`') or inside conditions (e.g. '`$mouseX > 0.5`'). For instance, it is possible to set that if someone enters a room, the light should go off. Or if the mouse is pressed on a certain x and y coordinates, another page should be loaded.

4) ***Self-awareness***

Self-awareness describes a set of meta characteristics belonging to states and tasks that are automatically added to the blackboard and become available to users. For example, states can have a status (e.g., is it running? Is it inactive?), sound tasks can have the current play position, the volume, and the audio pan as properties, etc. This feature can be useful in several ways. For example, we can set a transition to occur when all tasks within a certain state are completed (e.g. '`$State.Done`'). Or yet, it is possible to have the volume of an audio file to increase as its playback position increases.

5) ***Live development & Reuse***

Finally, ZenStates also borrows two additional features from [4] and [5]:

- **Live development:** Any element (e.g. tasks, transitions, and states) can be modified at runtime, allowing quicker process of experimentation and prototyping;
- **Reuse:** Previously defined interactive behaviors can easily be reused via nested state machines (that can be saved into files, exchanged, and then later imported). Here, we also expect to align with the principle of reuse as introduced by [26].

IV. Implementation

The design process of ZenStates has been iterative and user-centered with media artists. Steps included: a) *analysis of existing alternatives*; b) *interviews with expert users*; c) *paper prototypes*; d) *development of scenarios to assess the potential of the tools in solving problems*; e) *observing the compositional process of two new media works*; f) *functional prototypes*; and g) *user-interface refinement over the course of think-aloud protocol sections with novice and expert users*. While a full description of this process is beyond the scope of this paper, its direct result is the specification model as presented in this paper.

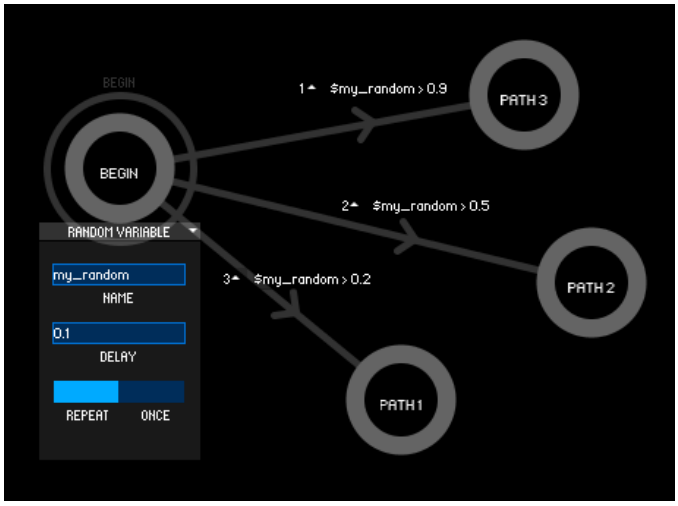

Fig. 4. Transitions have priorities attached to them. In this case, the blackboard variable '`$my_random`' generates a random number (between 0 and 1) which is then used to select a random path. The first transition to be checked is '`$my_random > 0.9`' because of its priority '1'. If false, transition with priority '2' would then be checked. This process goes on until all transitions are checked. In case none is satisfied, the current state is executed once again before the next round of checking (repeat button is enabled). Transitions can be added, edited, and deleted via direct manipulation.

To validate this model, we implemented all features described here in a functional software prototype. In this prototype, abstract elements (namely the state machine, states, tasks, transitions, and the blackboard) are reified–according to the principle of reification introduced in [26]–to graphical objects that can be directly manipulated by the user [17]. These visual representations become the core of the user interaction and also allow users to infer the current state of the system at any time during execution: the current running state is painted green; states already executed are red; inactive states are gray (see Fig. 2). All images presented on this paper are actual screenshots of this functional prototype, also demonstrated in the supplementary video.

The prototype has been implemented in Java, using Processing[3] as an external library for the graphics. The project is open-source and the source code is available online[4]. We stress that, as a prototype, this system is sill under development and its usability needs improvement.

[3]http://processing.org/

[4]https://github.com/jeraman/zenstates-paper-vlhcc2018

V. Evaluation

We argue ZenStates proposes an expressive (i.e., allow to develop a wide diversity of scenarios) yet easy-to-understand specification model for creative interactive environments. We have tested these claims in two steps. First, we have probed ZenStates' expressive power by designing 90 *exploratory scenarios*. Second, we have conducted a *user study* investigating ZenStates specification model in terms of its capacity to quickly and accurately describe interactive environments.

A. Exploratory scenarios

To explore the expressive power of ZenStates, we have developed 90 unique exploratory scenarios. These scenarios implement atomic audiovisual behaviors with different levels of complexity, based on a constrained set of inputs (either mouse, keyboard, or both), outputs (either background color of the screen, the sound of a sinewave, or both), and blackboard variables. The chosen behaviors are often used in the context of music/media arts. Examples include: Sinusoidal sound generators with user-controlled frequency (c.f., the project Reactable [5]) as implemented in the scenario *'mouse_click_mute_mousexy_freq_amp'*; and contrasting slow movements and abrupt changes to create different moments in a piece (c.f., 'Test pattern' by Ryoji Ikeda [6]) implemented in the scenario *'random_flickering_random_wait_silence'*) . The full list of exploratory scenarios–with implementation and video demonstration–is available as supplementary material[7].

While small, these atomic scenarios can be further combined to one another via hierarchical state machines and transitions. Therefore, it is possible to create a potentially infinite number of new scenarios–much more complex than the atomic ones. This diversity illustrates the expressive power of ZenStates–especially considering the constrained set of input, outputs and blackboard variables that were used.

B. User study

We investigated if ZenState's specification model makes the development of interactive environments easier to understand. This model was compared to the specification model used by two popular interactive environments development tools: Puredata (hereafter, the *dataflow* model), and Processing (hereafter, the *structured* model).

1) ***Hypothesis:*** Our hypothesis is that Zenstates allows users to understand interactive environments specifications more *accurately*–that is, ZenStates would reduce the number of code misinterpretations by users–and *faster*–that is, ZenStates would reduce the amount of time necessary to understand the code–compared to the alternatives.

2) ***Subjects and Materials:*** We recruited 12 participants (10 male, 1 female, and 1 not declared), aged 29 years old on average (*min* = 22, *max* = 46, *median* = 26, *SD* = 7.98). Participants were all creative EUPs with previous experience with interactive environments tools (e.g. media artists, composers, designers, and technologists), either professionals or students. Their expertise on computer programming ranged from novices to experts (*min* = 2 years, *max* = 23 years, *mean* = 7.67, *median* = 6, *SD* = 5.48). The most familiar languages for them were Python and Max/MSP (both with 4 people each), followed by Processing (2 people).

Regarding the experimental material, we have selected 26 of our exploratory scenarios representing a wide diversity of usage scenarios (i.e., either using blackboard variables; single input; and multimodal input). Each scenario was then specified using the three evaluated specification models (ZenStates, dataflow, and structured), resulting in 78 specifications. All these scenarios and their specifications are available online[8].

3) ***Procedure:*** Our experimental procedure is based on [27], adapted and fine-tuned over a preliminary pilot with 6 participants. The resulting procedure is composed of 3 blocks–one block per specification model tested–of 6 trials.

In each block, participants were introduced to a specification model as presented in Figure 5. Participants were also given a small printed cheatsheet containing all possible inputs (i.e., mouse and keyboard), actuators (i.e., screen background color, and a sinewave), and language specific symbols that could appear over the trials. At this point, participants were allowed to ask questions for clarification, and were given some time to get used to the model.

After this introduction, participants were introduced to one interactive environment specification–either ZenStates, dataflow, or a structured language–and to six different interactive environments videos showing real behaviors. Their task was *to choose which video they thought that most accurately corresponded to the specification presented*, as shown in Figure 6. Participants were instructed to be the most accurate, and to chose a video as quick as they could–without sacrificing accuracy. Only one answer was possible. Participants were allowed two practice trials and the cheatsheet could be consulted anytime.

Participants repeated this procedure for all three evaluated alternatives. Presentation order of the models was counterbalanced to compensate for potential learning effects. Similarly, the order of the six videos was randomized at each trial.

Our measured dependent variables were the *decision accuracy* (i.e., the percentage of correct answers), and the *decision time* (i.e., the time needed to complete the trial). As in [27], the decision time was computed as the trial total duration minus the time participants spent watching the videos. The whole procedure was automatized via a web application[9].

[5]https://www.youtube.com/watch?v=n1J3b0SY_JQ

[6]https://www.youtube.com/watch?v=JfcN9Qhfir4

[7]https://github.com/jeraman/zenstates-paper-vlhcc2018/tree/master/scenarios

[8]https://github.com/jeraman/zenstates-paper-vlhcc2018/tree/master/data%20collection/html/scenarios

[9]https://github.com/jeraman/zenstates-paper-vlhcc2018/tree/master/data%20collection

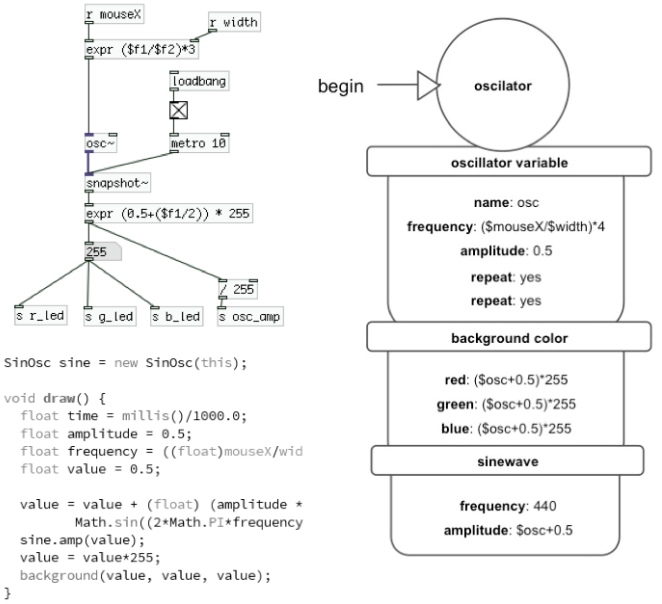

Fig. 5. The three specification models presented to the participants: dataflow (top-left), structured (bottom-left), and ZenStates (right).

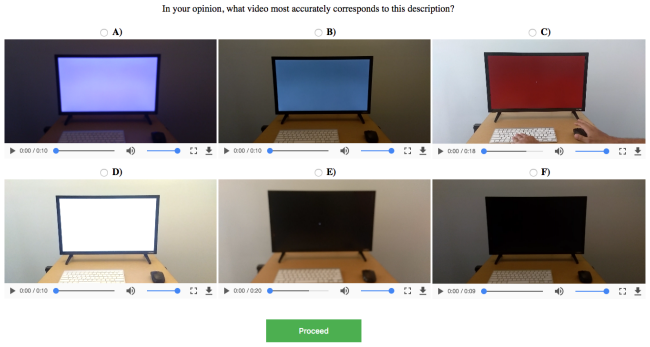

Fig. 6. Being introduced to one specification, participants needed to choose which one among six videos they thought that most accurately corresponded to the specification presented.

Finally, participants were asked about the specification models they had considered the easiest and the hardest to understand, followed by a short written justification.

4) ***Results:*** Data from HCI experiments has often been analyzed by applying null hypothesis significance testing (NHST) in the past. This form of analysis of experimental data is increasingly being criticized by statisticians [28], [29] and within the HCI community [30], [31]. Therefore, we report our analysis using estimation techniques with effect sizes[10] and confidence intervals (i.e., not using p-values), as recommended by the APA [33].

Regarding *decision time*, there is a strong evidence for ZenStates as the fastest model (mean: 40.57s, 95% CI: [35.07,47.27]) compared to the dataflow (mean: 57.26s, 95% CI: [49.2,67.5]), and the structured model (mean: 70.52s,

[10]Effect size refers to the measured difference of means–we do not make use of standardized effect sizes which are not generally recommended [32].

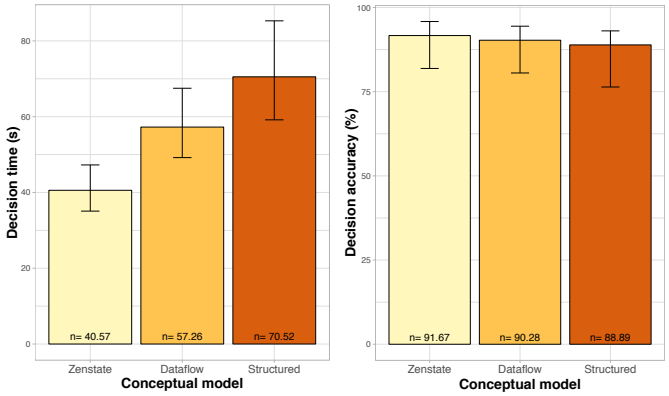

Fig. 7. *Decision time* (left) and *Accuracy* (right) per specification model. Error Bars: Bootstrap 95% CIs.

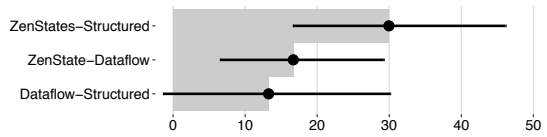

Fig. 8. *Decision time* pairwise differences. Error Bars: Bootstrap 95% CIs.

95% CI: [59.16,85.29]), as shown in Figure 7. We also computed pairwise differences between the models (Fig. 8) and their associated confidence intervals. Results confirm that participants were making their decision with ZenStates about 1.4 times faster than with the dataflow and 1.8 times faster than with the structured model. Since the confidence interval of the difference between dataflow and structured models overlaps 0, there is no evidence for differences between them.

Regarding *decision accuracy*, participants achieved 91.67% (95% CI: [81.89,95.83]) accuracy with ZenStates, 90.28% (95% CI: [80.55,94.44]) with the Dataflow model, and 88.89% (95% CI: [76.39,93.05]) with the structured model (see Fig. 7(right)). These results show quite high accuracy with all the specification models overall, although there is no evidence for one of them being more accurate than the other.

Finally, regarding the final questionnaire data, ZenStates was preferred by participants as the *easiest model to understand* (8 people), followed by the structured (3 people), and the dataflow (1 person) models. Participants written justifications reported aspects such as *"graphic display"*, *"code compartmentalization"*, and *"abstraction of low-level details"* as responsible for this preference. Similarly, dataflow (8 people) was considered the *hardest model to understand*, followed by structured (3 people), and ZenStates (1 person).

VI. Limitations

Our studies also revealed limitations in our specification model and its software interface, namely:

- **The blackboard:** Currently represented as a two-columns table on the top right of the screen, containing variable name and its value. While this initial approach

fulfills its goals (i.e., enhancing communication), it has limitations. A first limitation deals with representing a large amount of sensor data in the screen. For example, if a 3D depth camera is attached to the system (tracking x, y, and z points for all body joints), all detected joints would be added to the blackboard. For users interested in specific information (e.g. hand x position), the amount of visual data can be baffling. Another limitation concerns lack of support to high-level features (e.g. derivatives, and averages), and filtering, which are often as useful as raw sensor data. Further research is needed to improve the blackboard on these directions;

- **Physical environment alignment:** ZenStates assumes that the physical environment (i.e., sensor input, hardware for output media) and the tasks it supports (i.e., sound-related, light-related) is static and consistent, and that it would remain consistent along the execution. This assumption is reasonable when dealing with standard interaction techniques (e.g. WIMP tools for desktop, as in SwingStates [2]). However, because we are dealing with potentially more complex setups, it is possible that the assumption is no longer possible in certain cases. For example, in a certain artistic performance, some sensors might be added, or some light strobes removed during the performance. How to maintain this environment-software consistency in these dynamic cases? How should ZenStates react? These are open questions that need to be addressed in future developments;
- **Interface usability and stability:** The evaluation performed so far focuses only on readability of our specification model, not addressing ease of use or usability of the prototype interface that implements the model. We reason that ease of use and usability are not as relevant in such prototype stage as already-known problems would show up, limiting the evaluation of our specification model. At this stage, readability seems more effective as it could make specifications easier to understand, and potentially easier to learn, reuse, and extend. Nevertheless, we highlight that the usability of such interface would play a significant role in the effectiveness of our model. Examples to be improved include the obtuse expression syntax, using '$' to instantiate variables, and the small font size;

In addition to addressing these problems, we also plan to explore the usage of ZenStates in other creative contexts and to implement principles that could improve general support to creativity inside ZenStates (see [10] for examples).

VII. Conclusion

In this paper, we have analyzed the state of the art of development tools for programming rich interactive behaviors, investigating how these tools could support creative end-user programmers (e.g., media artists, designers, and composers) who typically struggle with technical development. As a solution, we introduced a simple yet powerful specification model called ZenStates. ZenStates combines five key contributions– 1) the blackboard, for communication; 2) the tasks, for concrete fine-tunable behaviors; 3) the prioritized guard-condition-based transitions; 4) self-awareness; and 5) live development & reuse–, exploring those in the specific context of interactive environments for music and media arts.

Our evaluation results suggest that ZenStates is expressive and yet easy to understand compared to two commonly used alternatives. We were able to probe ZenStates' expressive power by the means of 90 exploratory scenarios typically found in music/media arts. At the same time, our user study revealed that ZenStates model was on average 1.4 times faster than dataflow model, 1.8 times faster then the structured model in terms of *decision time*, and had the highest *decision accuracy*. Such results were achieved despite the fact participants had no previous knowledge of ZenStates, whereas alternatives were familiar to participants. In the final questionnaire, 8 out of 12 participants judged ZenStates as the easiest alternative to understand.

We hope these contributions can help making the development of rich interactive environments–and of interactive behaviors more generally–more accessible to development-struggling creative communities.

Acknowledgment

The authors would like to thank Sofian Audry and Chris Salter for their support and numerous contributions, and everyone involved in the project *'Qualified Self'* for their time.

References

[1] R. Blanch and M. Beaudouin-Lafon, "Programming rich interactions using the hierarchical state machine toolkit," in *Proceedings of the working conference on Advanced visual interfaces - AVI '06*. New York, New York, USA: ACM Press, 2006, p. 51.

[2] C. Appert and M. Beaudouin-Lafon, "SwingStates: adding state machines to Java and the Swing toolkit," *Software: Practice and Experience*, vol. 38, no. 11, pp. 1149–1182, sep 2008.

[3] S. Oney, B. Myers, and J. Brandt, "ConstraintJS: Programming Interactive Behaviors for the Web by Integrating Constraints," in *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*. New York, New York, USA: ACM Press, 2012, p. 229.

[4] ——, "InterState: A Language and Environment for Expressing Interface Behavior," in *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*. New York, New York, USA: ACM Press, 2014, pp. 263–272.

[5] K. S.-P. Chang and B. A. Myers, "Creating interactive web data applications with spreadsheets," in *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*. New York, New York, USA: ACM Press, 2014, pp. 87–96.

[6] A. J. Ko, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, S. Wiedenbeck, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, and H. Lieberman, "The state of the art in end-user software engineering," *ACM Computing Surveys*, vol. 43, no. 3, pp. 1–44, apr 2011.

[7] F. Paternò and V. Wulf, Eds., *New Perspectives in End-User Development*. Cham: Springer International Publishing, 2017.

[8] B. P. Bailey, J. A. Konstan, and J. Carlis, "Supporting Multimedia Designers: Towards more Effective Design Tools," in *Proceedings of Multimedia Modeling*, 2001, pp. 267–286.

[9] B. Myers, S. Y. Park, Y. Nakano, G. Mueller, and A. Ko, "How designers design and program interactive behaviors," in *IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, sep 2008, pp. 177–184.

[10] B. Shneiderman, "Creativity support tools: accelerating discovery and innovation," *Communications of the ACM*, vol. 50, no. 12, pp. 20–32, 2007.

[11] M. W. Krueger, "Responsive environments," in *Proceedings of the June 13-16, 1977, national computer conference on - AFIPS '77*. New York, New York, USA: ACM Press, 1977, p. 423.

[12] D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, jun 1987.

[13] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman, "Practical trigger-action programming in the smart home," in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14*. New York, New York, USA: ACM Press, 2014, pp. 803–812.

[14] M. Puckette, "Max at Seventeen," *Computer Music Journal*, vol. 26, no. 4, pp. 31–43, dec 2002.

[15] J. Noble, *Programming Interactivity: A Designer's Guide to Processing, Arduino, and openFrameworks*, 2nd ed. O'Reilly Media, 2012.

[16] B. Victor, "Learnable programming," 2012. [Online]. Available: http://worrydream.com/LearnableProgramming/

[17] B. Shneiderman, "Direct Manipulation: A Step Beyond Programming Languages," *Computer*, vol. 16, no. 8, pp. 57–69, aug 1983.

[18] B. Hempel and R. Chugh, "Semi-Automated SVG Programming via Direct Manipulation," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology - UIST '16*. New York, New York, USA: ACM Press, 2016, pp. 379–390.

[19] J. Jacobs, S. Gogia, R. Měch, and J. R. Brandt, "Supporting Expressive Procedural Art Creation through Direct Manipulation," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems - CHI '17*. New York, New York, USA: ACM Press, 2017, pp. 6330–6341.

[20] J. F. Pane, C. A. Ratanamahatana, and B. A. Myers, "Studying the language and structure in non-programmers' solutions to programming problems," *International Journal of Human-Computer Studies*, vol. 54, no. 2, pp. 237–264, feb 2001.

[21] F. Cuenca, K. Coninx, D. Vanacken, and K. Luyten, "Graphical Toolkits for Rapid Prototyping of Multimodal Systems: A Survey," *Interacting with Computers*, vol. 27, no. 4, pp. 470–488, jul 2015.

[22] P. Dragicevic and J.-D. Fekete, "Support for input adaptability in the ICON toolkit," in *Proceedings of the 6th international conference on*

[31] P. Dragicevic, F. Chevalier, and S. Huot, "Running an HCI experiment in multiple parallel universes," in *Proceedings of the extended abstracts of the 32nd annual ACM conference on Human factors in computing*

Multimodal interfaces - ICMI '04. New York, New York, USA: ACM Press, 2004, p. 212.

[23] J.-Y. L. Lawson, A.-A. Al-Akkad, J. Vanderdonckt, and B. Macq, "An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components," in *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems - EICS '09*. New York, New York, USA: ACM Press, 2009, p. 245.

[24] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot, "STATEMATE: a working environment for the development of complex reactive systems," *IEEE Transactions on Software Engineering*, vol. 16, no. 4, pp. 403–414, apr 1990.

[25] C. Appert, S. Huot, P. Dragicevic, and M. Beaudouin-Lafon, "FlowStates: Prototypage d'applications interactives avec des flots de données et des machines à états," in *Proceedings of the 21st International Conference on Association Francophone d'Interaction Homme-Machine - IHM '09*. New York, New York, USA: ACM Press, 2009, p. 119.

[26] M. Beaudouin-Lafon and W. E. Mackay, "Reification, polymorphism and reuse," in *Proceedings of the working conference on Advanced visual interfaces - AVI '00*, no. May. New York, New York, USA: ACM Press, 2000, pp. 102–109.

[27] K. Kin, B. Hartmann, T. DeRose, and M. Agrawala, "Proton++: A Customizable Declarative Multitouch Framework," in *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*. New York, New York, USA: ACM Press, 2012, p. 477.

[28] M. Baker, "Statisticians issue warning over misuse of P values," *Nature*, vol. 531, no. 7593, pp. 151–151, mar 2016.

[29] G. Cumming, "The New Statistics," *Psychological Science*, vol. 25, no. 1, pp. 7–29, jan 2014.

[30] P. Dragicevic, "Fair Statistical Communication in HCI," ser. Human–Computer Interaction Series, J. Robertson and M. Kaptein, Eds. Cham: Springer International Publishing, 2016, pp. 291–330.

systems - CHI EA '14. New York, New York, USA: ACM Press, 2014, pp. 607–618.

[32] T. Baguley, "Standardized or simple effect size: What should be reported?" *British Journal of Psychology*, vol. 100, no. 3, pp. 603–617, aug 2009.

[33] G. R. VandenBos, *APA dictionary of psychology*. Washington, DC:: American Psychological Association, 2007.