SoftwareX 6 (2017) 255–260

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

The Sound Design Toolkit

Stefano Baldan [a,*], Stefano Delle Monache [a], Davide Rocchesso [a,b]

[a] Department of Architecture and Arts, Iuav University of Venice, Dorsoduro 2206, 30123, Venice, Italy
[b] Department of Mathematics and Computer Science, University of Palermo, Via Archirafi 34, 90123, Palermo, Italy

ARTICLE INFO

Article history:
Received 18 January 2016
Received in revised form 14 November 2016
Accepted 9 June 2017

Keywords:
Sonic interaction design
Sound synthesis
Procedural audio

ABSTRACT

The Sound Design Toolkit is a collection of physically informed sound synthesis models, specifically designed for practice and research in Sonic Interaction Design. The collection is based on a hierarchical, perceptually founded taxonomy of everyday sound events, and implemented by procedural audio algorithms which emphasize the role of sound as a process rather than a product. The models are intuitive to control – and the resulting sounds easy to predict – as they rely on basic everyday listening experience. Physical descriptions of sound events are intentionally simplified to emphasize the most perceptually relevant timbral features, and to reduce computational requirements as well.

Code metadata

Current code version	075
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-16-00016
Legal Code License	GPL
Code versioning system used	git
Software code languages, tools, and services used	C
Compilation requirements, operating environments & dependencies	Compilation: Gnu Make. Environments: Cycling '74 Max (Win/Mac) and PureData (Win/Mac/Linux)
If available Link to developer documentation/manual	https://github.com/SkAT-VG/SDT/releases/download/075/SDT_APIdoc-075.pdf
Support email for questions	stefanobaldan@iuav.it

Software metadata

Current software version	075
Permanent link to executables of this version	https://github.com/SkAT-VG/SDT/releases/download/075/SDT-075.zip
Legal Software License	GPL
Computing platforms/Operating Systems	Linux, OS X, Microsoft Windows
Installation requirements & dependencies	none
If available, link to user manual — if formally published include a reference to the publication in the reference list	none
Support email for questions	stefanobaldan@iuav.it

1. Motivation and significance

The Sound Design Toolkit (SDT) is a software package providing a set of perceptually founded sound models, for the interactive generation of a variety of basic and complex acoustic phenomena such as interactions between solid objects, interactions involving liquids or gasses, and machines. The SDT can be framed as a virtual

* Corresponding author.
E-mail addresses: stefanobaldan@iuav.it (S. Baldan), sdellemonache@iuav.it (S. Delle Monache), roc@iuav.it (D. Rocchesso).

http://dx.doi.org/10.1016/j.softx.2017.06.003

Foley box, filled with a rich palette of virtual noisemakers, readily available to the sound designer to sketch and prototype sonic interactive behaviors [1]. The tools of the kit are physics-based procedural audio models [2], designed from first principles and exhibiting physically meaningful parameters. The natural application frame is to be found in sound design for multisensory interactive systems (e.g., games) where action–sound behaviors are more important than sounds *per se*.

The sketching and prototyping character of the SDT emerged out of more than ten years of research in sound synthesis and design [3]. The initial collection of sound models was developed within the EU project Sounding Object (SOb) as a set of *externals* (plug-ins) and *patches* (programs) for Pure Data,[1] a widely used open source visual language. Over the years, many explorations in sonic interaction design pushed the development of a collection of sound models and control layers that would go beyond the musical metaphor, and respond to the requirements of design thinking and practice [1]. The whole package was ported to the Max environment,[2] a commercial visual programming language made by Cycling '74, similar to Pure Data, yet offering a richer graphical user interface and a larger set of features [4].

In the scope of the research activities of the ongoing EU project SkAT-VG (Sketching Audio Technologies using Vocalizations and Gestures),[3] the SDT software architecture has been deeply revised and the palette of sound models further extended. The collection, already including impact, friction and derived processes such as crumpling, rolling and bouncing, has been expanded with the families of aerodynamic interactions (continuous turbulences, explosions), liquids, and machines (combustion engines, DC motors). The aim is to have a palette of sounding objects to cover a relevant mixture of acoustic phenomena, as they are found in major applications of sound design.

The framework has been developed according to an ecological approach to everyday listening, and organized to reflect the current knowledge on the perception and categorization of everyday sounds [5,6]. Indeed, the base assumption is that humans hear *sound events* rather than sounds per se, being able to extract meaningful information about their surroundings from their everyday listening experience [7].

The SDT synthesizers make use of physically informed procedural audio algorithms [2], which generate sound from a simplified computational description of the sound producing event, rather than relying on sample-based techniques and wavetable manipulation. The simplification is instrumental to achieve a two-folded objective: 1) providing an affordable environment for real-time applications on ordinary computer and embedded systems, and 2) *cartoonification*, that is the improvement of perceptual clarity and understanding by means of simplified auditory representations (i.e., cartoon sounds), through the exaggeration of their salient features [8].

For instance, the pitched and "blooping" sound generated by an air bubble in water can be approximated by a sinusoidal oscillator with decreasing amplitude and rising frequency, instead of trying to faithfully model the complex fluid-dynamics involved in the process. Although heavily simplified, the model is nevertheless informed by physics, but at a higher level of abstraction: Resonant frequency and decay time of the digital oscillator depend on the radius of the simulated bubble, as it would happen in the corresponding real world phenomenon. The cartoonified bubble model can therefore be controlled by physically relevant parameters while retaining and exaggerating the key features of the original sound [9]. This approach emphasizes the role of sound as a behavior, as a process rather than a product.

Certainly, the SDT is not the only software package available for procedural sound synthesis and design. The prominent work by Andy Farnell [2], available as a collection of Pure Data patches, shares with SDT the same philosophy of synthesizing sounds by designing behaviors and processes. Farnell proposes a loose organization of models that are entirely expressed in Pure Data visual language, thus relying on that hosting environment.

Pruvost and colleagues recently proposed a procedural audio framework for the physics-based sound synthesis of solid interactions [10]. Similarly to the SDT, this system exploits the action–object paradigm for continuous control and sound synthesis, yet facilitating perceptual relevance rather than physical accuracy. In addition, the real-time audio generation is driven by a game engine. The software package, however, has not been released to the public.

The French company Audiogaming[4] developed an interesting proprietary bundle providing a wide palette of environmental procedural audio plugin units.[5] Their sound synthesizers make use of a mixed approach of physical sound modeling and wavetable manipulation. In particular, the software package has been optimized to work with audio middleware solutions for game engines.

Finally, there are open-source libraries for audio signal processing and algorithmic synthesis that include physics-based sound synthesis modules, the most widely known being STK [11]. This is oriented to the development of music synthesis and audio processing software, and it offers a wide range of software objects, ranging from simple filters to full-scale synthesizers. It definitely lacks the taxonomic organization of SDT, and it is more suitable for computer music programmers than for sound designers.

Several strategies and techniques have been proposed to make the high-dimensional spaces of synthetic sounds easier to explore interactively, and directly exploitable in human–machine interfaces (see [12] for a survey). Although considerable effort has been put to expose model parameters that are meaningful to humans, the so-called mapping layer is left at the periphery of the SDT, and strategies for mapping gesture to sound are not described in this paper.

2. Software description

2.1. Software architecture

The system architecture follows a modular and hierarchical structure, composed of three layers:

1. A core library coded in ANSI C, with few and widely supported dependencies, exposing a clean and streamlined API to all the implemented sound models, sound processors and audio feature extractors;
2. A set of wrappers for Max (version 6 or above) and Pure Data, providing access to most of the SDT framework features by means of externals;
3. A collection of Max patches and help files, providing a user-friendly GUI and an extensive user documentation for the whole framework.

The core components of SDT are written following the principles of object-oriented programming. Each component consists of an opaque data structure, defined as a type and storing the internal state of the object (acting as a class), and a collection of functions operating on that data type (acting as object methods). The header

[1] http://puredata.info.
[2] http://cycling74.com.
[3] http://skat-vg.eu.
[4] http://www.audiogaming.net/.
[5] http://lesound.io/.

files of the API contain references to the type definitions but not to the fields of the data structures, and only to those functions which are meant to be public, enforcing encapsulation to some extent. The core components can be compiled to a shared library under all the supported operating systems (framework for Mac OS X, DLL for Windows, shared object for Linux).

The object-oriented structure of the core library closely mirrors the structure of the software development kits for Max and Pure Data. To embed an SDT object inside a Max or Pure Data object, it is sufficient to embed an SDT data type inside the data structure of the external, and to call the corresponding SDT methods inside the methods of the external. Externals are also compiled as dynamically linked modules for all the available platforms.

The externals are combined in *patches*, visual programs that define a data flow to obtain the desired output, out of a given input. Externals are placed on a canvas, and displayed as boxes exposing their respective sets of inlets and outlets. By connecting inlets and outlets through virtual cables, externals can exchange and process data, producing the desired program [4].

The reuse of the same core objects in different environments is a choice made with portability in mind, and allows consistent behavior across different platforms and operating systems. Although the integration with Max is the most actively supported, the C API exposed by the core library can be used on its own in a wide variety of contexts. In general, the software architecture emphasizes the malleability and modularity of the sound models as basic bricks to design the sound of more complex machines and mechanical interactions.

For instance, rubbing and scraping sounds are achieved by imposing a friction model controller `[sdt.scraping~]` to the modal resonator `[sdt.modal]`, in frictional coupling `[sdt.friction~]` with another object `[sdt.inertial]`. Moreover, several example patches of compound sound models, such as photocopiers, whipping, and fridge hums, are included. For example, windshield wiper sounds are achieved by coupling the models of electric motor and friction [13].

The SDT source code is available through a dedicated Git repository[6] and it is licensed under GNU General Public License. The compilation process relies on GNU Make,[7] with custom Makefiles available for each specific platform.

2.2. Software functionalities

Fig. 1 shows the organization of the synthesis models available in the Sound Design Toolkit. It is a bottom-up hierarchy of sound models, where the first level presents the basic algorithms with the corresponding Max externals, suitable for the generation of a large family of simple sound events. The second level highlights the temporally patterned events (with the corresponding Max externals), basic textures and processes, that can be straightly derived from the temporal control of the low-level models.

The available sound models are grouped according to a criterion of causal similarity, in four main classes of sounds, that is (1) vibrating solids, (2) liquids, (3) gasses, and (4) machines. In addition, the gray arrows describe the direct dependencies between the sound models and the *timbral families* in the small boxes. The concept of timbral family is defined as a specific configuration of one or more models, describing a perceptually relevant category of sound, unambiguously discriminated in terms of interaction, and temporal and timbral properties [14].

Basic solid interactions.

Basic solid interaction algorithms share a common, modular structure "resonator–interactor–resonator", representing the interaction between two resonating objects. This is implemented by connection of externals, such as `[sdt.inertial]` ↔ `[sdt.impact~]` ↔ `[sdt.modal]`.

Although object models can behave in different ways and be implemented through different algorithms, they all must expose one or more *pickup points*. Each pickup point gives information about *displacement* and *velocity*, and can as well be used to apply an external *force* to the resonator.

Resonators continuously update their state according to an input force, coming from an interactor and/or an external source. Object displacement and velocity can be read at any time from one or more pickup points. The available resonator models are:

- *Inertial mass:* Simulates a simple inertial point mass, mostly used as exciter for modal resonators;
- *Modal resonator:* Physical model of a set of parallel mass–spring–damper mechanical oscillators, with each oscillator representing a resonant frequency of the object.

Interactor algorithms read the state of exactly two pickup points, one for each interacting resonator, and apply a force accordingly. The available interactors are:

- *Impact:* Simulates a non-linear impact, by computing the contact force from the total compression, namely the relative displacement between the two contact points [15];
- *Friction:* Simulates a non-linear friction, by computing the friction force from the relative velocity between the two contact points [16].

Numerical simulations of the non-linear, continuous-time systems described above require an appropriate discretization strategy. In particular, the non-linear terms cause numerical and stability issues if not treated properly. Legacy resonators were discretized with a bilinear transformation, an implicit approximation method yielding instant mutual dependency between resonator states and interactor forces. This led to a delay-free, non-computable loop in the discrete-time equations, which needed to be solved by Newton–Raphson approximation [15]. Current resonators use the impulse invariance method, an explicit discretization strategy which does not cause delay-free loops, leading to simpler, more efficient and flexible implementations [17].

Compound solid interactions. Many complex sound textures can be modeled as stochastic series of micro-impacts [8,9]. Different distributions of time and amplitude of the impacts give different and peculiar timbral signatures to the resulting textures. The SDT provides three compound solid interaction algorithms, whose outputs are used as control signals for the single impact model. The modeled interactions include the sound categories of *rolling*, *crumpling/breaking* and *scraping*.

Liquid sounds. The main responsible for acoustic emission in water and other liquids is gas trapped in bubbles, rather than the liquid mass itself. The acoustic behavior of a population of independent, spherical bubbles is well known, and such bubbles are simulated by a polyphonic bank of exponentially decaying sinusoidal oscillators updated by a stochastic algorithm [18].

Continuous turbulences. Flowing gasses are practically silent until they meet an obstacle that forces them to deviate, generating turbulence. In the SDT, turbulent behavior is simulated by means of filtered noise. Three different types of turbulence are simulated, each with a different filtering strategy: *Noise* caused by solid surfaces, *howling* caused by cavities, and *Kármán vortices* caused by thin objects.

6 https://github.com/SkAT-VG/SDT.git.
7 https://www.gnu.org/software/make/.

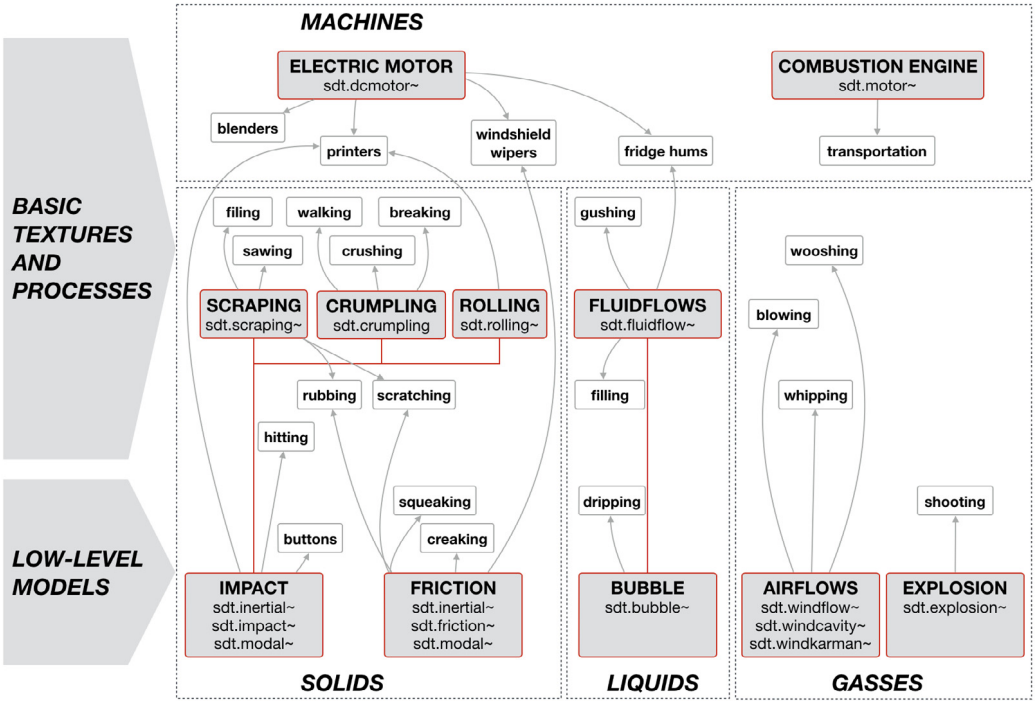

Fig. 1. The SDT taxonomy of sound models. The bottom-up hierarchy represents the dependencies between low-level models and temporally patterned textures and processes, for the four classes of sounds: solids, liquids, gasses, and machines.

Explosions. Explosions are characterized by a powerful shock-wave causing a blast wind, with a sudden positive peak in pressure followed by a negative expansion tail. The SDT algorithm simulates explosions with a Friedlander waveform, which closely approximates this behavior. As real world explosions are never perfectly impulsive, scattering and other types of turbulence are rendered by a feedback delay network reverberator [19].

Machines. The main timbral aspect of machine sounds is the presence of an engine, producing pitched tones based on its duty cycle. The Sound Design Toolkit provides models for:

- *Combustion engines:* Four periodic waveforms, representing the moving parts of each combustion chamber (intake and outtake valves, piston, fuel ignition), model the engine block. These waveforms are then used as input signals or feedback gain modifiers for a set of interconnected digital waveguides, which model the pipes composing the exhaust system [20];
- *Electric motors:* A sum of sinusoidal partials renders the pitched tones generated by the rotor and by the connected gears. Rotor partials are also used as amplitude envelope for filtered noise, to model the sound produced by motor brushes on the commutator ring. Frequency modulation of those partials simulates imperfections in the balance of the rotor and mechanical stress. A comb filter reproduces chassis resonances, while continuous filtered noise emulates air noise coming from the rotating parts and from the cooling fan.

Processing and analysis. The toolkit also includes a set of sound processing algorithms (such as pitch shifting and reverb) which have been exploited to enrich spatial and environmental attributes of the basic sound models. For instance, a feedback delay network reverb unit has been used to thicken and propagate micro-textures in rubbing/scraping sound modeling [19]. Several audio feature extractors (fundamental frequency estimator, pitch clarity, signal magnitude, zero crossing rate, statistical moments of the spectrum, onset) have also been developed to support the audio signal control of the sound synthesis models, for instance through vocalizations. These features found their application in the development of *miMic*, described in Section 4.

3. Illustrative examples

In its Max release, the SDT has been organized as *package*, that is a folder with a prescribed structure aimed at bundling objects, media, patchers, and resources for distribution, usually installed in the `Max/Packages` folder. The palette of sound models, available as Max externals, is accessible in the *SDT Overview* patch from the Max *extras* menu. The sound models are arranged according to the type of interaction primitive (i.e., vibrating solids, liquids, gasses) and the hierarchy of dependencies represented in Fig. 1. From the same menu, the *SkAT-VG Overview* patch provides example patches of compound sound models (i.e., *timbral families*). Each external is provided with a *help* patch, with a self-explanatory GUI, the detailed description of input, output, attributes and arguments, and a few basic presets to browse.

Fig. 2 shows the help patch for the impact sound model, that is a nonlinear impact between one `[sdt.inertial]` resonator and one `[sdt.modal]` resonator, according to the characteristics of the collision set in the `[sdt.impact~]` interactor. The UI controls allow the adjustment of:

- Mass, applied force and initial velocity of the striking object;
- Frequencies, amplitudes and decay times of the resonating modes of the struck object;
- Stiffness, dissipated energy and contact shape of the impact.

The example in Fig. 2 shows a metal sound, produced with only three modes of resonance set at 500 Hz, 509 Hz and 795 Hz. By reducing the *global decay factor* to a value of 0.02, the metal hit turns into a wooden knock. Changing the *mass* value to 0.1 together with the *global frequency factor* to 0.7 would affect the perceived size of the wooden struck object. By setting the frequency factor value to 4, the resulting impact recalls the sound of ceramics.

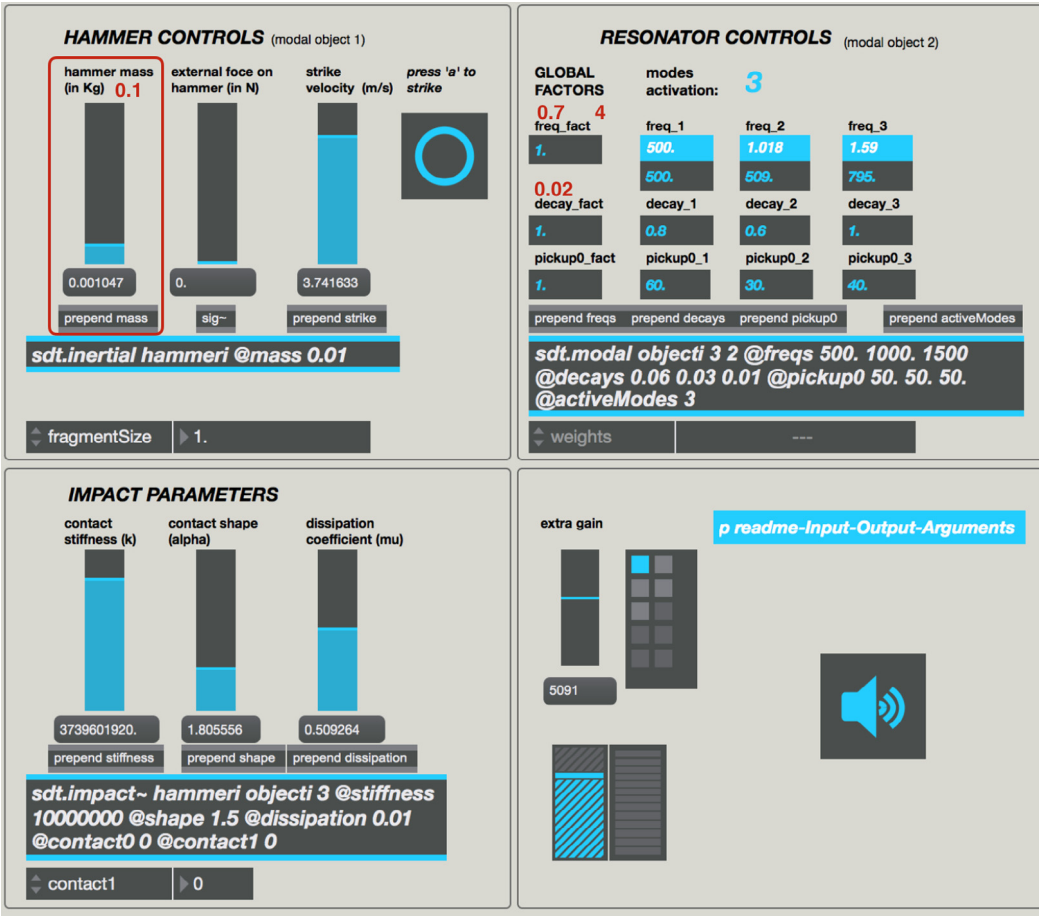

Fig. 2. The help patch of the impact sound model, describing a non-linear impact between one inertial and one modal object. With a few changes (values in red) it is possible to achieve a wide variety of sounding objects and materials. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4. Impact

The development of the SDT goes along with research in sound-mediated interaction and design. The toolkit has been used in a set of basic design exercises, around a wide range of explorations on auditory perception in interaction, such as rhythmic slicing, mechanical connection in continuous screwing, controlling perceptual hierarchies in auditory pattern design, and exploiting sensory contradiction in the action–sound loop. Experimentations with the SDT have facilitated the systematization of a basic design language made out of a lexicon of a few sound–action bricks [21,22].

Modern theories on embodiment debate on the essence of internal representations, as forms of previously experienced perception–action binds. Put in sound design terms, one can argue that the limits of our sonic imagery are our experiences. Indeed, recent experiments on perception of vocal imitations, identification and categorization of sound events have highlighted the primary role of basic mechanisms of physical sound production, such as impacts, frictions, gas turbulence, fluids, and rotary mechanisms [5]. Similarly, the physiomechanical production of vocal sounds can be analyzed in terms of turbulence, myoelastic vibration, impacts, and phonation. The SDT taxonomy of sound models and its design rationale, as described in Section 2.2, take place at the crossroad of these research tracks. The sound design implication is that to actually compose sounds is to exploit a reduced, emerging, basic lexicon of physically grounded phenomena, that can be further arranged and outlined in potentially infinite variations [21]. In this respect, *S'I Fosse Suono*[8] (lit. If I were sound), is an artistic proof-of-concept of vocal sketching, showing a collection of sixteen self-audio portraits, presented in (i) their original non-verbal vocal version, (ii) the SDT-translated version, (iii) a realization using granular manipulations of the recorded vocalizations. The installation was realized in collaboration with the sound designer Andrea Cera.

These concepts have been further explored in the development of *miMic*,[9] a microphone augmented by inertial sensors for vocal and gestural sketching [23]. In this application, the SDT is used to synthesize sonic sketches through vocalizations and gestures. Evaluating the effectiveness of SDT in design contexts is a difficult and time-consuming endeavor that requires the conduction and analysis of design exercises. Several design workshops have been organized for this purpose, and the analysis of designers' behaviors is ongoing research work [24].

Use of the software is being promoted among sound designers and artists through workshops and other dissemination initiatives. Worthy of notice is the use of SDT sound models in an artistic realization by Robin Meier.[10] The SDT is also being extensively used in the sonic interaction design community for design research and educational purposes. The new software architecture makes the SDT prone to experimentation on sound modeling, numerical discretization and signal processing.

8 S'i' Fosse Suono (2015) – http://www.skatvg.eu/SIFosse/.
9 https://vimeo.com/142351022.
10 Fossil Records (2015) – http://robinmeier.net/?p=2049.

The API exposed by the core SDT library has been extensively documented in Doxygen, to facilitate integration of the SDT features into different applications in the near future.

5. Conclusions

The Sound Design Toolkit is an open-source, cross-platform software extension serving as a workbench for cross-disciplinary research in sound perception, synthesis, and design. It offers a family of sound models based on physical descriptions of sound-producing phenomena, organized in a taxonomy that makes sense to humans. As the sound models do not need to be programmed, but only acted on through continuous manipulation, future work will be aimed at facilitating the use of SDT among practitioners.

Acknowledgments

Stefano Papetti has been the main developer of the SDT legacy code. We acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number: 618067.

References

[1] Delle Monache S, Polotti P, Rocchesso D. A toolkit for explorations in sonic interaction design. In: Proceedings of the 5th audio mostly conference: A conference on interaction with sound. New York, NY, USA: ACM; 2010. p. 1:1–7. http://dx.doi.org/10.1145/1859799.1859800.
[2] Farnell A. Designing Sound. Cambridge, MA: Mit Press; 2010.
[3] Rocchesso D. Sounding objects in europe. New Soundtrack 2014;4(2):157–64.
[4] Cipriani A, Giri M. Electronic music and sound design: Theory and practice with Max and MSP. Rome, Italy: Contemponet; 2010.
[5] Lemaitre G, Dessein A, Susini P, Aura K. Vocal imitations and the identification of sound events. Ecol Psychol 2011;23(4):267–307.
[6] Houix O, Lemaitre G, Misdariis N, Susini P, Urdapilleta I. A lexical analysis of environmental sound categories. J Exp Psychol Appl 2012;18(1):52. http://dx.doi.org/10.1037/a0026240.
[7] Gaver WW. What in the world do we hear?: An ecological approach to auditory event perception. Ecol Psychol 1993;5(1):1–29.
[8] Rocchesso D, Bresin R, Fernström M. Sounding objects. IEEE Multimedia 2003;10(2):42–52.
[9] Rath M, Rocchesso D. Continuous sonic feedback from a rolling ball. IEEE Multimedia 2005;12(2):60–9.
[10] Pruvost L, Scherrer B, Aramaki M, Ystad S, Kronland-Martinet R. Perception-based interactive sound synthesis of morphing solids' interactions. In: SIGGRAPH Asia 2015 Technical briefs. New York, NY, USA: ACM; 2015. p. 17:1–4. http://dx.doi.org/10.1145/2820903.2820914.
[11] Cook PR, Scavone G. The synthesis toolkit (STK). In: Proceedings of the international computer music conference; 1999. p. 164–6.
[12] Csapó A, Wersényi G. Overview of auditory representations in human-machine interfaces. ACM Comput Surv 2013;46(2):19:1–23. http://dx.doi.org/10.1145/2543581.2543586.
[13] Baldan S, Monache SD, Mauro DA, Rocchesso D, Lachambre H. Automatic system for the generation of sound sketches, deliverable of project SkAT-VG. Iuav University of Venice; 2015 URL http://skatvg.iuav.it/?page_id=388.
[14] Lemaitre G, Voisin F, Scurto H, Houix O, Susini P, Misdariis N, Bevilacqua F. A Large set of vocal and gestural imitations, deliverable of project SkAT-VG. Ircam Institut de Recherche et Coordination Acoustique/Musique; 2015 URL http://skatvg.iuav.it/?page_id=388.
[15] Papetti S, Avanzini F, Rocchesso D. Numerical methods for a nonlinear impact model: a comparative study with closed-form corrections. IEEE Trans Audio Speech Lang Process 2011;19(7):2146–58.
[16] Avanzini F, Serafin S, Rocchesso D. Interactive simulation of rigid body interaction with friction-induced sound generation. IEEE Trans Speech Audio Process 2005;13(5):1073–81.
[17] Zambon S. Accurate sound synthesis of 3D object collisions in interactive virtual scenarios [Ph.D. thesis]. Verona, Italy: University of Verona; 2012.
[18] Doel KVD. Physically based models for liquid sounds. ACM Trans Appl Percept 2005;2(4):534–46. http://dx.doi.org/10.1145/1101530.1101554.
[19] Rocchesso D, Baldan S, Dell. Monache S. Reverberation still in business: thickening and propagating micro-textures in physics-based sound modeling. In: Proc. of the 18th international conference on digital audio effects; 2015. p. 12–8.
[20] Baldan S, Lachambre H, Delle Monache S, Boussard P. Physically informed car engine sound synthesis for virtual and augmented environments. In: Proc. 2nd VR workshop on sonic interactions for virtual environments. France: Arles; 2015. p. 1–6. http://dx.doi.org/10.1109/SIVE.2015.7361287.
[21] Delle Monache S, Rocchesso D. Bauhaus legacy in research through design: the case of basic sonic interaction design. Internat J. Des 2014;8(3):139–54.
[22] Rocchesso D, Polotti P, Delle Monache S. Designing continuous sonic interaction. Int J Des 2009;3(3):13–25.
[23] Rocchesso D, Mauro DA, Delle Monache S. miMic: The microphone as a pencil. In: Proceedings of the TEI'16: Tenth international conference on tangible, embedded, and embodied interaction. ACM; 2016. p. 357–64.
[24] Erkut C, Rocchesso D, Monache SD, Serafin S. A case of cooperative sound design. In: Proceedings of the 9th nordic conference on human-computer interaction, NordiCHI '16. New York, NY, USA: ACM; 2016. http://dx.doi.org/10.1145/2971485.2996472. 83:1–83:6.