BLOCKCOMPILER — A RESEARCH TOOL FOR PHYSICAL MODELING AND DSP

Matti Karjalainen

Laboratory of Acoustics and Audio Signal Processing
Helsinki University of Technology, Espoo, Finland
matti.karjalainen@hut.fi

ABSTRACT

This paper describes an experimental research tool for block-based physical modeling and DSP computation. The goals of the development have been high abstraction level and flexibility in model specification without compromising computational efficiency in real-time simulation and application execution. To achieve both goals, the Lisp language is used for symbolic manipulation of computational block structures and C language for compilation of efficient executables. The primary motivation for this tool has been to enable flexible generation of physical models where two-directional interaction between elements is needed. A particular feature of the system is support for mixed modeling by combining digital waveguides, finite difference schemes, wave digital filters, as well as traditional block-based DSP algorithms.

1. INTRODUCTION — WHY YET ANOTHER BLOCK-BASED SYSTEM?

This work emerged from the need to easily write efficient simulation and sound synthesis programs for different kinds of acoustic and audio applications using a multi-paradigm approach. The idea of block-based computation itself is commonly utilized; there is a multitude of software environments such as Max/MSP and jMax, Pd, LabView, Simulink, etc., that are based on wiring of DSP blocks, using a graphical interface or textual scripting. There are also software environments particularly for simulating distributed systems (FEM, BEM, FDTD) or lumped element systems (circuit simulators). None of them is, however, a truly multi-paradigm environment for flexible yet efficient simulation and real-time synthesis of physical systems.

2. OVERVIEW OF THE BLOCKCOMPILER

The BlockCompiler is an experimental software environment using block objects and their interconnection networks for high-level specification of computational models. The first level of objects supports conventional DSP and one-directional signal data flow between objects. This includes elementary blocks such as adders, multipliers, nonlinear functions, filters, transformations, and sound I/O through PortAudio sound drivers. In addition to synchronous signal flow, there is a possibility of parametric control flow that supports asynchronous communication.

A more advanced level supports modeling of physical (two-way) interactions. The elements are connected through two-way ports that carry physical signal variables, such as force, pressure, velocity, voltage, current, etc. In this sense the system works as a circuit and network simulator. The structures may also be 2D and 3D meshes, which is important in simulating for example musical instruments or acoustic spaces. The models are based on Digital Waveguide (DWG) [1] and Finite Difference (FDTD) [2] principles. Lumped element systems can be added through Wave Digital Filter (WDF) [3] principles.

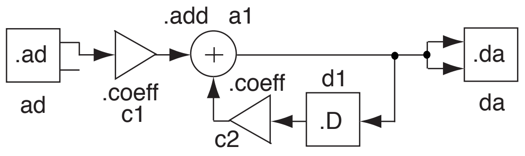


Figure 1: Simple low-pass filter from sound driver input to output.

2.1. Simple DSP scripting example

A full model specification in BlockCompiler is called a *patch*, which consists of interconnected *block-item* units.

Figure 1 shows a simple DSP example of a low-pass filter from sound input (ad-converter) to sound output (da-converter). This patch can be created by scripting in Lisp notation:

```
(patch ((ad (.ad))               ; sound in
        (c1 (.coeff 0.0666))     ; coeff
        (a (.add :inputs 2))     ; adder
        (c2 (.coeff -0.8668))    ; coeff
        (d1 (.d))                ; unit delay
        (da (.da)))              ; sound out
   (connect (out ad 0) (in c1))
   (connect (out c1) (in a 0))
   (connect (out a) (in d1))
   (connect (out d1) (in c2))
   (connect (out c2) (in a 1))
   (connect (out a) (in da 0))
   (connect (out a) (in da 1)))
```

where forms in parentheses such as `(.ad)` create related objects and assign them to local variables such as `ad`. Forms `(connect out in)` connect an output to an input. The first output or input of a block is used unless index or symbol name is also given, such as `(in a 1)` for the second input of the adder (index 1). The same patch can we written more compactly in short-form notation by applying the chaining function (->) by:

```
(patch ((a (.add)))
  (-> (.ad) (.coeff 0.0666) a (inputs (.da)))
  (-> a (.d) (.coeff -0.8668) (in a 1)))
```

Function forms `(-> obj1 obj2 ...)` connect these block objects from output to input, output to input, etc. This allows for compact scripting of patches, and powerful symbolic manipulation ability by the Lisp language provides high flexibility.

2.2. Properties of the BlockCompiler

• **Macro blocks** can be defined as combinations of more elementary blocks to hide the details of new macro class objects. The macro blocks can be utilized further hierarchically as elements of new macro definitions. Parameters can be given to specify the properties of a macro block during instantiation. A simple example of macro block definition, implementing the low-pass filter used in a patch related to Fig. 1, is:

```
(def-macro-block ((block .lpf) &key
                   coeff1 coeff2)
  (let ((c1 (.coeff coeff1))
        (c2 (.coeff coeff2))
        (a (.add)))
   (-> c1 (in a 0) (.d) c2 (in a 1))
   (set-inputs (in c1))
   (set-outputs (out a)))
```

which defines a block class named `.lpf` and a make-function of the same name for the low-pass filter part of the patch. Keywords given after `&key` are available to specify the properties of the macro-block instances. Elementary blocks `c1` and `c2`, i.e., the filter coefficients, as well as adder `a`, are specified and linked next. Finally, the inputs and outputs of the macro block are set as inputs and outputs of the elementary blocks.

The low-pass filtering patch of Fig. 1 can be made now by:

```
(defparameter lpstream
  (patch ()
    (-> (.ad)
        (.lpf :coeff1 0.0666 :coeff2 -0.8668)
        (inputs (.da)))))
```

• **Data types** (short, long, float, double) and corresponding array types are available for runtime signal data. Both scalar and vectorized operations are supported. On the level of model specification (in Lisp) there is strong support for object-based programming and scripting.

• **Data flow** between block outputs and inputs is multirate synchronous. This means that each block can be given an individual sample rate as an integer multiple of fraction of the master rate of the patch. Polyphase synchronization is possible.

Blocks can have parametric inputs (`param` inputs) that support asynchronous communication with or without handshaking. Combined with the multirate feature this makes possible to optimize the rate of control parameter updating.

• **Scheduling**. When a patch has been specified and instantiated as an interconnected set of blocks, the operations in it are scheduled. This is done by walking through the blocks hierarchically and setting them into a computable order, i.e., checking whether the inputs and outputs are ready for the operation. Arbitrary graphs are allowed unless there are delay-free loops, which condition is reported as an error.

• **Code generation and compilation**. Each block class has a method that generates C code for computation of the block. Elementary (non-macro) blocks are specified so that C-code generation is defined by pseudo-C, which is C source code except data references being to Lisp specifications of the data items.

A scheduled patch generates inline C code from the ordered block operations and writes it into a file in the form of a single C function and related data definitions. When the source file is written, it will be compiled on the fly by a call to GCC compiler. For simple-to-moderate size patches the compilation takes only a fraction of a second to a few seconds. Finally a function pointer from the compiled file is made and prepared for execution.

• **Patch streaming and stepping**. There are two modes of executing a patch: streaming and stepping. In the real-time streaming mode a patch is timed by sound driver sampling rate by linking sound input blocks (`.ad`) and output blocks (`.da`) to other DSP or physical model blocks. Streaming *patchx* is started by function `(run-patch` *`patchx`*`)`. While real-time streaming is running, the patch is fully controllable from Lisp, allowing for highly flexible control and inspection of patch behavior.

In the stepping mode a single sample step is computed every time a form `(step-patch` *`patchx`*`)` is executed. This makes it possible to do non-realtime processing as well as debugging real-time patches.

• **Graphic user interface**. Presently the BlockCompiler doesn't have a graphic editor for creating patches. The preference to programming and scripting the patches was given since the software was developed primarily for research purposes, where textual programming is found more powerful than graphical specification. Since controlling of patch parameters is an important feature of any real-time synthesis environment, graphic controllers such as sliders are available already now. An example thereof is presented below in the form of an extended Karplus-Strong string model. It is probable that a full-blown graphic user interface will be added in the future.

• **Model export**. An important property of the BlockCompiler is support for exporting patch models to other execution platforms. This is based on the fact that BlockCompiler is basically a C code generator. So far the code export feature is fully implemented for Mustajuuri [4], an audio signal processing platform.

3. DISCRETE-TIME PHYSICAL MODELING

The main motivation for BlockCompiler was the potential to create computationally efficient physical models in a flexible way. Instead of one-directional data flow, true physical models require two-directional interaction via the ports of physical blocks.

In time-domain discrete-time modeling of physical systems the task is to convert the underlying (partial) differential equations into approximating difference equations then to be solved. Formulation of the solution as DSP algorithms makes them computable by efficient software tools for real-time simulation. We next discuss a systematic DSP formulation of the two main physical modeling approaches of interest, the digital waveguides (DWG) and the finite difference time-domain schemes (FDTD), and then add wave digital filters (WDF) to this modeling framework.

In a one-dimensional lossless medium the wave equation is written

$$y_{tt} = c^2 y_{xx} \tag{1}$$

where y is a wave variable, subscript tt refers to second partial derivative in time t, xx to second partial derivative in place variable x, and c is speed of wavefront in the medium of interest.

3.1. Wave-based modeling

The 1-D traveling wave formulation is based on the d'Alembert solution of propagation of two opposite direction waves, i.e.,

$$y(t,x) = \overrightarrow{y}(t - x/c) + \overleftarrow{y}(t + x/c) \tag{2}$$

where the arrows denote the right-going and the left-going components of the total waveform. Assuming that the signals are bandlimited to half of sampling rate, the traveling waves can be sampled without losing any information by selecting T as the sample interval and X the position interval between samples so that $T = X/c$. Sampling is applied in a discrete time-space grid in which n and m are related to time and position, respectively. The discretized version of Eq. (2) [1] becomes:

$$y(n,m) = \overrightarrow{y}(n-m) + \overleftarrow{y}(n+m) \tag{3}$$

It follows that the wave propagation can be computed by updating state variables in two delay lines by

$$\overrightarrow{y}_{k,n+1} = \overrightarrow{y}_{k-1,n} \quad \text{and} \quad \overleftarrow{y}_{k,n+1} = \overleftarrow{y}_{k+1,n} \tag{4}$$

i.e., by simply shifting the samples to the right and left, respectively. This kind of discrete-time modeling is called Digital Waveguide (DWG) modeling [1]. Since the physical wave variables are split explicitly into directional wave components, we will call such models *W-models*.

The next step is to take into account the global physical constraints of continuity by Kirchhoff type of rules. This means to formulate the scattering junctions of interconnected ports, with given impedances and wave variables at related ports. For a scattering junction, where the physical variables are sound wave pressure P and volume velocity U, and when a parallel admittance model [1] of N ports is utilized, the Kirchhoff constrains become

$$P_1 = P_2 = \ldots = P_N = P_J \tag{5}$$

$$U_1 + U_2 + \ldots + U_N + U_{\text{ext}} = 0 \tag{6}$$

where P_J is the common pressure of coupled branches and U_{ext} is an external volume velocity to the junction. When port pressures are represented by incoming wave components P_i^+, outgoing wave components by P_i^-, admittances attached to each port by Y_i, and

$$P_i = P_i^+ + P_i^- \quad \text{and} \quad U_i^+ = Y_i P_i^+ \tag{7}$$

the junction pressure P_J can be obtained as:

$$P_J = \frac{1}{Y_{\text{tot}}}(U_{\text{ext}} + 2\sum_{i=0}^{N-1} Y_i P_i^+) \tag{8}$$

where $Y_{\text{tot}} = \sum_{i=0}^{N-1} Y_i$ is the sum of all admittances to the junction. Outgoing pressure waves, obtained from Eq. (7), are then $P_i^- = P_J - P_i^+$. The resulting junction, a *W-node*, is depicted as a DSP structure in the N_1 node of Fig. 2 (top). When admittances Y_i are frequency-dependent, this diagram can be interpreted as a filter structure where the incoming pressures are filtered by the corresponding wave admittances Y_i times two, and their sum is filtered further by $1/Y_{\text{tot}}$ to get the junction pressure P_J.

Two special cases can be noticed on the basis of Eq. (8). First, a (passive) loading admittance is the case with Y_i where no incoming pressure wave component P_i^+ is associated. This needs no computation except including Y_i in Y_{tot} because $P_i^+ = 0$, see the left-hand termination, a *W-admittance*, in Fig. 2. Another issue is the external velocity U_{ext} effective to the junction. This is connected directly to the summation at the junction node.

The W-node in Fig. 2 is coupled through *W-ports* to the neighboring elements (port 3 is uncoupled). The right-hand side block

[1]Models can be formulated as well for impedances instead of admittances and for series connection, and different physical variable pairs can be used.

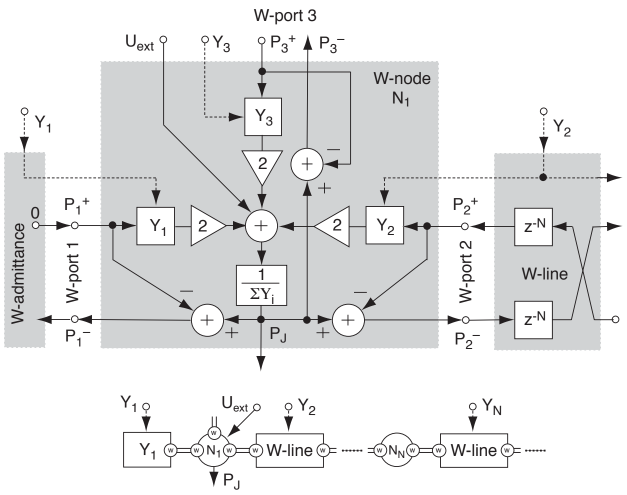


Figure 2: Top: A 3-port scattering junction (W-node N_1). Incoming pressures are P_i^+ and outgoing ones P_i^+. W-port 1 is connected to termination W-admittance Y_1 and port 2 to a two-directional delay line (W-line). Admittance controls are marked by dashed lines. Bottom: Block diagram with abstracted blocks and how they can be connected to form a 1-D DWG waveguide.

is a two-directional delay line, a *W-line*, of admittance Y_2. The bottom part of the figure depicts a block diagram abstraction of the DSP structure. It also characterizes how waveguides are built as structures of W-line elements connected by W-node junctions.

Notice that the admittances in Fig. 2 may be real-valued or frequency-dependent so that Y_i and the impedance $1/\sum Y_i$ can be realized as FIR or IIR filters, or just as real coefficients if all attached admittances are real. In the latter case, if we skip the external velocity U_{ext} of Eq. (8), we may write the equation using scattering parameters α_i as $P_J = \sum_{i=0}^{N-1} \alpha_i P_i^+$, where $\alpha_i = 2Y_i/Y_{\text{tot}}$. This and other special forms of scattering [1] are efficient computationally when admittances are real-valued, but in a general case it is practical to implement computation as shown in Fig. 2 so that the term $1/\sum Y_i$ is a common filter.

Dashed lines in Fig. 2 are parametric controls for admittances of the network elements. If the DSP blocks are grouped as shown, which is natural in an object-based formulation, the junction W-nodes actually contain most of the computation by implementing wave scattering. The W-node is delegated its admittance parameters through ports from the network elements, W-lines and W-admittances. In a time-varying case the admittance filters (blocks Y_1, Y_2, and Y_3) as well as the inverse of their sum $1/\sum Y_i$ must be updated when the admittance control parameters change.

3.2. Finite difference modeling

In the most commonly used way to discretize the wave equation by finite differences the partial derivatives in Eq. (1) are approximated by second order finite differences

$$y_{xx} \approx -(2y_{x,t} - y_{x-\Delta x,t} - y_{x+\Delta x,t})/(\Delta x)^2 \tag{9}$$

$$y_{tt} \approx -(2y_{x,t} - y_{x,t-\Delta t} - y_{x,t+\Delta t})/(\Delta t)^2 \tag{10}$$

By selecting the discrete-time sampling interval Δt to correspond to spatial sampling interval Δx, i.e., $\Delta t = c\Delta x$, and using index notation $k = x/\Delta x$ and $n = t/\Delta t$, Eqs. (9) and (10) result in

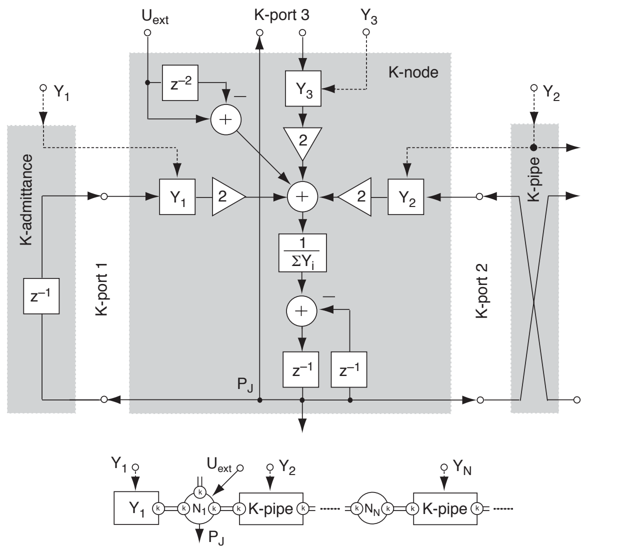


Figure 3: Top: Digital filter structure for finite difference approximation of a two-port scattering node with port admittances Y_1 and Y_2. Only total pressure P_J (K-variable) is explicitly available. Bottom: Block diagram with abstracted blocks and how they can be connected to form a 1-D FDTD waveguide.

$$y_{k,n+1} = y_{k-1,n} + y_{k+1,n} - y_{k,n-1} \tag{11}$$

which is a *K-model*, i.e., using Kirchhoff type of variables, not wave components. From form (11) we can see that a new sample $y_{k,n+1}$ at position k and time index $n+1$ is computed as the sum of its neighboring position values minus the value at the position itself one sample period earlier.

The behavioral similarity of DWGs vs. FDTDs [7], although being computationally different formulations (W- vs. K-models), hints to expand Eq. (11) to a FDTD type scattering junction for arbitrary port admittances. For a parallel admittance model, corresponding to Eq. (8), Eq. (11) can be formulated for N ports as

$$P_{\mathrm{J},n+1} = \frac{2}{Y_{\mathrm{tot}}} \sum_{i=0}^{N-1} Y_i P_{i,n} - P_{\mathrm{J},n-1} \tag{12}$$

This is a waveguide mesh formulation as discussed in [5]. Figure 3 depicts a DSP formulation of one such 3-port scattering *K-node* and the way to terminate port 1 by *K-admittance*, Y_1. This corresponds to the W-model in Fig. 2, except that a wave traveling to the left reflects back from Y_1 one unit delay later than in the DWG case. Notice the feedback through a unit delay. There can be any number of ports attached to a node also here as for a DWG junction. The bottom part of Fig. 3 depicts a block diagram abstraction which shows the conformity with the DWG in Fig. 2.

An essential difference between DWGs of Fig. 2 and FDTDs of Fig. 3 is that while DWG junctions are connected through two-directional delay lines (*W-lines*), FDTD nodes have two unit delays of internal memory, and delay-free *K-pipes* connect ports between nodes (see the right-hand side block in Fig. 3). The DWG and FDTD junction nodes and ports are not directly compatible because they use different type of wave variables.

More details about FDTD modeling vs. DWG modeling are presented in [6].

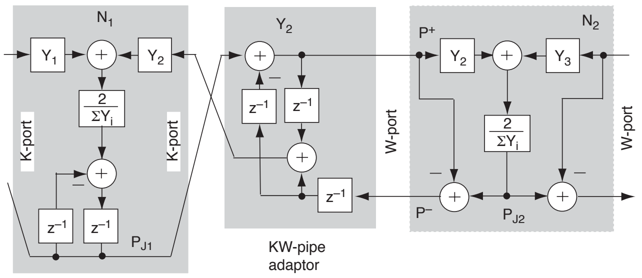


Figure 4: FDTD node (left) and a DWG node (right) forming a part of a hybrid waveguide. Y_i are wave admittances of W-lines, K-pipes, and adaptor KW-pipes between junction nodes. P_J are junction pressures, P^+ and P^- are wave components.

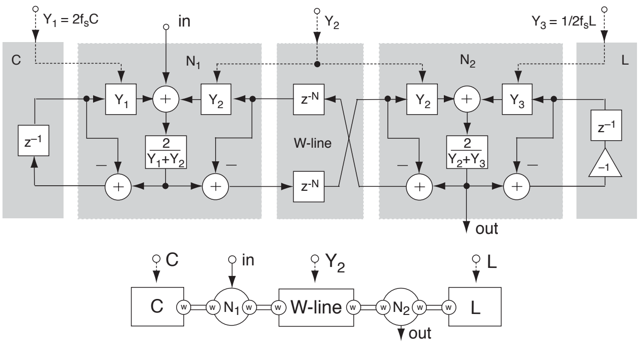


Figure 5: A simple DWG+WDF resonator where a DWG delay line is terminated with a WDF capacitor (left) and inductor (right).

3.3. Interfacing DWGs and FDTDs

The next problem is how to interface wave-based and FDTD-based submodels. In [**?**] it was shown how to interconnect a lossy 1-D FDTD waveguide with a similar DWG waveguide into a mixed model using a proper interconnection element (adaptor). As a generalization, it is possible to make any hybrid model of K-elements (FDTD) and W-elements having arbitrary wave admittances/impedances at their ports.

Figure 4 shows how this can be done in a 1-D waveguide between a K-node N_1 (left) and a W-node N_2 (right). The role of the KW-pipe in the middle of Fig. 4 is to adapt the K-type port of an FDTD node and the W-type port of a DWG node. It is delay-free in left-to-right direction and contains delay in the opposite direction.

The equivalence of W- and K-variable based models and the availability of the adaptor allow now to implement mixed models where both of the approaches can be applied, depending on which one is more useful in a problem at hand. Generally the DWG elements are preferable in 1-D modeling due to good numerical properties and possibility of arbitrary (including fractional) delays, while the FDTDs are more efficient in 2-D and 3-D structures, being however more critical in numerical accuracy.

3.4. Interfacing wave digital filters with DWGs

An additional technique to the mixed modeling framework above is to adopt Wave Digital Filters (WDF) [3] as discrete-time simulators of lumped parameter elements. Being based on W-modeling, i.e., wave variables, they are computationally compatible with the W-type DWGs [7]. A WDF *resistor* is basically just a real-valued termination (see Y_1) in Fig. 2, but WDF *capacitors* and *inductors*,

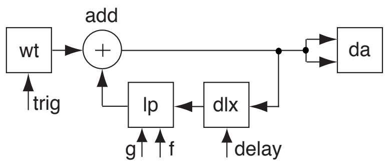
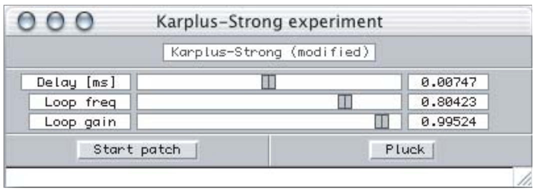


Figure 6: Block diagram of an extended Karplus-Strong model as a case of semiphysical modeling.



Karplus-Strong (modified)

Delay [ms]	0.00747
Loop freq	0.80423
Loop gain	0.99524

Figure 7: Control dialog for the Karplus-Strong model.

as well as *ideal transformers* and *gyrators*, etc., are useful additional components [3].

As a physically bound choice for the case of this study, a WDF capacitor is realized as a feedback from V^- wave of a port back to V^+ through a unit delay, having a port admittance $2f_sC$. A WDF inductor is a feedback through a unit delay and coefficient -1, having a port admittance $1/2f_sL$. Here C is capacitance, L is inductance, and f_s is sample rate (cf. [7]).

Figure 5 shows an example of a model where a DWG delay line is terminated by a WDF capacitor at the left hand side and by a WDF inductor at the right hand side.

WDF elements, being W-models, are not directly compatible with DFTDs that are K-models. However, the compatibility can be realized through a KW-adaptor element if needed.

4. PHYSICAL MODELING EXAMPLES

In this section we describe by case examples how BlockCompiler is used for physically based modeling. The first example is an extended Karplus-Strong string model with user interface for parameter control. The second example shows how a string instrument is built by coupling the strings via a common bridge impedance. The third example is a waveguide formulation of a vocal tract speech synthesizer, and the final case is to show how to make a mixed FDTD+DWG waveguide mesh.

An extended Karplus-Strong string model, depicted in Fig. 6 by a block diagram, is an example of 'semiphysical' modeling. The patch is created by script:

```
(defparameter ks-string
  (patch ((d (.slider
               :title " Delay [sec]" ; slider
               :low 0.001 :high 0.02 ; delay
               :init-value 0.0075))  ; [sec]
          (f (.slider
               :title " Loop freq"   ; slider
               :low 0.0 :high 1.0    ; freq
               :init-value 0.80))
          (g (.slider
               :title " Loop gain"   ; slider
               :low 0.9 :high 1.0    ; gain
               :init-value 0.995))
          (trig (.trig-button))
```

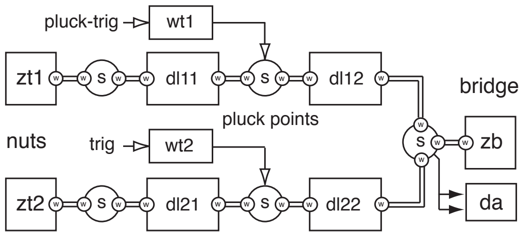


Figure 8: Block diagram of two strings coupled through a common bridge impedance (zb).

```
          (wt (.wtable :data *noise-data*))
          (dlx (.delay :delay-time 0.02
                       :control '.lg3))
          (add (.add ))
          (lp (.lp1)))
    (-> d (.smooth) (param dlx 'delay))
    (-> f (.smooth) (param lp 'freq))
    (-> g (.smooth) (param lp 'gain))
    (-> trig (param wt 0))
    (-> wt add dlx lp (in add 1)
        (inputs (.da)))
    (patch-window
     (:v (.text-box "Karplus-Strong String")
         (sliders (list d f g) :length 200)
         (buttons (list (.start/stop) trig)))
     :title "Karplus-Strong experiment"
     :view-position #@(350 150)
     :view-size #@(450 113))))
```

The resulting patch is a single-delay-loop structure with a controllable length delay line (by Lagrange interpolation of order 3, .lg3) and a first order low-pass loop filter with DC gain and low-pass frequency control, the string model being triggerable from a wavetable (filled by a noise burst in this case).

The feature of intrest in this case is the use of parameter control blocks by sliders in a graphic user interface window depicted in Fig. 7. The sliders (`d`, `f`, and `g`) in the script are connected to parameter inputs through smoothing (`.smooth`) interpolators to avoid gliches. Finally the patch control window is created by a layout specification (`patch-window ...`).

As an example of true physical modeling by digital waveguide approach, Fig. 8 shows a string instrument of two strings connected by a common bridge impedance. The model is composed of delay lines, junction nodes, impedances, and triggerable excitation wavetables. On the left hand side of Fig. 8 we can see (nut) terminations (`zt`) of the two strings. Delay lines (`dl`) make the strings in two parts so that the (series junction) nodes in the middle are insertion points for pluck (force) excitation from wavetables (`wt`). True physical interaction of a string and a plucking object could be modeled as well but for simplicity we show here a wavetable excitation case. On the right hand side the bridge impedance (`zb`) is a common termination for both strings, implementing a physically correct connection of these elements by a series junction node. The velocity of the bridge node is the model output that could be further processed by a filter for sound radiation from the body.

Notice that the nodes (junctions) can accommodate any number of attached DWG delay lines with arbitrary, even time-varying impedances. The formulation is general in the sense that it can

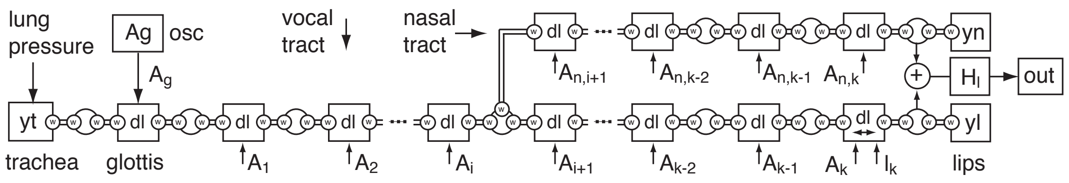


Figure 9: DWG transmission-line speech production model, including nasal-tract, made of (mostly) constant length sections.

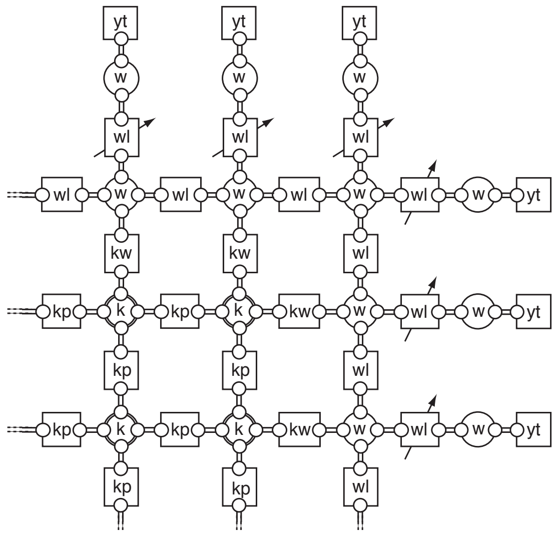


Figure 10: Part of a 2-D waveguide mesh composed of K-type FDTD elements (left bottom): K-pipes (kp) and K-nodes (k), W-type DWG elements (top and right): delay-controllable W-lines wl, W-nodes (w), and terminating admittances (yt), adaptor elements (kw) to make a mixed model.

be used as a circuit/network simulator having arbitrary (spatially) distributed elements such as DWGs and FDTDs as well as lumped elements such as WDF capacitors, inductors, and resistances in series or parallel connections. Acually any impedances/admittances, expressed by z-transforms (FIR or IIR forms), can be used.

As another case of physical modeling, Figure 9 illustrates an advanced speech production transmission-line model, which however follows traditional guidelines. The tract is divided into a set of constant length sections whereby acoustic admittances $Y(i)$ can be controlled according to their cross-sectional area dependency $Y(i) = A(i)/\rho c$, where ρ is air density and c is speed of sound. Parametric control of vocal tract shape can be based for example on mapping from articulatory parameters, or by contextual lookup and interpolation in time. Fine-tuning of the tract length can be made at the lips by a single controllable fractional delay line section.

The glottis is realized as a single section of vocal tract with varying area, controlled by a glottal waveform oscillator. Lung pressure makes the volume flow through the glottis to be in relation to its opening. More advanced nonlinear models of self-oscillation can also be experimented easily.

The termination in the model of Fig. 9 at lips and nostrils includes a filter H_l for detailed lip pressure to far-field radiation function. Other functionalities that can be experimented relatively easily are for example generation of turbulence frication and bursts in constrictions and during the opening of occlusion.

As a 2-dimensional case of physical modeling, Fig. 10 depicts a part of a rectangular mesh structure that is composed of FDTD elements for efficient and memory-saving computation and DWG elements for boundaries. Such model could be for example a membrane of a drum or in a 3-D case a room enclosed by walls.

In Fig. 10 the elements denoted kp are K-type pipes between K-type nodes. Elements kw are K-to-W adaptors and elements fd are (controllable) fractional delays. Elements yt are terminating admittances. In a general case the admittances of each element can be different, thus allowing for instance for non-homogeneous membrane models.

5. SUMMARY

This paper has described an experimental software environment called the BlockCompiler, which has been developed primarily as a flexible tool for physical modeling and real-time computation of such models. The present prototype of the BlockCompiler runs only on the Macintosh OS X operating system, but all its software components should be possible to be ported to other major platforms (Linux, Windows).

Additional documentation on BlockCompiler can be found in: `http://www.acoustics.hut.fi/software/BlockCompiler`.

6. ACKNOWLEDGEMENTS

This study is part of the Academy of Finland project “Technology for Audio and Speech processing” (SA 53537) and is also related to the EU ALMA project (IST-2001-33059).

7. REFERENCES

[1] J. O. Smith, “Principles of Waveguide Models of Musical Instruments,” in *Applications of Digital Signal Processing to Audio and Acoustics*, ed. M. Kahrs and K. Brandenburg, Kluwer Academic Publishers, Boston 1998.

[2] J. Strikverda, *Finite Difference Schemes and Partial Differential Equations*, Wadsworth and Brooks, Grove, Ca, 1989.

[3] A. Fettweis, “Wave Digital Filters: Theory and Practice,” *Proc. IEEE*, 74(2), pp. 270–372, 1986.

[4] http://www.tml.hut.fi/∼tilmonen/mustajuuri/

[5] S. Van Duyne and J. O. Smith, “Physical Modeling with the 2-D Digital Waveguide Mesh,” *Proc. Int. Computer Music Conf. (ICMC’93)*. Tokio, Japan, 1993, pp. 40–47.

[6] M. Karjalainen, “Mixed Physical modeling: DWG + FDTD + WDF,” Accepted to *Proc. IEEE WASPAA 2003*.

[7] S. D. Bilbao, *Wave and Scattering Methods for the Numerical Integration of Partial Differential Equations*, PhD Thesis, Stanford University, May 2001.

[8] M. Karjalainen, C. Erkut, and L. Savioja, “Compilation of Unified Physical Models for Efficient Sound Synthesis,” Proc. *IEEE ICASSP’2003*, Hong Kong, 2003.