A TYPE-BASED APPROACH TO GENERATIVE PARAMETER MAPPING

Josué Moreno
School
Department

Vesa Norilo
Sibelius Academy
Centre for Music & Technology
Helsinki, Finland

ABSTRACT

This paper presents a type-based strategy for automating parameter mapping. It approaches the problem of integrating a disparate set of control signals into a musical application while minimizing the amount of boilerplate code the user has to write in order to connect control signals to unit generators.

The method is based on polymorphic mapper components and signal metadata that allows *typed* connections. Concepts for a semantically meaningful parameter mapping system for musical applications are explored and implemented in the Kronos programming language.

1. INTRODUCTION

Much of digital instrument building is point-to-point routing of signals. Often, these signals will need to be adapted on the go: for an instrument to be playable, its control surface must be mapped so that most of its parameter space is musically meaningful, maximizing expressiveness while being as economical as possible.

This results in carefully arranged adaptation layers, where the output from a control is scaled, translated and transformed to a range that is appropriate for an audio generator. Changing either the control or the generator typically means that the entire adaptation layer must be redesigned.

Little work has been done to standardize the fundamental mapping algorithms [8]. Likewise, few approaches have been made towards an automatization of mapping, with the exception of a neural networks based gesture identification and mapping libraries [10], which are not a general purpose solution.

With no standard framework, instrument designers are constantly reinventing the wheel, re-implementing mapping algorithms whenever creating a new instrument. Many mapping operations are very common, so it makes sense to design a general purpose software library that includes these operations [8].

Not all the composers or instrument designers have the necessary background to envision proper mapping schemes, resulting in linear and oversimplified parameter translation. These are rarely musically satisfying [1].

The authors propose a method to perform automatic parameter mapping based on an adaptation layer that can automatically reconfigure itself based on the semantics of both its input and output.

This paper is organized as follows; in Section 2, *Previous Work*, the current state of parameter mapping methods is examined. In Section 3, *Type-basd mapping*, the the principles of such mapping strategy will be stated and implementation details discussed. Finally, in Section 4, *Examples*, the authors will demonstrate the features of type-based mapping.

2. PARAMETER MAPPING STRATEGIES

2.1. Fundamentals of Data Mapping

Data mapping is the process of creating data element mappings between two distinct data models. It consists of data transformation or data mediation between a data source and a destination.

Data-driven mapping is a recent approach in data mapping and involves simultaneously evaluating actual data values in two data sources using heuristics and statistics to automatically discover complex mappings between two data sets.

Semantic Mapping consists of, given a data set, constructing a projection matrix that can be used for mapping data elements from one dimensional space into another, being or not not of the same dimension count. It is an alternative to *random mapping, principal components analysis* and *latent semantic indexing methods*. [6]

Mapping in musical applications, as in the work done by Miranda, has been mainly focused on several strategies formalized as [2]; *One to one; One to many; Many to one; Many to many*. These have been implemented by the composers themselves or within a software application that may carry aesthetic implications. Also, this categorization is not so much a mapping strategy as an overrall observation about different input/output configurations one may encounter.

2.2. Survey of the Field

Previously, Steiner has made work towards a Catalog of Mapping algorithms for the purpose of developing a Mapping Framework within Pd. [8] Continuing on previous developments such as the [hid] library [9], MnM for Max/MSP [4] and ESCHER [7].

In these frameworks interaction between mapping modules is based on standardizing all parameter ranges to a

linear range of $[0,1]$. Any information on if the range should be multidimensional, logarithmic or bipolar is dropped.

An interesting automatization of the mapping process is presented in [10]. The method is based on machine learning via neural networks that can extrapolate a mapping formalism from a data set of inputs and desired outputs. However, this method is mostly geared towards gesture recognition and not intended as a general mapping solution.

These approaches require that the user is an expert on mapping strategies. They provide only the fundamental building blocks for the design of such.

Research done on *Simplicital Interpolation* [3] is of value here as an example of Random Incremental Mapping in the domain of instrument design. These interpolation techniques may produce reasonable intermediate outputs for intermediate inputs out of discretely sampled data. Still, the authors believe that interpolation is not satisfying in a more general purpose mapping engine in which, given the amount of possible scenarios, the nature of the input data and the required output values, might not make sense within an interpolation context.

2.3. Aesthetic Considerations and Goals

Most of the research done in the field of Data Mapping for musical applications carries aesthetic implications. The spectrum of available solutions range from highly developed systems that have a built-in workflow and aesthetic to low level collections of primitives that present a steep learning curve as well as require a plenty of boilerplate programming to achieve simple tasks.

The aim of this research is to automate a lot of the low level programming associated with mapping while remaining as neutral as possible as to how the user wants to map parameters. Type-based metadata is offered as a means for the mapping framework to be smarter, freeing up the user to concentrate on solving the aesthetic problems rather than the technical details.

3. TOWARDS TYPE-BASED MAPPING MONADS

In computer science terminology, parameter mapping can be described as a monad. It is essentially a processing pipeline of functions, each of which transforms the data in some way. In addition, monads can adapt to the context of each sequence step, depending on the function in question and the data being processed. The monad is a very broad concept, but its applicability to mapping is explained in the rest of this section.

3.1. Basic Mapping

Let us consider a simple parameter mapping of a MIDI note to an oscillator, with the addition of a vibrato-producing LFO. A straightforward implementation of this in a low level modular environment might look like the mockup in Figure 1.

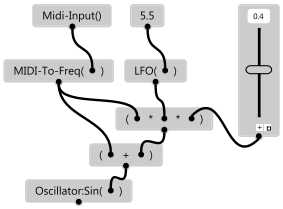

Figure 1. Low level mock implementation of a MIDI controlled Oscillator with vibrato

In most existing implementations, all the signals consist of plain numerical data. The user is expected to understand that a conversion from a MIDI note number to frequency in hertz is required. The LFO outputs a standard range, typically $[-1,1]$. More domain knowledge is required in order to understand how to scale this into a proper modulation depth; we assume a slider input of $[0,1]$ and treat this as a modulation depth percentage of the fundamental frequency. Finally, the efficient DSP implementation of an oscillator operates in cycles per sample – requiring a final adaptation step from hertz.

The example given is artifically low level to demonstrate the assumptions that tend to be present in even a simple patch. For most adaptation steps, we must actively think about the semantics of both input and output. This knowledge must be reflected in our choice of processing primitives; many conversion primitives are present solely to define the *types* or *units* to convert between.

All this knowledge already exists in the patch. A MIDI input, a slider and an LFO have a well-defined output range. Likewise, the oscillator has a well-defined input semantics. Why, then, does the user need to repeat all this by programming each transformation step manually?

3.2. Typed Signals and Polymorphic Primitives

To enable smart behavior for mapping primitives, the authors propose *typed signals* and *polymorphic primitives* that generate a *mapping monad*. Metadata, such as an appropriate unit, parameter range or curvature can be attached to a signal to endow it with desired semantics. This is called type; in contrast to raw data, *type* carries an annotation of what the data means.

Primitives can then be *overloaded* so that they respond differently according to the type of the incoming signal. Such functions are called *polymorphic*; a staple of functional and object oriented programming. The distinct implementations of the polymorphic function are called *overloads*.

This allows for a higher level parameter mapping framework where the concepts are nearer to human thought and further from the fundamental mathematical and logical operations carried out by a computer, and the signal processing framework can deduce most low level adaptation steps from the context.

3.3. The Type-Primitive Matrix

Consider a processing chain that performs parameter mapping from a source to a final destination. Each link is a named process with a certain semantic meaning; such as "limit range", "smooth" or "modulate".

A framework with N mapping primitives and M signal types can be envisioned as a NxM matrix where each cell represents the polymorphic overload for a mapping primitive x on an input signal of type y. Each overload has an output signal of type z. Since z can influence overload resolution of the next step in the processing chain, any mapping primitive can exert influence over the entirety of the chain following it.

This has huge potential for automating the mapping process. For example, when a control signal source changes, typically a large part of the following parameter mapping chain must be adjusted. With appropriate type annotations and overloads, this can happen automatically.

3.4. Taming the Combinatorics

The Type-Primitive matrix in Section 3.3 grows as a product of the number of types and primitives in the system. Implementing such a framework can quickly become non feasible due to the number of overloads required. Several strategies exist to deal with this.

3.4.1. Subtypes

If the signal types can naturally fit a hierarchical category, subtyping can be leveraged. Common traits of several types can be collected into a supertype, and overloads can consider this instead of every subtype separately. Many object oriented systems implement subtypes.

3.4.2. Generics

Generics are the approach chosen by the authors for the implementation of this framework is generic programming, as it is one of the main techniques of the Kronos[5] environment. Similar results could be achieved with the C++ template metacompiler or a dynamically typed language such as Javascript.

A generic mapping primitive is one that doesn't explicitly define a fixed overload for each type, but rather uses pattern matching to perform overload resolution. Such an overload can be selected based upon some salient traits of the incoming type – for example, whether the type supports ordering or a certain subset of arithmetic.

Some primitives may even delegate type specialiation to their components. For example, a smoothing filter doesn't necessarily care about the precise type of the signal passing through, as long as sufficient arithmetic for the type is defined – in this case, summation and multiplication. For details on this mechanism, the reader is referred to prior work[5].

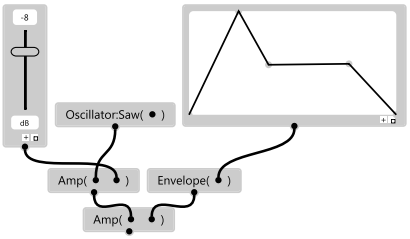

Figure 2. Amplifiers driven by a decibel-format slider and a linear $[0, 1]$ envelope.

4. EXAMPLE IMPLEMENTATION

4.1. About Kronos

For this research, the Kronos programming environment[5] is used to provide the type system, polymorphism and generative capabilities required by the mapping monads.

Kronos is a special purpose programming language intended for musical signal processing tasks. It is targeted to musicians and music technologists who want to create customized signal processing solutions. Anything from effect processors to virtual instruments can be built. The Kronos engine offers a powerful functional programming language, yet it can be easily used with a visual, patching-based interface.

4.2. Simple Case

First we consider a simple amplitude control from a variety of sources. To motivate polymorphism, an overloaded amplifier module is demonstrated. Two instances of the amplifier are included in the patch. One of them is controlled by a slider set in decibels, the other by a envelope with a gain coefficient range of $[0, 1]$. By using the semantics of *amplification* instead of a multiplication, the meaning of this simple program becomes obvious; in this context, a level parameter is decibels is easy to understand.

With a standard multiplication, there's no context from which to deduce how to handle decibels – as multiplying by a decibel quantity has its own, separate meaning. This context is then provided by the end user, by the means of inserting the appropriate mathematics to compute a linear gain coefficient from an incoming decibel before the multiplication. The authors contend that this is something that should be left to software.

4.3. Multidimensional

In this example, disparate real time control signals are combined into a single three-dimensional parameter. Although the example is contrived, it serves to demonstrate some of the automation provided by the framework.

The example is shown in Figure 3. Three inputs are generated by listening to a MIDI continuous controller, analyzing an audio input and receiving an OSC event stream. All these inputs can assign correct type information to

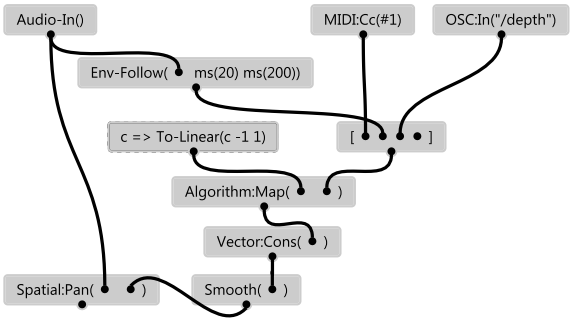

Figure 3. Mapping three disparate parameters to a Spatial Panner

their output signal, describing the range of the signal. In addition, the Envelope-Follower module receives two time constants annotated as milliseconds.

Algorithm:Map is used to transform all three parameters at once, applying a transformation to a linear range of $[-1,1]$. Because the x, y and z components have associated type information, the polymorphic $To-Linear$ function can deduce the appropriate scaling automatically. The transformed values are combined into a single vector coordinate, which is used as a coordinate for a spatial panner. A simple smoothing is applied by a filter – which can maintain the vector coordinate semantics due to the generic approach described in Section 3.4.2.

The spatializer can also leverage polymorphism; type context can imply whether the control parameter is given in polar or cartesian form, and if the panning happens in 2 or 3 dimensions.

5. CONCLUSIONS

In this paper, the authors have demonstrated some of the possibilities and advantages of using polymorphism as a mapping strategy based upon signal semantics. Types are used as metadata that aid composed mapping monads to configure themselves to account for disparate parameter ranges.

We have the advantage of the generative nature of Kronos programming language. To expand upon this we apply a type system to mapping. By abstracting the math into software objects, mapping can be approached as a system of higher level human logic.

Upon this, the authors are developing a framework for Generative Music Composition and Performance based on live audio and sensor data input. By making the mapping process smarter and quicker to prototype we can enable another way of interacting and conceiving generative music.

The framework along with the Kronos compiler will be made freely available to interested parties. It is conceived as a Kronos library, but given its extensive OSC and MIDI communication capabilities, interaction with various software packages is feasible. Finished adaptation layers can also be built into object code that can link into any system that supports a C language interface.

6. REFERENCES

[1] E. M. Alexis Kirke, "Evaluating Mappings for Cellular Automata Music," in *Interdisciplinary Centre for Computer Music Research (ICCMR)*, Plymouth, UK, 2006.

[2] E. R. Eaton J. Miranda, "New Approaches in Brain-Computer Music Interfacing. Mapping EEG for Real-Time Musical Control," in *Proceedings of Creativity at the Intersection of Music and Computation*, University of Plymouth, UK, 2012.

[3] C. Goudeseune, "Interpolated mappings for musical instruments," in *Organised Sound*, 2002, p. 7(2):8596.

[4] F. B. R. Muller and N. Schnell., "MnM: a Max/MSP mapping toolbox." in *Proceedings of the Conference on New Interfaces for Musical Expression (NIME05)*, Vancouver, Canada, 2005.

[5] V. Norilo, "Introducing Kronos - A Novel Approach to Signal Processing Languages," in *Proceedings of the Linux Audio Conference*, F. Neumann and V. Lazzarini, Eds. Maynooth, Ireland: NUIM, 2011, pp. 9–16.

[6] T. B. L. Renato Fernandes Corra, "Dimensionality Reduction of very large document collections by Semantic Mapping," in *Proceedings of the 6th International Workshop on Self-Organizing Maps*, Center of Informatics, Federal University of Pernambuco, Brasil, 2007.

[7] M. W. N. Schnell and J. B. Rovan., "Escher-modeling and performing composed instruments in real-time." in *IEEE Systems, Man, and Cybernetics*, 1998.

[8] H. C. Steiner, "Towards a catalog and software library of mapping methods," in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME06)*, Paris, France, 2006.

[9] H.-C. Steiner, "[hid] toolkit: a unified framework for instrument design," in *Proceedings of the 2005 International Conference on New Interfaces for Musical Expression (NIME05)*, Vancouver, BC, Canada, 2005.

[10] C. A. C. T. and H. C., "Real-time Gesture Mapping in Pd Environment using Neural Networks," in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME04)*, Japan, 2004.