Auto-documentation mathématique pour le traitement du signal avec Faust

Karim Barkati, Yann Orlarey

Auto-documentation mathématique pour le traitement du signal avec Faust

Karim Barkati
Grame*
karim.barkati@gmail.com

Yann Orlarey
Grame
yann.orlarey@grame.fr

1. INTRODUCTION

Dès 1984, Donald Knuth soulignait l'importance de la documentation pour la programmation, en rapport avec son concept de « programmation littérale » [1] [1] : "*I believe that the time is ripe for significantly better documentation of programs [...].*" [2] Un quart de siècle plus tard, où en est-on de la programmation littérale en informatique musicale ? Il semblerait qu'à la différence des langages de programmation généraux qui utilisent `doxygen` ou `javadoc`, les langages dédiés à l'informatique musicale manquent encore d'un système de documentation intégré. En outre, les principaux langages d'informatique musicale qui permettent de faire du traitement du signal – comme Max/MSP, Pure Data, ou SuperCollider – ne produisent pas de documentation mathématique du traitement du signal alors que rien ne s'y oppose techniquement. Enfin, au-delà de la programmation littérale qui demande un effort supplémentaire aux programmeurs, qu'en serait-il d'un système d'*auto*-documentation ?

Le compilateur FAUST répond à ces questions grâce à la conception et à l'implémentation récente d'un mécanisme d'*auto-documentation mathématique* pour le traitement du signal. En effet, le compilateur FAUST est aujourd'hui capable de calculer une documentation complète à partir d'un programme FAUST, qui dénote un bloc-diagramme de traitement du signal, sous la forme d'une liste exhaustive de formules mathématiques en LaTeX.

2. OBJECTIFS

On peut dénombrer quatre objectifs ou usages principaux pour l'auto-documentation mathématique :

1. *la préservation*, soit préserver les processeurs de signaux sous forme mathématique, de façon indépendante de tout langage informatique ;
2. *la validation*, soit apporter une aide au débogage en montrant les équations mathématiques telles qu'elles sont effectivement calculées et normalisées après la phase de compilation ;
3. *la pédagogie*, soit présenter un nouveau type de supports d'enseignement grâce au pont automatique entre le code source et les formules mathématiques pour le traitement du signal ;
4. *la publication*, soit produire du matériel de publication, en préparant des formules LaTeX et des blocs-diagrammes SVG faciles à intégrer dans des articles.

Le premier objectif de l'auto-documentation mathématique, à savoir la préservation à long terme, repose sur l'hypothèse forte que *le langage mathématique a toutes les chances de durer beaucoup plus longtemps que n'importe quel langage informatique*. Cela signifie qu'une fois imprimée sur papier, une documentation mathématique devient un document hautement préservable, car la sémantique d'un programme de traitement du signal est entièrement traduite dans deux langages indépendants de tout langage et de tout environnement informatiques : le langage mathématique, pour la plus grande part, et le langage naturel, employé pour structurer la présentation du document en direction d'un lecteur humain et pour préciser la définition de certains éléments mathématiques spécifiques (comme les symboles que nous utilisons pour dénoter les opérations sur les entiers).

En conséquence, d'une part la documentation mathématique est auto-suffisante au regard de la réimplémentation d'un programme de traitement du signal et d'autre part, elle devrait le rester pour des décennies et probablement davantage !

3. CONTEXTE

3.1. Un langage fonctionnel pour les signaux audio

FAUST est un langage compilé dédié au traitement du signal audio en temps réel [3]. Le nom FAUST vient de *Functional AUdio STream*. Son modèle de programmation combine deux approches : la programmation fonctionnelle et la composition de blocs-diagrammes. On peut voir FAUST comme un langage structuré de blocs-diagrammes avec une syntaxe textuelle.

Typiquement, un processeur de signal FAUST p est une fonction qui prend en entrée des signaux (eux-mêmes des fonctions du temps appartenant à $\mathbb{S} = \mathbb{N} \mapsto \mathbb{R}$) et produit en sortie des signaux, tels que $p : \mathbb{S}^n \mapsto \mathbb{S}^m$.

FAUST a été conçu comme un langage de spécification dirigé par la sémantique ; c'est un langage générique à double titre car d'une part FAUST s'appuie sur un paradigme purement fonctionnel et d'autre part son algèbre de

* Karim Barkati travaille actuellement à l'Ircam.

1 . *« Literate programming »*.

2 . « Je crois que le moment est venu pour une documentation des programmes significativement meilleure. »

3 . http://faust.grame.fr.

blocs-diagrammes lui fournit un modèle générique pour les processus synchrones.

De plus, son approche haut-niveau permet au compilateur de déployer à la fois des optimisations poussées, en particulier avec la vectorisation [2] et la parallélisation [3] (dont OpenMP et du *work stealing* [4]), et de démultiplier les plates-formes et langages cibles (Actionscript Flash, ALSA, CoreAudio, Jack, Qt, GTK, CSound, iPhone, LADSPA, LLVM, LaTeX, Matlab, Max/MSP, Octave, OSS, Pure, Pure Data, Q, Snd-RT, SynthFile, SuperCollider, VST, VSTi, dll).

3.2. Du point de vue de la préservation

La préservation des œuvres de musique contemporaine est un problème à la fois épineux et urgent dans le domaine de la musique interactive utilisant l'informatique en temps réel [5]. Des décennies de pièces musicales interactives réalisées depuis les années 1980 risquent malheureusement de ne plus être rejouables et les pièces d'aujourd'hui sont aussi menacées d'être perdues, au tout du moins très coûteuses à remonter, dans seulement cinq ou dix ans !

Le phénomène d'obsolescence et ses conséquences dans le domaine de l'informatique musicale sont clairement identifiés, mais il n'y a pas de solution simple. En effet, les logiciels et le matériel utilisés pour la musique en temps réel évoluent constamment afin de tirer parti des progrès technologiques et informatiques, car les technologies temps réel sont souvent très gourmandes à la fois en mémoire et en vitesse, sollicitant du traitement du signal lourd. En particulier, les langages de programmation audio pour le temps réel évoluent rapidement et de nombreux langages apparaissent d'une manière incompatible avec les langages précédents ou avec les versions précédentes.

Par conséquent, en informatique musicale interactive, cette perpétuelle course à l'innovation a conduit à un cycle d'obsolescence très court typique des pièces musicales en temps réel.

Comme nous l'avons mentionné précédemment, la préservation est le premier objectif de ce type de documentation mathématique, mais il faut préciser que nous ne nous référons pas à de simples enregistrements, ainsi que l'indique l'annexe technique ASTREE [6] :

> « Il est à noter que l'objectif de la pérennisation de la création numérique ne se limite pas à la pérennisation *du résultat* de cette création (sous la forme d'un enregistrement vidéo, audio, etc.), mais qu'il nécessite la pérennisation *du dispositif interactif* qui permet de le générer. »

Le développement de l'auto-documentation mathématique des programmes FAUST à GRAME a été soutenu par le projet de recherche français ASTREE, un acronyme pour « Analyse et synthèse des processus en temps réel » [4].

4. En collaboration avec quatre laboratoires : IRCAM, GRAME, MINES ParisTech et CIEREC.

FAUST, en tant que langage de traitement du signal auto-documenté, permet de préserver la partie synchrone de systèmes interactifs musicaux, en tentant de mettre les processeurs de signaux « en clair » pour le temps réel, comme Jean-Claude Risset l'appelle de ses vœux [7] :

> « La vitesse du changement technologique rend vite caducs les appareillages spécialisés, ce qui gêne *la constitution d'un corpus de savoir-faire*, *l'approfondissement d'une tradition* de composition et d'interprétation et d'écoute et *la décantation de "classiques"*. Aussi est-il important d'assurer la portabilité des dispositifs technologiques de réalisation sonore, surtout si celle-ci est réalisée en temps réel et non "sur support", c'est-à-dire sous forme d'un enregistrement.
>
> Bien des démarches significatives sont mal comprises, ignorées ou oubliées. La musicologie est en effet mal armée pour rendre compte des démarches de l'informatique musicale, manquant de documents immédiatement exploitables comme une partition traditionnelle. Or, la plupart du temps, ces documents existent, mais *ils doivent être mis "en clair"*. »

Ainsi, nous pouvons dénombrer de nombreux enjeux d'une telle stratégie de préservation innovante qui utilise *l'abstraction mathématique* pour tendre vers une documentation robuste et pérenne :

- augmenter la durée de vie des œuvres temps réel ;
- favoriser les progrès de l'informatique musicale ;
- rendre à l'histoire de l'art des pièces rejouables ;
- rétablir un partage et une mémoire collective ;
- améliorer et diffuser les stratégies de préservation ;
- faciliter l'échange de pièces du répertoire contemporain entre les centres et les pays.

4. AUTO-DOCUMENTATION

Cette section présente les principaux éléments de l'auto-documentation de FAUST, ainsi que des extraits commentés de la documentation mathématique générée avec le mode automatique.

4.1. Génération de la documentation mathématique

La façon la plus simple de générer la documentation mathématique complète est d'exécuter sur un fichier FAUST le script `faust2mathdoc` :

```
faust2mathdoc myfaustfile.dsp
```

Le fichier PDF est alors généré dans le répertoire approprié : `myfaustfile-mdoc/pdf/myfaustfile.pdf`.

Précisément, l'exécution de la commande `faust2mathdoc` produit une arborescence organisée en cinq sous-répertoires, pour accueillir l'ensemble des fichiers utilisés et produits :

```
▼ noise-mdoc/
  ▼ cpp/
    ◇ noise.cpp
  ▼ pdf/
    ◇ noise.pdf
  ▼ src/
    ◇ math.lib
    ◇ music.lib
    ◇ noise.dsp
  ▼ svg/
    ◇ process.pdf
    ◇ process.svg
  ▼ tex/
    ◇ noise.aux
    ◇ noise.log
    ◇ noise.pdf
    ◇ noise.tex
```

Chronologiquement, le script shell `faust2mathdoc` appelle d'abord le compilateur FAUST avec l'option `faust --mathdoc` appropriée pour la documentation mathématique, ce qui génère :

- un répertoire de haut niveau suffixé par "`-mdoc`",
- 5 sous-répertoires (`cpp/`, `pdf/`, `src/`, `svg/`, `tex/`),
- le fichier LaTeX de la documentation mathématique,
- des fichiers SVG pour les blocs-diagrammes.

Puis le script termine le travail du compilateur FAUST,

- en déplaçant le fichier C++ produit dans `cpp/`,
- en convertissant tous les fichiers SVG au format PDF avec la commande `svg2pdf`,
- en appelant la commande `pdflatex` sur le fichier LaTeX pour compiler le fichier PDF final,
- en déplaçant le fichier PDF final dans `pdf/`.

4.2. La structure du document produit

Le document PDF final, produit automatiquement, est structuré en quatre sections qui sont par défaut intitulées en anglais respectivement :

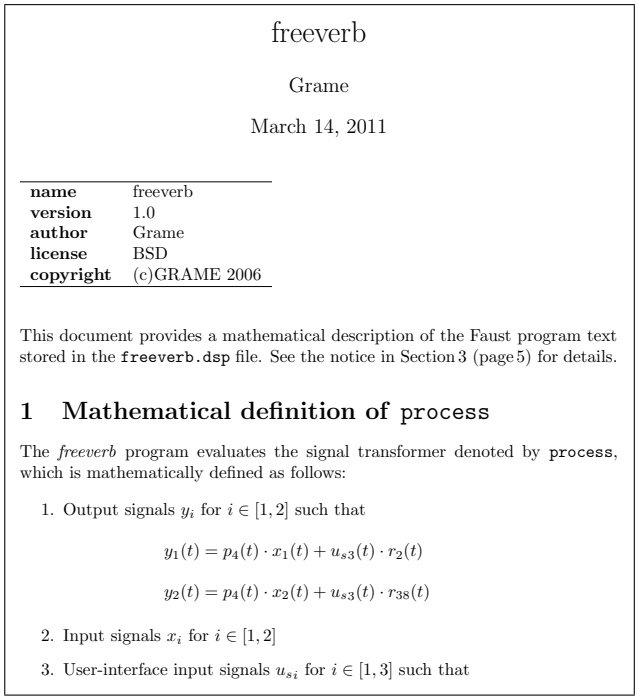
1. Mathematical definition of `process`
2. Block diagram of `process`
3. Notice
4. Faust code listings

Il est possible de spécifier une autre langue en utilisant les options `-mdlang` ou `--mathdoc-lang` suivies d'un argument de deux lettres (`en`, `fr`, `it`, `de`, etc.). FAUST propose à ce jour quatre langues : l'anglais, le français, l'italien et l'allemand. La structure du document devient alors la suivante lorsqu'elle est compilée en français :

1. Définition mathématique de `process`
2. Bloc-diagramme de `process`
3. Notice
4. Code Faust

4.3. Une première page

Tout d'abord, pour donner une idée, examinons la première page d'une documentation mathématique grâce à la figure 1 qui montre la première page du document PDF



freeverb

Grame

March 14, 2011

name	freeverb
version	1.0
author	Grame
license	BSD
copyright	(c)GRAME 2006

This document provides a mathematical description of the Faust program text stored in the `freeverb.dsp` file. See the notice in Section 3 (page 5) for details.

1 Mathematical definition of `process`

The *freeverb* program evaluates the signal transformer denoted by `process`, which is mathematically defined as follows:

1. Output signals y_i for $i \in [1,2]$ such that

$$y_1(t) = p_4(t) \cdot x_1(t) + u_{s3}(t) \cdot r_2(t)$$

$$y_2(t) = p_4(t) \cdot x_2(t) + u_{s3}(t) \cdot r_{38}(t)$$

2. Input signals x_i for $i \in [1,2]$
3. User-interface input signals u_{si} for $i \in [1,3]$ such that

Figure 1. Première page du document `freeverb.pdf`.

généré à partir du programme FAUST `freeverb.dsp` (sachant que les marges sont réduites ici).

Les éléments d'en-tête sont extraits des métadonnées suivantes, déclarées dans le fichier-source en FAUST :

```
declare name        "freeverb";
declare version     "1.0";
declare author      "Grame";
declare license     "BSD";
declare copyright   "(c)GRAME 2006";
```

La date de la compilation de la documentation est insérée sous le titre et un peu de texte est ajouté pour présenter chaque section et le document lui-même. Ainsi, en plus du langage mathématique, le document s'appuie sur le langage naturel mais on peut légitimement faire l'hypothèse que ce dernier aussi durera bien plus longtemps que n'importe quel langage de programmation actuel.

4.4. Définition mathématique de `process`

La première section du document produit contient la définition mathématique complète de `process`. De toute évidence, le calcul de la présentation des formules mathématiques est la partie la plus importante de l'auto-documentation mathématique.

4.4.1. Compilation sémantique

Pour développer la sortie LaTeX de la documentation mathématique, au lieu d'utiliser de simples substitutions (*pattern-matching*), nous avons étendu le compilateur FAUST de l'intérieur, en réimplémentant les classes principales afin de pouvoir imprimer les équations mathématiques sous une forme *normalisée*. Cela signifie que, à l'instar de la

4. Intermediate signals p_i for $i \in [1, 8]$ and r_1 such that

$$p_1(t) = k_1 \cdot \max(0, u_{s1}(t))$$

$$p_2(t) = \cos(p_1(t))$$

$$p_3(t) = 2 \cdot p_2(t)$$

$$p_4(t) = 0.5 \cdot \frac{\sin(p_1(t))}{\max(0.001, u_{s2}(t))}$$

$$p_5(t) = (p_4(t) - 1)$$

$$p_6(t) = (1 + p_2(t))$$

$$p_7(t) = 0.5 \cdot p_6(t)$$

$$p_8(t) = \frac{1}{1 + p_4(t)}$$

$$r_1(t) = p_8(t) \cdot (x_1(t-1) \cdot (0 - (p_6(t) - x_2(t))) + p_7(t) \cdot x_1(t) + p_7(t) \cdot x_1(t-2) + p_5(t) \cdot r_1(t-2) + p_3(t) \cdot r_1(t-1))$$

5. Constant k_1 such that

$$k_1 = \frac{6.28318530717959}{f_S}$$

Figure 2. Quelques formules rendues (`HPF.pdf`).

Lettre	Type de signal
$y(t)$	Signal de sortie
$x(t)$	Signal d'entrée
$u_b(t)$	Signal d'interface-utilisateur de bouton
$u_c(t)$	Signal d'interface-utilisateur de case à cocher
$u_s(t)$	Signal d'interface-utilisateur de glisseur
$u_n(t)$	Signal d'interface-utilisateur de boîte-nombre
$u_g(t)$	Signal d'interface-utilisateur de VU-mètre
$p(t)$	Signal intermédiaire de paramètre (calculé au taux de contrôle)
$s(t)$	Signal intermédiaire simple (calculé au taux d'échantillonnage)
$r(t)$	Signal intermédiaire récursif (dépend d'échantillons précédents $r(t-n)$)
$q(t)$	Signal intermédiaire de sélection (sélecteurs à deux ou trois branches)
$m(t)$	Signal intermédiaire de mémoire (délai d'un échantillon)
$v(t)$	Signal intermédiaire de table (lecture ou lecture/écriture)
$k(t)$	Signal constant

Table 1. Nommage et typage des signaux.

sortie en C++ du compilateur, la sortie LaTeX est calculée *après* la compilation des processeurs de signaux, bénéficiant ainsi de toutes les simplifications et normalisations que le compilateur FAUST est capable de faire.

Dit autrement, la sémantique mathématique d'un programme FAUST est normalisée par le compilateur avant qu'il la traduise dans le langage LaTeX, ce qui est rendu possible par le principe de compilation sémantique appliqué dans FAUST.

4.4.2. Typage des signaux

Le compilateur FAUST utilise un typage interne pour une compilation efficace des processeurs de signaux et nous avons réutilisé ce typage pour le découpage en sous-équations de l'auto-documentation mathématique.

Quelques formules sont montrées sur la figure 1 (extraite du fichier `freeverb.dsp`, une réverbération) et sur la figure 2 (extraite du fichier `HPF.dsp`, un filtre passe-haut), telles qu'elles sont imprimées dans les documents PDF produits correspondants. Sur la figure 1, on peut voir la définition de trois types de signaux, tandis que sur la figure 2 on peut voir deux autres types, et ce sont au total les cinq familles de signaux qui sont utilisées pour les sections de la définition mathématique d'un processeur de signal :

1. « Output signals »,
2. « Input signals »,
3. « User-interface input signals »,
4. « Intermediate signals »,
5. « Constant signals ».

En réalité, l'extension *documentator* du compilateur FAUST gère davantage de types de signaux et utilise pleinement les capacités de marquage des signaux du compilateur FAUST pour découper plus finement les équations. Un découpage élaboré est très important pour la lisibilité du document, sans quoi il y aurait une seule formule pour `process`, potentiellement extrêmement longue ! Le *documentator* pousse cette recherche de lisibilité un peu plus loin que le découpage en cinq familles de signaux, en utilisant des lettres différentes selon les types internes et des indices numériques pour construire l'identificateur de chaque sous-équation (le nom du membre de gauche).

Les indices sont faciles à comprendre : sur la figure 1 par exemple, les mentions comme « $y_1(t)$ », « $y_2(t)$ » et « Input signals x_i for $i \in [1, 2]$ » indiquent clairement que le bloc-diagramme de `freeverb` possède deux signaux d'entrée et deux signaux de sortie, c'est-à-dire qu'il est un transformateur de signal stéréophonique. Le choix des lettres en fonction des types internes de signaux, un peu plus complexe, est spécifié dans le tableau 1.

4.4.3. Typographie mathématique automatique

Le problème du saut de ligne automatique en LaTeX pour l'impression des équations trop longues a été résolu par le packetage `breqn` que nous utilisons.

La figure 3 montre le code LaTeX (`HPF.tex`) correspondant aux formules affichées sur la figure 2 (`HPF.pdf`). Concernant le code LaTeX, nous insérons chaque équation dans un environnement `dmath*` qui prend en charge les sauts de ligne, et nous insérons chaque famille de signaux dans un environnement `dgroup*` qui gère quant à lui l'alignement vertical des équations (généralement autour du

```
        \begin{dmath*}
                p_{6}(t) =  \left(1 + p_{2}(t)\right)
        \end{dmath*}
        \begin{dmath*}
                p_{7}(t) = 0.5 * p_{6}(t)
        \end{dmath*}
        \begin{dmath*}
                p_{8}(t) = \frac{1}{1 + p_{4}(t)}
        \end{dmath*}
    \end{dgroup*}


    \begin{dgroup*}
        \begin{dmath*}
                r_{1}(t) = p_{8}(t) *  \left(x_{1}(t\!-\!1) *  \left(0 -
  \left(p_{6}(t) - x_{2}(t)\right) \right)  + p_{7}(t) * x_{1}(t) + p_{7}(t) *
x_{1}(t\!-\!2) + p_{5}(t) * r_{1}(t\!-\!2) + p_{3}(t) * r_{1}(t\!-\!1)\right)
        \end{dmath*}
    \end{dgroup*}

\item Constant $k_$ such that
    \begin{dgroup*}
        \begin{dmath*}
                k_{1} = \frac{6.28318530717959}{f_S}
        \end{dmath*}
    \end{dgroup*}
```

Figure 3. Code LaTeX des formules (`HPF.tex`).

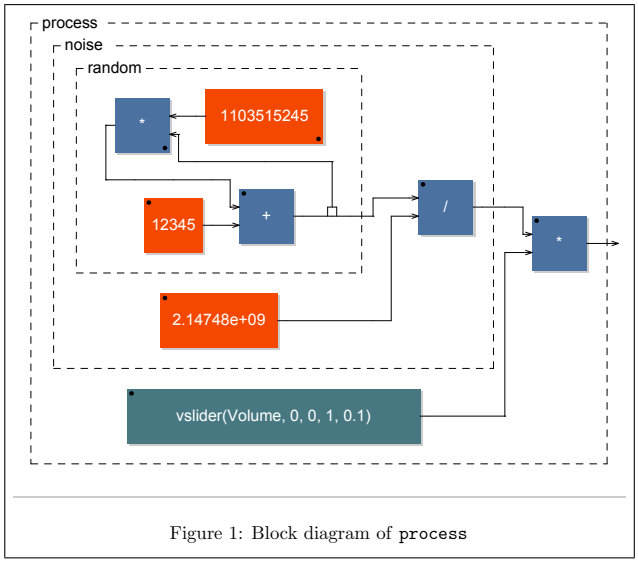


Figure 1: Block diagram of `process`

Figure 4. Un bloc-diagramme rendu (`noise.pdf`).

signe égal).

Par exemple, on peut remarquer sur la figure 2 que la longue formule $r_1(t)$ imprimée a été correctement et joliment disposée sur deux lignes, tandis que l'équation LaTeX correspondante r_1(t) de la figure 3 ne comporte explicitement aucun signe de saut de ligne (la double contre-oblique en LaTeX) et se présente bien d'un seul tenant.

4.5. Bloc-diagramme de `process`

La deuxième section présente le bloc-diagramme de plus haut niveau de `process`, c'est-à-dire un bloc-diagramme qui tienne sur une page. Le schéma approprié est calculé par la partie du compilateur FAUST qui gère la sortie SVG.

La figure 4 montre le bloc-diagramme calculé à partir du fichier `noise.dsp` (un générateur de bruit). Par défaut, le bloc-diagramme de haut-niveau de `process` est généré en SVG, converti au format PDF via l'utilitaire `svg2pdf` (en utilisant la bibliothèque graphique 2D Cairo), puis nommé et inséré dans la deuxième section de la documentation en tant que figure LaTeX flottante, pour être référençable.

Sur la figure 4, les cadres en pointillés délimitent graphiquement les fonctions nommées, les couleurs varient selon le type des éléments dessinés (processeurs, constantes, objets d'interface utilisateur, etc.) et le petit carré vide représente un retard d'un échantillon.

4.6. Notice

La troisième section, qui présente la notice pour éclairer l'ensemble de la documentation, est divisée en deux parties :

- un en-tête commun systématique (cf. figure 5) ;
- un corps mathématique variable (la figure 6 montre un exemple à partir du fichier `capture.dsp`).

Afin de rendre la relecture la plus intelligible possible dans un futur potentiellement éloigné, la première partie de la notice utilise le langage naturel de manière intensive pour expliciter la documentation, en donnant à la fois des informations contextuelles – telles que le numéro de version du compilateur, la date de compilation, une présentation synoptique, les adresses URL des sites officiels de FAUST et de SVG, l'arborescence des répertoires de la documentation générée – et des explications fondamentales sur le langage FAUST lui-même, sa sémantique mathématique (dénotationnelle) – y compris l'identifiant `process`, la sémantique des signaux et des transformateurs de signaux.

En particulier, comme on peut le remarquer sur la figure 6, chaque fonction ou opérateur susceptible de ne pas être standard est défini, comme la fonction de « cast » $int(x)$ ou encore les opérations entières dénotées par des opérateurs cerclés ($\oplus$, $\ominus$ et $\odot$ sur la figure 6).

4.7. Code Faust

La quatrième et dernière section fournit le code FAUST complet. Tout le FAUST est inséré par défaut dans la documentation : le code source principal ainsi que toutes les librairies nécessaires, en utilisant le système de *pretty-printer* fourni par le paquetage LaTeX `listings` avec un `lstset` spécifique pour les mots-clés du langage FAUST, comme le montre la figure 7.

Vous pourriez vous demander pourquoi nous imprimons le code FAUST dans la documentation alors que le langage FAUST est également concerné par notre « mot d'ordre » selon lequel le langage mathématique durera beaucoup plus longtemps que n'importe quel langage informatique... Il s'agit principalement d'ajouter un élément d'aide supplémentaire pour la contextualisation ! En effet, selon les algorithmes de traitement du signal et les implémentations, certains codes FAUST peuvent s'avérer extrêmement précieux pour la compréhension des formules mathématiques imprimées, toujours dans la perspective de réimplémenter le même algorithme avec d'autres langages dans

3 Notice

- This document was generated using Faust version 0.9.36 on March 14, 2011.
- The value of a Faust program is the result of applying the signal transformer denoted by the expression to which the `process` identifier is bound to input signals, running at the f_S sampling frequency.
- Faust (*Functional Audio Stream*) is a functional programming language designed for synchronous real-time signal processing and synthesis applications. A Faust program is a set of bindings of identifiers to expressions that denote signal transformers. A signal s in S is a function mapping[1] times $t \in \mathbb{Z}$ to values $s(t) \in \mathbb{R}$, while a signal transformer is a function from S^n to S^m, where $n, m \in \mathbb{N}$. See the Faust manual for additional information (http://faust.grame.fr).
- Every mathematical formula derived from a Faust expression is assumed, in this document, to having been normalized (in an implementation-dependent manner) by the Faust compiler.
- A block diagram is a graphical representation of the Faust binding of an identifier I to an expression E; each graph is put in a box labeled by I. Subexpressions of E are recursively displayed as long as the whole picture fits in one page.
- The `BPF-mdoc/` directory may also include the following subdirectories:
  - `cpp/` for Faust compiled code;
  - `pdf/` which contains this document;
  - `src/` for all Faust sources used (even libraries);
  - `svg/` for block diagrams, encoded using the Scalable Vector Graphics format (http://www.w3.org/Graphics/SVG/);
  - `tex/` for the LaTeX source of this document.

Figure 5. En-tête commun de la notice (`BPF.pdf`).

- $\forall x \in \mathbb{R}$,

$$\mathrm{int}(x) = \begin{cases} \lfloor x \rfloor & \text{if } x > 0 \\ \lceil x \rceil & \text{if } x < 0 \\ 0 & \text{if } x = 0 \end{cases}.$$

- This document uses the following integer operations:

operation	*name*	*semantics*
$i \oplus j$	integer addition	normalize$(i+j)$, in $\mathbb{Z}$
$i \ominus j$	integer substraction	normalize$(i-j)$, in $\mathbb{Z}$
$i \odot j$	integer multiplication	normalize$(i \cdot j)$, in $\mathbb{Z}$

Integer operations in Faust are inspired by the semantics of operations on the n-bit two's complement representation of integer numbers; they are internal composition laws on the subset $[-2^{n-1}, 2^{n-1}-1]$ of $\mathbb{Z}$, with $n = 32$. For any integer binary operation $\times$ on $\mathbb{Z}$, the $\otimes$ operation is defined as: $i \otimes j = \mathrm{normalize}(i \times j)$, with

$$\mathrm{normalize}(i) = i - N \cdot \mathrm{sign}(i) \cdot \left\lfloor \frac{|i| + N/2 + (\mathrm{sign}(i)-1)/2}{N} \right\rfloor,$$

where $N = 2^n$ and $\mathrm{sign}(i) = 0$ if $i = 0$ and $i/|i|$ otherwise. Unary integer operations are defined likewise.

[1] Faust assumes that $\forall\, s \in S, \forall\, t \in \mathbb{Z}, s(t) = 0$ when $t < 0$.

Figure 6. Partie variable d'une notice (`capture.pdf`).

4 Listing of the input code

The following listing shows the input Faust code, parsed to compile this mathematical documentation.

Listing 1: `noisemetadata.dsp`

```
//-----------------------------------------------------------------
// Noise generator and demo file for the Faust math documentation
//-----------------------------------------------------------------

declare name      "Noise";
declare version   "1.1";
declare author    "Grame";
declare author    "Yghe";
declare license   "BSD";
declare copyright  "(c)GRAME 2009";


random = +(12345)~*(1103515245);


noise   = random/2147483647.0;


process = noise * vslider("Volume[style:knob]", 0, 0, 1, 0.1);
```

Figure 7. Faust code listing.

plusieurs décennies.

5. MODE MANUEL

Vous pouvez spécifier vous-même la structure de la documentation au lieu d'utiliser le mode automatique, avec cinq balises de type XML. Cela permet de modifier la présentation et d'ajouter vos propres commentaires, non seulement sur `process`, mais aussi sur toute autre expression que vous souhaitez documenter plus particulièrement. Notez que dès que vous déclarez une balise `<mdoc>` à l'intérieur de votre fichier FAUST, la structure par défaut du mode automatique (décrite à la section 4.2) est ignorée et tout le travail de présentation des sections LaTeX vous revient !

Voici les six balises spécifiques au mode manuel :

- `<mdoc></mdoc>` pour ouvrir un champs de documentation dans le code faust,
  - `<equation></equation>` pour obtenir les équations d'une expression FAUST,
  - `<diagram></diagram>` pour récupérer le bloc-diagramme de haut-niveau d'une expression FAUST,
  - `<metadata></metadata>` pour référencer les métadonnées spécifiées en FAUST,
  - `<notice />` pour insérer une notice des formules mathématiques effectivement imprimées,
  - `<listing [attributes] />` pour insérer le code-source des fichiers FAUST appelés, avec des attributs pour gérer trois options de mise en page (pour plus de détails, cf. référence FAUST [8]).
    @ `mdoctags=[true|false]`
    @ `dependencies=[true|false]`
    @ `distributed=[true|false]`

6. ASPECTS PRATIQUES

6.1. Conditions d'installation

Voici un résumé des conditions d'installations pour générer la documentation mathématique :

- `faust`, évidemment !
- `svg2pdf`, appartenant à la bibliothèque graphique 2D Cairo, pour convertir les blocs-diagrammes SVG ;
- `breqn`, un paquetage LaTeX pour la gestion automatique des sauts de lignes des équations trop longues ;
- `pdflatex`, pour compiler le fichier LaTeX en PDF.

6.2. Exemples en ligne

Pour voir des exemples concrets de cette auto-documentation mathématique qui capture la sémantique mathématique de programmes FAUST, vous pouvez consulter en ligne deux fichiers PDF :

- http://faust.grame.fr/pdf/karplus.pdf (documentation automatique),
- http://faust.grame.fr/pdf/noise.pdf (documentation manuelle).

7. CONCLUSIONS

Nous avons présenté dans cet article un système innovant pour l'auto-documentation mathématique de processeurs de signaux programmés en FAUST. Nous croyons fortement qu'une telle fonctionnalité d'auto-documentation, plus ou moins raffinée, sera bientôt un point crucial pour les langages de programmation à venir, en conjonction avec l'essor des langages compilés et spécifiques à un domaine (ou DSL, *domain-specific languages*).

En effet, les DSL compilés conçus dans l'optique d'être des langages de spécification permettent de laisser les aspects techniques et de mise en œuvre au compilateur, via notamment des techniques de compilation sémantique. De plus, en amont, la perspective de l'auto-documentation pourraient encourager les concepteurs de langages et d'outils informatiques musicaux à établir une sémantique très claire dès le début, ce qui pourrait devenir une bonne pratique à moyen terme, car il est difficile d'obtenir une sémantique claire à posteriori.

Ainsi, les bénéfices des DSL compilés et auto-documentés sont nombreux : principalement la préservation, l'échange, la flexibilité, l'analyse et la maintenabilité.

Comme limitation, il faut préciser que notre modèle actuel d'auto-documentation mathématique est seulement capable de préserver la partie *synchrone* du logiciel de musique en temps réel, mais pas encore la partie *asynchrone*, parce qu'aucun modèle événementiel consensuel n'existe pour le moment. Il faudra sans doute un effort collectif international de plusieurs années, en collaboration étroite, pour pouvoir répondre à ce défi de la modélisation de la partie asynchrone.

Enfin, rêvons d'outils musicaux simples, bien définis, auto-documentés et interopérables.

8. REMERCIEMENTS

Ce travail a bénéficié d'une aide de l'Agence nationale de la recherche, dans le cadre du projet ASTREE, qui porte la référence n° ANR-2008-CORD-003-01.

Nous remercions l'ensemble des partenaires du projet ASTREE, et particulièrement Pierre Jouvelot de MINES ParisTech pour ses commentaires pointus.

9. REFERENCES

[1] D. Knuth, "Literate programming," *The Computer Journal*, vol. 27, no. 2, pp. 97–111, 1984.

[2] N. Scaringella, Y. Orlarey, D. Fober, and S. Letz, "Automatic vectorization in faust," *JIM, editor, Actes des Journées d'Informatique Musicale JIM2003, Montbeliard*, 2003.

[3] Y. Orlarey, D. Fober, and S. Letz, "Parallelization of audio applications with faust," in *Proceedings of the SMC 2009-6th Sound and Music Computing Conference*, 2009, pp. 23–25.

[4] S. Letz, Y. Orlarey, and D. Fober, "Work stealing scheduler for automatic parallelization in FAUST," *Work*, 2010.

[5] N. Bernardini and A. Vidolin, "Sustainable live electro-acoustic music," in *Proceedings of the International Sound and Music Computing Conference*, Salerno, Italy, 2005.

[6] Y. Orlarey, D. Fober, A. Bonardi, F. Rousseaux, L. Pottier, P. Jouvelot, and F. Rousseau, "ASTREE Description of Work," Ircam, Grame, Armines-Cri, Cierec, Tech. Rep., 2008.

[7] J.-C. Risset, *Rapport de mission Art-Science-Technologie*. Ministère de la Culture et de la Communication, 1998.

[8] Y. O. et al., *Faust Quick Reference*, Grame – Centre National de Création Musicale, Lyon, Août 2010.