Abstraction du processus temps réel : une stratégie pour la préservation à long terme

Karim Barkati

Numéros / n° 2 - automne 2012

« Abstraction du processus temps réel : une stratégie pour la préservation à long terme »

Karim Barkati

Résumé

Les recherches décrites ici abordent la problématique de la préservation à long terme du processus temps réel dans la création contemporaine utilisant le numérique. En effet, nous avons développé une stratégie d'*abstraction*, laquelle consiste à générer automatiquement une documentation mathématique qui explicite la sémantique d'un processus, représentée uniquement à l'aide de la notation mathématique et du langage naturel. L'objectif et l'enjeu de cette approche sont à situer dans le statut auto-suffisant de cette documentation, en tant que support autonome pour la réimplémentation.

Introduction

Dans la création musicale contemporaine, le numérique est omniprésent et le risque de voir disparaître les objets numériques constitutifs des oeuvres est reconnu depuis déjà de nombreuses années (Pennycook, 2008 ; Bonardi *et al.*, 2008 ; Bernardini et Vidolin, 2005). Dans le projet ASTREE, nous visons un élément majeur de la création numérique : le *processus temps réel* qui se trouve au coeur de l'interaction homme-machine et dont la préservation à long terme reste éminemment problématique en raison de sa nature logicielle, entravée par un réseau de dépendances elles-mêmes logicielles, matérielles et intellectuelles difficiles à préserver. Or ce processus temps réel est considéré dans le domaine musical comme « l'instrument de musique » d'aujourd'hui, participant de « l'orchestre numérique » (Manoury, 1998) et sa préservation apparaît comme une tâche indispensable dans la perspective légitime de la reproduction future des oeuvres.

Notons d'emblée 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. Une méthode de pérennisation de ces processus temps réel, qui prennent par exemple la forme de *patchs* Max/MSP ou Pure Data, permettrait d'éviter les migrations incessantes et coûteuses de ces objets, les incertitudes sur leur avenir et finalement le risque de voir disparaître une grande partie de la création numérique contemporaine.

Pour répondre à cette problématique, le projet ASTREE s'appuie sur l'hypothèse d'une « lingua franca » pour les processus temps réel, qui se présente concrètement comme un langage de programmation fonctionnel, en l'occurrence le langage Faust (Orlarey *et al.*, 2009). Cette *lingua franca* permet, pour ce qui nous intéresse, la production d'une documentation pérenne sous forme d'un document papier, imprimable à partir d'un document électronique pouvant être enregistré dans un format PDF archivable. Ce document, en tant qu'abstraction du programme d'origine, doit permettre la réimplémentation du processus originel par un acteur humain ne disposant que de la connaissance du langage naturel et de la notation mathématique.

Nous avons étendu le compilateur Faust afin qu'il soit capable de générer unedocumentation mathématique d'un processus temps réel telle qu'elle permette sa réimplémentation, cette caractéristique unique constituant un gage de pérennité pour les développements effectués dans ce langage. Avec ce dispositif, le projet ASTREE vise à préserver la partie synchrone de systèmes interactifs musicaux, en tentant de représenter « en clair » les processeurs de signaux pour le temps réel, comme Jean-Claude

Risset l'appelle de ses voeux (Risset, 1998).

1. L'abstraction comme stratégie pour la préservation à long terme

1. 1. Problématique de la préservation du logiciel à long terme

En matière de préservation du logiciel, plusieurs stratégies peuvent être appliquées, parmi lesquelles la migration et l'émulation (Hoorens *et al.*, 2007), ou la virtualisation (Lorie, 2002). À ces stratégies, le projet ASTREE propose d'ajouter une stratégie supplémentaire : l'*abstraction*. Celle-ci consiste à produire une expression abstraite de l'objet originel sous forme d'une représentation intellectuelle utilisant un socle de connaissances largement partagées, à savoir le langage naturel et un sous-ensemble standard de la notation mathématique, de façon à réduire drastiquement à la fois l'implicite et les dépendances.

De fait, les stratégies de migration, d'émulation et de virtualisation maintiennent le logiciel dans un réseau de dépendances, comme la plate-forme cible, l'émulateur, la machine virtuelle, le système d'exploitation, les bibliothèques logicielles, les ordinateurs compatibles, ou même les connaissances liées à ces nouvelles cibles, qui forment autant d'obstacles à la visée originelle de pérennisation de ces stratégies. Au contraire, l'abstraction permet de s'affranchir de toute dépendance matérielle ou logicielle, en produisant une représentation complète et autonome du processus originel, c'est-à-dire une représentation exempte de toute connaissance implicite ou non communément partagée. Le principal objectif que nous poursuivons est de pérenniser les développements effectués, en en produisant une documentation dans laquelle toutes les connaissances implicites sont rendues explicites au seuil d'un socle commun, en s'appuyant sur des paradigmes de représentation pérennes, à savoir le langage naturel et la notation mathématique ? les seuls à pouvoir être considérés comme tels au regard de la communauté visée. Au reste, le langage Faust s'efface lui-même dans ce passage à la sémantique mathématique en tant que connaissance.

En définitive, cette abstraction doit donc permettre la réimplémentation du processus ultérieurement, dans un temps potentiellement aussi lointain que peuvent le permettre les technologies de préservation de documents papiers imprimés (au format A4).

1. 2. Statut de la documentation par rapport aux normes et aux standards

En matière de préservation de données numériques, le principal standard est la norme OAIS [(1)]. Elle définit notamment des notions importantes comme l'Information de Représentation, la Base de Connaissances et la Communauté Désignée, trois notions qui sont inextricablement liées. La documentation que nous décrivons ici remplit la fonction d'Information de Représentation telle qu'elle est définie dans ce standard, dans la mesure où elle doit permettre la recréation de l'objet originel. Nous définissons ici la Communauté Désignée comme celle des ingénieurs aptes à comprendre le langage naturel et la notation mathématique, qui forment donc quant à eux la Base de Connaissances partagée par la Communauté Désignée.

De surcroît, en amont de la documentation papier, le document numérique généré utilise des formats standards et ouverts susceptibles de faciliter la préservation à moyen terme : PDF [(2)] pour le document lui-même (ISO 32000-1 :2008), SVG [(3)] pour les blocs-diagrammes (recommandation du W3C) et Latex pour les sources (standard de fait dans la communauté scientifique, implémentant les règles typographiques courantes). Nous avons testé la conformité de plusieurs documents avec le standard PDF/A [(4)] spécialisé pour l'archivage et constaté quelques avertissements mineurs, qui sont maintenant en cours de correction.

1. 3. La génération de code multi-cible pour la

préservation à moyen terme

Enfin, en deçà du long terme, l'expression d'un processus en langage Faust permet aussi la recompilation de ce processus vers un grand nombre de plates-formes logicielles, dont Actionscript Flash, ALSA, CoreAudio, CSound, dll, Jack, Qt, GTK, iPhone, Java, LADSPA, Matlab, Max/MSP, Octave, OSS, Pure, Pure Data, Q, Snd-RT, SynthFile, SuperCollider, VST, VSTi, via la génération de code en C, C++, Java et LLVM. Ici encore, on peut considérer le compilateur Faust comme une passerelle d'abstraction, mais cette fois-ci en tant qu'intermédiaire traducteur dans la « Babel musicale » du traitement du signal audionumérique. À ce niveau, Faust permet en effet d'augmenter les chances de survie à moyen terme d'un processus temps réel par la démultiplication de son déploiement, en tant que vecteur de migration logicielle multi-cible.

2. Génération de la documentation

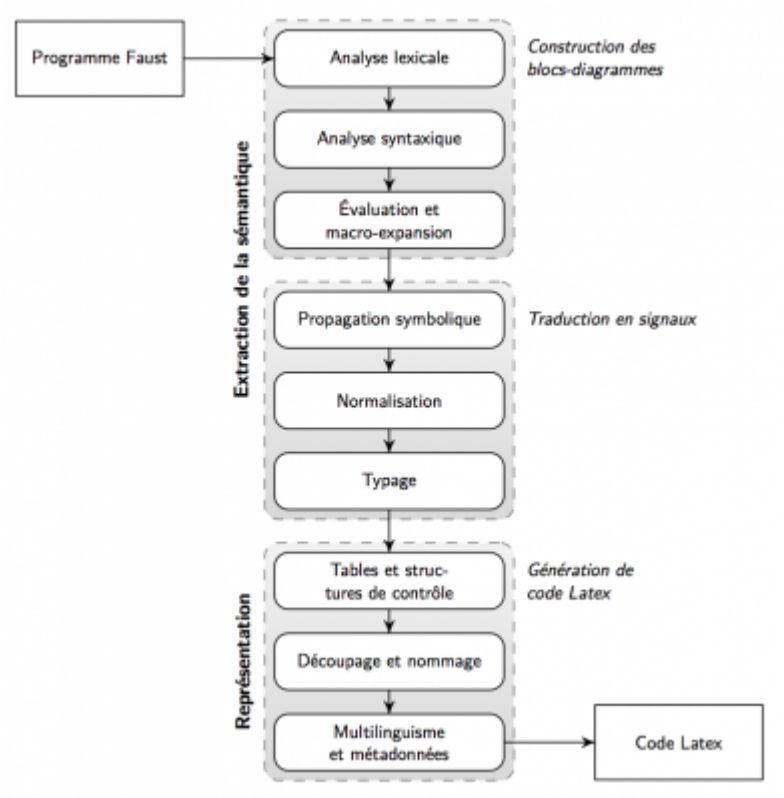
Avant de pouvoir obtenir le document PDF final, il faut d'abord générer le contenu dans un langage de formatage de texte adapté à la fois à la notation mathématique et au langage naturel, ce qui motive notre choix du langage Latex. La génération de la documentation mathématique à partir d'un programme Faust se décompose alors en deux problématiques principales : l'extraction de la signification d'un programme Faust et la représentation de cette information (cf. Figure 1).

2. 1. Le langage Faust et le *documentator*

Faust, acronyme de *Functional AUdio Stream*, est un langage compilé dédié au traitement du signal audionumérique en temps réel. Faust a été conçu comme un langage de *spécification* dirigé par la sémantique, combinant le paradigme de programmation fonctionnelle ? le plus proche du langage mathématique ? et une algèbre de blocs-diagrammes correctement définie (Orlarey *et al.*, 2004).

La partie *documentator* du compilateur permet de générer de façon automatique une documentation mathématique d'un processus programmé en Faust, grâce à la généricité de ce langage pour les processeurs de signaux ; nous entendons par « généricité » la propriété de permettre de décrire non pas simplement un algorithme mais la sémantique complète d'un programme.

Figure 1



Chaîne de compilation de la documentation mathématique

2. 2. Extraction de la sémantique

Le compilateur Faust calcule la sémantique d'un processeur de signaux décrite par un programme Faust en deux grandes étapes : la construction des blocs-diagrammes et leur traduction en signaux (cf. Figure 1).

Pour la construction des blocs-diagrammes, les phases d'analyses lexicale et syntaxique restent ordinaires, tandis que la phase d'évaluation et de macro-expansion fait appel au lambda-calcul (alpha-conversion, bêta-réduction, etc.) et à l'expansion des itérateurs Faust de blocs-diagrammes (*par*, *seq*, *sum*, *prod*).

La traduction en signaux des blocs-diagrammes ainsi obtenus donne ensuite un codage informatique exact de leur sémantique dénotationnelle, telle qu'elle est présentée dans (Jouvelot et Orlarey, 2011). Typiquement, si on a le bloc-diagramme 'A :B' (une composition séquentielle) et si la sémantique des processeurs 'A' et 'B' est *'A' : x -> g(x)* et *'B' : y -> f(y)*, alors on a *'A :B' : x -> f(g(x))*, ou plus précisément avec le temps *'A :B' : x(t) -> f(g(x(t))*. Après cette phase de propagation symbolique, les signaux sont normalisés en leur appliquant un ensemble de règles de réécriture (par exemple *x*1 -> x* ; *x+x -> 2*x*), jusqu'à arriver en *forme normale* (Dershowitz et Jouannaud, 1989), c'est-à-dire dans une forme où plus aucune règle ne peut encore être appliquée. Enfin, le compilateur déduit le typage (statique et implicite dans Faust), pour annoter les signaux (entiers, flottants, constants ou non, d'interface utilisateur, etc.). Cette phase sert à plusieurs objectifs, dont l'optimisation, la parallélisation, la preuve formelle (comme l'interprétation abstraite pour le calcul des intervalles) et la documentation présentée ici.

Pour pouvoir imprimer les équations mathématiques sous une forme normalisée, nous avons dû étendre le compilateur Faust en reprogrammant les classes principales. Cela signifie que, à l'instar de la génération de code audio C++ du compilateur Faust, la sortie Latex est calculée *après* la compilation des processeurs de signaux.

2. 3. Représentation de la sémantique

Le document généré automatiquement est structuré en quatre sections (présentées ci-dessous), précédées d'un en-tête qui se compose du titre, de la date de la compilation de la documentation, des métadonnées du fichier Faust (*name*, *version*, *author*, *license*, *copyright*, etc.) et d'un texte de présentation. En outre, l'implémentation du *documentator* abstrait chacune des chaînes de caractères susceptible d'apparaître dans une documentation de façon à gérer le multilinguisme et propose à ce jour l'anglais, le français, l'italien et l'allemand. Il propose aussi un mode « manuel » présenté dans (Barkati et Orlarey, 2011).

2. 3. 1. Définition mathématique de *process*

La première section du document contient la définition mathématique complète du processeur principal *process*. L'approche *automatique* de la documentation mathématique soulève un problème spécifique, au niveau de l'automatisation du choix des points de découpage pour une présentation des équations qui soit lisible par un humain. De fait, du point de vue de la machine, l'équation de *process*, potentiellement très longue, suffit à elle seule à décrire l'ensemble du processus mais d'une façon indigeste pour l'humain. À l'inverse, découper systématiquement au plus court n'aboutirait qu'à un grand nombre d'équations atomiques donc peu porteuses de sens individuellement.

Finalement, le choix que nous avons arrêté pour le découpage automatique des équations repose sur le système de typage des signaux du compilateur. Les cinq types de signaux suivants supportent les cinq 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

D'autres sous-types permettent d'améliorer encore la relecture en attribuant des indices alphabétiques et/ou numériques aux identificateurs des équations. Par ailleurs, le problème du saut de ligne automatique en Latex pour l'impression des équations trop longues a été levé par le packetage *breqn*, comme le montre le code Latex de la figure 2 (hpf.tex) qui correspond à la formule de $r_1(t)$ affichée sur la figure 3 (hpf.pdf).

Figure 2

```
\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*}
```

Code Latex d'une formule longue (hpf.tex)

Figure 3

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

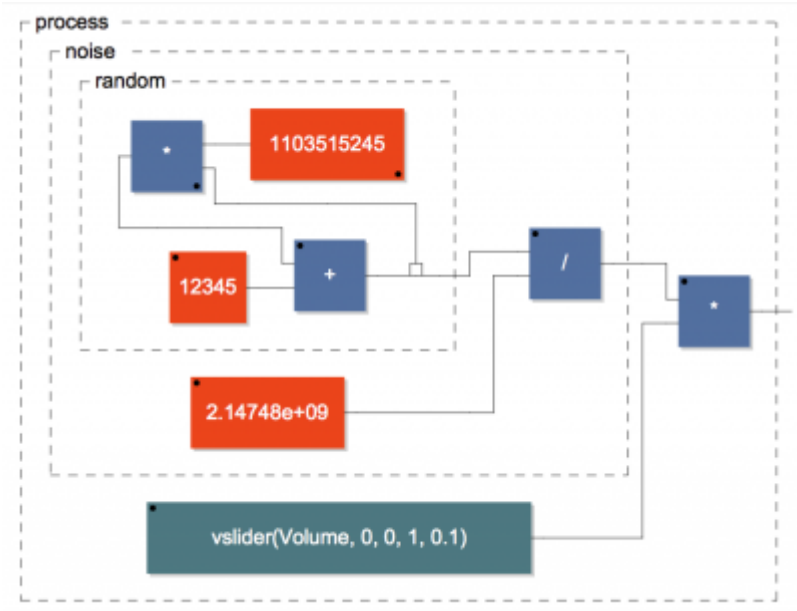
$$k_1 = \frac{6.28318530717959}{f_S}$$

Quelques formules rendues (hpf.pdf)

2. 3. 2. Bloc-diagramme de *process*

Le bloc-diagramme de *process* (cf. Figure 4) illustre l'organisation schématique pour le plus haut niveau du processeur ; les cadres en pointillés délimitent les fonctions nommées, les couleurs varient selon le type des éléments dessinés (opérateurs, constantes, objets d'interface utilisateur, etc.) et le petit carré vide représente un retard d'un échantillon.

Figure 4

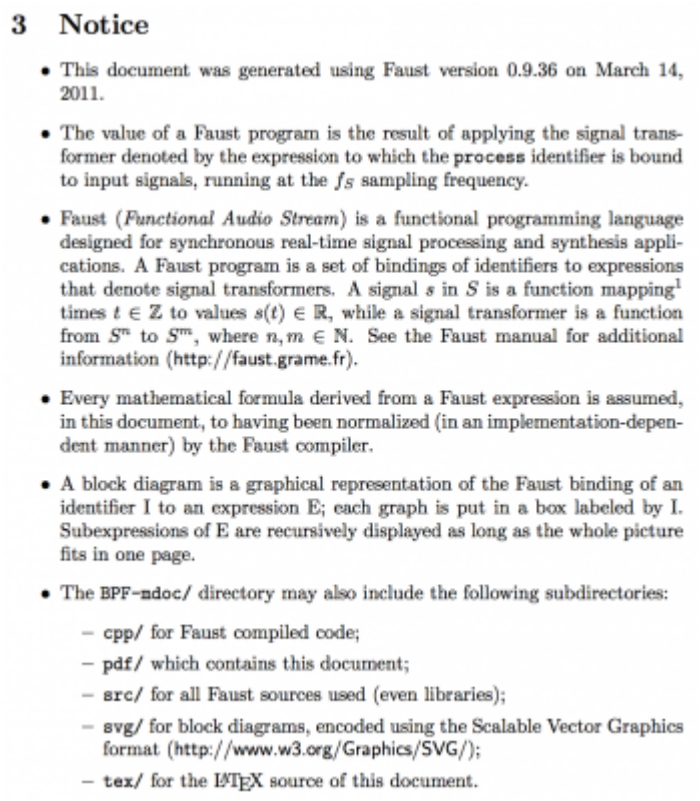


Un bloc-diagramme SVG

2. 3. 3. Notice

La notice explicite plusieurs points de la documentation. Recalculée à chaque compilation, elle donne d'une part des informations contextuelles, telles que la version du compilateur, la date de compilation, une présentation synthétique, l'adresse du site de Faust, l'arborescence des répertoires de la documentation numérique et une présentation du langage Faust lui-même, dont la sémantique des signaux et des transformateurs de signaux (cf. Figure 5). D'autre part, elle définit chaque opérateur susceptible de ne pas être standard, comme la conversion entière et l'addition en complément à deux sur 32 bits (cf. Figure 6).

Figure 5

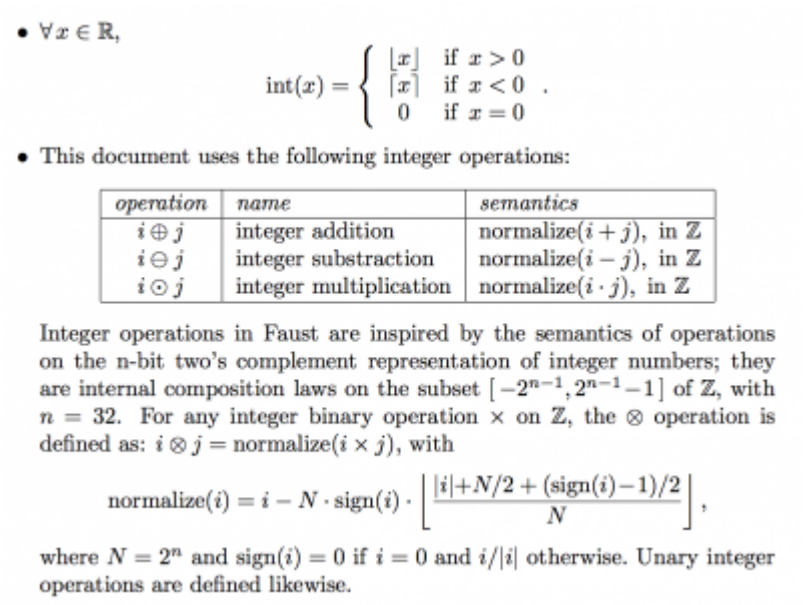


3 Notice

- 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.

Extrait d'une notice (partie contextuelle)

Figure 6



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

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

- 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.

Autre extrait d'une notice (sur des opérations entières)

2. 3. 4. Code Faust

La quatrième et dernière section fournit le code Faust complet : le code source principal ainsi que toutes les librairies nécessaires, avec un système de coloration syntaxique. D'une part, ce code peut restituer ce qui passe d'intentionnalité dans le code Faust du programmeur d'origine, à travers les noms de fonctions, l'organisation du code et les commentaires. D'autre part, ce code extensif constitue un intermédiaire de pérennisation à court ou moyen terme du programme, c'est-à-dire tant que le compilateur Faust restera disponible.

3. Validation de la documentation

Dans le cadre du projet ASTREE, des travaux de validation visant à comparer les processus recompilés avec les processus originels ont été effectués. Ces travaux ont compris aussi bien des travaux de validation technique (différence arithmétique des signaux de sortie entre les processus originaux et recompilés), que des travaux de validation « en situation », comprenant la reprise d'oeuvres musicales du répertoire contemporain (dont « En Echo » de Manoury) en situation de concert (Bonardi, 2011).

3. 1. Objectifs et conditions

L'objectif est ici de tester artificiellement la pérennité à long terme de la documentation mathématique, en termes d'*utilisabilité*, c'est-à-dire de capacité à permettre une réimplémentation de l'objet originel. Nous ne testons pas la pérennité du document numérique produit, en l'occurrence un fichier PDF imprimable en A4, mais celle de son contenu en tant que représentation intellectuelle, sachant que cette documentation peut être conservée et transmise sous forme papier. Nous avons donc tenté de nous rapprocher de la situation d'obsolescence en respectant les trois conditions de test suivantes : différence dans les connaissances, différence dans les outils utilisés et absence de contact direct. Ainsi, nous avons soumis à deux ingénieurs externes les mêmes documentations mathématiques, correspondant à deux objets de test (un effet d'écho et un programme de synthèse additive), dans des conditions proches de celles retenues.

1. Concernant la différence dans les connaissances, le développeur A possède des notions en traitement du signal (il a suivi des modules du master ATIAM [5]), mais ses connaissances ne sont pas les mêmes que celles habituellement partagées par la communauté des réalisateurs en informatique musicale. Le développeur B n'a pas de notions en traitement du signal. Les codes rendus ne sont d'ailleurs pas « temps réel », signe de la non-appartenance de ces développeurs à notre communauté d'informatique musicale.
2. Concernant la différence dans les outils utilisés, le développeur A a utilisé Matlab et le développeur B Mathematica, alors que les objets originaux ont été développés en Max/MSP (avant portage en Faust) pour le premier objet et directement en Faust pour le second. Là, non seulement les outils diffèrent, mais les paradigmes de programmation aussi.
3. Concernant l'absence de contact direct, nous avons soumis la documentation originelle aux développeurs et ils nous ont renvoyé le code développé sans qu'aucun autre contact au sujet de cette documentation n'ait eu lieu. Dès le début, les consignes données aux développeurs ont été strictes de ce point de vue.

3. 2. Objets testés

Deux objets simples ont été choisis pour les tests de réimplémentation, de manière à réduire le temps nécessaire à l'expérience :

1. un programme de synthèse additive, sans entrée, développé originellement en Max/MSP puis traduit en Faust via un traducteur [6] développé à l'Ircam ;
2. un programme d'écho développé en Faust et possédant des entrées et des sorties.

Les documentations de ces deux objets ont été générées de manière automatique ; toutefois, le code Faust original a été légèrement modifié de manière à effacer les indices trop évidents (par exemple, le mot « addsynth » qui aurait éventuellement pu mettre le développeur sur la voie de la synthèse additive).

3. 3. Questionnaire

Nous avons soumis le questionnaire suivant aux deux développeurs afin d'approfondir la validation.

1. La documentation vous a-t-elle permis de réimplémenter le processus originel ?
2. Quelles difficultés, liées à la documentation où à des insuffisances de celle-ci, avez-vous rencontrées durant la réimplémentation du processus ?
3. Avez-vous été aidé dans cette tâche en faisant appel à des ressources extérieures à la documentation mathématique et différentes de celles que vous utilisez habituellement dans votre activité (expert, personne physique, ressource web spécifique au traitement du signal audio, etc.) ?

   a) Non, je n'ai utilisé que les ressources mises en oeuvre habituellement dans mon activité.
   b) Oui, j'ai utilisé des ressources extérieures (préciser lesquelles) :

1. À votre avis, une évolution de la documentation permettrait-elle de résoudre ce ou ces difficultés (pour chacune des difficultés rencontrées, avec éventuellement une piste pour ces évolutions ? non obligatoire) ?
2. À l'issue de cette expérience, diriez-vous que :

   a) La réimplémentation n'a posé aucun problème.
   b) La réimplémentation a posé des problèmes mineurs qui peuvent être résolus avec des

améliorations de la documentation.
c) La réimplémentation a posé des problèmes majeurs qui ont nécessité l'aide de ressources extérieures.
d) La réimplémentation a été impossible.

3. 4. Résultats de la validation

À l'issue des tests, il apparaît que la documentation permet effectivement la réimplémentation du processus originel, puisque les deux développeurs ont réussi à réimplémenter les processus. Néanmoins, les tests mettent en lumière certaines lacunes de deux ordres, identifiées de façon très similaires par les deux développeurs :

- des lacunes mineures dans la documentation, correspondant principalement au manque de définition de certains symboles mathématiques ou à l'ordonnancement des équations ;
- un problème qui semble plus important, à savoir l'absence de jeu de données de test, qui permettrait au développeur de vérifier la validité de sa réimplémentation.

Il semble aisé de répondre aux lacunes identifiées comme mineures, par simple ajout des définitions afférentes et réorganisation de la documentation. De plus, les développeurs ont pu trouver ces définitions par eux-mêmes, sur Internet, ce qui montre que les définitions manquantes sont disponibles et partagées par une large communauté.

En revanche, l'absence de jeu de données de tests constitue un problème plus complexe qui pose deux questions : comment produire un jeu de données de test ? Comment s'assurer de sa pertinence ? La réponse à ce problème nécessite des réflexions et des développements supplémentaires, que nous avons initiés. Plus précisément, nous envisageons une recommandation de « bonne pratique » qui consisterait à programmer un double du processus principal, paramétré de façon à produire immédiatement un signal pertinent pour la documentation. Dans le cas d'un effet, ce signal de sortie devrait représenter la réponse impulsionnelle. On inclurait dans la documentation une liste d'échantillons, dont le nombre reste à déterminer, en fonction de la calculabilité des temps caractéristiques en jeu et du degré de pertinence attendu.

Conclusion et travaux futurs

Dans cet article, nous avons montré comment il était possible de générer automatiquement une documentation mathématique d'un processeur logiciel de traitement du signal à partir d'un langage de programmation fonctionnel, en l'occurrence le langage Faust, susceptible de le pérenniser pour le long terme. En effet, nous avons mis en évidence le fait important que cette documentation s'appuie uniquement sur des paradigmes pérennes tels que le langage naturel et la notation mathématique. Nous avons par ailleurs validé expérimentalement comment une telle documentation est susceptible de rester utilisable dans le futur, en constatant la réussite de la réimplémentation par des ingénieurs ayant des connaissances différentes des nôtres, disposant d'outils différents et ne recevant pas d'informations supplémentaires. Nous avons finalement évalué dans quelle mesure cette documentation était exempte de connaissances implicites, de dépendances matérielles et de dépendances logicielles. Ceci fonde notre approche de pérennisation par *abstraction*, c'est-à-dire par explicitation complète de la sémantique du logiciel avec des représentations communes, sur un support papier, constituant ainsi une documentation autonome pour la préservation à long terme.

De surcroît, nous remarquons qu'une telle documentation rencontre d'autres usages en pratique : en tant que support à la pédagogie montrant explicitement les formules mathématiques qui correspondent au programme d'un processus, en tant que support au débogage pour la même raison et en tant que support préparatoire à la publication, puisque les équations sont compilées et récupérables dans le langage Latex, le standard de fait pour la publication scientifique. Le compilateur Faust est aussi capable de générer du code vectoriel et parallèle et de compiler un processeur pour différentes plates-formes, y compris pour *smartphone*. Ces caractéristiques font que Faust est aujourd'hui intégré dans plusieurs formations supérieures, dont les universités de Stanford (États-Unis), de Maynooth (Irlande), de Saint-Étienne et de Paris 8 (France).

Pour les travaux futurs, nos expériences de validation nous ont permis de détecter quelques lacunes dans la documentation actuelle qui feront l'objet de travaux complémentaires, notamment en ce qui concerne la constitution automatique d'un jeu de données de test.

---

1. *Open Archival Information System* ou Système ouvert d'archivage d'information. Standard ISO 14721 :2003.

2. *Portable Document Format*, un format standard ouvert de description de pages d'impression.

3. *Scalable Vector Graphics*, un format XML de représentation graphique vectorielle.

4. Le format standardisé PDF/A est une version restrictive de PDF, conforme à la norme ISO 19005-1.

5. *Acoustique, traitement du signal et informatique appliqués à la musique*.

6. Prototype développé dans le cadre du projet ASTREE.

Pour citer ce document:
Karim Barkati, « Abstraction du processus temps réel : une stratégie pour la préservation à long terme », *RFIM* [En ligne], Numéros, n° 2 - automne 2012, Mis à jour le 28/09/2012
URL: http://revues.mshparisnord.org/rfim/index.php?id=184