Préservation de processus temps réel

Alain Bonardi, Jérôme Barthélemy, Guillaume Boutard, Raffaele Ciavarella

Préservation de processus temps réel

Vers un document numérique

Alain Bonardi*, Jérôme Barthélemy*, Guillaume Boutard*, Raffaele Ciavarella*

** Institut de Recherche et de Coordination Acoustique Musique,*
1, Place Igor-Stravinsky, 75004 Paris

{jerome.barthelemy, alain.bonardi, guillaume.boutard, raffaele.ciavarella}@ircam.fr

RÉSUMÉ. *La création musicale contemporaine - comme d'autres formes de création, telle la scénographie - fait largement appel à des dispositifs numériques. La préservation de ces dispositifs est problématique en raison de leur forte dépendance au système sous-jacent (hardware et software), de même que leur compréhension et leur interprétation dans le cadre d'une analyse de type musicologique. Nous proposons ici une approche systématique de ce problème, basée d'une part sur les normes OAIS (Open Archive Information System) et CIDOC-CRM (International Committee for Documentation Conceptual Reference Model), et d'autre part sur une algèbre de blocs diagrammes, afin de tendre à conférer à ces objets un véritable statut de document qui puisse être lu et interprété par des êtres humains, et qui puisse garantir leur pérennité à long terme.*

ABSTRACT. *Contemporary musical creation, as well as other forms of creation, like scenography, is widely based on digital processes. Preservation of these digital processes is difficult, due to their large dependency on underlying systems (software and hardware), as well as their understanding and interpretation in the course of musicological studies. We propose here a systematic approach to the problem, based on one hand on the OAIS and CIDOC CRM standards, and on the other hand on a block-diagram algebra, in order to try to confere to these objects a real status of document that can be read and understood by human beings , and guarantee its long-term preservation.*

MOTS-CLÉS : *patch, document, création musicale, constitution de connaissances, préservation, authenticité.*

KEYWORDS: *patch, document, musical creation, knowledge building, preservation, authenticity.*

1. Introduction

L'introduction des techniques numériques dans la production artistique contemporaine, et notamment dans la production musicale, a eu des conséquences majeures, notamment sur les pratiques créatrices, en permettant de nouvelles formes d'interactivité et de nouvelles possibilités d'expression ; mais elle fait également naître de nouveaux problèmes, plus particulièrement en ce qui concerne la pérennité des objets musicaux ainsi créés.

La pérennité de la création musicale a pu être garantie depuis plusieurs siècles grâce d'une part à la notation musicale, et d'autre part à la conservation des instruments de musique, ce qui implique la préservation et la transmission des connaissances et des techniques nécessaires à leur utilisation. En ce qui concerne les processus numériques, la situation est différente : il n'existe pas aujourd'hui de standard de notation pour ces objets, et leur expression est fortement dépendante du système sous-jacent utilisé. En l'absence du concepteur, il devient extrêmement difficile, voire impossible, de documenter ces processus. La réalisation de portages ou de migrations est de même extrêmement difficile, et nécessite la plupart du temps de mobiliser les compétences de ceux qui ont participé à l'élaboration du dispositif.

Toutefois, une part importante de l'activité créatrice est dédiée à la réalisation de ces processus numériques, qui sont au cœur de la pensée musicale contemporaine. Il est dès lors capital de les conserver, de les documenter, de comprendre leur logique, de saisir comment ils participent à l'élaboration de la pensée musicale contemporaine, et de leur donner le statut de document. Pour cela, il importe notamment de développer une méthodologie permettant l'expression de ces processus dans une forme indépendante du système sous-jacent, et de développer les méthodes qui permettront leur analyse – notamment en termes de variants et d'invariants – , et qui devraient permettre à terme de construire une classification organologique de ces processus numériques, comparable à la classification organologique des instruments de musique.

Cette contribution est donc centrée sur la problématique de la préservation des données numériques (logiciels, échantillons etc…), bien que la préservation des informations associées, telles que provenance ou contexte soit à considérer. La relation avec ces informations associées est par ailleurs effectuée en utilisant le modèle CIDOC-CRM, en usage pour la documentation muséographique.

2. L'objet « processus temps réel » ou patch

Les *patchs* sont des modules logiciels de commande (par exemple de déclenchement et de synchronisation de dispositifs numériques) et de traitement numérique du signal utilisés dans les arts de la performance au sens large, comme la musique, la danse, le théâtre, la vidéo, ou encore les oeuvres associant plusieurs arts. Ils reprennent pour la plupart les paradigmes des dispositifs électroniques matériels

utilisés sur scène depuis une cinquantaine d'années (par exemple les processeurs d'effets tels que les réverbérations). Ces *patchs* tirent leur efficacité et leur importante diffusion de leur fonctionnement temps réel (à l'instar des appareils électroniques qu'ils remplacent) et de leur facilité de mise en œuvre grâce à un langage graphique.

À titre d'exemple, nous présentons ci-dessous un module créé avec le logiciel Max/MSP, produisant deux sinusoïdes, à 100 et 200 Hz, et les additionnant, l'amplitude du résultat étant atténuée (multipliée par 0.25). L'aspect du *patch* fait penser à un montage électrique, comme ceux élaborés sur la paillasse d'un laboratoire de physique, avec des appareils qui sont les modules rectangulaires (exemple de l'objet cycle~ qui produit une sinusoïde, à l'instar d'un générateur) et des câbles, qui sont ici les connections entre objets. Il est extrêmement facile de réaliser ces *patchs*, sans programmation, simplement en incorporant en quelques clics de souris des objets prédéfinis, et en les reliant par des connections. Il est de plus possible de programmer de nouveaux objets.

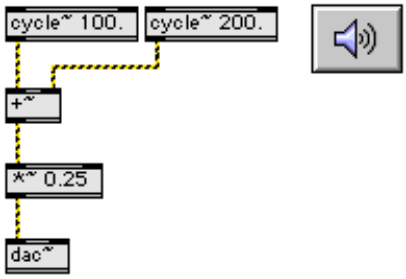


Figure 1. *Exemple de patch Max/MSP produisant la somme de deux sinusoïdes*.

Historiquement, ce type d'approche est apparu à la fin des années 1980. À cette époque, des centres de recherche musicale comme l'IRCAM cherchaient des solutions pour enrichir les musiques mixtes, associant musiciens humains et sons électroniques. Jusqu'alors, une bande magnétique accompagnait les instrumentistes, imposant une temporalité figée. Le chercheur Miller Puckette développa alors une première version d'un logiciel intitulé Patcher, qui préfigurait les deux logiciels les plus utilisés aujourd'hui : Max/MSP et PureData. Le *patch* marque le début des approches interactives dans lesquelles la machine est « en attente » d'informations venant du musicien (ou du performer au sens large), informations acquises par des capteurs. L'une des premières pièces du répertoire IRCAM à exploiter ces possibilités est *Jupiter*, de Philippe Manoury (1987), associant une flûte (dotée de capteurs et d'un microphone pour obtenir un certain nombre de paramètres du jeu du flûtiste) à un dispositif d'informatique musicale temps réel. Plusieurs logiciels ont ensuite vu le jour, s'ouvrant au cours des dernières années à la vidéo et à l'image tri-dimensionnelle, comme Isadora, très utilisé dans le monde de la danse.

Par rapport aux paradigmes traditionnels de la notation et de l'instrument de musique, le patch se positionne d'une manière nouvelle. On peut certes le considérer comme un instrument de musique « virtuel », mais chaque patch est un cas particulier de traitement et perd toute universalité liée à la notion d'instrument.

D'autre part, le patch contient implicitement une sorte de « partition », qui est le « scénario » temporel de ses traitements. Mais les catégories musicales liées à l'instrument et à la partition ne sont jamais explicitement exprimées, elles sont encodées, enfouies sous une description orientée traitement du signal temps réel.

3. La préservation des processus temps réel

Un institut de production comme l'IRCAM possède à son catalogue plus de 500 œuvres. Chacune de ces œuvres vit à travers un cycle de production complexe. C'est lui qui va permettre les évolutions en engendrant de nouvelles expressions du même concept, ces termes étant entendus selon les définitions FRBRoo (Object Oriented Functional Requirements for Bibliographic Records) (F20 Self-Contained Expression et F46 Individual Work) (cf. documentation en ligne du modèle FRBRoo). Ce cycle consiste en des re-productions successives qui sont autant de réinterprétations de l'œuvre au travers des évolutions technologiques. Chaque expression de l'électronique, outre le patch Max/MSP, va consister en une multitude de fichiers comprenant échantillons, fichiers de configuration ou de synchronisation, etc., dont l'utilisation peut ou non être liée à la fonction originelle du type (par exemple, un fichier AIFF utilisé comme flux de données de contrôle). Ce cycle entretient par là-même l'œuvre. Or, un petit nombre d'entre elles sont régulièrement l'objet d'une nouvelle performance. Soumises à l'obsolescence rapide des technologies que la musique mixte implique, elle deviennent de plus en plus difficilement reproductibles. Il devient donc nécessaire d'instaurer une méthodologie adéquate.

Outre la complexité inhérente au statut des œuvres mixtes, celles-ci peuvent atteindre un niveau de complexité supplémentaire par l'adjonction d'œuvres vidéos, chorégraphiques et littéraires. Une œuvre comme *Avis de tempête* de Georges Aperghis, ne peut être envisagée sans questionner le lien entre le livret de Peter Szendy, la partition et l'électronique ; une œuvre comme *Double Bind?* de Unsuk Chin sans questionner le lien entre le geste instrumental et l'électronique. Cette complexité affecte la gestion de l'authenticité.

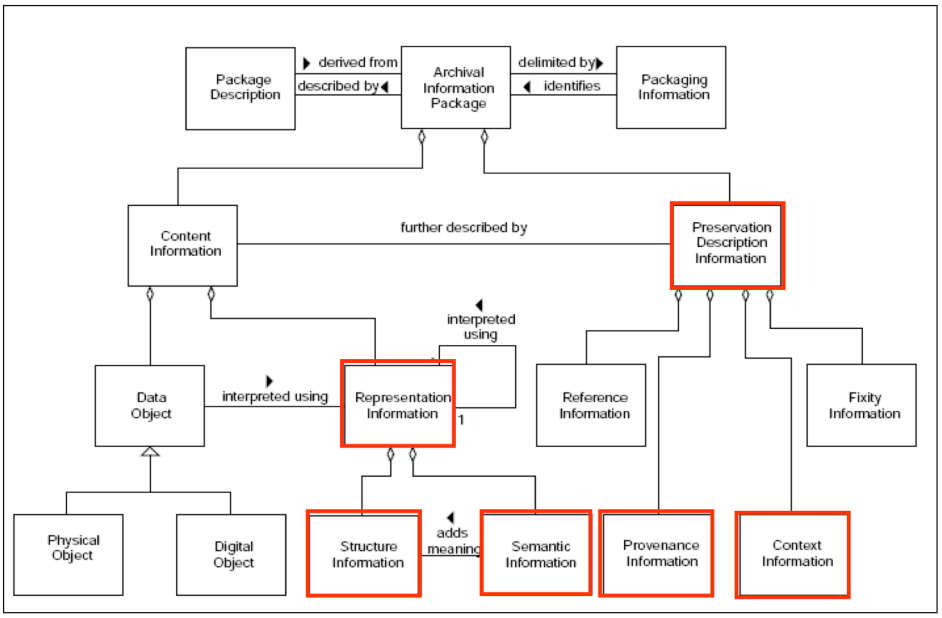


Figure 2. *Paquet d'informations archivé (vue détaillée) du modèle OAIS, montrant (entourés en gras), les concepts que nous approchons plus spécifiquement dans cet article.*

Une méthodologie adéquate sur la base du modèle OAIS (documentation en ligne du modèle OAIS : http://public.ccsds.org/publications/archive/650x0b1(F).pdf) doit être développée. Il n'entre pas dans les objectifs de cet article d'expliciter en détail les éléments conceptuels du modèle, mais, dans l'optique de la préservation des données descriptives de l'authenticité, il convient de mettre l'accent sur la provenance et le contexte du PDI (Preservation Description Information). En effet, ces informations ne peuvent pas être fournies de manière automatique, et il ne peut exister de méthodologie générale pour les exprimer (contrairement aux informations de référence et fixité). Il appartient donc au gestionnaire de l'archive de fournir la méthodologie adéquate pour les exprimer.

Dans notre cas, nous avons choisi d'exprimer les concepts de provenance et de context, en les modélisant sur la base des ontologies muséographiques CIDOC CRM (documentation en ligne du modèle CIDOC CRM) et son extension bibliographique FRBRoo (documentation en ligne du modèle FRBRoo).

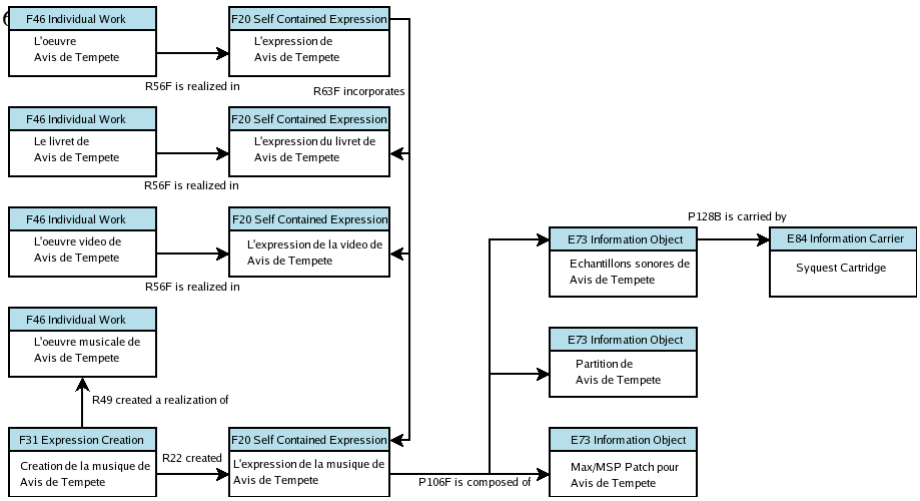


Figure 3. *Une partie de la modélisation de la provenance et du contexte pour l'œuvre Avis de Tempête de Georges Aperghis*.

Les concepts de provenance et de contexte pouvant être considérés comme sécants, leur définition est fournie sur la base de requêtes (Doerr *et al.*, 2007). Il devient, à l'extrême, possible de définir la provenance comme un sous-ensemble du contexte. La gestion de l'authenticité est un processus qui doit être défini en amont à travers un protocole (documentation en ligne du modèle conceptuel de référence CIDOC). Ce processus doit prendre en compte de façon récursive le protocole, incluant la définition de la provenance et du contexte de façon itérative en fonction du cycle de vie de l'œuvre, et la définition du protocole, définissant les critères de comparaison d'un système de migration. On donnera pour exemple de problématique le portage en 1999 d'une œuvre de 1985 de Luigi Nono : *A Pierre, dell'azzurro silencio, inquietum* ; après la mort du compositeur, le transfert en numérique sous Max/MSP du traitement analogique original (Battier *et al.*, 2000) avait soulevé plusieurs contestations sur son authenticité. D'après un des réalisateurs en informatique musical responsable du portage, Olivier Pasquet, la polémique fut résolue après l'intervention de l'assistant musical de Luigi Nono, André Richard, par une modification dans le patch des spécifications premières des filtres utilisés.

4. Information de représentation : l'approche de l'IRCAM

Un autre concept majeur défini par le modèle OAIS est le concept d'Information de Représentation. Il s'agit ici de fournir les informations qui vont permettre de décoder le flot numérique, et de le re-présenter.

De la même manière que pour le contexte et la provenance, il n'existe pas de méthodologie générale pour extraire l'information de représentation. La

méthodologie adéquate doit impérativement être développée par le gestionnaire de l'archive. C'est cettte méthodologie que nous décrivons ici.

Un tour d'horizon des différentes démarches entreprises dans les différentes institutions concernées montre que les efforts de préservation se répartissent selon quatre types :

– La préservation au sens de la sauvegarde matérielle d'éléments liés aux œuvres, que nous n'étudierons pas ici, ce type de processus étant le plus largement documenté. Nous avons par exemple décrit la base Mustica dans des publications précédentes (Bonardi *et al.*, 2007).

– L'émulation de dispositifs techniques disparus.

– La migration d'une plateforme technique ancienne à une autre plus récente.

– La virtualisation des processus.

4.1. *Les approches d'émulation et de migration*

L'émulation est certainement l'une des approches les plus difficiles. Bernardini et Vidolin (Bernardini *et al.*, 2005) citent l'exemple d'*Oktophonie* de Stockhausen, qui nécessite pour son exécution un ordinateur ATARI-1040 ST qui n'existe plus (sauf chez les collectionneurs). Il existe des émulateurs de cet ordinateur sur d'autres machines, mais personne ne sait si le programme Notator utilisé par Stockhausen à l'époque fonctionne sur un émulateur. Les communautés d'utilisateurs/collectionneurs d'ATARI sur le Web pourraient être d'une grande utilité dans ces démarches d'expérimentation pour remonter l'œuvre.

La migration est un processus qui consiste à reprendre une version précédente d'un patch pour l'adapter à une évolution technique, comme le changement de système d'exploitation, une évolution technique majeure permettant de nouveaux traitements, ou encore une incompatibilité entre versions d'un même logiciel. L'activité de migration est la plus répandue lorsque l'on remonte des œuvres. De nombreux compositeurs ont eu leurs œuvres portées d'un environnement technique à un autre. Toutes les institutions dans le champ des arts électroniques font face à des nécessités de migration. À l'IRCAM, les pièces importantes utilisant des ordinateurs Next ont été migrées vers des ordinateurs Macintosh à la fin des années 1990. Avec des résultats jugés de manière très variable selon les compositeurs : la même procédure de migration a ainsi abouti pour certains à un résultat trop proche de l'original (et du coup ne tirant pas assez parti des nouvelles technologies mises en œuvre) ; d'autres le trouvent trop éloigné, n'ayant pas assez à voir avec l'original.

La migration, en général motivée par des raisons techniques, peut également être perçue comme un appel à re-création pour le compositeur. Le compositeur et théoricien Jean-Claude Risset (Risset *et al.*, 2002) a créé une nouvelle œuvre, *Resonant Sound Spaces*, avec l'aide d'Antonio de Sousa Dias, Daniel Arfib, Denis

Lorrain et Laurent Pottier, qui avaient formalisé informatiquement certains aspects de ses œuvres précédentes. Le travail de Sousa Dias (De Sousa Dias, 2003) a été fondamental pour Risset, tout particulièrement sa programmation de structures harmoniques avec des outils temps réel.

Au cours des dernières années, les migrations ont essayé d'éviter les solutions propriétaires. Bullock et Coccioli (Bullock *et al.*, 2005), et Miller Puckette (Puckette, 2004) ont mis l'accent sur la nécessité de standards et de formats ouverts pour sauvegarder tous les fichiers impliqués dans un processus de création. Ils recommandent par exemple de sauvegarder les *patchs* produits dans Max/MSP au format texte ASCII plutôt qu'en binaire, certes plus rapide à charger ; de même, toute la documentation devrait être écrite en format « text » ; enfin, les contenus sonores devraient être sauvegardés sous un format de fichier WAV de base.

4.2. *La virtualisation*

Dans la perspective d'une préservation à long terme des processus numériques, nous privilégions la virtualisation des processus, en tant que technique qui permet de rendre l'objet numérique indépendant d'une implémentation particulière sous-jacente.

Cette approche n'est pas toutefois sans poser de nombreux problèmes. En première analyse, et en matière d'exécution de programmes temps réel, de nombreux comportements ne sont pas documentés, ou insuffisamment. Par exemple les programmes temps réel emploient un « scheduler » (ordonnanceur) qui régule l'exécution des différents composants du programme dans le temps. L'algorithme employé par le scheduler est la plupart du temps non documenté dans le cas de logiciels tels que Max/MSP ou PureData. Toutefois, les résultats obtenus sont dépendants de cet ordonnanceur. Il importera donc, en complément de la virtualisation, de documenter les résultats attendus du processus, et de manière à permettre la comparaison avec une exécution ultérieure du processus sur une machine différente.

Au delà des ces problèmes pratiques, il apparaît à l'évidence une difficulté majeure : en effet, tout processus de virtualisation implique une série de modifications majeures apportées au document original, aussi bien en termes de forme (syntaxe) que, potentiellement du moins, en termes de sémantique. C'est pourquoi nous développons une méthodologie précise, décrite ci-après, afin de garantir la conformité de cette représentation virtuelle à l'original, conformité qui passe notamment par l'établissement d'une relation biunivoque entre l'original et la représentation virtuelle.

5. Les développements en cours

5.1. *Architecture du système et choix fondamentaux*

5.1.1. L'ordinateur virtuel universel

L'idée centrale de la solution que nous développons à l'IRCAM, est de convertir les patchs en des programmes virtuels exécutables pour un « ordinateur virtuel universel » (UVC, Universal Virtual Computer), dont le concept a été développé à l'origine par Raymond Lorie (Lorie, 2001). L'UVC est suffisamment simple pour rester pertinent sur une période importante. Dans la mesure où l'UVC peut être appliqué sur n'importe quel ordinateur, les programmes sur l'UVC deviennent indépendants de la technologie.

Un programme UVC peut être écrit pour décoder tout format de fichier spécifique. Nous avons déjà développé des décodeurs pour les formats Max/MSP et PureData. Pour que cette solution soit utilisable pour l'interprétation des patchs, un traducteur (Décodeur) a été développé qui tourne sur l'UVC et traduit le patch en question en une « vue logique de données » (LDV, Logical Data View). La LDV est une description structurée de l'objet numérique, générée selon un schéma donné, le « schéma logique de données » (LDS, Logical Data Schema).

Pour analyser ou exécuter l'objet dans dix, quinze, ou même cent ans, un émulateur UVC pourra être écrit qui fera tourner un décodeur Max/MSP, PureData ou Reaktor, générant une LDV. Puisque le LDS est lui aussi préservé, les futurs programmateurs seront capables de comprendre la LDV et de développer des visualiseurs ou des interpréteurs.

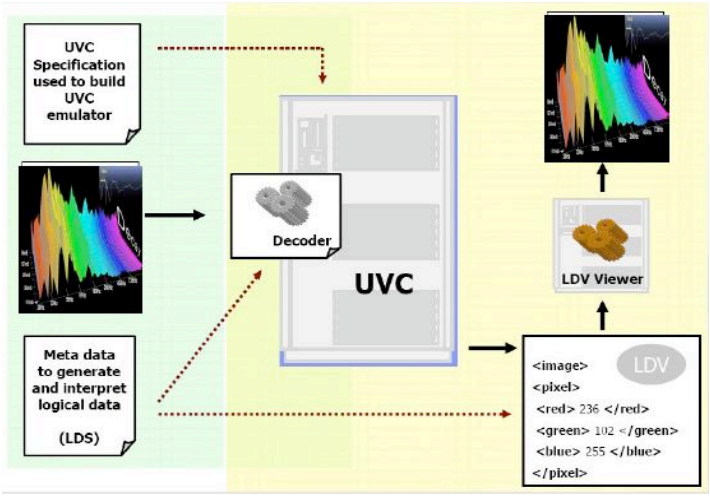


Figure 4. *Schéma de l'architecture UVC.*

5.1.2. Le principe de base de l'UVC

Les segments sont au cœur du concept utilisé à l'intérieur de l'UVC. Les segments sont des espaces linéaires de mémoires adressables avec potentiellement une quantité illimitée de registres associée à chaque segment.

Les programmes écrits pour l'UVC utilisent cette mémoire basée sur des segments. Ces programmes UVC devraient consister en plusieurs fragments de code, appelés sections, qui sont stockés dans des segments séparés. Une section se compose de plusieurs instructions UVC, qui, organisées en séquence, forment une routine, effectuant une certaine opération sur les données. Chaque section peut effectuer des appels à une ou plusieurs sections, liant ainsi le programme dans sa totalité. L'utilisation de sections rend le design des programmes UVC plus flexible. Les programmes peuvent être découpés en différentes parties.

5.2. *Modélisation algébrique et réduction de complexité*

En partant de ces principes et avant de commencer les développements, une phase préliminaire de recherche a été menée, premièrement afin de prouver que les langages utilisés (langages de patch comme Max/MSP) peuvent être modélisés algébriquement, et deuxièmement de vérifier certaines hypothèses permettant d'effectuer une réduction de complexité avant transformation.

Des matériaux intermédiaires, générés après le processus de réduction, nous avons extrait une description formelle, par application de la théorie générale des langages. Cette phase préliminaire fournit une description grammaticale et syntaxique du langage (LDS, Logical Data Scheme). Cette description est exprimée en XML, un langage auto-descriptif (préservable par lui-même).

Nous avons par ailleurs eu à sélectionner un langage de destination pour la virtualisation. Nous avons choisi le langage FAUST (Orlarey *et al.*, 2002), pour ses capacités d'expression, de concision et de cohérence, et aussi pour le fait que ce langage développé depuis 2000 par le GRAME possède déjà un compilateur capable de générer du code C++.

Le LDS est aussi utilisé en entrée pour générer le code source pour le codec spécifique. Le code source est généré dans un langage portable et compilé séparément. A la fin de cette phase, nous sommes capable d'appliquer ce codec sur n'importe quel patch afin de produire son double virtuel. Afin de s'assurer de la validité du processus nous avons appliqué une conversion inverse et vérifié la consistance avec l'original. Cette méthode implique le développement d'un nouveau codec (celui de la transformation inverse) pour tout langage source.

Nous avons développé plusieurs outils, sur la base de la LDV, pour analyser de façon normalisée nos matériaux sources et fournir leur Information de Représentation au sens de la norme OAIS (RI, Representation Information). La RI

est à la base du processus de préservation, elle est le moyen de fournir aux utilisateurs futurs la capacité de comprendre les données.

Enfin, un visualisateur LDV a été développé. Il démontre la possibilité de représenter les données de la même façon que le programme original le fait.

La production d'un nouvel environnement de développement reste toutefois en dehors de notre champ de recherche, et n'est pas dans nos objectifs, mais nous espérons que notre travail pourra être intégré par les développeurs d'environnements de développement afin d'assurer la pérennité des productions futures de leurs utilisateurs.

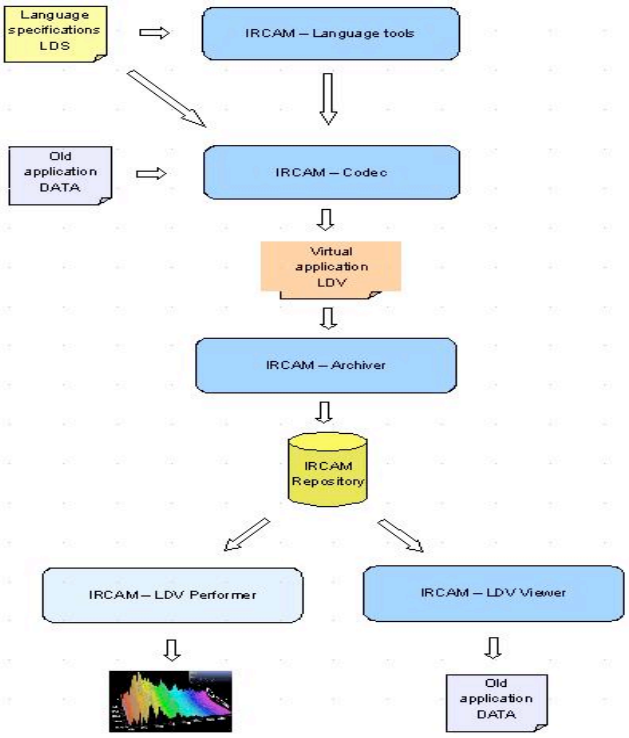


Figure 5. *Phases du processus*.

5.3. Outils

5.3.1. Réduction de complexité du langage

De par la nature des patchs et en raison de leur complexité, nous estimons utile la réalisation d'une phase préliminaire de réduction de complexité.

Une telle réduction de complexité est rendue possible par l'application du système de règles mathématiques connu sous le nom d'algèbre de bloc diagramme. Toutefois, ceci implique un processus relativement complexe comprenant l'identification des nœuds, des divers chemins (aller/retour, continu/non continu) et le calcul du gain correspondant. Ceci implique un processus complexe où jusqu'à 33 règles ont été introduites (Karayanakis, 1995) qu'il pourrait être nécessaire d'appliquer.

Nous avons donc appliqué un nouveau concept : le flux principal/périphérique et les règles de décalage correspondantes.

5.3.1.1. Les bases des blocs diagrammes

En général, le bloc diagramme d'un système linéaire invariant dans le temps est constitué de quatre composants, le signal, le bloc (avec la fonction de transfert), le point d'addition et le point de séparation.

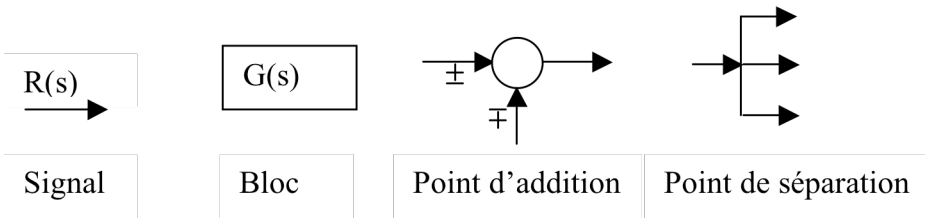


Figure 6. *Les composants d'un bloc diagramme*.

L'algèbre de bloc diagramme est basée sur des opérations simples telles que les blocs en séries/récursifs, les blocs parallèles et les boucles de rétro-action.

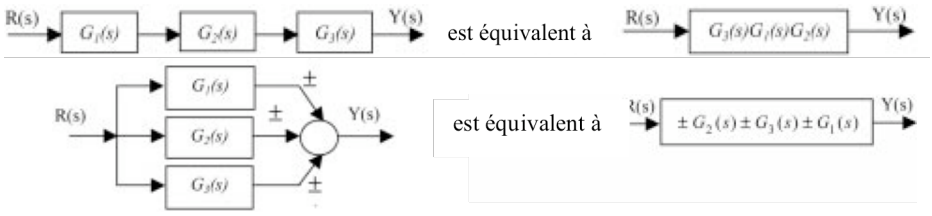
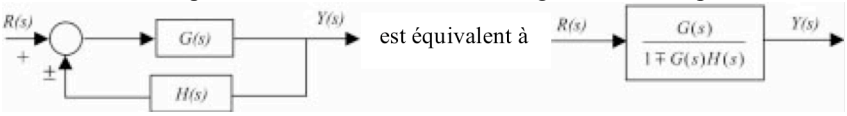


Figure 7. *Les bases de l'algèbre de bloc diagramme*.

Les blocs en série se combinent entre eux par multiplication et les blocs en parallèle par addition. Le bloc, une fois combiné, est interchangeable dans les deux cas. Notons que la compréhension des propriétés des jonctions additives en série est cruciale dans la simplification de certains blocs diagrammes : les points d'addition



en série peuvent être échangés, combinés et séparés algébriquement.

L'équivalence des jonctions additives en série avec l'opérateur d'addition mathématique va de soi.

5.3.1.2. La simplification des blocs diagrammes par l'algèbre de bloc diagramme

Un facteur de complexité est lié à l'existence de points d'addition/séparation à l'intérieur d'une boucle. Une simplification du bloc diagramme peut toujours être obtenue au moyen du déplacement de ces points de la façon appropriée. Dans la pratique courante de simplification par l'algèbre de bloc diagramme, en addition aux trois règles décrites (figure 7), de nombreuses autres règles sont introduites dans la littérature concernant le déplacement des points d'addition/séparation. Chaque règle implique une paire de blocs diagrammes équivalents. L'équivalence peut facilement être vérifiée en traçant le signal en entrée et en sortie et en vérifiant la concordance des signaux de sortie dans les deux cas (Knuth, 1973).

5.3.1.3. Une nouvelle approche dans la simplification des blocs diagrammes

Le flux principal/périphérique est défini ainsi : à un point d'addition ou de séparation, des signaux arrivent et partent. Un flux principal est défini comme allant dans une direction unique, qu'il soit arrivant ou sortant du point d'addition/séparation, le reste étant considéré comme flux périphérique. Avec l'introduction des concepts de flux principal/périphérique, les points d'addition/séparation peuvent facilement être déplacés en vue d'une simplification en appliquant les règles de décalage suivantes:

- Pour supprimer un bloc G d'un flux principal, le bloc G doit être ajouté à tous les flux périphériques.

- Pour supprimer un bloc G d'un flux périphérique, un bloc G doit être ajouté au flux principal et un bloc 1/G doit être ajouté à tous les autres flux périphériques.

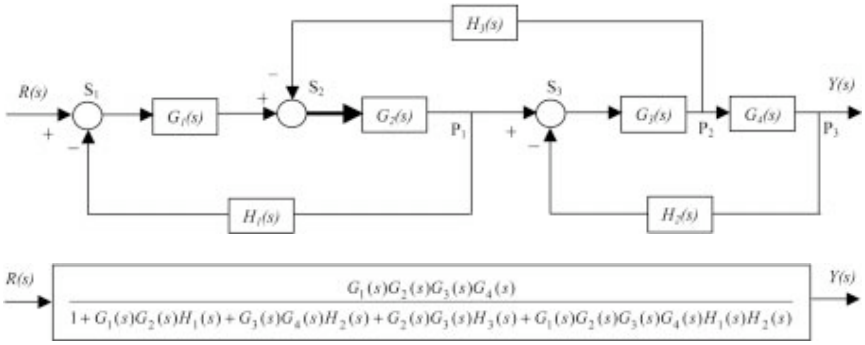


Figure 8. *Réduction d'un diagramme de bloc par les règles de décalage*.

Premièrement, nous identifions le flux principal et les flux périphériques de chaque bloc selon les points d'addition/séparation. D'après la définition, la ligne la plus foncée (figure 8) représente le flux principal dans tous les cas.

L'équivalence entre une paire de bloc diagrammes est alors facilement obtenue en application d'une seule règle – la règle de décalage.

De l'analyse précédente, on voit que les concepts de flux principal/périphériques et les règles correspondantes de décalage éliminent le besoin d'introduction d'une douzaine de règles d'algèbre de bloc diagramme dans la simplification de bloc diagramme.

5.3.2. Outil de description de langage

Pour décrire le langage source nous avons utilisé la forme BNF (Backus-Naur Form), un « méta-langage » ou langage de description de langages, à la base développé pour décrire le langage ALGOL en 1960. Elle est généralement la base de la description des langages de programmation et de définition de leurs analyseurs syntactiques. Afin de rendre le développement plus flexible nous avons adopté le format G pour les descriptions de langage, un format de description commun proposé par Jean Bovet (Bovet, 2003) et devenu un standard *de facto*.

Ainsi nous avons été capable de prototyper, simuler, déboguer et naviguer dans les éléments grammaticaux.

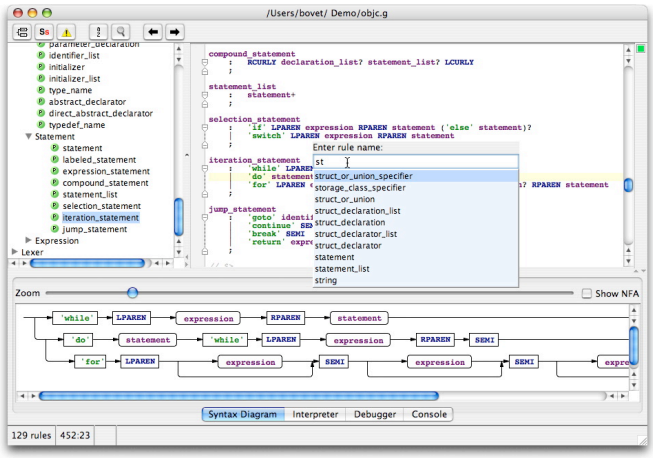


Figure 9. *Outil pour la description de langage et la simulation*.

5.4. *Génération du codec*

Nous avons structuré notre codec comme un traducteur de langage moderne (compilateur), afin qu'il se divise logiquement en cinq parties:

- le frontal du compilateur,
- le moteur de construction de la table des symboles,
- le moteur intermédiaire,
- le générateur de code,
- l'optimisateur.

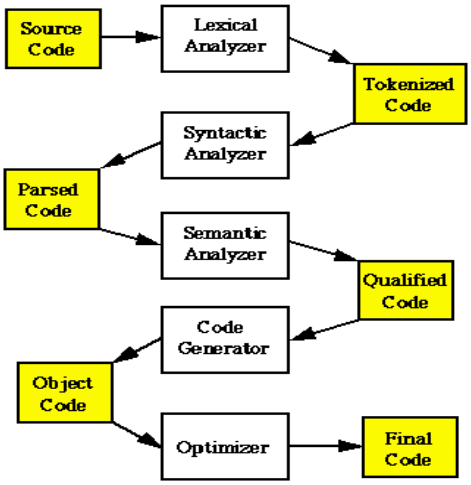


Figure 10. *Schéma de génération du codec*.

Le frontal du compilateur est composé d'un scanneur et d'un parseur qui lit le code source et construit une représentation du code source sous forme d'un arbre syntaxique abstrait (Abstract Syntax Tree, AST). Après conversion de la source en AST, le frontal résout les déclarations de symboles, procède à une analyse sémantique et construit la table des symboles et autres structures de données en support. Toutes les structures de données sont décrites dans le format XML dans des fichiers temporaires privés. La sortie du frontal est un AST où chaque nœud est annoté avec soit un type soit une information de symbole.

La table des symboles est une des structures de données clefs du compilateur. Contrairement à l'AST qui peut être supprimé après que le graphe d'écoulement ait été construit, la table des symboles « vit » tant que le code source est compilé. La portée du langage et l'absence d'identifiant unique à l'intérieur d'un domaine complique la construction de la table des symboles.

Le moteur intermédiaire effectue des transformations d'arbre à arbre et construit le graphe de contrôle de flux (control flow graph) des blocs de base sur lesquels travaille l'optimisateur. Cette phase est aussi rendue persistante par l'utilisation de fichiers XML privés.

Le générateur de code produit des instructions pour le processeur cible. Nous utilisons le langage Java pour construire le générateur de code (Java), de manière à pouvoir le compiler sur presque n'importe quel système et à assurer en une certaine

mesure sa pérennité. La phase de génération de code effectue aussi une optimisation en fonction de la machine, ceci incluant une optimisation de type « peep-hole » et une planification de type load/store.

L'optimisateur construit la structure de données qui décrit l'usage des variables tout au long du graphe de contrôle de flux pour chaque méthode (habituellement appelé flux de données global, global data flow). Cette information est utilisée pour optimiser les références de données globalement à l'intérieur d'une méthode.

5.5. *Vérification du caractère bi-univoque de la transformation*

Afin de certifier la validité d'une transformation nous devons prouver la cohérence de la fonction de transfert. Pour ce faire, et pour la même raison pour laquelle nous avons besoin de la transformation inverse, nous avons décidé de vérifier le processus en utilisant une méthode de validation de transfert bi-univoque.

Le but de cette méthode est de valider la fonction de transfert, en fournissant la preuve qu'il est à tout moment possible de revenir à la version originale à partir de la transformée. Il n'entre pas dans les objectifs de cette méthode de valider la conformité de la transformée (ou des résultats obtenus grâce à celle-ci) à l'original. La validation de ces résultats est l'objet du développement d'une méthodologie complète, visant à garantir l'authenticité des objets transformés [Guercio 2007].

Selon cette méthode, deux codecs sont produits pour chaque traduction, le premier génère la LDV à partir du langage source et le second génère le même code source à partir de la LDV. Cette méthode n'est pas applicable à tous les langages mais nous avons démontré son applicabilité dans le cas des langages Max/MSP et PureData.

La source originelle et celle générée en retour sont comparées – par exemple en utilisant l'outil standard DIFF, et aucune différence ne doit être trouvée en principe. Toutefois, en pratique, il existe des différences entre l'original et la copie, celles-ci portant sur des différences dont la sémantique n'est pas majeure, par exemple, une différence dans l'ordre dans lequel les objets apparaissent dans l'original et dans la copie (il s'agit uniquement ici bien sûr que de l'ordre dans le fichier texte, et non de l'ordre des objets dans le bloc-diagramme). Cette différence peut avoir des conséquences dans l'exécution du programme, et notamment en fonction des règles appliquées par le scheduler (l'ordonnanceur). Il importera donc de documenter ces différences afin de permettre leur analyse ultérieure si nécessaire.

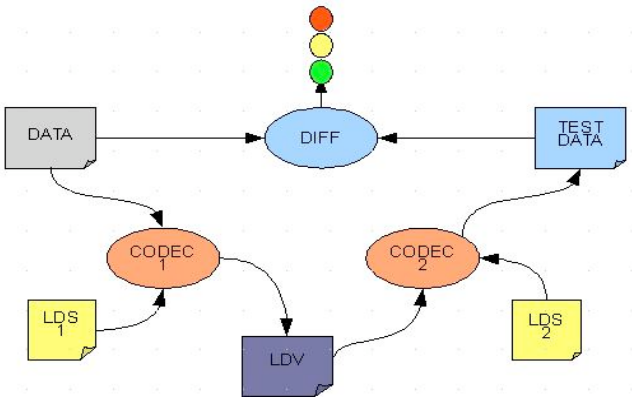


Figure 11. *Schéma de mise à l'épreuve*.

5.6. *Outils d'analyse automatique du code*

Nous développons deux types d'outils d'analyse automatique du code généré : des outils d'analyse statique du code d'une part, et des outils d'analyse des performances à l'exécution d'autre part.

La classe des outils d'analyse statique du code doit fournir, dans des schémas structurés, plus d'informations sur la structure interne de l'objet, incluant notamment une description de l'environnement, la structure hiérarchique interne de l'objet, la structure des fichiers et les références aux libraries externes utilisées. Les requêtes sur le contenu seront possibles dans un but de localisation de versions plus récentes d'un composant ou de versions du même composant sur un système d'exploitation différent. Des valeurs de hachage sont associées à chaque composant afin de permettre leur identification précise.

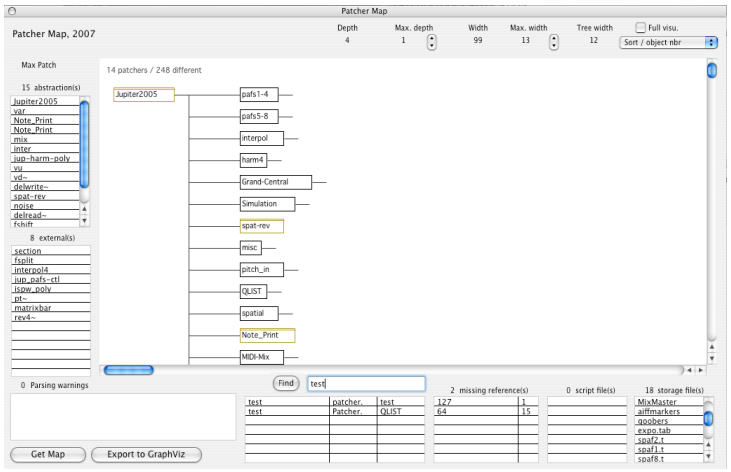


Figure 12. *Outil d'analyse statique*.

La classe d'outil concernant l'exécution fournira le moyen de redimensionner de façon adéquate le matériel informatique et de surveiller les performances. Ces outils sont spécifiquement orientés vers le traitement musical ou graphique, ils fournissent donc un contrôle direct sur des structures complexes comme les bus, les pipe-lines et tous les formats de données spécifiques.

5.7. *Le visualisateur LDV*

Nous avons développé un visualiseur à partir de notre description LDV afin de prouver la conformité à l'original des possibilités de visualisation, sans perte sémantique. Ce visualisateur, qui existe actuellement sous forme d'un prototype, ne permet qu'un nombre limité d'opérations par l'utilisateur, et ne permet notamment pas d'opérations d'édition, telles que les modifications du graphe ou la création de nouveaux graphes. Toutefois, nous prévoyons de le faire évoluer en lui appliquant des outils d'analyse dans un but de recherche, notamment musicologique.

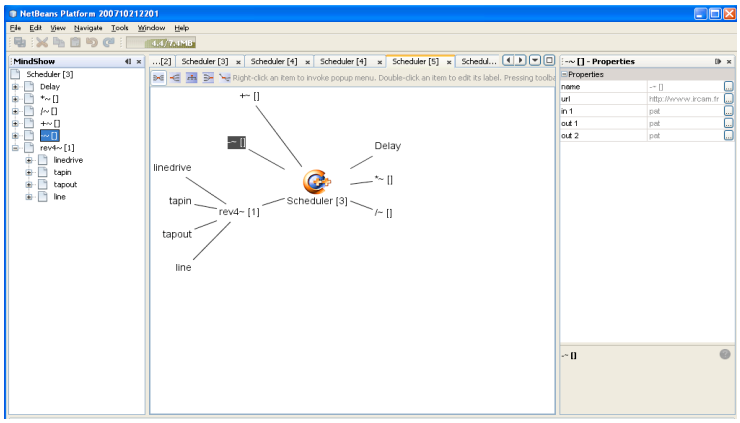


Figure 13. *Le visualisateur LDV Max/MSP*.

5.8. L'environnement d'exécution LDV

Nous n'avons pas prévu actuellement de développer d'environnement d'exécution temps réel directement sur la base de notre représentation LDV. Ce type de développement – extrêmement complexe - est hors de notre propos et dépasserait par ailleurs le cadre que nous nous sommes fixé. Toutefois, la capacité de notre système à traduire les processus en langage FAUST permet de s'affranchir de cette nécessité. Il existe en effet un compilateur FAUST qui permet de générer du code C++ efficace, et il est aussi possible de générer du code PureData.

6. Perspectives et conclusion

Nous avons donc montré que nous sommes en mesure de décrire et d'interpréter les expressions originelles des processus temps réel, et de les réencoder sous une forme faisant appel à une syntaxe commune (XML en l'occurrence), et à un référentiel commun d'opérateurs, ouvrant la voie à une méthode générique de portage et de préservation de processus temps réel.

Toutefois, au-delà de l'analyse et de l'encodage des processus temps réel sous une forme commune, utilisant une syntaxe commune et un référentiel commun, comme décrit ci-dessus, il nous semble nécessaire de développer un langage à la fois concis, expressif, et cohérent, qui permette simultanément d'encoder les expressions des processus temps réel, et de permettre leur analyse, afin de développer l'organologie des instruments électroniques dont nous parlions en introduction. Il va de soi qu'un tel langage devra aussi permettre la réimplémentation du processus, et qu'il puisse être interprété ou compilé.

Le langage FAUST (pour Functional AUdio STreams) (Orlarey *et al.*, 2002) est un langage de programmation dédié à la création de processeurs de flux. Ce langage a été développé par le GRAME à partir de 2000 pour aider les programmeurs, mais aussi les compositeurs et les réalisateurs en informatique musicale à construire de tels processeurs de flux audio.

Le langage FAUST a pour caractéristique majeure d'être basé sur une algèbre de blocs-diagrammes. Elle est basée sur un jeu d'opérateurs de construction suffisamment générique pour permettre la construction de tout type de bloc-diagramme. Par exemple, l'opérateur de construction séquentiel « : » permet de connecter la sortie d'un bloc à l'entrée d'un second bloc. D'autres opérateurs sont disponibles : la récursion, la construction parallèle, le dédoublement, la fusion. Par exemple, l'expression FAUST suivante :

A : (B,C :D) :E

exprime le bloc diagramme ci-dessous :

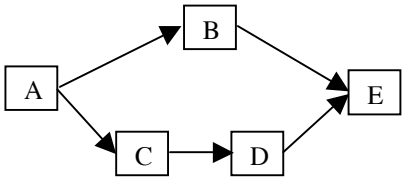


Figure 14. *Le bloc-diagramme correspondant à A : (B,C :D) :E.*

Ce langage possède les caractéristiques d'expressivité, de concision et de cohérence jugées nécessaires au but poursuivi. : exprimer de manière pérenne les processus temps réel utilisés dans nos réalisations musicales.

Expressivité : une telle algèbre de blocs-diagrammes possède la même expressivité que les systèmes graphiques de type Max/MSP ou PureData couramment utilisés par les réalisateurs en informatique musicale.

Concision : la forme algébrique retenue pour l'expression des blocs diagrammes est une garantie de concision.

Cohérence : il importe qu'une telle représentation puisse permettre la regénération du processus originel. Le langage FAUST offre cette possibilité, dans la mesure où un compilateur est actuellement disponible, qui permet de générer du code C++, ou bien une implémentation PureData.

Les travaux envisagés sont de deux types, comprenant tout d'abord le développement d'un système complet de transcription vers le langage FAUST, et ensuite l'application de méthodes d'analyse.

Tout d'abord, le développement d'un système complet permettant de transcrire les processus existants en langage FAUST permettra de disposer d'un corpus représentatif de données pour permettre la constitution d'une base de connaissances. Ce système s'appuiera bien entendu sur les outils décrits ci-dessus qui permettent l'analyse de l'existant, et seront complétés par un outil permettant leur encodage en langage FAUST.

A partir de ce corpus de données, on envisagera les développements suivants :

– En appliquant à ces représentations des techniques de *data mining,* rechercher les invariants ;

– Appliquer sur ces représentations des systèmes collaboratifs d'*annotation*, afin de construire la sémantique des traitements créatifs du signal (notamment en matière d'*effets musicaux, audio et éventuellement vidéo*, de dispositifs de *suivi temps réel) ;*
– Utiliser ces notations sémantiques et la notation FAUST afin de construire une *documentation* pertinente de ce patrimoine numérique.

Nous nous appuierons enfin sur cette documentation pour tenter de construire la classification des traitements numériques dont nous avons signalé en introduction la nécessité.

Ces diverses réalisations nous permettront de réaliser au moins partiellement l'environnement d'exécution LDV dont nous avons esquissé ci-dessus les principales lignes.

Remerciements

Ce travail a été partiellement soutenu par la Communauté Européenne, via le programme « IST » (Information Society Technology) du 6ème programme cadre pour le développement de recherche et de la technologie – projet CASPAR, contrat IST-033572.
Les auteurs sont seuls responsables du contenu de cet article, qui ne représente pas l'opinion de la Communauté Européenne, et la Communauté Européenne n'est pas responsable des utilisations qui pourraient être faites de ces contenus.

7. Bibliographie

Bernardini N., Vidolin A., « Sustainable Live Electro-acoustic Music », *Actes de la conférence internationale Sound and Music Computing Conference 2005*, Salerno, Italie, 2005.

Battier M., Faia C. H., Pasquet O., *Cahier d'exploitation de l'œuvre* A Pierre. Dell'Azzurro silenzio, inquietum *de Luigi Nono*, documentation IRCAM, 28 pages, 2000.

Bonardi, A. Barthélemy, J., Le patch comme document numérique : support de création et de constitution de connaissances pour les arts de la performance, *Actes du 10ème Colloque International sur le Document Electronique (CIDE 10)*, Nancy, 2007.

Bovet J., *ANTLR Description Language*, 2003.

Bullock J., Coccioli L., Modernising Live Electronics Technology in the Works of Jonathan Harvey, *Actes de la conférence internationale International Computer Music Conference*, Barcelona, Spain, 2005.

De Sousa Dias A., « Transcription de fichiers MusicV vers Csound au travers de Open Music », *Actes des dixièmes journées d'informatique musicale* (JIM 2003), Montbéliard, 2003.

Doerr M., LeBoeuf P., « Modelling Intellectual Processes: The FRBR - CRM Harmonization », *Actes de la conférence DELOS 2007*, 2007.

Guercio, M., Barthélemy, J., Bonardi, A., « Authenticity Issue in Performing Arts using Live Electronics », *Actes de la quatrième conférence internationale Sound and Music Computing Conference (SMC 07)*, Lekfada, Grèce, 2007.

Karayanakis, N. M., *Advanced System Modeling and Simulation with Block Diagram Languages*, CRC Press, 1995.

Knuth D. E., *Fundamental algorithms*, Vol. 4, Addison Wesley, 1973.

Knuth D. E., *Fundamental algorithms*, Vol. 5, Addison Wesley, 1973.

Lorie R., « Preserving Digital Information, an Alternative to Full Emulation », *Zeitschrift für Bibiothekswesen und Bibliographie*, Frankfurt, Germany, 2001.

Mason S. J., Feedback theory - some properties of signal flow graphs, *Proceedings IRE*, 41(9) p. 1144-1156, 1953.

Mason S. J., Feedback theory - further properties of signal flow graphs, *Proceedings IRE*, 44(7), p. 920-926, 1956.

Orlarey Y., Fober D., Letz S., « An algebra for block diagram languages », *Actes de la conférence internationale International Computer Music Conference 2002* (ICMA, éditeur), Gothenburg, Suède, 2002.

Puckette M., « New Public-Domain Realizations of Standard Pieces for Instruments and Live Electronics », Actes de la conference internationale International Computer Music Conference, Miami, 2004.

Risset J.-C., Arfib D., De Sousa Dias A., Lorrain D., Pottier L., « De Inharmonique à Resonant Sound Spaces », Actes des neuvièmes journées d'informatique musicale, Marseille, Gmem, 2002.

Documentation en ligne

Modèle FRBRoo, http://cidoc.ics.forth.gr/docs/frbr_oo/frbr_docs/FRBR_oo_V0.8.3.pdf

Modèle OAIS, http://public.ccsds.org/publications/archive/650x0b1.pdf

Modèle CIDOC CRM, http://cidoc.ics.forth.gr/docs/cidoc_crm_version_4.2.2.pdf

Définition du modèle conceptuel de référence CIDOC, http://cidoc.ics.forth.gr/docs/cidoc_crm_version_4.2.2.pdf