Cristallisation d'applications musicales par collaboration

Dominique Fober, Stéphane Letz, Yann Orlarey, Thierry Carron

Cristallisation d'application musicales par collaboration

Dominique Fober - Stéphane Letz - Yann Orlarey - Thierry Carron

GRAME
Laboratoire de recherche
9, rue du Garet BP 1185
69202 LYON CEDEX 01
Tél (33) 04 720 737 00 Fax (33) 04 720 737 01
Email : [fober, letz, orlarey]@rd.grame.fr

MIL PRODUCTIONS
BP 256
69659 Villefranche Cedex
tél/fax : (33) 04 740 251 95
Email : mprod@hol.fr

RÉSUMÉ. Le développement de la communication et de la collaboration inter-applications conduisent à l'élaboration de systèmes de plus en plus modulaires, basés sur des composants élémentaires et spécialisés qui sont amenés à collaborer pour produire des comportements émergents complexes. Dans le domaine musical, MidiShare fournit les fondements à de tels systèmes en prenant en compte les besoins spécifiques des applications musicales. L'étape ultérieure de la collaboration entre applications consiste naturellement à fournir des systèmes permettant de la simplifier et de l'automatiser. C'est en ce sens que nous présentons un nouveau concept : la *"cristallisation de programme"* qui consiste à assembler dynamiquement des applications séparées.

MOTS-CLÉS. architecture logicielle, communication, collaboration, temps-réel.

1 Introduction

En informatique, la notion de communication prend une importance croissante. Elle s'est notamment développée avec l'apparition des architectures parallèles et distribuées pour en permettre une meilleure exploitation. Mais les concepts de communication, présents dans d'autres domaines scientifiques, fournissent également des outils puissants pour la modélisation de phénomènes naturels ou le développement d'applications complexes. Aujourd'hui, la communication s'impose comme une nécessité pour l'ensemble des applications qui vont vers une spécialisation toujours croissante et sont de ce fait, contraintes à la collaboration pour toutes les tâches qui échappent à leurs compétences.

Dans le cadre de nos travaux sur les architectures logicielles pour la musique [1] , nous avons développé un système d'exploitation musical, *MIDIShare* [2], dont une des caractéristiques principales est de favoriser la communication inter-applications [3] et par là leur collaboration. Le modèle de communication de MIDIShare est connexionniste. Il peut être dynamiquement configuré par les applications ou l'utilisateur. Il est le fondement de la richesse du système et de son extensibilité : d'emblée, toute nouvelle application se place dans un cadre collaboratif et vient enrichir les possibilités offertes par celles déjà existantes. Les applications peuvent alors être vues comme des agents communicants, interagissants et capables de produire un comportement émergeant. La possibilité pour l'utilisateur de se constituer ainsi des méta-applications, lui restitue une dimension de programmabilité de la machine.

Le problème qui se pose dès lors, est celui des outils permettant de faciliter la mise en oeuvre de cette collaboration, voire de la figer pour des configurations se révélant intéressantes. Il peut en effet être rapidement fastidieux d'activer et de régler toutes les applications engagées dans une même collaboration, cela peut être rendu impossible par leur localisation sur des machines différentes. L'étape suivante dans le développement de la collaboration passe donc par un concept nouveau : la cristallisation de programme.

2 Le contexte

Les principes de la collaboration inter-applications ont été l'objet d'un certain nombre de travaux. Ces travaux concernent deux directions différentes. La première, applicable au niveau utilisateur, repose sur les langages de script, qui permettent d'activer des applications et de les contrôler de manière externe. La seconde repose sur la notion de document composite, associant les données et les logiciels permettant de les éditer.

2.1 Les architectures orientées document.

Considérant qu'il est plus naturel de centrer le travail de l'utilisateur sur le document plutôt que sur les outils qui lui permettent de le manipuler, des architectures orientées document telles que OLE (Object Linking and Embedding) développée par Microsoft ou encore plus récemment OpenDoc [Piersol, 1994] développée par Apple, ont donc vu le jour. Le propos de ces architectures est de rendre transparente la collaboration des applications autour d'une tâche précise, centrée sur le concept de document. Cette collaboration se traduit dans les faits par la contribution de chaque application dans son domaine (textuel, graphique...) à l'élaboration du document. Il s'agit donc d'un mode de collaboration figé autour d'un concept unique, et dont les principes de mise en oeuvre ne sont pas accessibles à l'utilisateur.

2.2 Les langages de scripts

Le shell Unix [5] est basé sur deux concepts simples et puissants : les fichiers et les flots de données. Les programmes fonctionnent comme des filtres : ils prennent un flot de données en entrée et produisent un flot en sortie. Le système de communication entre ces programmes est constitué de 'pipes', qui permettent de rediriger le flot de données sortant d'une application vers l'entrée d'une autre. Un langage de script, le shell, permet sous une forme textuelle, d'activer les applications et de décrire le cheminement du flot de données.

Plus récemment, d'autres travaux ont été mené dans le cadre d'OSA (Open Scripting Architecture) [6] avec pour but de prendre en charge l'automatisation, l'intégration et la personnalisation des applications et des services systèmes. OSA repose notamment sur un système de communication à messages distribués : les Apple Events [7], et sur un langage de script : AppleScript [8]. Les messages agissent au niveau utilisateur sur les objets de l'application tels que les fenêtres, ou encore tout autre objet ou service que l'application voudra rendre accessible à ces messages. Le langage de script permet d'activer les applications et de restituer leur comportement par le biais de ces messages.

Dans le domaine musical, le principal problème soulevé par ces systèmes est qu'ils ne prennent pas en compte la dimension du temps. Le système Unix est totalement inadapté au temps réel. Sur Macintosh, le temps de transmission d'un 'Apple Event' est élevé et n'est pas déterministe : il augmente avec l'occupation du temps CPU de la machine.

3 La cristallisation de programme

Nous présenterons plusieurs solutions permettant de faciliter la collaboration des applications musicales. Parce qu'elle bénéficie d'une mise en oeuvre pour un ensemble d'applications, la cristallisation par collaboration sera ensuite plus détaillée.

3.1 Concepts

La cristallisation de programme consiste, dans un environnement collaboratif ouvert, à construire une application équivalente à l'ensemble des applications en cours d'utilisation. L'idée recouvre 3 abords différents, qui peuvent se révéler complémentaires : il s'agit de restitution, d'agrégation ou de synthèse du comportement collaboratif des applications. Les modes de cristallisation correspondants seront qualifiés de faible à forte.

3.2 Cristallisation collaborative

Elle représente la possibilité d'enregistrer l'état et le comportement d'un ensemble d'applications, pour pouvoir ensuite restituer cet état et piloter les applications correspondantes de manière à reproduire le

même comportement. Ce mode de cristallisation suppose une collaboration étendue de la part des applications, qui doivent permettre leur contrôle de manière externe. Il est qualifié de faible parce que le résultat de la cristallisation est à lui seul incomplet pour restituer le comportement des applications, la présence et la collaboration de celles-ci sont également nécessaires. Il peut être cependant plus utile que des modes de cristallisation plus forts, dans le cas de collaboration pour des systèmes répartis sur un réseau par exemple, puisqu'il permet de conserver également la topologie du système.

3.3 Cristallisation par agrégat

Elle représente pour une application, la possibilité d'accéder aux ressources code qui produisent le comportement de chacune des applications de l'environnement, de s'approprier ce code par recopie, et de restituer le comportement global de l'environnement en l'exécutant. La cristallisation résulte ici dans l'agrégation en une application, des ressources de l'ensemble des applications de l'environnement.

3.4 Cristallisation par synthèse

Elle représente pour une application, la possibilité d'accéder à la description fonctionnelle du comportement de chacune des applications de l'environnement, de combiner ces descriptions et de restituer le comportement équivalent en évaluant le résultat de cette combinaison. La cristallisation résulte ici dans la synthèse du comportement des applications. Elle repose sur un formalisme commun à l'ensemble des applications et présente l'avantage d'être plus compacte que la cristallisation par agrégat.

4 La cristallisation collaborative

Elle repose sur le concept de méta-application, vue comme une application virtuelle dont on considère que les composants sont les applications réelles dont on veut cristalliser le comportement. Cette application virtuelle sera manipulée comme un unique objet logiciel. Elle diffère cependant d'une application réelle par le fait essentiel qu'elle est déclinable à l'infini, en fonction des composants actifs lors de la cristallisation.

4.1 Services de la méta-application

Le fonctionnement de la méta-application repose sur la collaboration de ses différents composants, rendue possible par le biais d'un protocole de communication commun. Considérons l'ensemble des services fournis par cette application virtuelle comme un ensemble de commandes auxquelles elle répond, il suffit alors d'identifier ces commandes dans un protocole de communication commun et de les distribuer à chacun des composants pour que les services correspondant existent. Leur implémentation réelle dépend alors de chaque composant et de sa réponse à chaque message. L'activation des services pourra se faire grâce à un ou plusieurs composants spécialisés, communs à l'ensemble des applications virtuelles, qui auront pour responsabilité de distribuer les messages correspondants (figure 1).

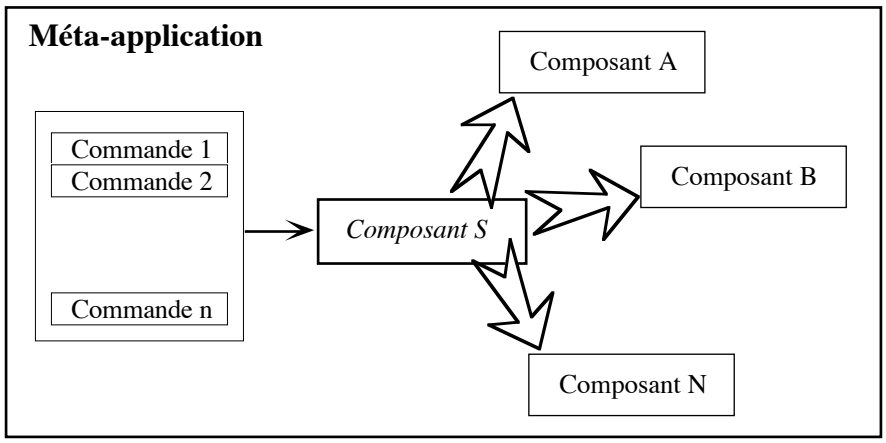

figure 1

La cristallisation elle-même, ie l'identification des différents composants de la méta-application en vue de leur manipulation future, sera réalisée par un composant spécialisé particulier, capable de collecter l'état courant du système, pour le stocker dans un document qui permettra ensuite de le reconstituer.

4.2 Exemple de protocole

Nous allons examiner un exemple de définition de protocole permettant d'implémenter les services élémentaires de toute application (tableau 1).

Commandes	*Services*
"Launch"	ouverture de l'application
"Quit"	fermeture de l'application
"Load"	restauration d'un état particulier
"Save"	sauvegarde d'un état particulier

tableau 1

Ces services vont pourvoir être implémentés grâce aux 4 messages suivants :

Messages	*Descriptions*	*Réponses*
"GetIdMsg"	requête d'identification	identification du composant
"QuitMsg"	requête de fermeture	-
"GetStateMsg"	demande d'état	état du composant
"StateMsg"	état d'un composant	-

tableau 2

Nous allons supposer que tous les services précédemment définis sont pris en charge par un composant unique qui assure également l'étape de cristallisation.

4.2.1 Cristallisation des composants

L'étape de cristallisation consiste à identifier tous les composants actifs à un moment donné et à stocker dans un document les informations nécessaires à la restitution de ces composants. Ce document constitue la seule occurrence réelle de la méta-application. Il sera constitué de la manière suivante :

- émission du message "GetIdMsg" qui demande à chaque composant de renvoyer son identité. Ce message est distribué à toutes les applications, y compris celles qui n'implémentent pas le protocole.
- initialisation d'un délai d'attente des réponses.
- traitement des réponses et sauvegarde dans un document. L'absence de réponse est interprétée comme l'absence de composant actif.

4.2.2 Ouverture de la méta-application

Consiste à ouvrir le document correspondant, à identifier ses composants et à les ouvrir individuellement.

4.2.3 Fermeture de la méta-application

Consiste simplement à émettre le message "QuitMsg". Les composants recevant ce message sont censés quitter le système à réception du message.

4.2.5 Sauvegarde d'un état

L'étape de sauvegarde consiste à demander à tous les composants actifs de renvoyer leur état :

- émission du message “GetStateMsg”. A réception de ce message, un composant est censé répondre par le message “StateMsg”, contenant en en-tête l’identification de l’application suivi de la description de son état.
- initialisation d’un délai d’attente des réponses.
- sauvegarde des réponses dans un document.

4.2.4 Restauration d’un état

Consiste à ouvrir le document correspondant, à identifier les composants et à leur renvoyer leur état sous la forme du message “StateMsg” précédent. A réception de ce message, un composant est censé prendre l’état correspondant.

5 Un exemple d’implémentation sur l’architecture de MidiShare

Un exemple d’implémentation d’un protocole permettant la cristallisation de composants logiciels a été réalisé par la Sté Mil Productions à travers un ensemble d’applications (figure 1) basées sur l’architecture de MidiShare. Ces applications ainsi que le protocole associé reposent sur la métaphore du studio d’informatique musicale : il est considéré que le résultat de la cristallisation représente l’équivalent d’un ensemble d’appareils installés dans un studio, ainsi que des connexions qui peuvent exister entre ces appareils. Sur la base de cette métaphore, le protocole permet des manipulations courantes du studio telles que mise en service, extinction du studio, sauvegarde et restauration des connexions, sauvegarde et restauration de l’état des composants.

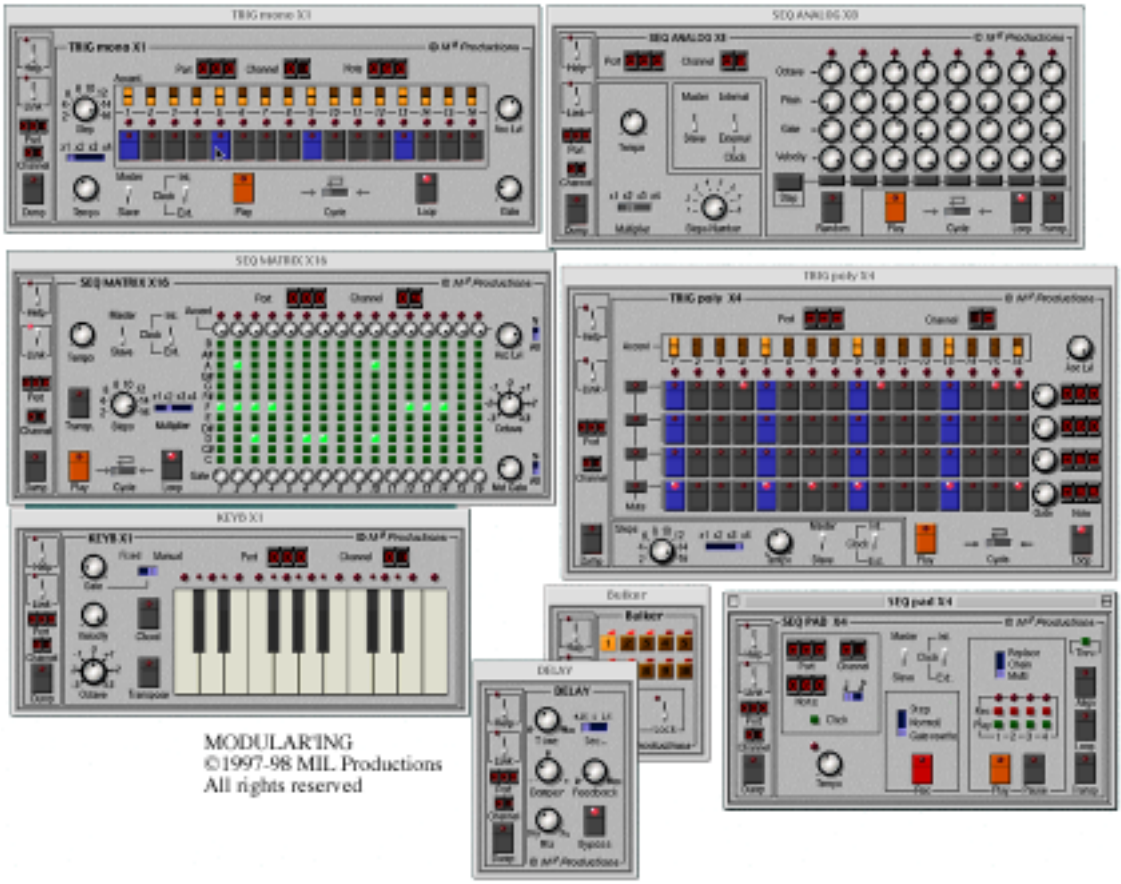

figure 2

L'implémentation bénéficie de l'architecture de MidiShare pour ce qui concerne la communication et la gestion des connexions entre applications. Toute la partie communication est une surcouche de la norme MIDI, de telle sorte que l'échange des messages correspondant au protocole n'est pas localisé à une machine. Ce protocole est appelé MAG (pour MIDI Application Glue).

5.1 Le protocole MAG

Il permet d'implémenter les services élémentaires définis ci-dessus (4.2 Exemple de protocole) grâce aux 3 messages suivants :

Messages	*Descriptions*	*Réponses*
DumpRequest	demande d'état	message Dump
Dump	état	-
QuitRequest	demande de fermeture	-

tableau 3

Ces messages sont transmis sous la forme de System Exclusive MIDI et ont le format suivant :

SysEx	F0
Identificateur constructeur	<1 octet>
Identificateur protocole	<n octets>
Identificateur message	<n octets>
Données spécifiques au message	<0 - n octets>
Fin de SysEx	F7

Seul le message d'état "Dump" contient des données spécifiques qui sont les suivantes :

Identificateur de l'application	<8 octets>
Signature de l'application	<4 octets>
Données spécifiques à l'application	<0 - n octets>

- L'identificateur est un champ qui dépend de la machine hôte. Il doit contenir les informations nécessaires pour identifier le processus correspondant à l'application (au sens du système d'exploitation de la machine), et pour la localiser en tant que fichier.
- La signature est un champ indépendant machine, qui permet d'identifier toutes les applications identiques dans le sens où elles peuvent être des implémentations, des copies ou des instances différentes de la même application. Une application n'est censée accepter que les messages qui portent sa signature.
- Les données spécifiques contiennent l'état interne de l'application. Elles peuvent être interprétées par toutes les applications qui portent la même signature.

5.2 Le document "méta-application"

Le concept de méta-application repose sur un unique document qui contient l'état total du système au moment de la cristallisation soit :

- l'identification des composants permettant de les réactiver.
- l'état des connexions du système, incluant tant les différents composants que toutes les applications compatibles MidiShare.
- l'état des différents composants.

Dans la métaphore du studio, le même document peut donc servir à :

- allumer le studio en une seule opération : en ouvrant tous les composants contenus dans le document.
- restaurer un câblage particulier du studio : en restaurant l'état des connexions entre les composants.
- restaurer un état particulier du studio : en restaurant l'état des différents composants.

Du point de vue de l'application virtuelle, nous avons réuni toutes les fonctionnalités d'une application réelle : plusieurs documents correspondant à une même configuration de studio représentent le concept usuel de document pour une même méta-application.

5.3 Les composants spécialisés

L'activation des services d'une méta-application se fait grâce à trois composants spécialisés :

- **Studio Maker** : prend en charge les services d'activation ("Launch") et de sauvegarde ("Save") d'une méta-application. Il génère le document correspondant à un état du système. Il restaure les composants du système et leurs connexions à partir d'un document mais ne modifie pas l'état interne des composants.
- **Studio Loader** : fournit le service de restauration d'un état ("Load") des composants à partir d'un document. Il ne restaure pas les composants qui ne seraient pas présents au moment de l'appel.
- **Studio Closer** : permet de quitter la méta-application ("Quit") en fermant tous les composants actifs au moment de l'appel.

6 Conclusion

Dans le domaine de l'informatique musicale, ces dernières années ont vu s'imposer des logiciels dont l'ambition croissante a eu pour résultat de complexifier singulièrement leur utilisation. L'accumulation et la centralisation de fonctionnalités dans un même logiciel ont généralement eu pour effet leur sous-exploitation en proportion. Tirant leçon de ce constat, les applications de l'informatique musicale s'orientent aujourd'hui vers des systèmes plus modulaires par le biais de "plug-ins". L'architecture sous-jacente est un modèle centralisé de gestion des composants où la dimension de programmabilité est difficilement accessible à l'utilisateur.
L'approche que nous avons présenté est originale dans le sens où elle est totalement répartie et peut se prêter aisément à des prolongements en réseau. A partir de concepts usuels, elle fournit à l'utilisateur une manière simple de construire de nouvelles applications à partir de celles existantes. Elle nous semble intéressante tant pour sa souplesse que pour son accessibilité. Elle peut se révéler riche de prolongements pour le concept même d'application et d'environnement logiciel.

Références

[1] Fober D., Letz S., Orlarey Y., *MidiShare, un système d'exploitation musical pour la communication et la collaboration.* Actes des Journées d'Informatique Musicale, JIM'95, Paris, 1995, pp.91-100

[2] Orlarey Y., Lequay H. *MidiShare : a Real Time multi-tasks software module for Midi applications.* Proceedings of the ICMC, ICMA, San Francisco, 1989, pp.234-237

[3] Orlarey Y. *Hierarchical Real Time Inter application Communications* Proceedings of the ICMC, ICMA, San Francisco, 1991, pp.408-415

[4] Piersol K., *A Close-Up of OpenDoc*. Byte, March 1994, pp 183-188

[5] Unix User's Manual. Berkeley, CA 1984

[6] Cook William R., Harris Warren H., *The Open Scripting Architecture: Automating, Integrating, and Customizing Applications*, 1994, White paper

[7] *Apple Event Registry: Standard Suites*, Developer Technical Publications, Apple Computer, Inc. 1992

[8] *AppleScript Language Guide*, Developer Technical Publications, Apple Computer, Inc. 1993