Migration d'INScore pour plateformes mobiles et pour le web

Guillaume Gouilloux

▶ **To cite this version:**

Guillaume Gouilloux. Migration d'INScore pour plateformes mobiles et pour le web. Environnements Informatiques pour l'Apprentissage Humain. 2015. dumas-01557054

HAL Id: dumas-01557054

https://dumas.ccsd.cnrs.fr/dumas-01557054

Submitted on 5 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL ASSOCIE DE LYON

MEMOIRE

présenté en vue d'obtenir

le DIPLOME D'INGENIEUR CNAM

SPECIALITE : INFORMATIQUE

OPTION : SYSTEMES D'INFORMATION

par

Guillaume GOUILLOUX

Migration d'INScore pour plateformes mobiles et pour le web

Soutenu le 11 décembre 2015

JURY

PRESIDENT :

M. Christophe PICOULEAU	**Professeur des Universités/Cnam Paris**

MEMBRES :

M. Bertrand DAVID	**Professeur des Universités E.R./EC Lyon** **Enseignant responsable/Cnam Lyon**
M. Claude GENIER	**Administrateur du Cern E.R.** **Enseignant Cnam Lyon**
Yann ORLAREY	**Directeur de la recherche GRAME**
Dominique FOBER	**Chercheur GRAME**

Remerciements

Je remercie le GRAME et l'ensemble de son personnel pour m'avoir accueilli au sein de l'équipe de recherche. Je tiens à remercier plus particulièrement Dominique Fober, chercheur et tuteur durant le stage, Yann Orlarey, directeur du département recherche, et Stéphane Letz, chercheur, pour leur disponibilité et leur gentillesse.

Je remercie Bertrand David pour ses enseignements et pour ses conseils durant la rédaction du mémoire.

Je remercie également le CNAM de Lyon et plus particulièrement Eléonore Gondeau pour son suivi et son implication durant ces années d'étude.

Enfin, je remercie ma famille et mes proches pour leur soutien et leurs encouragements durant ce cursus.

Liste des abréviations

API : Application Programming Interface ou interface de programmation.

IDE : Integrated Development Environment ou Environnement de développement intégré. Logiciel permettant de faciliter le développement des applications.

OS : Operating System ou système d'exploitation.

SDK : Software Development Kit ou kit de développement logiciel. Ensemble d'outils et de librairies permettant de développer avec un langage particulier ou pour une plateforme particulière.

SVG : Scalable Vector Graphics ou graphique vectoriel adaptable. Les images SVG sont un ensemble de graphiques vectoriels décrit en langage XML.

Glossaire

AR ou Abstract Representation : Terme utilisé dans le moteur Guido. La représentation abstraite d'une partition est obtenue par transformation du code texte guido. Elle ne comporte pas d'élément de mise en forme graphique.

Bytecode : Représentation intermédiaire d'un programme entre le langage de programmation et le langage machine. Cette représentation est utilisé pour réduire la dépendance d'un programme vis-à-vis du matériel et de permettre son exécution sur différentes architectures.

Device Graphique : Terme utilisé dans le moteur Guido. Interface de programmation mis à disposition par le moteur guido permettant de dessiner les partitions. Les multiples implémentations permettent de dessiner les partitions avec différentes technologies et sur différentes plateformes.

Framework : Ensemble de composants logiciels réutilisables.

GR ou Graphical Representation : Terme utilisé dans le moteur Guido. La représentation graphique d'une partition ajoute à la représentation abstraite un ensemble d'objets permettant de faire la mise en page de la partition.

Guido : Le moteur Guido permet de faire le rendu graphique d'une partition écrite au format guido. Le format Guido est un format texte d'écriture de la musique propre à ce moteur.

Scène : Terme utilisé dans l'application INScore. Une scène est un conteneur d'objets constituant la partition. Une scène est représentée par un espace graphique (une fenêtre sur ordinateur ou un onglet sur mobile). Une partition peut être composée de plusieurs scènes indépendantes ou synchronisées les unes par rapport aux autres.

Thread : Fil d'exécution d'un programme qui semble se dérouler en parallèle des autres fils d'exécution. Un programme peut utiliser plusieurs threads pour exécuter plusieurs tâches en même temps ou pour garder une interactivité avec les utilisateurs pendant un traitement.

Table des matières

Introduction

A l'heure où l'informatique s'immisce dans l'ensemble des actions de notre vie quotidienne par l'intermédiaire des objets connectés, qu'une grande majorité de personnes utilisent des smartphones, les applications pour ordinateurs doivent savoir s'adapter pour survivre. De nombreux services et applications sont également dématérialisés en étant disponibles sur internet sans aucune installation particulière de l'utilisateur, si ce n'est un navigateur.

Cette révolution n'est pas seulement une contrainte, mais est également un moyen d'étendre les fonctionnalités et de créer de nouvelles utilisations. Le domaine de la création musicale, toujours en quête d'innovation et de nouvelles formes d'expression, doit également s'adapter et intégrer ces changements. C'est au sein de l'équipe de recherche du Grame, Centre National de Création Musicale, que cette problématique est traitée.

Le portage d'une application existante vers les mobiles et le web n'est pas toujours une évidence. Les contraintes de ces plateformes n'en font pas forcément les hôtes idéals pour des applications interactives aux graphismes élaborés. C'est pourtant une application de création de partitions musicales interactives augmentées qui est portée sur le web et sur mobile. Ces portages font accéder les utilisateurs à un ensemble de possibilités tout à fait nouvelles et lèvent des verrous techniques tout en faisant naitre de nouvelles idées de créations. Dépendante d'un moteur de rendu de partition musicale, un travail préalable sur ce moteur sera donc nécessaire avant le portage de l'application cible.

Durant le premier chapitre, une présentation de Grame et des logiciels concernés par la migration est faite. Au chapitre 2, nous présentons les enjeux de la migration et les apports pour les applications tant sur le plan des fonctionnalités que de la disponibilité. Il est également question de la spécificité des plateformes visées avec une étude technique de chacune d'entre elles. Enfin, les chapitres 3 et 4 sont respectivement consacrés au portage du moteur de rendu de partitions musicales Guido et de l'environnement de création de partitions INScore.

Chapitre 1

Contexte de travail

Grame, Centre National de Création Musicale, m'a accueilli en qualité de stagiaire pendant neuf mois. Grame est une association et l'un des six centres nationaux de création musicale. Sa mission principale est de permettre la conception et la réalisation de nouvelles œuvres musicales. Fortement engagée dans un processus d'innovation, l'association comporte une équipe de recherche dédiée à l'informatique musicale dans laquelle ce travail a été réalisé. Cette équipe participe à des projets nationaux et internationaux de recherche en informatique musicale. Elle collabore avec les artistes et leur propose des logiciels et librairies qu'ils peuvent télécharger et utiliser librement. Parmi eux, la librairie Guido et le logiciel INScore, objets centraux du travail réalisé, permettent respectivement la représentation musicale traditionnelle et la création de partitions musicales augmentées.

1.1 – Présentation de Grame

1.1.1 – Historique et missions

Grame a été créé en 1982 par Pierre Alain Jaffrennou et James Giroudon, l'actuel directeur. Grame propose un ensemble de développements technologiques ainsi qu'une assistance technique et artistique aux personnes accueillies en résidence afin de leur fournir un environnement favorable à la création. Le développement de ses activités à l'international permet à Grame d'être présent dans de nombreux pays avec notamment une présence en Asie et en Amérique du Nord. Cette présence passe par différents types d'interventions, comme des concerts, mais aussi des masters-classes, conférences et expositions.

Les œuvres musicales contemporaines vont au-delà du simple concert. Elles utilisent souvent les arts numériques en ajoutant une dimension visuelle et des dispositifs de spatialisation ou d'interactions. Le département recherche développe et propose de nouvelles technologies permettant d'établir la relation entre ces différentes composantes de la création.

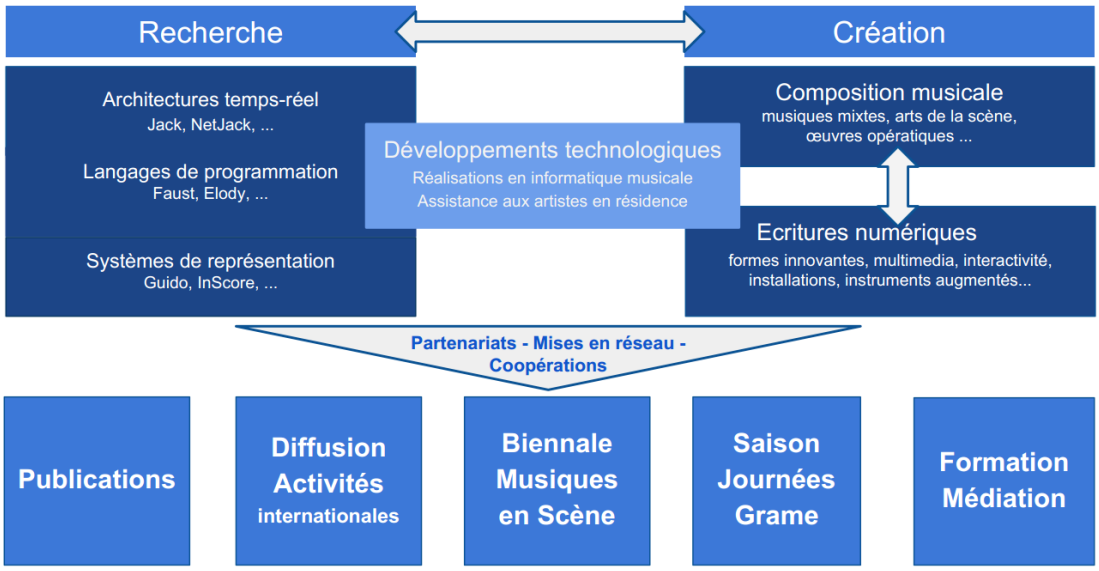


Figure 1 : Les activités de Grame

La diffusion et la production des activités de Grame passent bien entendu par des concerts, mais également par des publications scientifiques. Créée en 1992, la biennale « Musiques en Scène » est devenue une des manifestations les plus importantes en matière de création musicale. Enfin, Grame a une action pédagogique à plusieurs niveaux. Ces actions pédagogiques passent par des interventions en milieu scolaire pour sensibiliser le jeune public, mais aussi à une coopération pédagogique avec le CNSMDL (Conservatoire National Supérieur de Musique et de Danse de Lyon), l'université Lyon2 et son département de musicologie et l'université Jean Monnet de Saint Étienne.

1.1.2 – Le département recherche

La recherche en informatique musicale et l'avancée sur les technologies numériques sont indispensables à la création des œuvres. L'équipe de recherche assure la création, le développement et la maintenance d'un ensemble d'outils mis à disposition des artistes. Elle est constituée de 3 personnes permanentes. Elle accueille également de nombreux stagiaires et collabore régulièrement avec des personnes extérieures. L'équipe de recherche est membre de pôles de compétitivité comme Imaginove, permettant de rencontrer les acteurs

régionaux de l'innovation et de mettre en place des collaborations. Elle est également régulièrement en contact avec l'IRCAM (Institut de Recherche et Coordination Acoustique/Musique) et l'INRIA (Institut National de recherche en Informatique et Automatique).

La recherche est orientée autour de trois axes :

- Les systèmes de représentation de la musique et de la performance,
- Les systèmes communicants temps réels qui permettent la collaboration de systèmes audio ou midi,
- Les langages de programmation, orientés composition ou encore traitement du signal.

1.1.3 – Projets et logiciels

Grâce à son expertise dans le domaine de l'informatique musicale, Grame participe à de nombreux projets couvrant différents aspects et utilisations comme nous pouvons le voir sur la Figure 2.

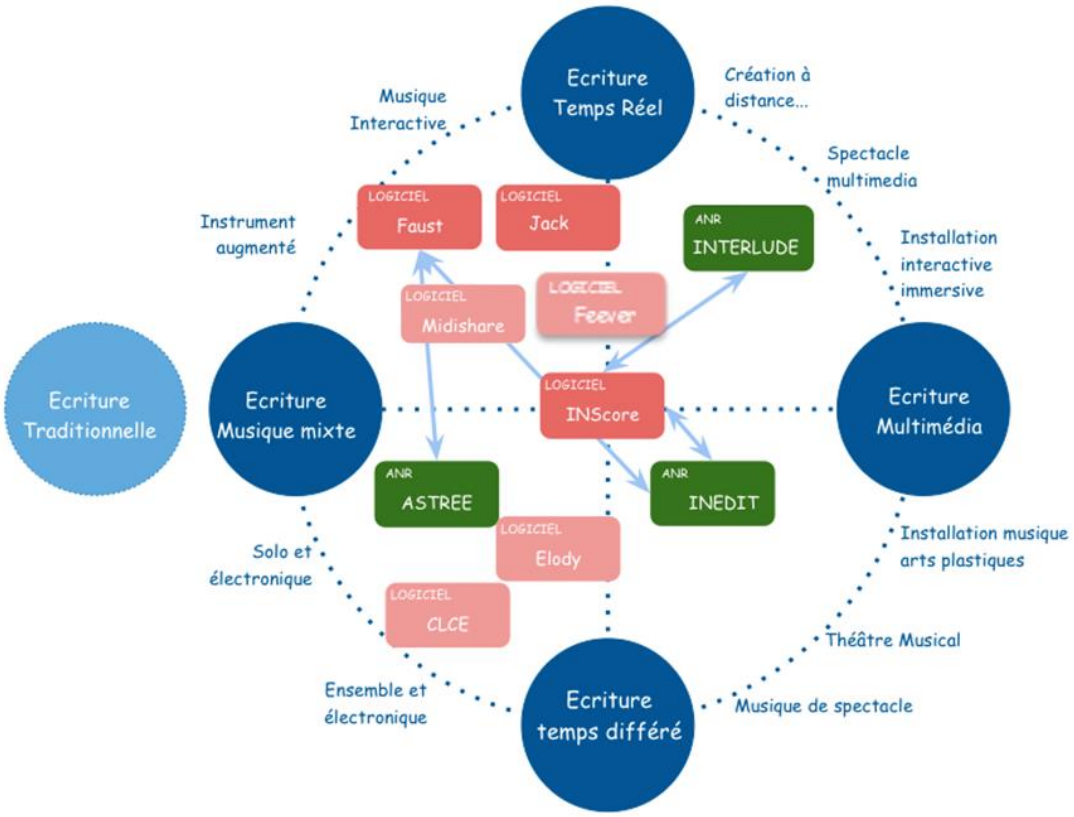


Figure 2 : Les projets, logiciels du Grame et leur orientation principale

L'équipe développe actuellement 3 logiciels principaux qui sont Faust, Jack et INScore. Ces logiciels évoluent en lien avec les projets auxquels participe Grame. Dernièrement, les quatre projets significatifs auxquels Grame a participé ont été Interlude, Astree, Inedit et Feever. Ces projets ont été financés par l'ANR et ont été réalisés notamment en collaboration avec l'IRCAM, le LABRI (Bordeaux), le CIEREC (Université Jean Monnet de Saint Etienne), ARMINES (MINES ParisTech), l'INRIA/IRISA.

Logiciels développés et proposés par Grame

Les logiciels ou librairies développés sont distribués en open source (sous licence GNU LGPL[1]). FAUST, INScore et les sous-projets qui en découlent concernent actuellement la quasi-totalité du temps consacré aux développements informatiques.

- FAUST (Functionnal AUdio STream) : ce projet regroupe un ensemble de techniques et d'outils permettant de décrire des algorithmes de synthèse et de traitement du signal dans un langage de spécification de haut niveau.
- INScore : logiciel de notation musicale permettant de créer des partitions musicales augmentées interactives et temps réel intégrant aussi bien la notation musicale traditionnelle que des objets graphiques arbitraires. Il permet de manipuler ces objets dans l'espace et dans le temps. Il s'appuie sur deux autres librairies développées et distribuées par Grame, la librairie Guido et libMusicXML.
- Jack : logiciel permettant aux différentes applications musicales de collaborer grâce à un système de câblage virtuel.

Deux autres logiciels significatifs ne sont maintenant plus maintenus :

- Le système temps réel midishare : système d'exploitation MIDI temps réel et multitâches disponible sous Windows, Linux et Mac OS.
- L'environnement de composition musicale Elody : ce logiciel permet la description et la manipulation algorithmique de structures musicales et de procédés

[1] La licence GNU LGPL permet notamment d'utiliser ce code dans un logiciel propriétaire sans être contraint de divulguer les sources. Plus d'information sur https://www.gnu.org/licenses/lgpl.html.

compositionnels, sans nécessiter pour autant de connaissances particulières en informatique.

Principaux projets

GRAME participe également à des projets ayant pour but la création musicale contemporaine. Les quatre projets présentés ici sont des projets soutenus par l'ANR (Agence Nationale de la Recherche)

- ASTREE : analyse et synthèse de traitements temps réels
- INTERLUDE : exploration et interaction gestuelle expressive avec des contenus musicaux
- INEDIT : interopérabilité des outils de création sonore et musicale couplant écriture du temps et écriture de l'interaction.
- FEEVER : solution globale et ubiquitaire pour le traitement du signal audionumérique.

A noter les deux anciens projets VENUS et IMUTUS qui ont servi de base à la création des librairies Guido et du logiciel INScore.

- VEMUS (Virtual European Music School) : Il s'agit d'un système ouvert et interactif d'apprentissage de la pratique instrumentale destiné aux niveaux débutant et intermédiaire. Il permet de créer des lieux virtuels de rencontre et de pratique musicale.
- IMUTUS (Interactive Music Tuition System) : système interactif pour l'enseignement à distance de la pratique instrumentale.

1.2 – Les logiciels concernés par la migration

L'application INScore est la principale cible. Cependant, INScore est intimement lié à la librairie Guido pour le rendu de la notation musicale classique qui constitue donc une étape préalable avant tout travail sur INScore.

1.2.1 – Librairie Guido

La librairie Guido Engine est basée sur le travail de thèse réalisé par Kai Renz en 2002 maintenant repris en maintenance évolutive par Grame. Le format de notation musicale Guido a été créé dans le cadre de la thèse. Ce format est une nouvelle approche de la notation musicale basée sur une représentation textuelle simple, utilisable et compréhensible par l'homme. Le format Guido est un langage de tag développé depuis 1996, flexible et extensible comme l'explique [Renz, 2002] en page 7. Il a également l'avantage d'être indépendant de la plateforme de l'utilisateur.

Un nouveau langage

Le langage Guido se concentre sur la représentation des concepts musicaux contrairement à d'autres représentations existantes orientées sur la représentation graphique, la performance ou encore l'analyse musicale. [Renz, 2002] compare en page 37 Guido aux différents formats musicaux existants, qui sont par ailleurs pour l'essentiel toujours utilisés aujourd'hui, bien que chacun ait évolué. L'approche utilisée par le formalisme Guido le rend utilisable dans d'autres contextes que celui d'origine, contrairement à la plupart des autres formalismes. Voici un exemple de notation Guido et de sa représentation graphique à l'aide de l'application GuidoEditor :

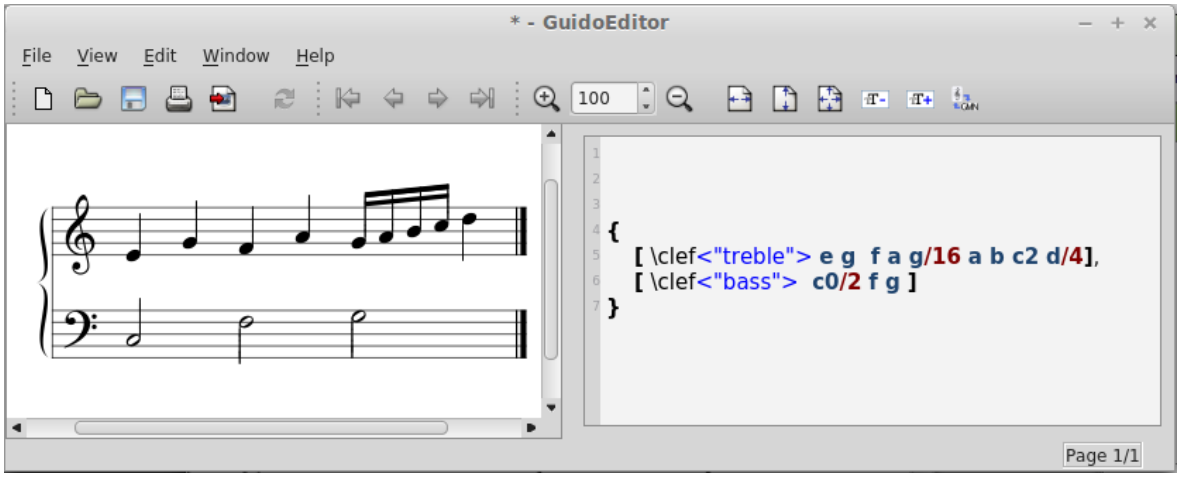


Figure 3 : Représentation graphique de la notation Guido

La simplicité de la notation permet d'avoir des performances correctes lors de la génération de la représentation graphique dans un contexte interactif. Malgré l'attachement de Guido à représenter le contenu musical, il est possible de décrire une représentation exacte du formatage de la partition avec des informations additionnelles. Ces informations destinées

uniquement à des fins de représentation graphique peuvent être retirées par des outils spécifiques.

Un langage adapté aux besoins

Dans la création musicale contemporaine, la musique est souvent générée par des outils développés par les compositeurs eux-mêmes, ou manipulée par leurs outils. Cette musique peut être créée en temps réel pendant le concert. Il fallait donc avoir un formalisme compréhensible par l'homme et dont la représentation graphique soit assez performante pour être faite en temps réel.

La musique générée peut être utilisée par d'autres logiciels. Le fait que le format Guido se concentre sur les concepts musicaux est donc également un atout. Le format plein texte permet de faciliter les échanges de données.

Il faut noter également que les logiciels commerciaux les plus répandus ne fournissent pas la description de leur langage d'encodage des partitions.

Fonctionnement du moteur

Le moteur Guido est une librairie développée en C/C++ dont l'API publique est en C. La conversion de code Guido en une représentation graphique se fait en plusieurs étapes :

- la transformation du code Guido en une représentation abstraite (Abstract Representation ou AR)
- la transformation de l'AR en représentation graphique (Graphical Representation ou GR)
- le dessin du GR sur un dispositif logiciel particulier (Graphical Device), pouvant être le dessin dans une image SVG, sur une surface en utilisant un driver natif de l'appareil, etc. Il existe plusieurs implémentations de Graphical Device pour pouvoir utiliser le moteur sur les différentes plateformes (Windows, Mac OS et Linux).

Les symboles musicaux sont représentés à l'aide d'une police de caractères spécifique appelée guido2. Les applications embarquant le moteur Guido doivent donc pouvoir utiliser d'une manière ou d'une autre cette police de caractères. Elles ont la possibilité d'embarquer la police ou de la charger depuis le système. Dans le deuxième cas, la police doit avoir été préalablement installée. Pour calculer les espacements entre les symboles, que ce soit pour

l'écriture de symboles musicaux ou pour l'écriture de texte, le moteur utilise les métriques de la police utilisée, c'est-à-dire la taille de chaque symbole. Les systèmes graphiques des systèmes d'exploitation courants (Windows, Mac OS X et Linux) fournissent un ensemble de directives permettant de calculer ces métriques facilement. La librairie Qt permet également de faire ces calculs.

Nous l'aurons compris, le moteur Guido se doit d'être performant pour le rendu graphique des partitions dans les contextes d'utilisation cités précédemment. Le rendu temps réel était déjà une des caractéristiques visées par [Renz, 2002].

1.2.2 – INScore

INScore, initié dans le cadre des projets ANR Interlude et Inedit, permet de créer des partitions interactives augmentées. Le terme de partition augmentée expliqué par [Fober, et al., 2010] est utilisé, car INScore permet non seulement de représenter des objets musicaux grâce au moteur GUIDO, mais également d'inclure des objets arbitraires comme du texte, des images, des signaux… et de les manipuler aussi bien dans le domaine graphique que temporel. Des synchronisations temporelles sur les objets du domaine graphique peuvent être créées. La synchronisation permet par exemple d'établir une correspondance entre la lecture d'une ressource audio et une partition musicale. Le temps de la ressource audio est exprimé en frame alors que le temps d'une partition est la noire.

Un tel exemple est proposé par [Fober Dominique, et al., 2012] où la représentation graphique du signal et la représentation graphique d'une partition sont synchronisées.

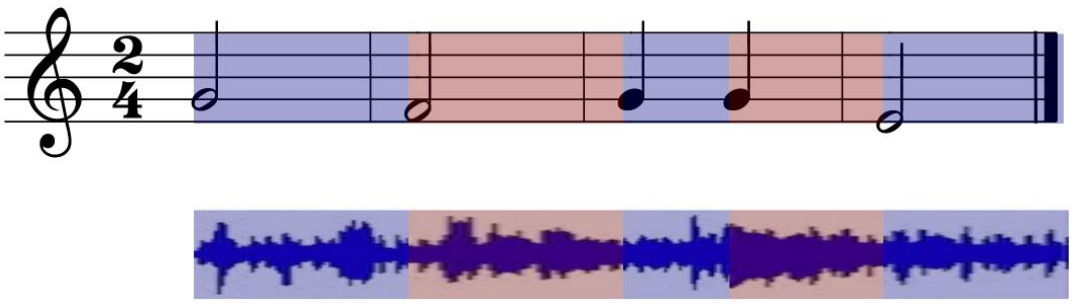


Figure 4: Synchronisation d'un signal audio et d'une partition

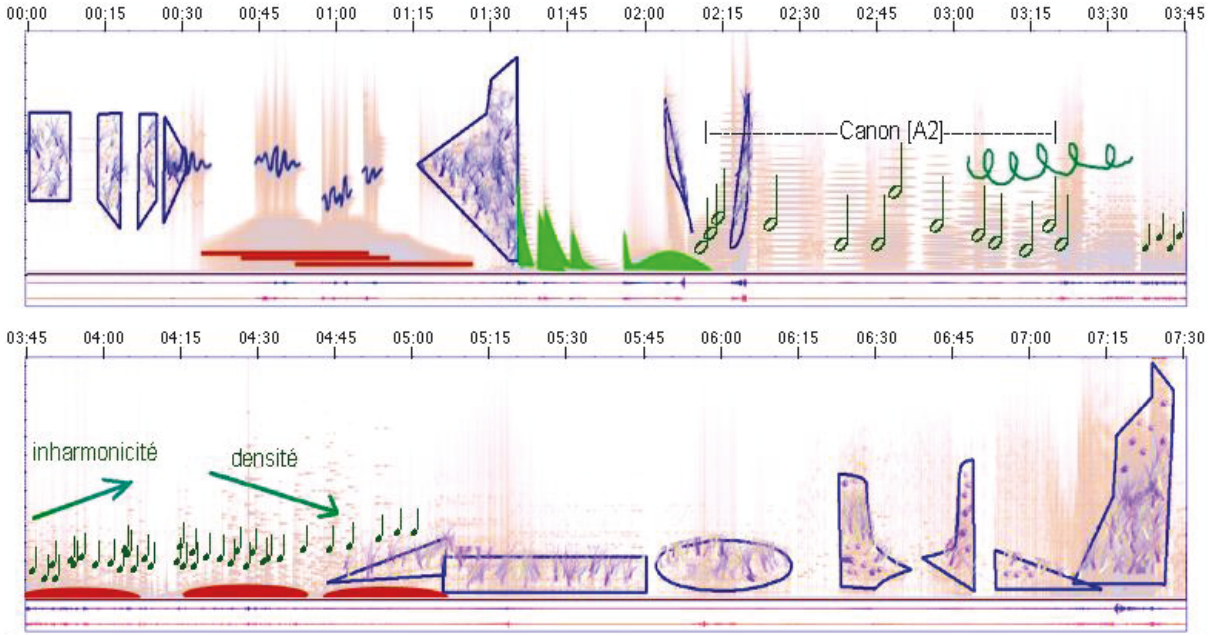
Cet exemple illustre bien l'ouverture que donne INScore sur l'utilisation combinée de musique électronique et d'instruments lors de performances. Cette facilité de mise en œuvre permet aux compositeurs de s'affranchir d'une partie des problématiques techniques et de

créer plus librement. Nous pouvons imaginer également des cas d'utilisation à titre pédagogique. Un curseur pourrait défiler sur la partition à mesure que l'œuvre est écoutée permettant aux élèves de s'habituer à suivre des partitions.

INScore inclut une dimension interactive par la manipulation temps réel de l'ensemble de ses objets. Ils peuvent être manipulés par l'intermédiaire de messages ou d'actions sur l'interface graphique. L'utilisation d'objets graphiques arbitraires est devenue indispensable dans la composition moderne qui utilise non seulement la notation standard, mais vise aussi à donner des instructions plus générales aux musiciens pour créer des ambiances sonores ou utiliser des dispositifs électroniques. La Figure 5 montre un extrait de la partition de la pièce Turenas de John Chowning, qui intègre un ensemble de conventions en plus de la notation musicale standard.

Figure 5 : Partition de Turenas de John Chowning (transcription par Laurent Pottier)

La partition peut aussi être utilisée comme un moyen de représentation de la musique



générée par des algorithmes à destination du compositeur, mais également du public qui appréhendera plus facilement le concept de la pièce qu'il est en train d'écouter. C'est le type d'utilisation que [Carinola, 2015] en fait dans sa pièce flux aeterna, générant de la musique en continu disponible par internet, visible en Figure 6.

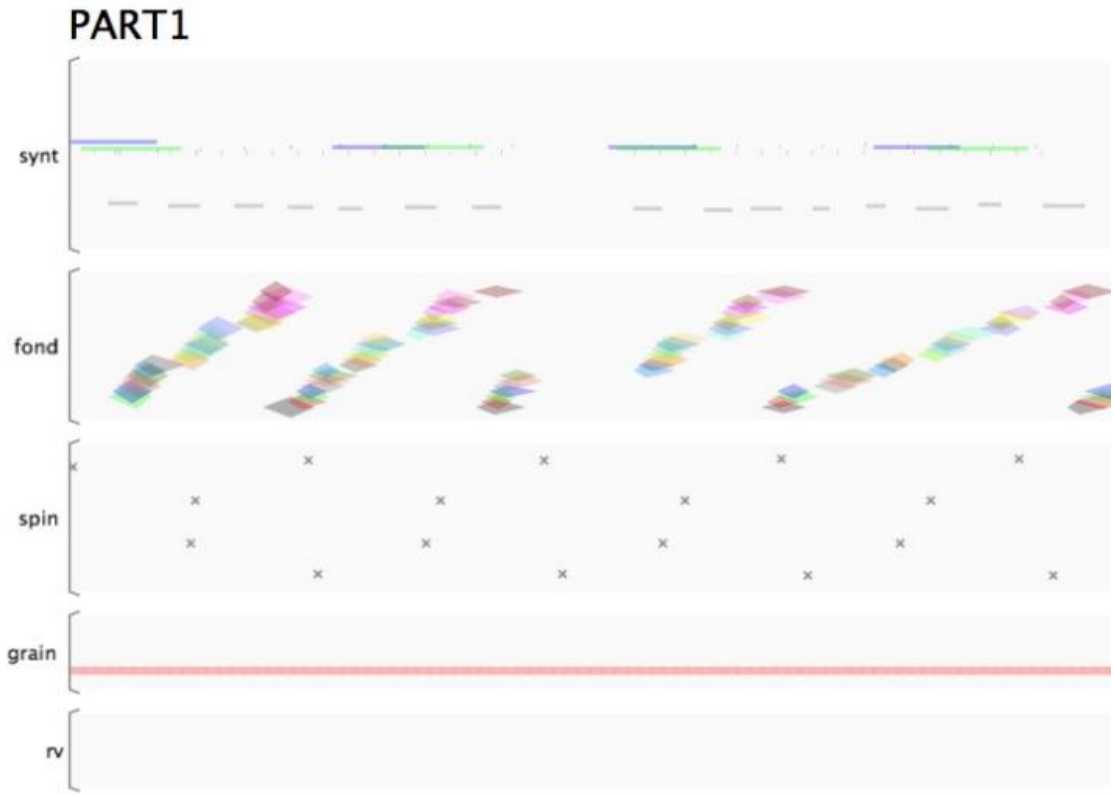


Figure 6 : Partition de représentation des opérations des algorithmes de génération de la musique [Carinola, 2015]

La représentation du geste utilisée à des fins d'expression sonore est un exemple des défis à relever dans la notation musicale moderne. En effet, comme le souligne [Fober, et al., 2015], peu de travaux ont été réalisés en prenant en compte une approche globale. Des pièces utilisent le geste depuis les années 1960, mais utilisent une codification qui leur est propre.

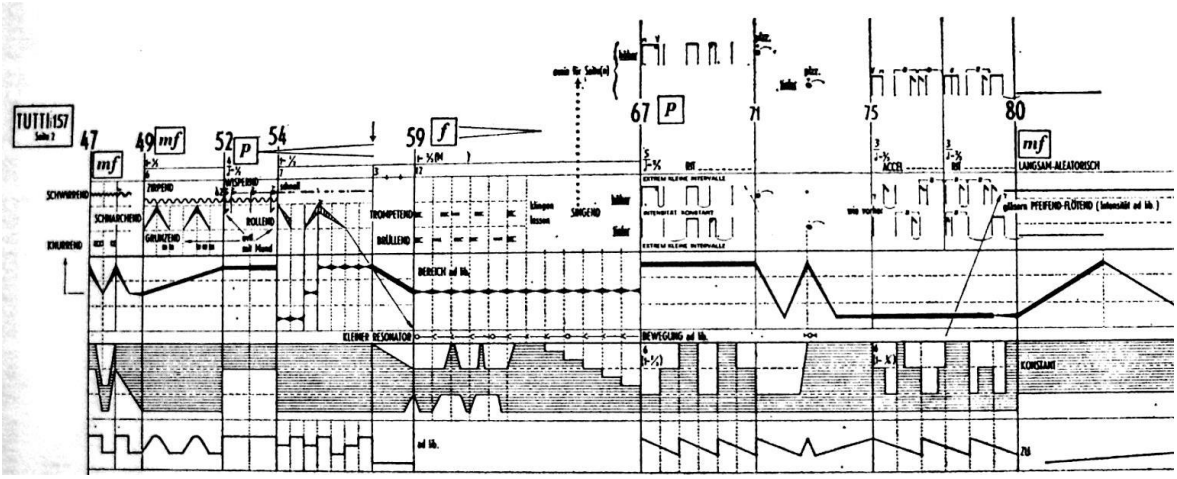


Figure 7 : Extraits de Mikrophonie 1 de Karlheinz Stockhausen

Dans l'extrait en Figure 7 de la partition de la pièce de Stockhausen, nous pouvons voir quatre lignes différentes, divisées elles-mêmes en trois sous-lignes. Le compositeur a établi une codification pour que les musiciens puissent effectuer les mouvements de microphone captant les sons d'un tam, instrument de percussion.

Fonctionnement d'INScore

INScore est avant tout un lecteur de partition. Il est bien sûr utilisé pour les créer, mais est rarement utilisé seul. Ce sont souvent des logiciels tiers qui vont piloter INScore par l'intermédiaire de message OSC (Open Sound Control). Il est néanmoins possible de glisser/déposer des fichiers de script sur une fenêtre d'INScore ou de glisser/déposer directement les lignes de script.

OSC est un protocole de communication initié par UC Berkeley Center for New Music and Audio Technology. [CNMAT, 2015] le présente comme un protocole simple et ouvert permettant un contrôle temps réel. Dans INScore, chaque objet est identifié par une adresse. Un objet est manipulable en créant un message composé de l'adresse de l'objet, un nom de message (c'est-à-dire une méthode) et 0 à n paramètres. Ce protocole est utilisé dans une implémentation utilisant UDP comme support réseau. UDP à l'avantage d'être non connecté et d'offrir des performances temps réels. En revanche, il ne permet pas de s'assurer de la livraison des paquets, il peut donc y avoir des pertes d'informations.

Une partition représentée par INScore est composée d'une ou plusieurs scènes. Une scène est un espace graphique conteneur des objets. Chaque scène possède sa propre fenêtre indépendante des autres. Les objets peuvent être eux-mêmes conteneur d'autres objets. L'application, chaque scène et objet, ont une adresse OSC unique permettant d'y accéder. L'ensemble des objets d'une partition peut être représenté par un arbre, chaque nœud ayant une adresse OSC. L'application et les scènes sont également des objets. Certains types d'objets peuvent avoir des nœuds statiques qui sont présents dès la création de l'objet et non supprimable. Ces nœuds statiques sont représentés en noir dans la Figure 8 ci-dessous. Les nœuds dynamiques sont quant à eux, représentés en bleu.

Le nœud `/signal` peut être vu comme un conteneur de signaux, permettant la représentation graphique des signaux.

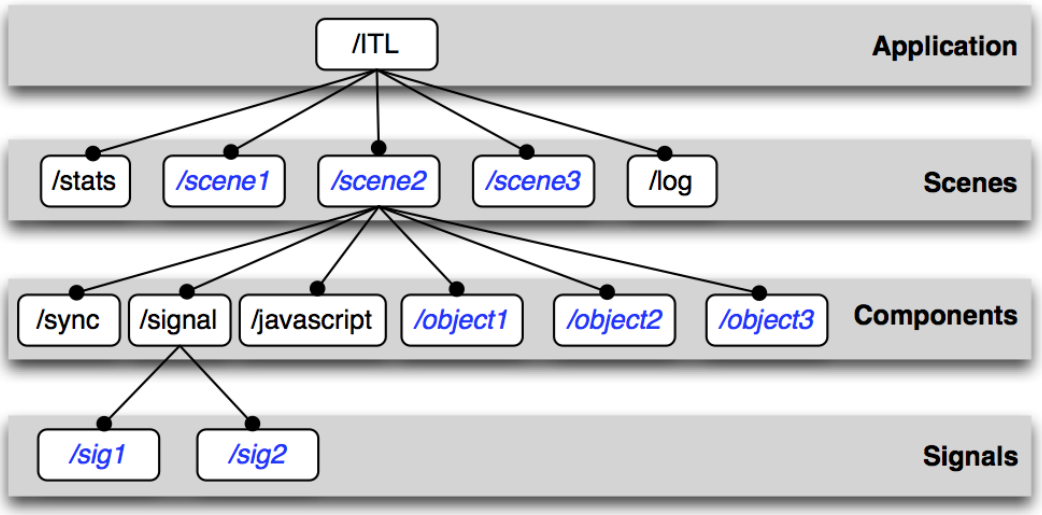


Figure 8 : L'espace d'adressage OSC [Fober, 2015]

L'application a été développée suivant l'architecture MVC (Modèle Vue Contrôleur). Cette architecture propose une solution à la gestion des interfaces graphiques des applications interactives. Elle propose un cadre de développement qui sépare les données de la présentation. La présentation des données et la logique de parcours de ces données sont développées dans des éléments externes au modèle métier. L'avantage d'une telle architecture est de permettre de faire une présentation différente des mêmes données sans avoir de modification du modèle. Comme le montre la Figure 9, elle est composée de 3 éléments principaux, le modèle, les vues et le contrôleur.

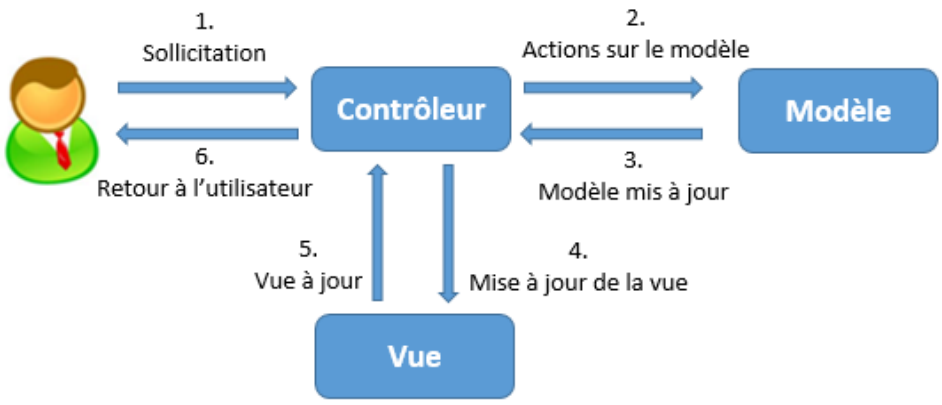


Figure 9 : Modèle théorique MVC

L'implémentation du modèle MVC par INScore nous est donnée par la Figure 10 :

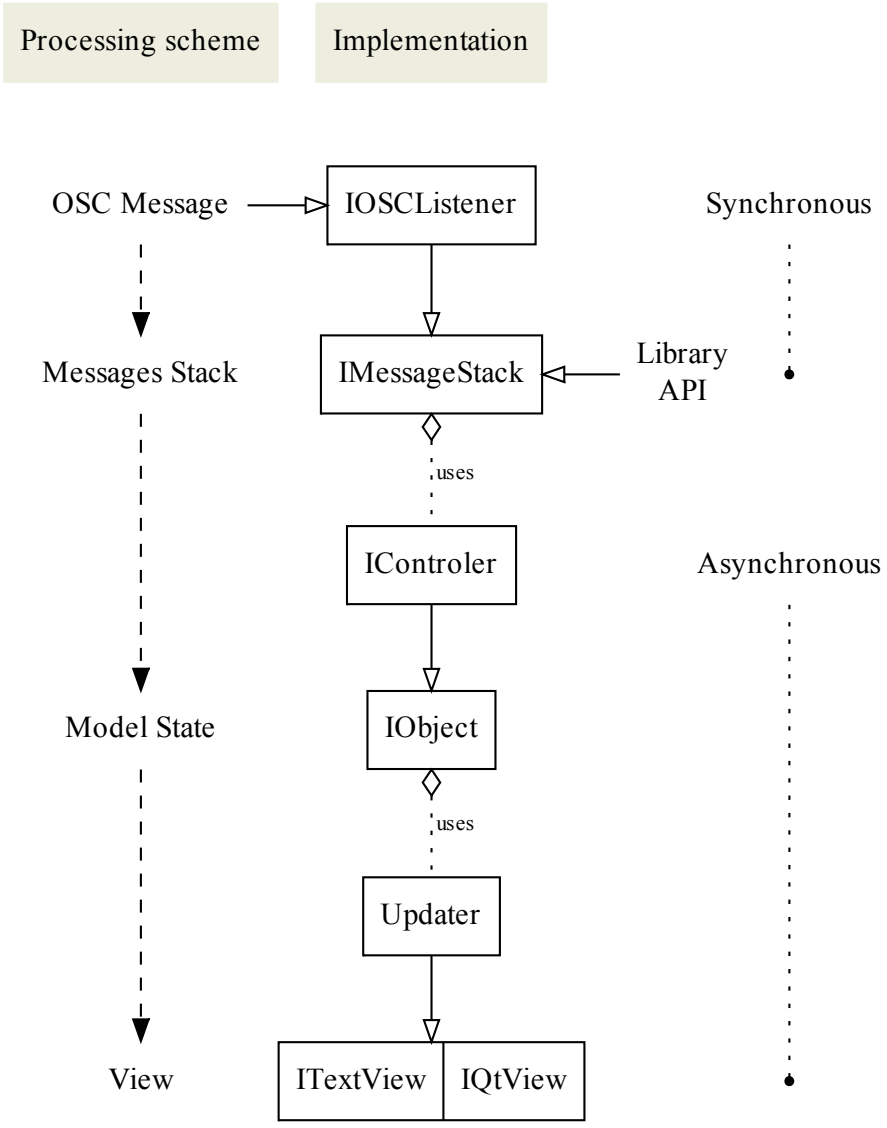


Figure 10 : Architecture MVC d'INScore [Fober Dominique, et al., 2012]

Les messages arrivent par l'écouteur OSC et sont stockés dans une pile de messages. Ces message sont les requêtes utilisateurs du modèle MVC figure XXX. Dans INScore, l'arrivée d'un message ne déclenche pas d'action du contrôleur, car la tactique adoptée est une lecture à intervalle réguliers de la pile de messages. Le traitement des messages est donc asynchrone. Le contrôleur distribue les messages aux différents objets concernés pour remettre à jour le modèle. Dans un deuxième temps, les vues (représentation graphique) sont mises à jour à partir de l'objet correspondant du modèle. L'objet « Updater » se charge de la mise à jour de la vue à partir du modèle. L'interface graphique en elle-même est développée à l'aide du framework Qt. Les vues sont donc des objets dépendants de ce framework.

Le framework Qt

Qt est un framework cross-platform pour le langage C++. Qt peut être utilisé librement pour les logiciels en licence LGPL. Il inclut un grand nombre de composants permettant de faire facilement des IHM, mais fournit également de nombreux modules facilitant les développements d'applications graphiques ou non. La composition modulaire du framework permet au développeur de choisir ce qui doit être inclus dans son projet. Il fournit également un IDE (Environnement de Développement Intégré) appelé QtCreator et un outil de gestion du processus de compilation cross-platform avec qmake. La Figure 11 donne un aperçu du framework et de ses principaux modules. De nombreux autres modules et outils ne sont pas représentés ici.

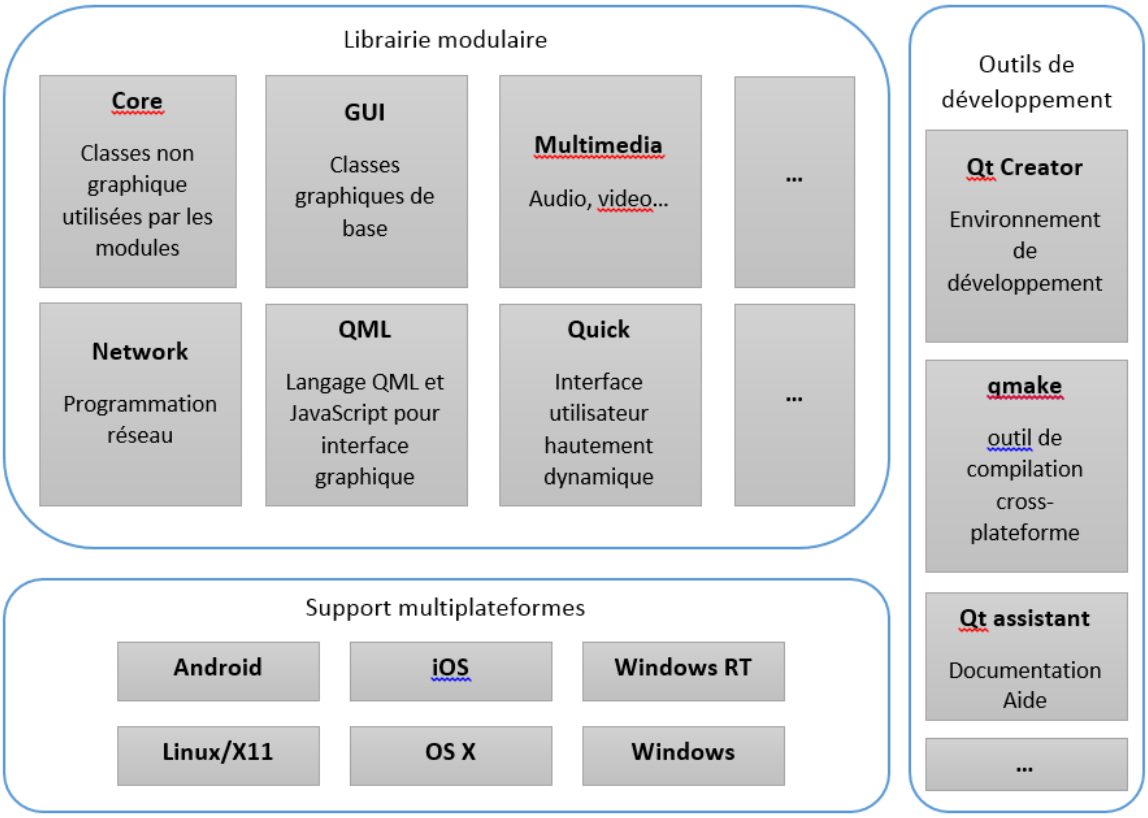


Figure 11 : Vue d'ensemble parcellaire du framework Qt

Principe de fonctionnement : la boucle d'évènement

Qt permet à l'utilisateur d'avoir des interactions avec le programme par l'intermédiaire de l'IHM. Chaque évènement est traité par une boucle d'évènement. Les objets communiquent entre eux par l'intermédiaire d'un système de signaux propre à Qt.

L'architecture Qt est basée sur une construction dirigée par les évènements. L'exécution de l'application se fait à l'intérieur d'une boucle qui traite les évènements internes ou externes à l'application. Cette boucle d'évènement est le thread principal. Les évènements ne sont pas traités en direct, mais sont stockés dans une file. Le rôle de la boucle d'évènement est de récupérer les évènements de la file et de les diriger vers leurs destinataires. Comme expliqué par [The Qt Company, 2015], il y a 4 types d'évènements :

- gestionnaire de fenêtrage.
- activité réseaux
- timers
- évènements postés par d'autres threads.

La boucle d'évènement doit pouvoir réagir rapidement aux sollicitations. Le traitement d'un évènement ne doit pas être long, sans quoi, l'application apparaitrait comme bloqué. Une des solutions pour les tâches longues est d'utiliser un autre thread pour ne pas bloquer le thread principal (la boucle d'évènements).

Les signaux

Le mécanisme de « signal » et « slot » est utilisé pour la communication entre les objets. Comme l'explique [The Qt Company, 2015], ce mécanisme propre à Qt est rendu possible par l'utilisation du Meta-Object System et de son compilateur Meta-Object Compiler. Les évènements et le mécanisme de signal/slot sont deux mécanismes différents pouvant accomplir les mêmes choses. Les signaux sont utilisés pour la communication inter-objet à l'intérieur de l'application. Ils sont plus simples à mettre en œuvre que les évènements.

Les signaux permettent d'avoir une alternative à la technique des callback. Le signal d'un objet peut être connecté à un ou plusieurs slot d'autres objets comme le montre la Figure 12.

Le traitement du slot est fait de manière asynchrone par rapport au signal, mais n'est pas traité dans un thread différent pour autant. Il faut donc, là aussi, éviter les traitements longs qui pourraient bloquer la boucle principale. Cette approche reste tout de même puissante, car les objets émetteurs et récepteurs n'ont pas se connaitre ni avoir une signature particulière. Il faut tout de même prendre garde à utiliser ce mécanisme à bon escient, car la traçabilité de l'exécution du code est rendue plus difficile.

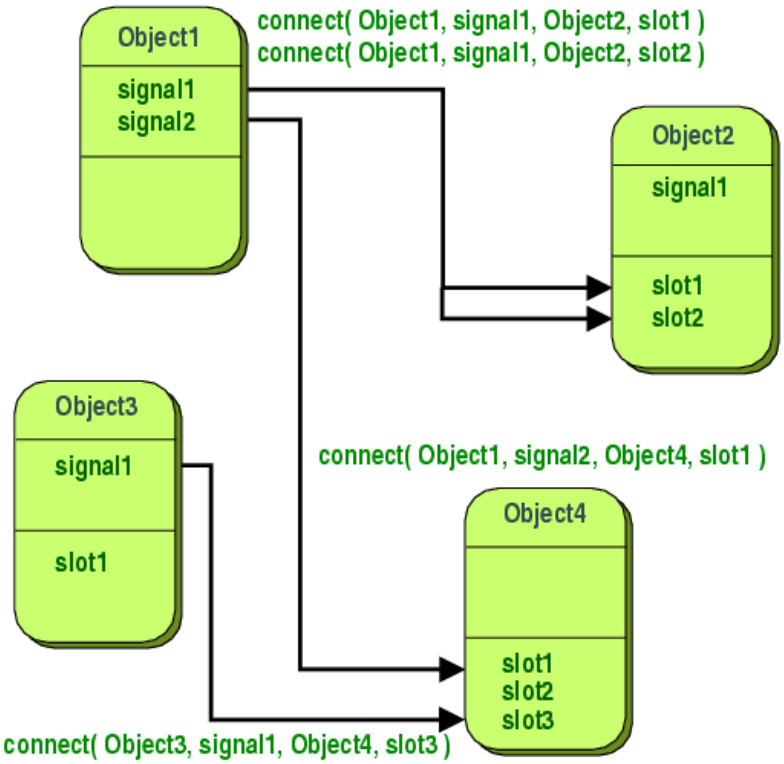


Figure 12 : Connexion des signaux des objets 1 et 3 aux slots des objets 2 et 4 [The Qt Company, 2015]

Les threads

A plusieurs reprises, nous avons pu voir que le parallélisme de l'exécution n'est qu'apparent, aussi bien avec les évènements qu'avec le mécanisme de « signal/slot ». Par conséquent, les exécutions de code prenant beaucoup de temps machine sont à éviter pour garder la réactivité et l'interactivité du logiciel.

Avant la version de 2011, le langage C++ ne fournissait pas d'API standard, c'est-à-dire disponible sur toutes les plateformes, pour créer des threads. Qt fournit une API haut niveau pour créer des threads et pour les manipuler, avec des classes permettant de créer des sémaphores, des mutex et des conditions d'attente. La création d'objets à l'intérieur de nouveaux threads permet de s'affranchir des contraintes précédemment citées. En se conformant aux préconisations de [The Qt Company, 2015], l'application peut être composée de multiples threads communiquant entre eux tel qu'illustré par Figure 13. Les objets peuvent communiquer à l'intérieur de l'espace d'exécution de leur thread, mais peuvent aussi utiliser les mécanismes d'évènement et de signal slot inter thread.

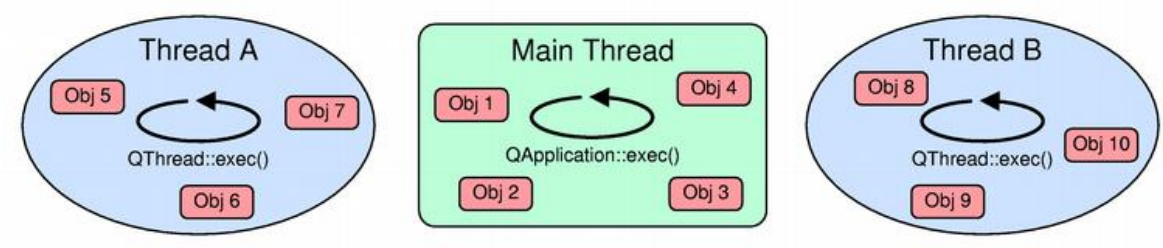


Figure 13 : Application utilisant deux threads supplémentaires à la boucle principale [The Qt Company, 2015]

Qmake, un outil de compilation cross-platform

À l'instar d'autres outils, qmake permet de simplifier la compilation d'une application, en fournissant une abstraction aux spécificités de chaque plateforme.

D'une syntaxe relativement simple comparé à d'autres produits (comme cmake), qmake peut être utilisé pour des projets n'utilisant pas le framework Qt. Qmake permet de générer des projets pour les différents environnements de développement (Visual Studio, Xcode ou codeblock) et est intégré à QtCreator. Outre sa souplesse, il permet donc au programmeur de ne pas changer ses habitudes en continuant à utiliser son environnement habituel.

Après cette présentation du contexte de travail, des projets concernés et des différentes technologies qu'ils utilisent, nous pouvons maintenant aborder l'objectif de ce travail. Nous allons commencer par voir les enjeux des migrations sur les différentes plateformes, puis les spécificités de celles-ci pour mieux appréhender le travail à fournir par la suite.

Chapitre 2

Enjeux, objectifs et contraintes

Les objectifs du travail visent à la fois à un déploiement sur plateformes mobiles et à une migration vers le Web du logiciel INScore. Le déploiement d'INScore sur des plateformes mobiles soulève à la fois des problèmes techniques et des questions de design, tant pour ce qui est de la présentation que de l'interaction avec la partition. La migration vers le Web pose quant à elle d'autres problèmes, liés d'une part à la migration nécessaire vers d'autres langages de programmation, et d'autre part à l'éclatement potentiel des ressources et des calculs. En effet, les architectures web modernes permettent de répartir les ressources sur différentes machines de façon complétement transparente pour les utilisateurs. La possibilité d'utiliser des contenus répartis sur différentes machines pour la création des partitions avec INScore permet une véritable intégration au monde du web. L'intégration de ces contenus externes permet également une répartition des calculs. La conservation de l'interactivité de la partition après un portage sur le web est un des défis à relever. Des perspectives nouvelles en matière d'usage, de collaboration et de design sont également étudiées.

2.1 – La nécessité de la migration

L'informatique a beaucoup évolué ces dernières années, les usages et les habitudes d'utilisations ont considérablement changé. La diffusion d'un logiciel passe par sa mise à disposition pour le plus grand nombre et donc à un déploiement sur un ensemble de plateformes parfois hétérogène. Dans le monde de la musique, le support des partitions commence à se diversifier. La tablette gagne sur le papier le fait de pouvoir contenir des milliers de partitions et d'être interactive. Son usage ne change pas fondamentalement les habitudes des musiciens, elle reste posée sur un pupitre. Son adoption devrait donc se faire rapidement. Du côté du web, la distribution et le partage de partitions commencent à voir le

jour et font émerger de nouvelles pratiques. Le propos est alors de trouver des solutions permettant de pallier les problèmes d'interaction tout en profitant des possibilités collaboratives offertes par internet.

La Figure 14 montre qu'en recherchant « music score app » sur internet, 39 millions de résultats sont retournés en étant associé avec le mot « Android », 12 millions avec « iOS » et 29 millions avec « iPad ». En utilisant « web » ou « internet », le nombre de résultats explose.

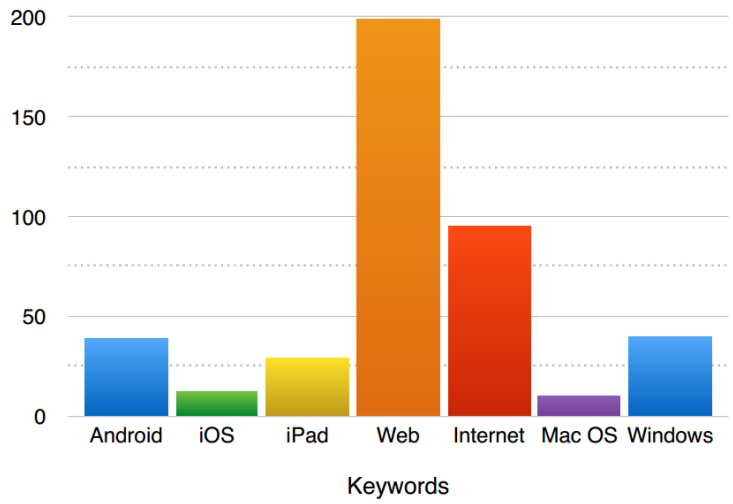


Figure 14 : Résultat de la recherche Google « music score app » associée à différents mots clefs.

Il est clair que le support des partitions est en migration sur les appareils mobiles, mais plus encore vers internet. Ce n'est pas seulement de visualisation dont il est question, mais également de création et d'édition de partitions.

2.1.1 – Le Web

L'évolution de technologies a permis de mettre en ligne des applications qui utilisent des interfaces utilisateur riches. Leur mise à disposition permet une ouverture au travail collaboratif et c'est également dans cette optique qu'INScore a vocation à être disponible sur le web.

L'avantage du web sur les autres plateformes qui utilisent différents systèmes d'exploitation, c'est que le navigateur peut être vu comme un type d'OS à part entière qui propose une abstraction du véritable système hôte. Les navigateurs intègrent d'ailleurs à chaque évolution de nouveaux services proposés par les OS. Les applications web ont donc l'avantage de

l'universalité théorique sur l'ensemble des plateformes. Malheureusement, les différences de comportement des navigateurs internet rendent parfois les choses plus compliquées dans la pratique, même s'il est vrai que les différences ont tendance à être moins marquées au fur et à mesure des versions.

La création d'interfaces riches sur le web est un des enjeux majeurs pour les applications. Rappelons que le web a été conçu initialement pour partager de la documentation au format texte, puis a évolué pour intégrer des images, des contenus multimédias, etc., pour arriver au web que l'on connait aujourd'hui. Différentes technologies qui pouvaient être intégrées au navigateur ont vu le jour (comme Adobe Flash ou Microsoft Silverlight) avant l'intégration récente de possibilités natives des navigateurs avec HTML5. Les utilisateurs attendent des applications interactives. Ces applications ont maintenant que peu de différences avec leurs homologues installées sur nos ordinateurs, comme nous pouvons le constater avec les traitements de texte et tableur en ligne par exemple.

La possibilité récente d'inclure des ressources réseau ouvre INScore au concept des mashup tel que les décrit [Plouin, 2011]. Les mashup sont des applications qui utilisent différentes ressources et différents services du web en les mixant pour créer un nouveau service ou une nouvelle application. De la même façon, INScore peut utiliser des ressources web (scripts, images, texte, page HTML…) et peut également, grâce à cette migration sur le web, mixer sa partition avec le résultat d'autres instances d'INScore tel que l'illustre la Figure 15.

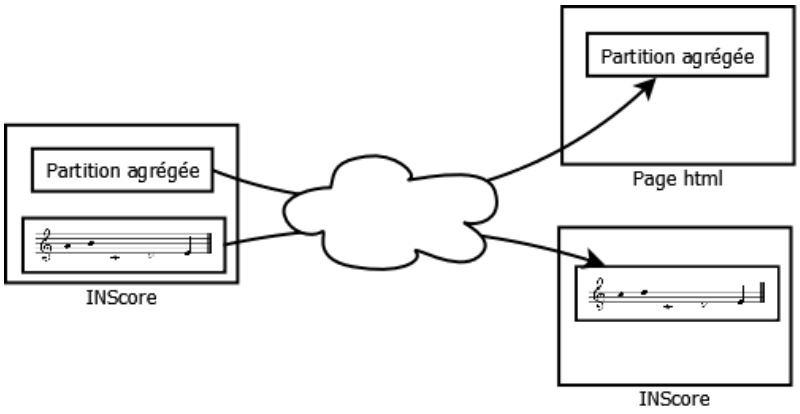


Figure 15 : Agrégation de ressource pour créer une partition

L'ouverture à un travail collaboratif passe d'abord par la mise à disposition de tous de l'application. Le pilotage à distance par l'intermédiaire du protocole OSC peut laisser penser que l'application était déjà prête à la collaboration. Elle ne l'était qu'en partie, car outre le

fait que l'on n'a pas le rendu à distance, le protocole OSC n'a pas de support HTTP. Il n'est donc pas possible de piloter INScore à travers le web.

2.1.2 – Les plateformes mobiles

La version mobile d'INScore est, plus encore que son homologue pour ordinateur, destinée à être pilotée. Les interfaces utilisateur des appareils mobiles n'utilisant pas le multifenêtrage, le glisser-déposer est impossible. La taille des écrans l'aurait de toute façon rendu difficile à manipuler voir inutilisable. Comme nous le verrons par la suite, une interface utilisateur d'accueil est toutefois nécessaire pour la version mobile. Cette interface permet un minimum de contrôle sur INScore. Elle est également nécessaire pour la validation de l'application avant la publication sur les marchés d'applications. En effet ces marchés d'applications effectuent une batterie de tests standards, l'affichage d'une simple scène vide serait donc rejeté.

À l'instar des concepts d'applications en cloud computing qui permettent une décentralisation en utilisant des clients riches, INScore peut être vu comme une RDA (Rich Desktop Application) alimentée par les flux de messages de différents logiciels. Les RDA sont des applications riches qui s'exécutent directement sur l'appareil. Il faut donc installer un runtime spécifique pour pouvoir exécuter l'application. L'avantage de ce type d'application est d'être intégré plus facilement au système, de proposer des fonctions natives, d'être plus interactives et de pouvoir fonctionner en mode déconnecté. En revanche, elles perdent la souplesse de l'application web qui est remise à jour sans aucune action de l'utilisateur.

La version mobile d'INScore suit la logique de décentralisation des applications qui passent de l'ordinateur à différents supports. Cette décentralisation sur mobile est en partie rendue possible par l'éclatement des ressources sur le réseau. La facilité de mise en ligne de ressources ou l'utilisation de dossiers réseau de stockage permet à l'utilisateur de partager facilement ses documents et donc de retrouver son environnement sur les différents appareils.

Dans le contexte de la musique, la tablette est séduisante par la facilité de transport qu'elle procure et par l'utilisation standard de la partition, c'est-à-dire posée sur un pupitre. Avec des logiciels adaptés tel qu'INScore, elle ouvre à de nouvelles possibilités pour les

compositeurs. Il devient simple d'avoir une installation permettant d'utiliser des partitions dynamiques.

L'interaction avec la partition est beaucoup plus facile avec une tablette qu'un ordinateur. En contexte de représentation, l'utilisation d'une souris par les musiciens est inappropriée alors que les écrans tactiles offrent une voie d'action beaucoup plus naturelle. Le compositeur peut donc imaginer des actions faites par les musiciens sur leur partition. Le résultat de ces actions peut être récupéré par l'instance d'INScore pilote ou transmis directement aux autres tablettes. L'exemple Figure 16 illustre ce cas d'utilisation avec un INScore maitre qui pilote les différentes tablettes des musiciens :

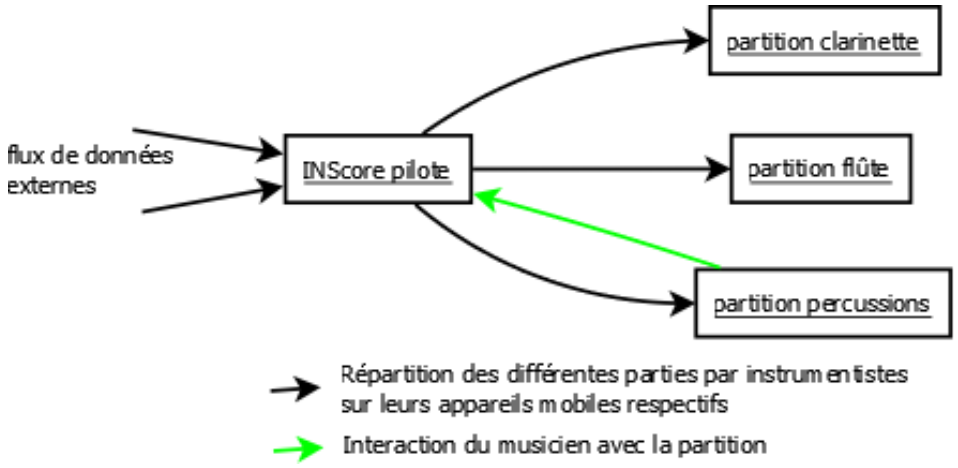


Figure 16 : Création en temps réel de parties séparées pour chaque instrument et récupération des interactions des instrumentistes

Enfin, une dernière raison pour migrer INScore sur appareil mobile est que certaines pièces récentes utilisent les gestes pour créer de la musique. Les valeurs détectées par les accéléromètres de l'appareil mobile servent à la génération de sons. La migration d'INScore sur mobile est l'étape préalable à l'utilisation de ces valeurs dans la représentation musicale. La Figure 17 montre une des utilisations qui pourrait en être fait. Dans le cas de cette figure, l'appareil mobile utilise en même temps un logiciel générateur de sons sur les mouvements et INScore. Ces applications captent les valeurs des accéléromètres. INScore convertit en message OSC à des fins d'affichage ou à destination d'un logiciel tiers. Bien sûr, la question de la représentation du geste est encore un autre problème qui n'est pas abordé ici.

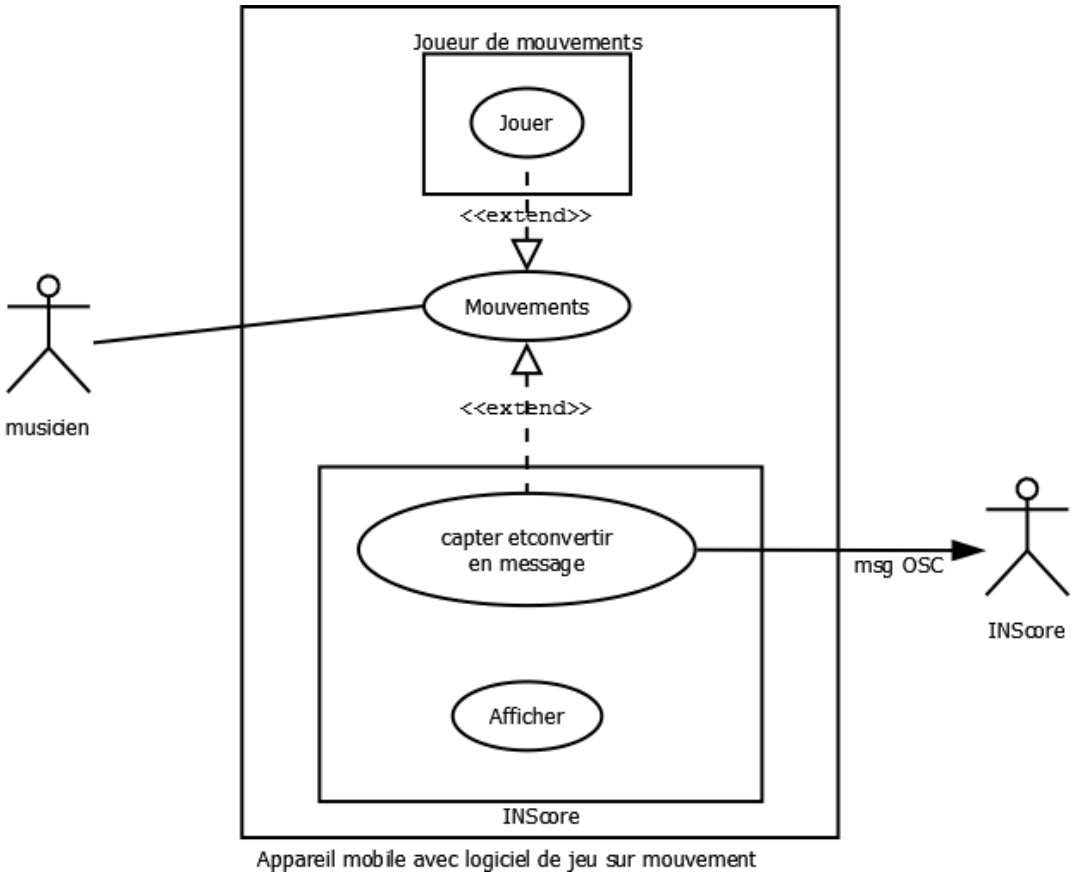


Figure 17 : Utilisation des gestes dans la musique

Dans le cas de la Figure 17, l'appareil mobile utilise en même temps un logiciel générateur de sons sur les mouvements et INScore. Ces applications captent les valeurs des accéléromètres. INScore convertit en message OSC à des fins d'affichage ou à destination d'un logiciel tiers. Bien sûr, la question de la représentation du geste est encore un autre problème qui n'est pas abordé ici.

2.1.3 – La collaboration

La mise sur le web d'INScore permet d'accéder à un nouvel ensemble d'utilisations. La première utilisation est la mise à disposition sur le web de la visualisation d'une partition dynamique. Le compositeur ou le système de composition crée la partition sur INScore. INScore permet par l'intermédiaire de son serveur embarqué de mettre à disposition la partition sur internet (voir Figure 18).

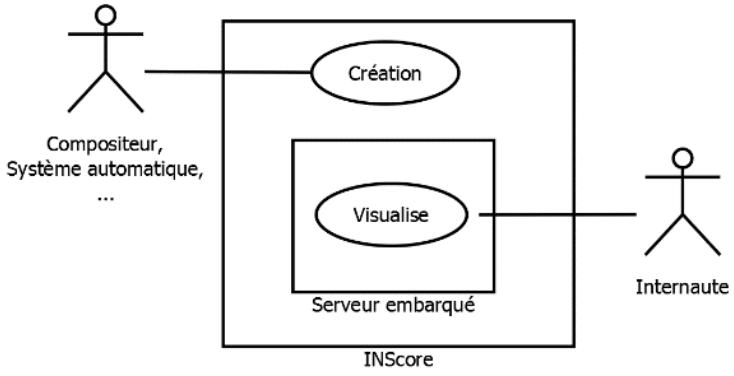


Figure 18 : Visualisation d'une partition à travers le web

Plusieurs situations peuvent tirer profit de ce nouvel accès. La pièce de Vincent Carinola présentée au chapitre 1 qui génère de la musique diffusée en continu sur le web peut également être suivie au niveau de sa partition. Un autre exemple est la diffusion du flux audio d'un concert pour lequel INScore est utilisé. La partition peut être diffusée en temps réel.

Dans ce premier cas d'utilisation, la composition était assurée en utilisant directement INScore sur l'ordinateur hôte. La composition collaborative s'avère beaucoup plus simple si le serveur embarqué dans INScore donne la possibilité de créer et manipuler les partitions à travers le web. Des compositeurs ou systèmes peuvent créer ensemble une partition tel que le montre la Figure 19.

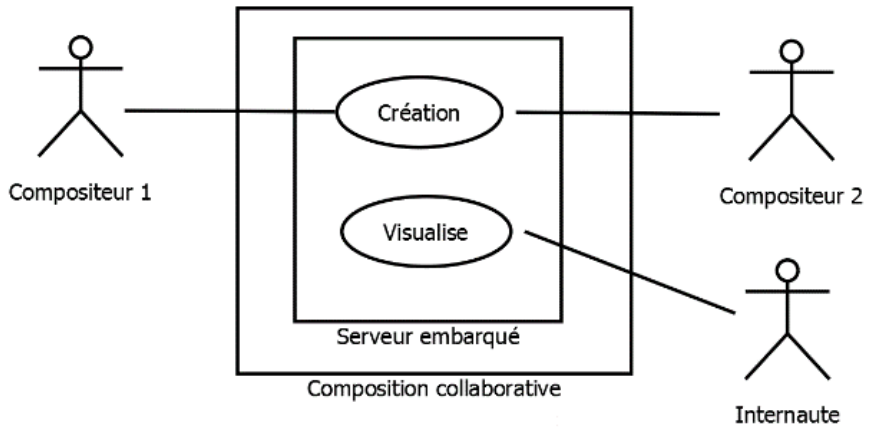


Figure 19 : Composition collaborative par le web

Les moyens techniques permettent déjà de faire des concerts où les musiciens sont répartis sur plusieurs lieux, comme dans la pièce « Miroirs Distants » de Jean Baptiste Barrière [Barrière, 2014], qui a permis de faire collaborer deux musiciens entre Lyon et New York pendant leur prestation. Le son et l'image de chacun étaient retransmis en direct dans leur

salle de concert respective. L'utilisation d'INScore sur le web permettrait d'ajouter la dimension de collaboration sur l'interaction avec la partition.

Enfin, les nouveautés d'INScore pour mobile et pour le web vont permettre de mettre facilement à disposition des contenus pour des évènements ponctuels. Le concert participatif, cas d'utilisation de la Figure 20, en est un exemple. Le public peut interagir sur le déroulement du concert ou de la pièce. Les actions du public peuvent être faites par l'intermédiaire de la version web de l'application ou sur la version mobile. Il est maintenant facile de faire installer sur les appareils mobiles une application en donnant un lien de téléchargement par un QRCode par exemple. Cette version mobile d'INScore contiendrait l'ensemble des ressources nécessaires au concert.

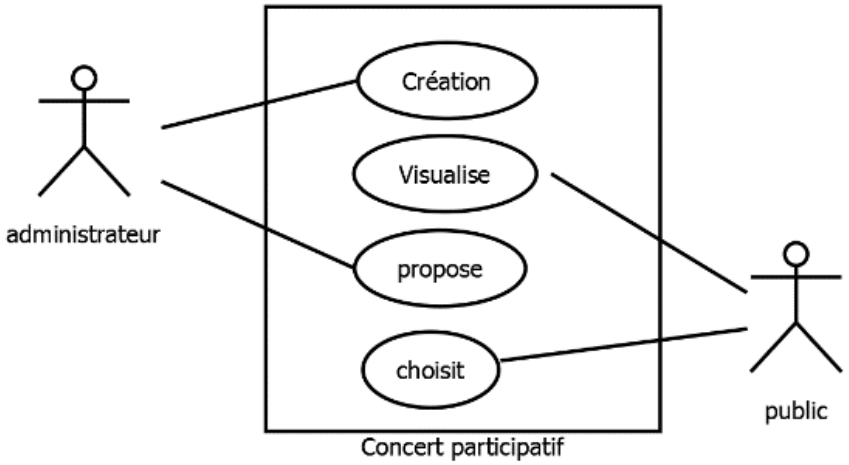


Figure 20 : Concert participatif à l'aide d'INScore

Le public peut donc visualiser le déroulement de la pièce sur son mobile et peut influer sur son déroulement en agissant sur celle-ci. Un administrateur, compositeur ou système informatique gère la partition en direct pour la mettre à jour et proposer les interactions.

Nous avons vu en quoi cette migration est nécessaire et quels en sont les enjeux et les apports pour la création musicale. Néanmoins, la variété des plateformes visées ne doit pas mettre en péril la maintenabilité des logiciels. Une seule personne permanente assure les développements et la maintenance des logiciels de notation musicale, il n'est donc pas question de réécrire les logiciels. De plus, la somme de travail que représente la migration devait rester cohérente par rapport aux 9 mois impartis. Nous allons maintenant aborder les spécificités des plateformes de destination pour en comprendre le fonctionnement et mieux appréhender le travail à effectuer.

2.2 – Spécificités techniques des plateformes

Les plateformes visées sont très hétérogènes, tant par les possibilités techniques qu'elles offrent que par les technologies utilisées. Rendre disponible sur le web une application développée pour un ordinateur de bureau traditionnel parait difficile tant la conception est différente, mais l'exposition sur le web une API minimale de l'application au travers de services web permet de donner un contrôle à distance et d'avoir les retours nécessaires à la gestion de ses actions. La création de services web pour les applications Guido et INScore est la première technologie utilisée pour la migration vers le web.

La deuxième technologie à destination du web est JavaScript. Bien sûr, il n'est pas question de réécrire des milliers de lignes de code en JavaScript. Cette problématique a été rencontrée par d'autres personnes et des technologies permettant la compilation de différents langages vers JavaScript sont apparues. Nous verrons le fonctionnement d'Emscripten pour porter un programme C/C++ en un programme JavaScript.

Enfin, les plateformes mobiles visées (Android et iOS) ont chacune leur environnement de développement et des contraintes qui leur sont propres.

2.2.1 – Les Services Web

Les Services Web permettent d'exposer un ensemble de fonctionnalités et de ressources sur le web. Le succès des Services Web tient dans leur capacité à faire communiquer des systèmes hétérogènes par l'utilisation d'un format commun pour l'échange des données. Ils permettent également de créer facilement des applications distribuées en faisant communiquer différents serveurs. Les Services Web sont donc souvent destinés à être utilisés par des applications plutôt que par les humains. Pourtant dans certains cas, il serait intéressant qu'ils soient utilisables et compréhensibles par l'homme.

Dans un premier temps, nous allons voir comment a été choisie la technologie utilisée pour le Service Web fonctionnant sur HTTP et quel en est son fonctionnement. Le format d'échange choisi ainsi que les bonnes pratiques sont également détaillés. Ensuite, une présentation des websocket est faite. Cette technologie permettant l'échange bidirectionnel entre client et serveur depuis du code JavaScript, est utilisée par la suite pour donner une alternative élégante au service web sur HTTP.

Le choix d'une technologie : REST versus SOAP

Les deux technologies principalement utilisées pour les Services Web sont REST (REpresentational State Transfert) et SOAP (Simple Object Access Protocol). Le premier est orienté ressources et permet aux agents d'interagir avec les ressources en les désignant par leur adresse et en utilisant une méthode du protocole HTTP. Le second a été créé pour pouvoir exécuter à distance des procédures et en récupérer le résultat. C'est un service de type RPC (Remote Procedure Call ou appel de procédure à distance).

Les services REST reposent sur le fonctionnement du web que nous utilisons tous les jours avec un navigateur internet. Lorsque nous naviguons sur un site internet, nous consultons une ressource désignée par son URL, c'est-à-dire une page du site. C'est exactement le même principe qui est utilisé avec REST. En utilisant les différentes méthodes HTTP, les ressources peuvent être complètement gérées à travers le service REST comme nous allons le voir par la suite avec les bonnes pratiques de ce type de services.

Les Services SOAP utilisent pour leurs échanges des enveloppes XML spécifiques. L'enveloppe de la requête permet de décrire la procédure à exécuter et de transmettre les données nécessaires. La réponse contient les données de retour de la procédure encapsulées dans une enveloppe XML. Il n'est donc pas facile d'utiliser SOAP sans passer par une application qui crée la requête et extrait les données de la réponse. Ce sont généralement des outils qui génèrent les clients de Service Web utilisé dans les applications.

Voici une comparaison entre la requête REST et SOAP d'un service web hypothétique permettant de consulter une photographie :

Requête REST :

```
GET /vacances2015/islande_001.png HTTP/1.1
Host: www.mesphotos.org
```

Requête SOAP :

```
POST / HTTP/1.1
Host: www.mesphotos.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 326

<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m=" www.mesphotos.org/vacances2015">
    <m:GetImage>
      <m:ImageName>islande_001.png</m:ImageName>
    </m:GetImage>
  </soap:Body>
</soap:Envelope>
```

La requête REST est nettement plus simple que la requête SOAP. Elle peut en outre être exécutée dans un navigateur. L'utilisation de REST est d'autant plus intéressante dans notre cas qu'elle s'intègre beaucoup plus facilement dans le cadre de la création d'applications en mashup comme évoqué précédemment, car l'utilisation est plus intuitive. Une des qualités de HTTP est sa simplicité qui en fait un protocole utilisable autant par des programmes que par des humains. C'est sur celle-ci que REST s'est appuyé.

La complexité du protocole SOAP permet en revanche d'avoir des fonctionnalités plus avancées. Il permet par exemple d'avoir une gestion de transactions sur plusieurs requêtes ou la gestion d'authentifications. Il faut noter également que SOAP pourrait très bien utiliser un autre protocole de transport que HTTP bien que ce soit le protocole le plus couramment utilisé. Ces fonctionnalités ne sont pas utiles dans l'utilisation que nous allons faire des web service. L'ensemble de ces raisons fait que REST est plus adapté à notre besoin.

Les bonnes pratiques REST

REST n'est pas une architecture, mais un ensemble de critères de conception qui permettent d'avoir une utilisation intuitive. Chaque fonction doit réaliser ce que son nom indique. Le

service REST doit être basé sur HTTP. Prenons donc tout d'abord le temps d'avoir un aperçu de ce protocole.

Le protocole HTTP est un protocole de communication client-serveur qui fonctionne en mode connecté et synchrone, c'est-à-dire qu'à chaque requête, une réponse est attendue. Le client ne peut pas envoyer une seconde requête avant d'avoir reçu la réponse à la première sans perdre celle-ci. Une requête se fait sur une URI particulière (une ressource). La réponse doit utiliser un des codes de retour HTTP. L'adjonction d'une des méthodes HTTP à la requête permet au serveur de déterminer l'action à effectuer. Des données peuvent également être jointes dans le corps de la requête comme l'explique [Fielding, et al., 1999] dans la spécification. La spécification du protocole HTTP nous donne le rôle de chaque méthode. C'est comme tel qu'elles devraient être utilisées avec un service REST afin d'avoir une interface uniforme et expressive. J'omets volontairement les méthodes TRACE et CONNECT qui ne sont généralement pas utilisées dans REST, ainsi que l'utilisation de méthodes personnelles qui ne vont pas dans le sens d'une sémantique partagée par les acteurs du web :

- GET : lecture d'une ressource
- HEAD : accès aux métadonnées d'une ressource
- POST : modification d'une ressource existante ou création d'une ressource en relation avec une autre ressource. L'URI de cette nouvelle ressource est choisie par le serveur.
- PUT : Création d'une nouvelle ressource en définissant son URI.
- DELETE : suppression d'une ressource.

Toujours dans l'idée du web sémantique, une URI (Universal Resource Identifier) doit être descriptive, c'est-à-dire être structurée et prédictible. C'est de cette façon qu'un client peut créer ses propres points d'entrée et utiliser le service d'une manière qui n'avait pas été prévu par son créateur. Elle doit aussi désigner un seul document. Cela parait évident, mais ce n'est pas toujours le cas. Prenez certains webmail, la même URI est utilisée quel que soit le mail consulté.

L'utilisation du code de retour HTTP permet d'analyser rapidement si le traitement s'est déroulé correctement.

Un bon service REST doit également respecter les deux propriétés suivantes issue de l'architecture orientée ressources (ROA : Resource Oriented Architecture) telle qu'elle est décrite par [Richardson, et al., 2007] :

Absence d'état :

Chaque requête se produit de manière isolée, le serveur ne s'appuie jamais sur des informations des requêtes passées. L'inconvénient de cette propriété est qu'il faut repasser l'ensemble des informations à chaque requête, comme le fait par exemple le moteur de recherche Google lorsque l'on change de page. En revanche, il n'y a pas d'expiration de délais côté client, ce qui supprime la gestion d'un grand nombre de condition d'erreur. C'est également plus facile de faire de la répartition de charge et de la mise en cache.

En conclusion, l'état de l'application doit être géré côté client. C'est lui qui gère sa progression dans l'application. L'état des ressources est géré côté serveur, car c'est le même pour tous les clients.

Connexité :

La connexité est la propriété des ressources à posséder des liens vers les états voisins ou d'intérêt par rapport à la requête. C'est une propriété qui est largement utilisée dans le web que nous connaissons, mais l'est beaucoup moins dans le cadre de web service. Ça pourrait se traduire par l'adjonction de l'URL des ressources lorsqu'une liste est fournie au client pour lui éviter d'avoir à les reconstruire.

En adjonction de ces propriétés, l'utilisation des méthodes HTTP devrait donner des garanties toujours selon [Richardson, et al., 2007]. Les méthodes GET et HEAD doivent assurer la sûreté. Ce sont des méthodes utilisées en lecture qui ne doivent pas occasionner de changement côté serveur. Souvent les serveurs écrivent dans un journal ou incrémente un compteur, c'est alors considéré comme un effet de bord. Les méthodes PUT de DELETE doivent être idempotentes, c'est-à-dire qu'une ou plusieurs exécutions de la même requête doit donner le même résultat. Ces deux garanties sont importantes dans un contexte comme le web où le réseau n'est pas toujours fiable. Les requêtes peuvent être réémises qu'elles aient été ou non déjà reçues. La méthode POST n'assure quant à elle aucune de ces deux garanties. Par exemple, l'idempotence ne peut être garantie avec un service d'incrémentation d'un nombre entier. La méthode POST devra être utilisée à bon escient.

Nous avons maintenant une vision idéale de l'architecture d'un service REST. Ce type de service est mis en pratique avec Guido pour faire un éditeur en ligne de partition et avec INScore pour le piloter à distance.

Présentation des Web sockets

Les web sockets permettent de communiquer à travers le réseau par l'intermédiaire d'un objet JavaScript [Flanagan, 2011]. Les web sockets utilisent leur propre protocole de communication, connecté et bidirectionnel, basé sur TCP. Les URL des web socket commence par « ws:// » ou « wss:// » pour les web sockets sécurisées.

Il est bien sûr possible d'utiliser HTTP pour communiquer depuis JavaScript, mais ce qui nous intéresse ici, ce sont les nouvelles possibilités offertes par une connexion longue durée entre un client et le serveur. Nous avons vu que HTTP est sans état. Il y a établissement de la connexion, une requête, la réponse puis la déconnexion. La connexion dans le cadre des web sockets est maintenue jusqu'à ce qu'une des deux parties la ferme.

La connexion bidirectionnelle permet au serveur d'envoyer des données aux clients sans être sollicité puisqu'il a connaissance de l'ensemble de ses clients. Les échanges entre clients et serveur ne sont pas synchrones contrairement à HTTP. Si un client envoie une requête qui implique une réponse, le message suivant reçu du serveur n'est pas forcément cette réponse. Cela peut être tout autre message du serveur.

Dans le contexte des partitions dynamiques avec INScore, la possibilité d'envoyer des mises à jour de la partition à l'initiative du serveur est séduisante. En revanche, la création d'un service web par l'intermédiaire des web sockets demande une conception inhabituelle par rapport à un service REST basé sur HTTP. L'utilisation des URI n'est pas possible pour désigner les ressources, puisqu'une web socket est associée à une seule URI. Il n'y a pas non plus la notion de méthode comme avec HTTP. Le couple requête / réponse est asynchrone, il va donc falloir trouver un moyen d'associer une réponse à la requête.

Nous verrons par la suite dans les adaptations d'INScore pour le web les solutions qui ont été mises en place.

2.2.2 – JavaScript

Le langage JavaScript est associé aux technologies du Web, mais l'évolution du langage et des moteurs d'exécution a permis à JavaScript de conquérir un terrain qui ne lui était pas destiné. JavaScript a maintenant un rôle côté serveur. Les librairies JavaScript peuvent être utilisées dans un serveur Node.js par exemple, qui ouvre de nombreuses possibilités. L'utilisation d'un navigateur n'est alors plus nécessaire pour exécuter le code JavaScript. La mise à disposition des moteurs d'exécution JavaScript aux développeurs leur permet d'intégrer le langage dans leurs programmes. Dans notre cas, JavaScript est utilisé comme un moyen de portage des applications vers de nouvelles plateformes exécutant du JavaScript. Le projet Emscripten est utilisé à cette fin.

Emscripten est un outil permettant de compiler du code C/C++ vers du JavaScript optimisé. Il utilise un ensemble de technologies existantes pour arriver à ce résultat. Comprendre l'ensemble de la chaine de compilation et les différentes technologies utilisées par Emscripten peut devenir indispensable avec des projets complexes pour la recherche de performance à l'exécution.

Le cheminement est en deux étapes :

- compilation du code C/C++ en assembleur LLVM (Low Level Virtual Machine) à l'aide du compilateur Clang.
- compilation de l'assembleur LLVM en JavaScript.

Grâce à la production de code optimisé asm.js, les applications ainsi compilées conservent une bonne rapidité d'exécution. Emscripten fournit également un ensemble de librairies standards pour permettre de compiler les projets ayant des dépendances. Le détail des outils et technologies utilisés va maintenant être présenté.

Le compilateur Clang et LLVM

La première étape est la compilation du code C/C++ en assembleur LLVM qui est une représentation intermédiaire du programme compilé indépendante des spécificités matérielles. Les développeurs du projet Emscripten se sont tournés vers le compilateur Clang, un sous projet de LLVM.

Le compilateur Clang permet de compiler du C, C++, Objective C et Objective C++. Il est disponible sous licence BSD, qui est compatible avec des produits commerciaux.

Le but de Clang est de fournir un nouveau frontend (compilation d'un langage défini vers une nouvelle représentation) pour les communautés Apple et LLVM. La communauté Apple est intéressée par Clang car GCC, le compilateur GNU utilisé traditionnellement sur les plateformes de type Unix, ne couvre pas forcément correctement les besoins de l'IDE d'Apple. Dans notre cas, nous nous intéressons à Clang pour sa sortie de compilation à destination de la communauté LLVM. Les avantages revendiqués par les développeurs de Clang [Naroff, 2007] sont la fourniture des diagnostics d'erreurs plus expressifs et une utilisation plus large que la simple compilation en fournissant un ensemble d'outils, comme l'analyse statique de code ou la génération de code. Il serait également plus performant, jusqu'à trois fois plus rapide que GCC.

Grâce à Clang, notre programme est maintenant converti en assembleur LLVM. LLVM est un ensemble de compilateurs et de chaines d'outils réutilisables dont le principal sous-projet est LLVM Core. Le terme LLVM est souvent utilisé en lieu et place de LLVM-Core par la communauté, ce qui peut rendre la compréhension de l'utilisation qui en est faite un peu difficile au premier abord.

Les librairies LLVM permettent d'avoir un optimiseur moderne et indépendant de la plateforme. Elles permettent également la génération de code. L'ensemble de LLVM est construit autour d'une représentation du code appelé LLVM Intermediate Representation (LLVM IR). La formalisation de LLVM IR étant bien documenté [LLVM Project, 2015], elle permet de créer son propre backend, c'est-à-dire de traduire le code LLVM IR en un langage machine spécifique. La compilation juste à temps permet de créer et exécuter du code machine spécifique à la demande, ce qui est essentiel pour les langages interprétés. LLVM IR est en forme Static Single Assignment (SSA), technique largement utilisé dans les compilateurs modernes, qui consiste à n'affecter qu'une seule fois chaque variable pour permettre la détection de variables inutiles et de code mort.

La production de code asm.js

La seconde étape de la compilation de notre programme C/C++ en JavaScript est la production de code asm.js à partir du bytecode LLVM. asm.js est un langage intermédiaire constitué d'un sous-ensemble de JavaScript permettant une exécution beaucoup plus rapide

que du JavaScript traditionnel. Le terme de langage intermédiaire est utilisé, car le code asm.js n'est pas destiné à être écrit par l'homme mais plutôt à être généré. L'avantage d'asm.js est qu'il peut être exécuté par n'importe quel moteur JavaScript. Asm.js est encore en cours de spécification, mais le dernier brouillon de travail [Herman, et al., 2014] permet d'avoir une bonne vue d'ensemble de cette technologie.

Pourquoi asm.js est plus rapide ?

Le sous-ensemble de JavaScript faisant partie d'asm.js a été sélectionné, car il permet de faire de la compilation anticipée. La lenteur à l'exécution du langage JavaScript provient en grande partie de l'allocation dynamique de mémoire et de l'utilisation du ramasse-miette. En effet, le principe du ramasse-miette est d'effectuer automatiquement une vérification des zones mémoires allouées dynamiquement afin de libérer les zones n'ayant plus aucune référence dans l'exécution du programme. La compilation anticipée permet quant à elle de traduire le programme en langage machine avant l'exécution et donc d'utiliser un espace mémoire restreint évitant l'allocation dynamique contrairement à la compilation à la volée, utilisée par les langages de script.

Le second facteur faisant d'asm.js un langage performant est l'optimisation du moteur d'exécution pour celui-ci. La performance d'un programme asm.js ne va donc pas dépendre uniquement de la machine utilisée, mais également du moteur JavaScript utilisé à l'exécution. La spécification et la promotion d'asm.js étant assurées par des employés de la fondation Mozilla, les navigateurs utilisant le moteur JavaScript de Mozilla (à savoir SpiderMonkey) ont de grandes chances d'avoir de très bonnes performances.

D'un point de vue technique, asm.js est construit autour des deux types entier et à virgule flottante et un espace mémoire d'allocation représenté par un tableau (typed array). Cet espace mémoire est alloué au démarrage du programme et est ensuite utilisé en guise de tas, pour faire référence aux langages plus communément utilisés.

Fonctionnement d'Emscripten

Le projet Emscripten a donc comme tâche principale de générer le code JavaScript optimisé asm.js à partir de l'assembleur LLVM. [Zakai, 2013] nous explique que cette étape est assez différente de ce qui est fait traditionnellement dans la compilation, puisqu'il s'agit d'un langage bas niveau (assembleur LLVM), vers un langage haut niveau (JavaScript). Un

travail de reconstitution des structures de contrôle utilisé dans JavaScript à partir de l'assembleur LLVM doit être fait. À l'instar des décompilateurs, Emscripten utilise son propre algorithme permettant de recréer les boucles (Relooper algorithm) en produisant un code optimisé, dont le détail peut être trouvé sur [Zakai, 2013]. L'utilisation du Closure Compiler [Google, 2015] (un compilateur JavaScript vers JavaScript) permet d'appliquer également plusieurs algorithmes d'optimisation pour supprimer le code mort, optimiser les variables par l'utilisation de la pile au lieu du tas et minimiser la taille du code.

Emscripten inclut un ensemble de librairies C/C++ standards pour permettre la compilation des projets. Dans le cas d'utilisation de librairies plus exotiques ou de framework trop complexe à porter, le développeur peut essayer de faire le portage lui-même ou alors de redévelopper ces fonctionnalités avec des outils et librairies disponibles avec Emscripten.

Prendre comme point de départ le code LLVM parait être plus compliqué que de partir directement du C/C++, mais les développeurs d'Emscripten voulaient pouvoir faire un projet ayant une portée plus large que le simple C/C++, en s'appuyant sur les nombreux compilateurs générant le code LLVM.

2.2.3 – Android

Le système d'exploitation Android a été spécialement conçu pour les appareils mobiles. Initialement créé par la société de conception de logiciels pour appareils mobile Android INC, ce système est maintenant la propriété de Google après le rachat de cette start-up en 2005 comme le rapporte [Bloomberg, 2005]. Bien que le projet Google Android soit un projet Open Source, celui-ci est piloté par Google. Seules les sources de la dernière version officielle d'Android sont disponibles publiquement. Les sources des versions en cours de développement restent fermées. N'importe quelle entreprise ou développeur particulier peut néanmoins participer au développement du système d'exploitation et soumettre des modifications qui subiront un processus de validation par Google.

Très solidement ancré dans le marché des plateformes mobiles, ce système est incontournable pour assurer la bonne diffusion d'une application. La part de marché est encore en augmentation d'après [Gartner, 2014] et 82 % des mobiles vendus utilisaient Android au troisième trimestre 2014.

Architecture

Le système Android est architecturé en différentes couches logicielles tel que l'illustre la Figure 21. Il est basé sur un noyau Linux sur lequel sont greffées différentes couches d'abstraction. Le langage cible de développement des applications Android est Java, langage qui s'exécute en utilisant une machine virtuelle, alors que le noyau Linux est en code natif C/C++. Les différentes couches logicielles vont permettre de faire un pont entre ces deux mondes et de pouvoir s'affranchir totalement du langage natif pour le développeur le désirant. L'orientation de l'architecture vers Java fait d'Android une plateforme Java et non Linux.

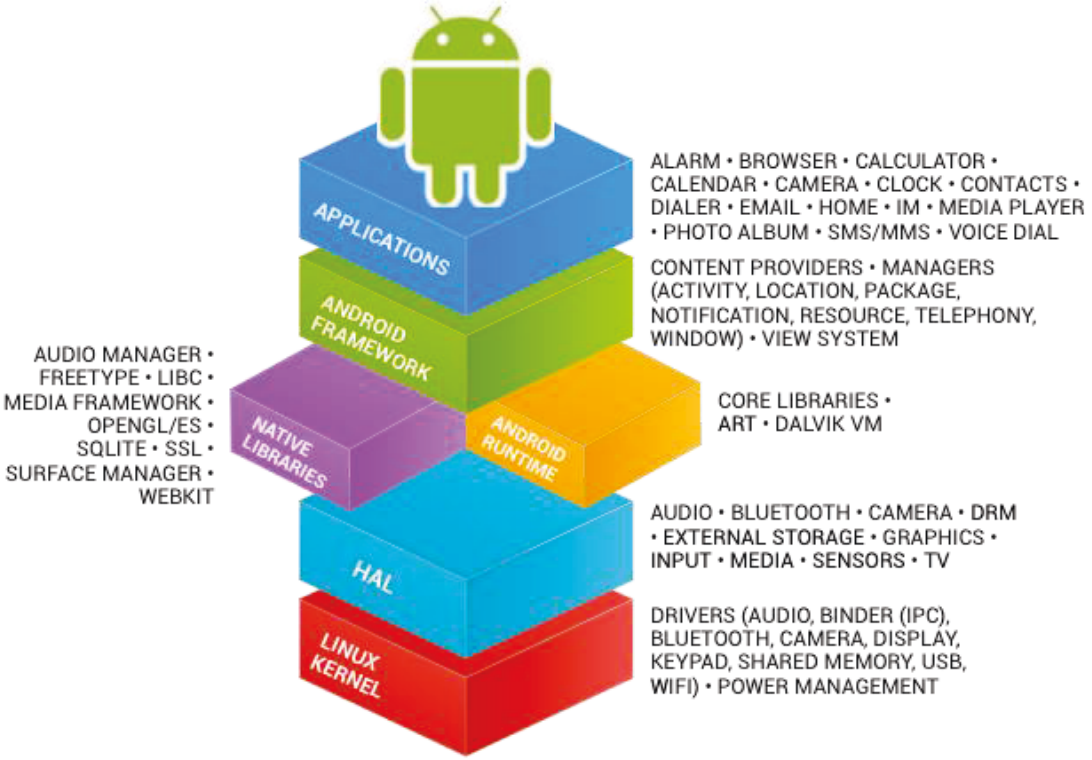


Figure 21 : Vue d'ensemble de l'architecture du système Android [Android source, 2015]

Noyau Linux

Le noyau Linux ne comporte que peu de modifications par rapport à un noyau standard. Les principales modifications concernent la gestion mémoire et les communications interprocessus. Il a en charge le fonctionnement de tous les périphériques matériels de l'appareil.

Couche d'abstraction matérielle (HAL Hardware Abstraction Layer)

Cette couche d'abstraction matérielle permet de s'affranchir des spécificités des drivers du matériel. Cette couche est implémentée en langage natif, c'est-à-dire en C/C++.

Librairies natives

Les librairies natives peuvent servir à toute sorte de choses. Elles fournissent des services aux couches supérieures. Elles ne sont pas uniquement dédiées au système, les applications peuvent installer et utiliser leurs propres librairies natives et communiquer avec elles par l'intermédiaire de JNI[2]. Il y a principalement des libraires de développement standard, des frameworks multimédias, pour la gestion de l'audio, et différents frameworks graphiques pour la gestion des interfaces utilisateurs.

Android Runtime

Le moteur d'exécution permet de faire fonctionner les applications. Il est composé de librairies standard Java et de librairies spécifiques à Android. Il inclut également la machine virtuelle qui permet d'exécuter le code java compilé. Cette machine virtuelle n'est pas la même que celle présente sur nos ordinateurs. La machine virtuelle Android exécute un bytecode java spécial au format DEX (Dalvik Executable). C'est au moment de la compilation des applications que le bytecode java standard est transformé en bytecode Dalvik. L'avantage d'utiliser ce bytecode Davlik est de s'affranchir des évolutions de java et d'avoir sa propre licence sur le format. Les avantages sur le plan des performances n'étaient à priori pas réels en 2010 [Vandette, 2010]. Depuis, le runtime Android a beaucoup évolué. La machine virtuelle Dalvik a été remplacée par ART (Android Runtime) comme le montre [Android source, 2015]. Ce runtime permet, lors de l'installation d'une application, de compiler une partie du bytecode Dalvik en langage natif. C'est ce qui est appelé la compilation anticipée ou AOT (Ahead-of-time compilation).

---

[2] JNI (Java Native Interface) est un mécanisme permettant de faire appel à des fonctions de code natif à partir d'un programme Java.

La Figure 22 montre les étapes de compilation que subit un programme Java pour être exécuté sur une plateforme Android.

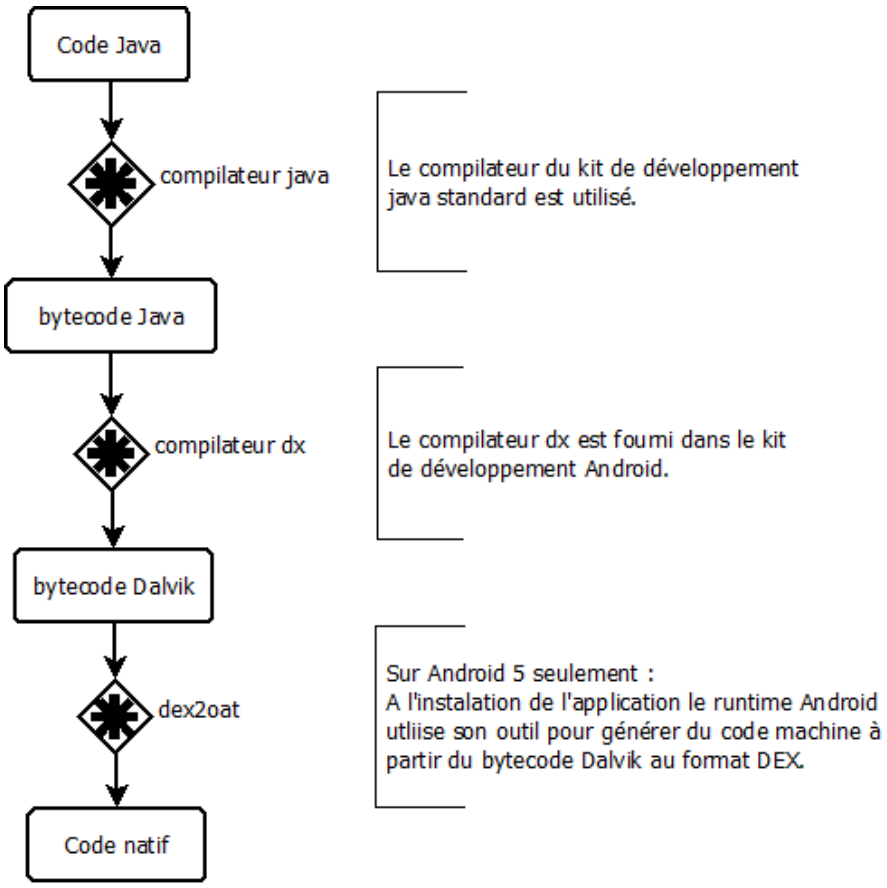


Figure 22 : Chaine de compilation d'une application

Android framework

Le framework Android est à destination des développeurs d'applications. Il fournit un ensemble de services pour simplifier les développements. C'est également lui qui fournit l'environnement d'exécution des applications. Les applications sont exécutées à l'intérieur de leur propre thread et obéissent au cycle de vie décrit par la Figure 23.

Le développeur doit s'adapter à ce cycle de vie, car il ne peut pas contrôler les évènements. Il est possible de sauvegarder et de restaurer l'état d'une application lorsqu'elle est détruite pour libérer de la place pour une application ayant besoin de mémoire.

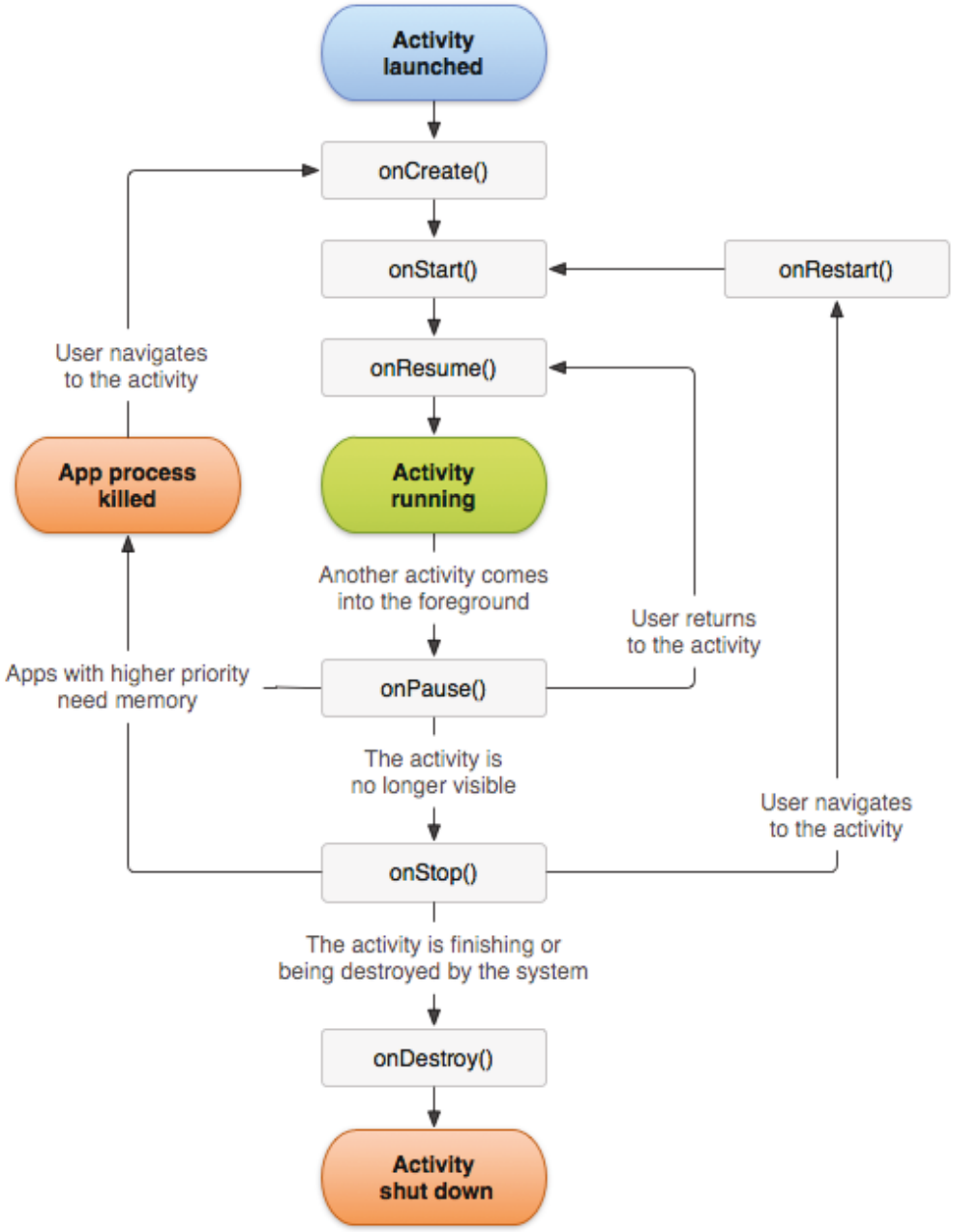


Figure 23 : Cycle de vie d'une application [Android developer, 2015]

Applications

Il s'agit de l'ensemble des applications de l'appareil. Il y a deux sortes d'applications :

- Les applications développées en Java, qui font potentiellement appel à leurs propres librairies natives par l'intermédiaire de JNI. Ce type d'application est recommandé par Google, car le système garde un contrôle sur elles.
- Les applications natives, développées uniquement en C/C++.

Les applications java ou native sont packagées sous forme de fichier apk. À l'instar des autres types de package java comme le jar, le war ou l'ear, un apk est une architecture de dossiers spécifique, zippée et contenant le code compilé et les ressources. Les applications possèdent un fichier de configuration « AndroidManifest.xml » qui permet de décrire des aspects techniques (quelle est la classe de base de l'application), des aspects cosmétiques (choix d'un thème de style) et des informations pour le déploiement telles que les versions du système Android supportées.

Android fournit un environnement de développement avec l'ensemble des outils nécessaires. Cet environnement de développement est disponible sur les trois principaux systèmes d'exploitation, Windows, Mac OS et Linux. Un outil permet de télécharger et installer différentes versions du SDK (Standard Development Kit) et du NDK (Native Development Kit) qui peuvent être utilisées en parallèle. Le développeur ayant plusieurs applications utilisant des versions différentes, peut ainsi faire la maintenance facilement. Le SDK contient les outils et librairies permettant de développer les applications Java pour Android. Le NDK est optionnel et sert à développer et compiler les parties natives des applications.

Distribution des applications

Les applications peuvent être distribuées de 3 manières différentes :

- par mail,
- par une mise à disposition sur un site web,
- par un marché en ligne comme Google Play.

Dans ces trois cas, les applications doivent tout de même remplir un certain nombre de critères. Je ne vais présenter que les plus importants, l'ensemble étant disponible sur [Android developer, 2015]. Tout d'abord les applications doivent être signées à l'aide d'un certificat fourni par Google. Le développeur crée lui-même le certificat avec les outils fournis. Une application ne peut être republiée pour mise à jour sur le Google Play Store qu'avec cette même clef. Un autre développeur ne peut donc pas modifier et publier à votre place votre application open source.

Un fichier de description de la configuration de l'application permet de déterminer les versions. Il y a tout d'abord la version de l'application elle-même que le développeur lui donne qui sert à évaluer si l'application est à jour. Il y a ensuite les versions minimum et maximum de l'API Android que l'application est capable de supporter. C'est au développeur

de correctement évaluer la version par rapport aux fonctionnalités qui sont utilisées. L'appareil bloque l'installation d'une application dont la version est non compatible avec son système.

Pour être publiée sur Google Play, l'application doit également fournir les icônes dans différentes résolutions afin d'être présentée correctement. Les pays de distribution, le prix, des captures d'écrans et autres vidéo promotionnelles peuvent également être fournis sur Google Play.

Afin d'avoir une certaine homogénéité dans la présentation et la qualité des applications, des séries de tests sont fournies par Google. Des tests spécifiques sont également disponibles suivant la nature de l'appareil, téléphone, tablette, etc. L'exécution de ces tests est à la discrétion du développeur, mais présente néanmoins l'avantage de pouvoir vérifier si les standards sont respectés.

2.2.4 – IOS

Initialement créé en 2007, le système iOS dérive de Mac OS X des ordinateurs Apple. Ce système a été développé pour les smartphones Apple, il a ensuite été décliné et adapté pour les autres appareils mobiles de la marque comme les tablettes, les lecteurs multimédia et récemment les montres connectées. Il s'appelait à l'origine iPhone OS et a été rebaptisé iOS en 2010. Le système iOS est un système propriétaire, mais le système sur lequel il est basé (Darwin) est open source. Ce système est principalement développé par Apple.

Bien que iOS soit en perte de vitesse par rapport à Android sur les téléphones selon [Gartner, 2014] avec 12.1 % des téléphones vendus au 3e trimestre 2014, il reste un des acteurs majeurs du marché. Android équipe également plus de tablettes vendues qu'iOS, mais la différence est moins flagrante que sur le marché du téléphone [Gartner, 2014].

Architecture

Étant basées sur Mac OS X, les mêmes couches logicielles sont trouvées dans le système iOS. Le noyau Darwin est également utilisé dans les deux systèmes d'exploitation. Ce noyau est une base de noyau Linux, donc écrit en C tout comme la majorité de la seconde couche. Les deux couches supérieures sont de leur côté, écrites en Objective C. Comme habituellement, la voie à privilégier, conseillée par Apple, est l'utilisation des librairies mises

à disposition au plus haut niveau, car elles cachent la complexité et permettent au système de garder un meilleur contrôle sur les applications. Le langage cible des applications iOS est donc le langage Objective C. [Apple Inc, 2014] donne le détail de l'architecture du système composé de quatre couches comme le montre la Figure 24.

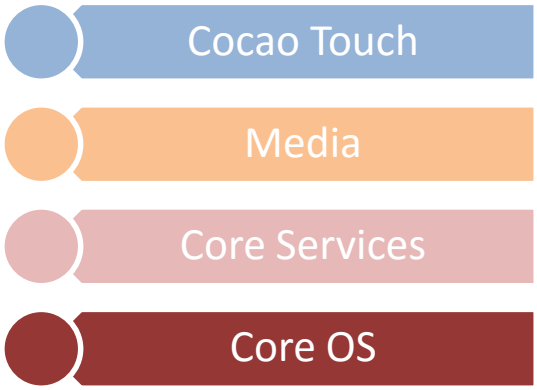


Figure 24 : Vue d'ensemble de l'architecture iOS [Apple Inc, 2014]

Les trois premières couches sont des abstractions du matériel et du noyau et ajoutent les propriétés graphiques au système. La dernière couche comprend le noyau ainsi qu'un ensemble de librairies, appelées framework dans le monde Apple, qui permettent sa manipulation.

Le cœur du système : le système d'exploitation (Core OS)

Cette couche est directement au contact du matériel. Elle comprend le système d'exploitation Darwin et un ensemble de librairies donnant les interfaces pour le pilotage de fonctionnalités spécifiques du matériel, comme le Bluetooth, ou lorsque l'appareil est connecté à des accessoires. Cette couche est moins destinée à être utilisée par les développeurs que par les couches supérieures.

Le système Darwin quant à lui, est un système d'exploitation sans interface graphique. Il est basé sur le noyau XNU. Ce noyau est lui-même le fruit du travail d'Apple qui a réalisé un noyau hybride mixant le micronoyau Mach et le noyau BSD. Cette approche a permis d'avoir une compatibilité avec le standard POSIX (interface portable de système d'exploitation qui est un standard d'interface de programmation système) par le noyau BSD. Les micronoyaux comme Mach apportent l'avantage d'une maintenance plus aisée et d'être plus facilement portable. La stabilité générale du système est également accrue. En effet, les services sont exécutés dans l'espace mémoire utilisateur et bénéficient d'une faible interdépendance.

Ainsi, la défaillance d'un service ne met pas en péril tout le système. Le revers de cette approche est que le système a des temps de latence élevés qui sont dus en partie à la communication des services à travers des interfaces trop lourdes à gérer, mais aussi au grand nombre d'appels système généré par les services en dehors du noyau. C'est ce qui a poussé Apple à construire son système d'exploitation sur une approche mixte.

Core Services

Cette couche fournit les services systèmes fondamentaux pour les applications. Les services fournis proposent des fonctionnalités et des niveaux très hétérogènes. Ainsi nous pouvons trouver des services réseau, de protection des données par cryptage, une API pour la programmation concurrente, le framework de gestion du modèle MVC (Modèle Vue Controleur) utilisé pour les interfaces graphiques et un ensemble de librairies de pur développement. D'un autre côté, des services beaucoup plus hauts niveaux sont également disponibles comme un service de stockage dans le cloud Apple (iCloud), accès aux comptes de réseaux sociaux, de publicité, de ventes d'application… Cette couche comprend également beaucoup d'encapsulation de fonctionnalités écrites en C/C++ dans des objets Objective C, qui est, rappelons-le, le langage cible pour le développement.

Media

La couche Media met à disposition l'ensemble des technologies graphiques, audio et vidéo qui peuvent être utilisées dans les applications multimédias. Une attention particulière est portée à la qualité des graphismes proposés par les applications. De nombreuses interfaces de programmation pour faire de la 2D ou de la 3D ainsi que des animations sont fournies. Des technologies de streaming audio et vidéo sont également incluses, mais sont spécifiques aux appareils Apple.

Cocoa Touch

Cette couche donne tous les services de haut niveau pour la programmation des applications. Elle permet de gérer facilement la présentation des applications en fournissant des objets de haut niveau et des moyens de personnalisation du style. Elle fournit également le framework MVC, illustré en Figure 25, qui permet de gérer l'affichage pour les applications voulant utiliser ces interfaces de haut niveau. Le détail du framework MVC est donné par [Apple Inc, 2014].

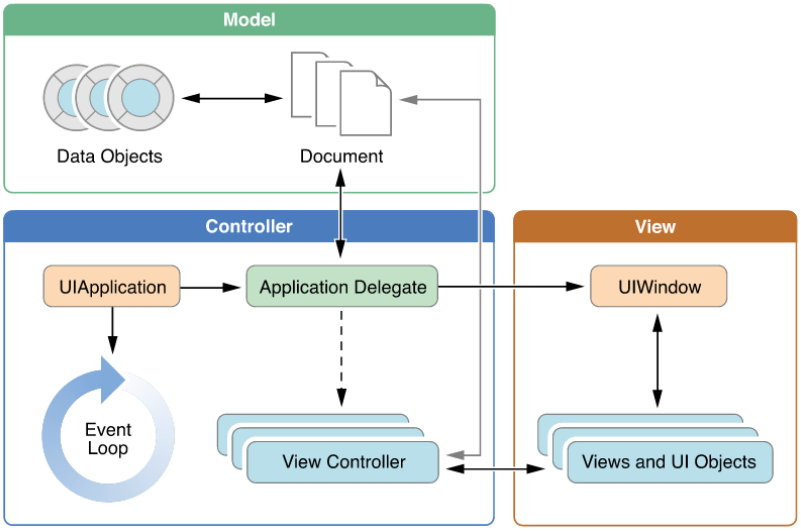


Figure 25 : Structure MVC intégré à iOS [Apple Inc, 2014]

La reconnaissance des gestes est incluse dans cette couche. Des gestes standards réalisés par les utilisateurs avec un ou plusieurs doigts peuvent être détectés et identifiés facilement. Il est ainsi simple d'implémenter une fonction de zoom par exemple.

Tout comme avec Android, les applications passent par différents états durant leur vie, suivant les actions de l'utilisateur, l'ouverture d'autres applications ou la mise en veille de l'appareil. Cet enchainement d'états doit être pris en compte lors de la réalisation de l'application. Le cycle de vie d'une application est donné par la Figure 26.

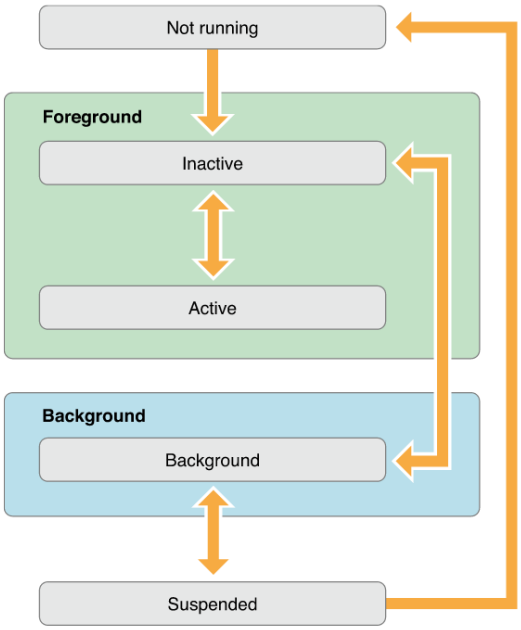


Figure 26 : Cycle de vie d'une application iOS [Apple Inc, 2014]

De nombreuses fonctionnalités des couches précédentes peuvent être retrouvées, mais sont présentées de manière simplifiée. Par exemple, une API de programmation d'applications concurrentes est proposée. Le programmeur aura donc tout intérêt à parcourir les différentes couches dans l'ordre du plus haut niveau au plus bas niveau pour trouver l'interface de programmation qui correspond le mieux à ses besoins.

Applications

Les applications sont développées en utilisant le SDK. Les outils de développement en eux-mêmes sont gratuits, mais il faut être inscrit et identifié comme développeur iOS pour pouvoir installer et tester ses applications sur un appareil iOS. Ces outils de développement sont uniquement disponibles sur Mac OS X. Une autre restriction est la version de système utilisé. Il n'est pas possible d'installer une version récente des outils de développement sur une version ancienne de Mac OS. Il faut donc avoir une certaine correspondance entre la version du SDK utilisé et la version de l'ordinateur utilisé pour le développement.

Les applications ont un format bien spécifique et sont packagées en un fichier avec l'extension « .app ». Un fichier de description de l'application doit être présent. Il contient des informations essentielles sur la configuration de l'application. Parmi ces informations, les besoins minimums de l'application en termes d'autorisations d'accès au matériel, mais aussi d'accès à certaines données logiciels doivent être décrits. L'ensemble des icônes utiles aussi bien à l'installation qu'à la publication sur le marché d'application (voir section suivante) est également renseigné.

Les applications n'ont pas de possibilités de manipuler des fichiers à l'intérieur du système. Chaque application a un endroit réservé pour la gestion de ses fichiers. Cette zone peut également être une zone de partage des fichiers qui pourront être échangés avec l'ordinateur par l'intermédiaire du logiciel dédié iTunes. Cette limitation permet de contourner les problèmes de sécurité.

Distribution des applications

Les applications peuvent être installées par le marché d'application Apple officiel, l'App Store, ou en la distribuant manuellement comme nous l'explique [Apple Inc, 2015]. Dans les deux cas, l'application doit avoir été signée avec un certificat fourni par Apple lors de l'enregistrement comme développeur iOS. Cette inscription, payante, permet non seulement

de tester ses applications en les installant sur des appareils, mais également d'avoir le certificat pour signer les applications et les soumettre sur le marché d'applications.

La distribution sur l'App Store se fait en plusieurs étapes. Les applications sont tout d'abord validées par Apple avant d'être distribuées. Une étape de bêta test peut être décidée par le développeur pour prendre en compte les retours des premiers utilisateurs avant la distribution officielle.

Les buts et enjeux de la migration sur chaque plateforme sont maintenant explicités. Nous avons pu voir que ces différentes migrations mènent à de nouvelles utilisations. Les cas d'utilisation vu précédemment entrainent de nouveaux besoins pour pouvoir être réalisés. Il ne s'agit donc pas uniquement de faire fonctionner les logiciels sur les plateformes, mais également d'apporter les modifications nécessaires pour qu'ils soient pleinement intégrés à l'environnement. Les technologies des plateformes cibles sont maintenant connues et nous avons un aperçu des possibilités de chacune, de leurs forces et leurs faiblesses. Nous allons maintenant voir les travaux réalisés sur le moteur Guido puis sur INScore.

Chapitre 3

Travaux réalisés sur le moteur Guido

Le travail sur le moteur Guido est préalable à celui sur l'application INScore. Aussi, les différentes interfaces de Guido avaient besoin d'être mises à jour. Bien qu'inutile pour INScore, le portage du moteur en une librairie JavaScript à l'aide d'Emscripten s'inscrit parfaitement dans la démarche d'ouverture des applications de Grame. Avoir le moteur sous forme de librairie JavaScript permet également de travailler sur la facilité d'intégration en créant des composants web. Une version Service Web de Guido existait. Elle n'est pas non plus nécessaire à INScore, mais nous verrons les modifications qui ont été apportées. Enfin, les versions du moteur pour iOS et Android sont présentées.

3.1 – JavaScript

Le portage du moteur Guido sous forme de librairie JavaScript est réalisé à l'aide d'Emscripten. Bien qu'Emscripten permette de faire la compilation de code C/C++ vers une librairie JavaScript, le moteur nécessite quelques adaptations de code pour être opérationnel. Différentes méthodes de dessin de la partition ont été également développées pour créer une image SVG ou dessiner dans un élément canvas d'HTML5. La question de performance est un des points d'interrogation de ce portage. Différentes méthodes de rendu graphique ont été testées et mesurées. Enfin, l'intégration dans une page HTML des différents composants nécessaires au rendu de partition peut être compliquée. C'est pourquoi un exemple de l'utilisation de composant web dans une page HTML 5 est présenté et mis à disposition des utilisateurs de la libraire.

3.1.1 – La création d'une API JavaScript

L'ensemble de l'API du moteur a été porté en JavaScript. L'API du moteur Guido est définie en C. Emscripten permet de faire une correspondance entre les fonctions C et les fonctions JavaScript par une déclaration de celles-ci dans le fichier de compilation. Une seconde solution est d'utiliser le module Embind qui permet d'utiliser des objets C++ en JavaScript. Ce module permet également de déclarer les structures C et de les utiliser en JavaScript. La conversion entre JavaScript et C++ est assurée par le module. L'utilisation de ce module permet donc de faire des passages d'arguments complexes aux méthodes exposées en JavaScript. Les déclarations pour faire la correspondance de nom entre C++ et JavaScript sont faites dans un fichier source C/C++.

C'est la deuxième solution qui a été choisie pour plusieurs raisons, et ce malgré l'obligation d'avoir l'API Guido en C++.

L'avantage de C++ sur C

La question de la création d'une API C++ plutôt que C pour le moteur Guido s'était déjà posée, mais elle constitue une interface de plus à maintenir pour les membres de Grame. Les apports de C++ sont tout de même intéressants notamment par la possibilité d'utiliser la surcharge de fonction et les arguments par défaut. L'API s'en trouve alors simplifiée. Le Tableau 1 donne un exemple de simplification par la surcharge alors que le Tableau 2 donne une simplification grâce à l'utilisation d'arguments par défaut :

`GuidoErrCode abstractExport(`
`  const GRHandler handle,`
`  int page,`
`  std::ostream& out);`
	Ces deux méthodes de la classe C++ ont le même nom, mais des arguments différents et une variable de retour différente.

`std::string abstractExport(`
`  const GRHandler handle,`
`  int page);`	

Tableau 1 : Surcharge de méthode en C++

GuidoErrCode **gr2SVG**(const GRHandler handle, int page, std::ostream& out, bool embedFont = true, const char* font = 0, const int mappingMode = 0);	Les trois derniers paramètres sont facultatifs. Le développeur ne les utilise que s'il ne veut pas utiliser les valeurs par défaut.

Tableau 2 : Arguments par défaut en C++

De plus, l'API Guido ne peut pas être modifiée pour garder une rétrocompatibilité du code des applications l'utilisant. Elle peut donc juste être étendue. Au paragraphe suivant, nous allons voir que l'utilisation des effets de bord est impossible avec Emscripten. Comme la surcharge de fonction n'existe pas en C, c'est un ensemble de fonctions qui aurait dû être ajouté avec des noms différents de la fonction d'origine. L'API du moteur aurait été illisible.

Un wrapper C++ a donc été créé. L'API Guido est composée de plusieurs modules. La solution choisie a alors été de créer une classe wrapper pour chaque module.

Les variables de retour par effet de bord

L'API C de Guido donne souvent les valeurs de retour par effet de bord. C'est un code d'erreur qui prend la place de la variable de retour. Ce code d'erreur n'est que très peu utilisé, nous pouvons donc nous en affranchir. L'API a donc été transformée sur le modèle du Tableau 3 (la valeur de retour est en rouge) :

Code C	Code C++
GuidoErrCode **GuidoGetPageDate**(CGRHandler inHandleGR, int pageNum, GuidoDate* date);	GuidoDate **getPageDate**(CGRHandler inHandleGR, int pageNum);

Tableau 3 : Modification des valeurs de retour (en rouge)

En JavaScript, Emscripten représente les pointeurs C/C++ comme des nombres entiers. Ces variables ne peuvent pas être exploitées en tant que telles, mais peuvent être utilisées pour être passées d'une fonction à l'autre comme des pointeurs opaques. En revanche, le pointeur sur la variable GuidoDate de la fonction C n'est pas exploitable, car il n'est pas possible de créer un pointeur sur un objet JavaScript.

Avec l'utilisation du module Embind et du wrapper C++, la structure GuidoDate peut être retournée par la méthode C++ et converti en objet JavaScript. Elle peut ensuite être utilisée en JavaScript comme tous les autres objets.

Des possibilités étendues

Nous venons de voir qu'Embind permet d'avoir en retour des méthodes des objets de la librairie manipulables côté JavaScript. Il est également possible de les créer et de les passer en arguments aux méthodes. L'utilisation de ce module nous permet d'avoir un contrôle total sur le moteur tout en gardant une bonne lisibilité puisque nous gardons les mêmes structures de paramètres qu'avec l'API C.

Plus généralement, les pointeurs, les tableaux et les chaines de caractères de style C sont problématiques à gérer en tant que paramètre de méthode avec Emscripten sans le module Embind. Les chaines de caractères par exemple, obligent le développeur qui utilise la librairie JavaScript à utiliser des fonctions spécifiques d'Emscripten pour finir le décodage et utiliser la chaine de caractères normalement. Embind permet de cacher cette complexité.

La technologie Embind permet également d'utiliser une partie de la STL (Standard Template Library : bibliothèque standard C++) en faisant des conversions automatiques.

3.1.2 – Le problème des polices de caractères

Nous l'avons vu lors de la présentation du moteur Guido, une police de caractères spécifique est utilisée pour représenter les symboles musicaux. Deux problèmes se posent alors en JavaScript :

- le calcul des métriques de la police de caractères,
- l'intégration de la police de caractères lors de la création d'une image SVG.

Le moteur calcule les métriques de la police pour définir l'espacement entre les symboles. Ce sont des primitives systèmes dépendantes du système hôte qui permettent le calcul des métriques de la police. En JavaScript, il n'y a pas de primitives permettant de faire ces calculs.

La bibliothèque NanoSVG a été testée à cette fin, mais les métriques calculées sont trop imprécises. Les métriques des polices ont donc été embarquées. Elles ont été calculées avec

un outil précis et sont utilisées directement par une classe dédiée à l'utilisation de la police musicale.

Le même problème existe pour la police texte. Les métriques de la police la plus utilisée dans Guido (Times) sont embarquées et utilisées, quelle que soit la police texte utilisée. Cette solution peut donc créer quelques décalages en cas d'utilisation de police texte exotique. Quasi idéal pour la police musicale qui n'est pas destinée à changer et évite du calcul, l'embarquement des métriques l'est moins avec le texte.

L'intégration de la police de caractères dans une image SVG permet de ne pas avoir à installer une police spécifique sur le système pour afficher correctement l'image. Le moteur embarque donc la police Guido au format SVG pour pouvoir l'intégrer lors de la génération de l'image.

3.1.3 – Les devices graphiques

Les devices graphiques sont utilisés par le moteur pour faire le rendu de la partition. Les implémentations des devices graphiques dépendent du système hôte ou plus généralement du système graphique utilisé. L'environnement JavaScript a donc nécessité le développement d'un nouveau device graphique.

En JavaScript, trois méthodes différentes peuvent être utilisées pour afficher les partitions :

- Export SVG et affichage de l'image résultante dans le navigateur.
- Export binaire et parcours de cet export par une implémentation de device JavaScript exécuté par le navigateur qui dessine dans un élément canvas HTML5. Cette implémentation n'est donc pas embarquée dans la librairie générée par Emscripten.
- Implémentation du dessin dans un canvas HTML5 directement dans la libraire Guido générée par Emscripten. Pour ce faire, Emscripten fournit des moyens d'intégrer du JavaScript à l'intérieur du code C/C++ [Emscripten Contributors, 2014]. Une macro C est utilisée (EM_ASM) pour pouvoir intégrer ce code JavaScript.

La première méthode a le mérite d'être relativement simple et d'être déjà éprouvée. Les exports SVG existent déjà depuis plusieurs versions du moteur. L'utilisation d'un canvas HTML5 est en revanche nouvelle. Elle permet un meilleur contrôle sur la partition et permet d'ajouter à posteriori des objets dans la partition. La deuxième version qui

embarque le code JavaScript dans la librairie évite le parcours de l'export binaire et a donc été créée pour des questions de performance.

3.1.4 – Exécution et performances

La question des performances et de l'utilisabilité de la librairie en JavaScript a été un des points d'interrogation du portage. Des mesures ont été faites avec les différentes options. Le moteur permet de tracer les temps d'exécution des différentes étapes de création de la partition, de la conversion code guido au rendu graphique. Deux partitions sont principalement utilisées pour faire ces tests :

- Einspeilung I de Emmanuel Nunes, pour Violon et électronique. C'est une partition d'une seule ligne très longue (fichier de 45 ko)
- Symphonia n°3 BWV 789 de Jean Sébastien Bach, partition pour piano avec une mise en page précise et de nombreuses indications (fichier de 12 ko)

La performance dépend fortement des options de compilation utilisées. Il y a plusieurs niveaux d'optimisation du code compilé, certaines permettant de réduire la taille du code. Emscripten désactive également par défaut les exceptions C++ pour des questions de performances.

Nous avons vu qu'Emscripten utilise un tableau ArrayBuffer comme mémoire virtuelle. Cette taille de mémoire peut être fixe ou variable. Lorsqu'elle est fixe, Emscripten peut faire beaucoup d'optimisation, mais si le programme demande plus de mémoire que celle disponible, l'exécution entre en erreur. Une option permet de déterminer si cette taille est fixe ou variable. Nous avons choisi d'utiliser une taille fixe, car la différence de performance est très importante entre une taille fixe et variable (facteur de 5 à 6 sur Einspeilung I et un navigateur Firefox) comme nous pouvons le voir dans le Tableau 4.

	Mémoire dynamique	**Mémoire fixe**
Génération de la représentation abstraite	1564 ms	206 ms
Génération de la représentation graphique	1244 ms	235 ms
Génération du SVG	1130 ms	327 ms
Total	**3938 ms**	**768 ms**

Tableau 4 : Différence de performance entre mémoire fixe et dynamique

La façon d'afficher la partition entraine également des changements de performance. Très logiquement, l'export binaire puis le parcours en JavaScript de cet export est la méthode la plus lente des trois. Les deux autres devices utilisant pour l'un l'export SVG et pour l'autre le device canvas HTML5 intégré à la librairie, ont des temps d'exécution pratiquement équivalents.

Bien sûr, un dernier facteur déterminant est le navigateur. Le compilateur Emscripten produit du code asm.js. Les navigateurs n'ayant pas un moteur JavaScript optimisé asm.js sont pénalisés. Les navigateurs ont tendance à accélérer le traitement après plusieurs affichages. Les mesures ne sont donc pas faites sur un premier chargement de partition.

La comparaison des navigateurs en utilisant l'export binaire puis le parcours en JavaScript pour le dessiner dans un canvas donne des résultats assez similaires avec les différents navigateurs sur la partition la plus grosse, mais montre des différences sur la partition la plus petite comme nous pouvons le voir dans le Tableau 5:

	Internet Explorer	**Firefox**	**Chrome**
Einspielung I	7.54 s	8.75 s	8.77 s
Synphonia n°3	4.04 s	2.04 s	1.31 s

Tableau 5 : Comparaison des navigateurs avec l'export binaire

Quant à Firefox, il fait moins bon usage du device canvas intégré à la libraire que ces concurrents. Globalement, le navigateur Chrome a eu les meilleurs résultats (Tableau 6).

	Internet Explorer	**Firefox**	**Chrome**
Einspielung I	6.72 s	7.59 s	3.97 s
Synphonia n°3	1.48 s	2.07 s	0.68 s

Tableau 6 : Comparaison des navigateurs avec le device canvas intégré

Le navigateur Internet Explorer a des performances en deçà de ces concurrents Firefox et Chrome avec l'utilisation des canvas HTML5, mais elles restent tout à fait convenables. La faiblesse d'Internet Explorer est l'affichage des images SVG comme nous pouvons le voir dans le Tableau 7.

	Internet Explorer	Firefox	Chrome
Einspielung I	**55.92**	**2.741**	**3.67**
Synphonia n°3	**4.04**	**0.242**	**0.508**

Tableau 7 : Comparaison des navigateurs en utilisant l'export SVG

À titre de comparaison, les images SVG des mêmes partitions ont été générées avec un outil C++. Nous pouvons voir sur le Tableau 8 que la librairie JavaScript est relativement efficace sur la plus imposante des deux partitions, mais l'est moins pour la plus petite. Cette caractéristique a également été constatée par l'équipe de recherche sur le portage d'applications audio. Les plus grosses applications ont également un temps d'exécution environ trois fois plus long en JavaScript.

	Outil C++
Einspielung I	1.138
Synphonia n°3	0.055

Tableau 8 : Temps de génération d'une image SVG en C++

Ces comportements sont bien sûr sujets à modification suivant les évolutions des navigateurs, mais permettent tout de même de noter qu'il faut faire une étude préalable de la méthode d'affichage et des navigateurs cibles ou du moteur d'exécution cible lors du développement d'une application avec cette librairie.

3.1.5 – Intégration dans une page HTML

Des exemples sont fournis aux utilisateurs de la librairie pour leur faciliter la prise en main du moteur. Une page web utilise une nouveauté d'HTML5, les Web Components ou Composants Web. Je vais maintenant présenter cette nouveauté et le travail qui a été fait.

Les composants web

Les Composants Web sont composés de 4 parties d'HTML5 faites pour être utilisées ensembles comme nous l'explique [Cooney, 2013]. Ces 4 parties sont des standards émergeants implémentés sur la plupart des navigateurs modernes. Elles peuvent être utilisées ensemble ou séparément. Ces 4 parties sont :

- les templates, qui permettent de créer un ensemble d'éléments qui pourront être réutilisés à volonté,
- le shadow DOM, qui permet d'encapsuler dans un élément racine du HTML, des styles CSS, du JavaScript ou des templates. Le DOM (Document Object Model) est l'arbre que représente le code HTML. Le principe mis en place ici est l'encapsulation dans une feuille de l'arbre (un élément HTML) d'un autre arbre (le shadow DOM),
- la création de nouveaux éléments, qui permet de déclarer de nouveaux éléments HTML utilisables directement dans la page web,
- le packaging, qui permet de distribuer les composants web en leur donnant un type mime spécifique (`application/package`) pour que les navigateurs puissent les interpréter. Cette partie n'est pas utilisée dans nos exemples, notre but étant de montrer comment encapsuler la mécanique du moteur dans un composant, mais pas de distribuer un composant finalisé.

L'intégration du moteur dans un composant

Le but de ce composant est de pouvoir intégrer en quelques lignes de code une partition dans sa page HTML. Le composant doit donc être dans un fichier externe à la page. Ce fichier contient un template HTML5.

Ce template contient :

- Le code JavaScript pour piloter le moteur
- L'enregistrement d'un nouvel élément HTML appelé « guido-viewer » qui est la racine du shadow DOM.
- Le code HTML et CSS du composant pour sa présentation.

Le template doit tout d'abord être importé dans la page HTML. L'élément peut ensuite être utilisé comme n'importe quel autre élément HTML standard. Le code du template est affiché à la position de l'élément. Les éléments peuvent posséder leurs propres attributs. L'élément guido-viewer utilise un attribut « `gmnCode` » pour afficher ou non un éditeur de code Guido à gauche de la partition. S'il n'y a pas d'éditeur de code Guido (comme sur l'exemple ci-dessous), le code doit être inséré dans le corps de l'élément. Lorsque l'éditeur est présent, le code saisi dans l'éditeur est transformé à la volée en représentation graphique de la partition tel qu'illustré en Figure 27.

```
<html>
    <head>
        …
        <link rel="import" href="template/guidoTemplate.html" />
    </head>
<body>
…
    <guido-viewer gmnCode="false">[d e f/8 a d b c h]</guido-viewer>
…
</body>
</html>
```

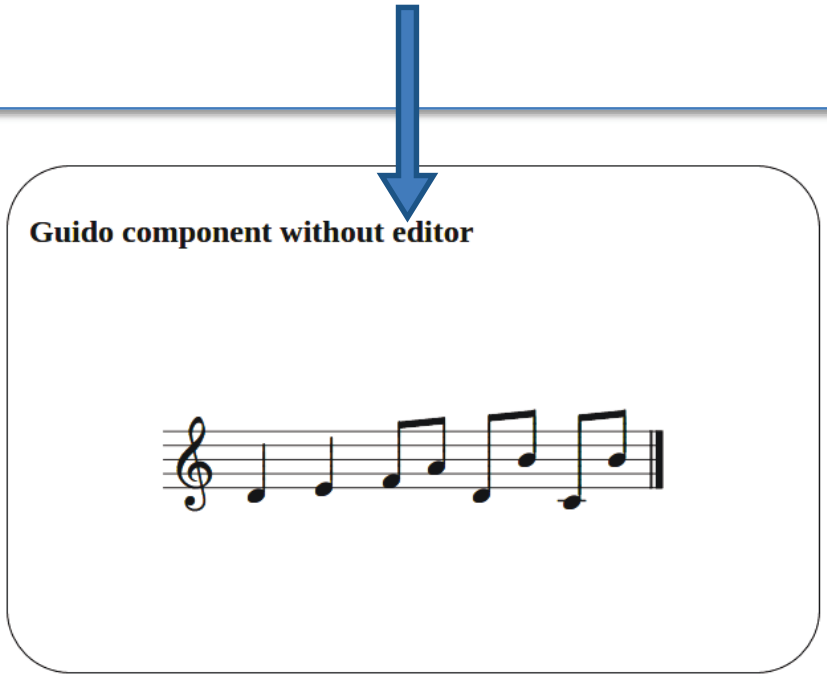


Figure 27 : Code et rendu graphique du composant web guido-viewer

Le code interne des composants utilisés dans une page est isolé du reste de la page. Il est encapsulé à l'intérieur de son élément racine et invisible pour l'extérieur (d'où son nom de shadow DOM). Il n'y a donc pas de risque d'avoir des conflits de nommage des éléments (un identifiant d'élément HTML doit être unique dans une page) ni d'avoir d'influence des styles CSS de la page sur le composant utilisé.

3.1.6 – Conclusion

Le fonctionnement de la version du moteur Guido en JavaScript correspond à nos attentes. Les différences de performance constatées entre les navigateurs sont moins importantes que ce que nous pensions.

Pour compléter ce portage et améliorer la consistance du moteur, quelques modifications pourraient encore être apportées. L'export binaire devrait toujours utiliser le même format. En effet, la représentation des nombres dépend de la machine sur laquelle s'exécute le

moteur (représentation LITTLE ou BIG ENDIAN). L'uniformisation dans l'un des deux formats permettrait l'échange des exports sans risquer de problèmes. La seconde modification porterait sur l'uniformisation de l'utilisation des polices textes dans le moteur. Une police texte par défaut est censée être utilisée, mais ce n'est pas toujours le cas. Cela permettrait de minimiser les potentiels problèmes avec les métriques de police fixes.

Les composants web sont très pratiques pour s'abstraire de la complexité. Leur intégration est très simple. Ils vont permettre de construire des pages web riches avec beaucoup plus de facilité que ce que nous proposaient les bibliothèques de composants JavaScript qui restaient toujours délicates à intégrer dans une page.

3.2 – Service Web

Le service web Guido existait déjà avant ma venue au Grame. L'ensemble des fonctionnalités n'était pas présent et un remaniement du code était nécessaire. Le serveur utilisait Libmicrohttpd et ce choix a été conservé. Ce serveur HTTP est tout d'abord présenté. Nous allons voir ensuite les modifications qui ont été apportées au service, qui pour la plupart ont été faites pour respecter les bonnes pratiques REST. Le dernier sujet abordé est la réentrance des algorithmes, propriété manquante au moteur Guido pour l'utiliser dans le cadre d'un serveur web sollicité par un grand nombre de clients.

3.2.1 – Serveur Web C/C++

Afin d'intégrer facilement le moteur dans un service web, le serveur Libmicrohttpd est utilisé. Libmicrohhtpd fait partie du projet GNU, un système d'exploitation constitué de logiciels libres. Ce serveur est présenté sous la forme d'une librairie C open source. Cette librairie, très petite (entre 100 et 150 ko suivant les versions), supporte tout de même de nombreuses fonctionnalités comme nous pouvons le lire sur la documentation [Grothoff, 2015].

La configuration du serveur permet de choisir son comportement pour le traitement des requêtes. Elles peuvent être traitées en parallèle avec un nombre configurable de traitements parallèles, ou en file d'attente avec une unique boucle de traitement.

Le code à produire pour le développement d'un serveur est d'assez bas niveau. Le développement d'une surcouche peut être une bonne approche si le serveur Libmicrohttpd est destiné à être utilisé dans plusieurs projets. Un autre point qui pourrait être amélioré par une surcouche est l'extension des fonctionnalités du serveur. Si les requêtes traitées sont amenées à évoluer régulièrement, un système d'ajout de méthodes de traitement et de configuration pourrait être développé. Ce système permet de séparer une partie de la logique métier du code de gestion et de fonctionnement serveur.

Ce système, illustré Figure 28, est assez simple à mettre en place, car il suffit :

- De créer un fichier de configuration qui associe une URL et une méthode HTTP à une méthode C++.
- De créer le dispositif d'appel de méthodes en fonction de la requête. L'utilisation de pointeur de fonctions parait tout à fait appropriée.
- De créer les méthodes de traitement de requête dans un nouveau fichier, à part du code source de gestion du serveur

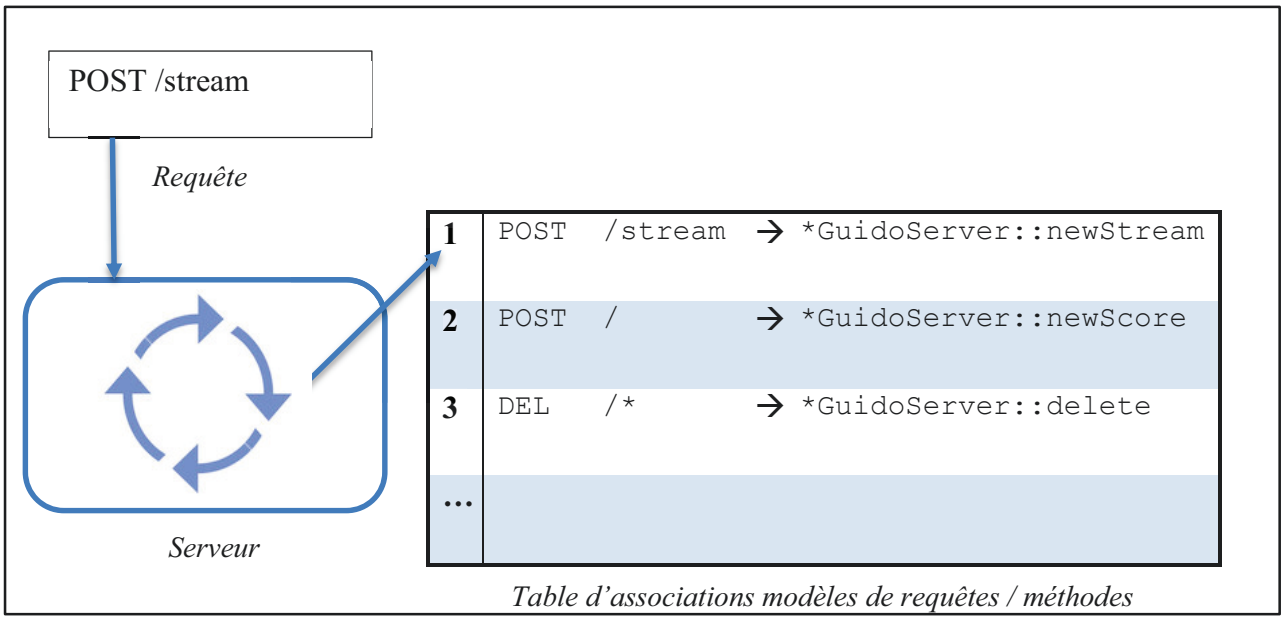


Figure 28 : Dispositif générique de traitement des requêtes

3.2.2 – Un Service Web REST

Le service web Guido est un service REST. Le serveur peut être installé sur des machines Mac OS X et Linux. La librairie étant un projet GNU, il n'est pas possible d'utiliser le compilateur Microsoft sans faire des adaptations. Le serveur n'est donc pas disponible sur Windows.

Ce service permet de créer une partition à partir de code Guido envoyé. La représentation graphique de la partition peut être récupérée dans différents formats d'image. Étant proches de l'API C de Guido, les informations sur la partition, comme le nombre de pages ou les mapping, peuvent également être récupérées par l'intermédiaire du Service Web dans des objets JSON.

Création d'une partition

La création d'une partition se fait par une requête HTTP et la méthode POST. Une session est alors créée avec le code Guido et la représentation abstraite de la partition. Ce choix a été fait pour éviter de renvoyer le code Guido et de recréer les représentations abstraites et graphiques qui prennent du temps de traitement. C'est l'identifiant de session qui est ensuite utilisé pour identifier la partition. Elle n'appartient pas forcément à un seul utilisateur, car cet identifiant peut être partagé entre plusieurs utilisateurs. Elle est pour l'instant stockée dans un tampon gardant les 100 dernières sessions et n'a pas de date d'expiration. Un système de cache sur disque est également disponible pour pouvoir recharger les partitions en cas d'arrêt du serveur.

La partition peut être vue comme une ressource telle que nous l'avons vu au paragraphe 2.2.1 sur les Service Web REST. Elle est créée avec la méthode POST, c'est bien le serveur qui en choisit l'identifiant. La réponse est un objet JSON avec l'identifiant de la partition et un code de retour HTTP 201 (« Created ») si la partition a pu être créée. Un code d'erreur 400 (« Error ») est renvoyé si la partition n'a pas pu être créée à partir du code Guido envoyé.

L'API « stream » de Guido permettant de créer une partition en ajoutant du code Guido au fur et à mesure est également disponible. Le Tableau 9 donne le détail de cette API :

Fonction	**URI**	**corps de requête**
Création	http://serviceGuido/stream	data=code Guido
Ajout	http://serviceGuido/stream/identifiant	data=code Guido
Initialisation	http://serviceGuido/stream/identifiant/reset	

Tableau 9 : Service REST Guido pour l'API "stream"

Cette API n'est pas strictement une API REST telle que [Richardson, et al., 2007] peuvent la décrire. La partition est identifiée par "`identifiant`" et peut être accédée en lecture comme toutes les autres partitions par l'URI "`http://serviceGuido/identifiant`" alors que la modification utilise l'URI "`http://serviceGuido/stream/identifiant`". Il y a donc deux URI pour accéder à la même ressource. Ce choix a été fait consciemment pour différencier la manipulation des partitions « stream » des autres. Cette différentiation ne nous parait pas être justifiée pour l'accès en lecture.

Récupération d'informations sur la partition

La récupération d'informations se fait par la méthode GET. Les options de mise en page peuvent être ajoutées en les passant en paramètres de requête. La représentation graphique de la partition doit alors être calculée. Cela va à l'encontre du principe de sûreté des requêtes GET et HEAD vu précédemment, mais en ne stockant pas la représentation graphique et ses options, les requêtes restent idempotentes.

La récupération de l'image d'une partition se fait avec une requête de la forme :

```
http://serviceGuido/identifiant/?option1=XXX&option2=XXX...
```

Pour récupérer une information particulière sur une partition, par exemple sa durée, la requête est complétée avec l'information demandée :

```
http://serviceGuido/identifiant/duration/?option1=XXX&option2=XXX...
```

Cette seconde forme de requête ressemble beaucoup à un service d'appel de procédure distante, mais elle nous a paru être la plus consistante pour ce type de requête.

La réponse de cette seconde forme est un objet JavaScript JSON. Les réponses utilisent les codes de retour HTTP, 200 (« Success ») en cas de réussite, 404 (« Not Found ») si l'identifiant n'a pas été trouvé, ou encore 400 (« Error ») si la requête est incorrecte.

Suppression d'une partition

La suppression d'une partition se fait logiquement avec la méthode HTTP DELETE et l'identifiant de la partition. La réponse est alors 200 (« success ») ou 404 (« not found »).

Conclusion

Le service web créé est assez consistant. Malgré quelques libertés, il respecte globalement les bonnes pratiques des services REST et propose une API intuitive et simple à utiliser. Le système de stockage des partitions devra certainement être revu suivant le type d'utilisation finale qu'il sera fait du service, mais fonctionne parfaitement dans un premier temps avec le nombre d'utilisateurs du service.

3.2.3 – Moteur réentrant

Dans un véritable contexte web, c'est-à-dire avec une possibilité de montée en charge lorsque le nombre de clients augmente, les serveurs utilisent des threads pour pouvoir traiter les requêtes en parallèle. Nous avons vu lors de la présentation de la librairie que Libmicrohttpd permet d'avoir ce comportement. Le moteur Guido quant à lui ne le permet pas.

Une fonction ou un algorithme pouvant être utilisé simultanément par plusieurs tâches est dit réentrant. Pour se faire, il ne doit pas y avoir de modifications de l'état interne global du programme pendant l'exécution de l'algorithme. Les seules modifications possibles sont à l'intérieur du contexte d'exécution de l'algorithme.

Le moteur Guido utilise des variables globales pour la gestion de paramètres de mise en page. Deux requêtes utilisant des mises en page différentes ne peuvent donc pas être traitées en parallèle. Des modifications ont été apportées pour fournir une API qui permette de créer une partition en fournissant l'ensemble des paramètres. Les extraits de code Figure 29 montrent la différence entre l'ancien code qui modifie le comportement général du moteur et le nouveau code qui n'a d'influence que sur la représentation graphique créée.

La première partie du travail pour rendre le moteur réentrant est terminé et l'interface de programmation est maintenant compatible avec la réentrance. Il reste tout de même plusieurs variables globales à intégrer dans les classes C++ pour que le moteur soit véritablement réentrant. Les options du serveur HTTP pourront ensuite être modifiées pour permettre de traiter les requêtes en parallèle.

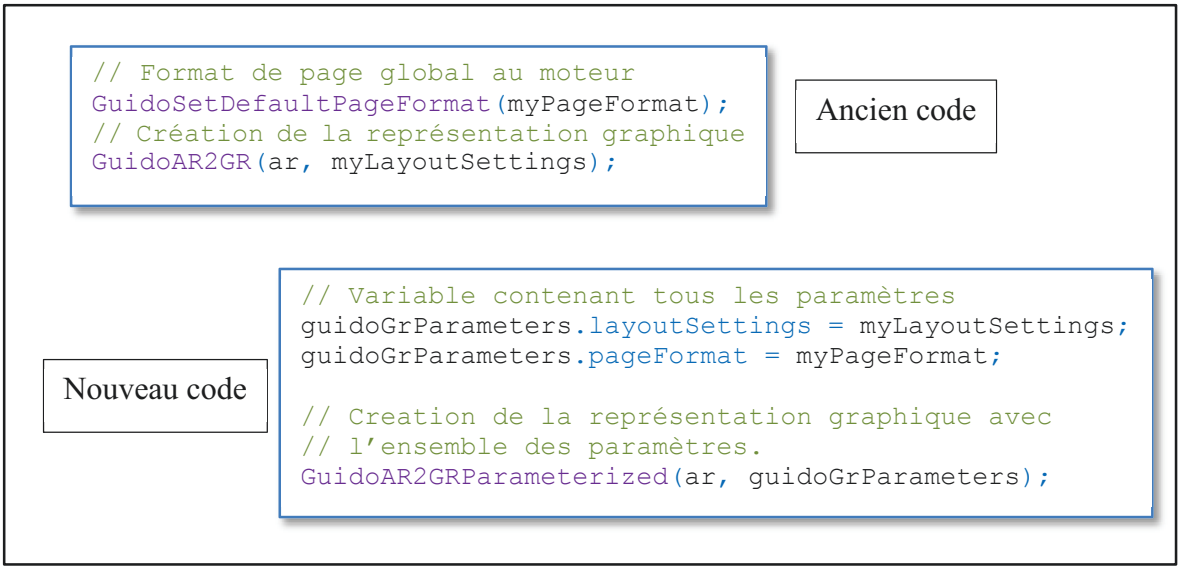


Figure 29 : Modification de l'API pour la réentrance du moteur Guido

3.3 – Interfaçage avec Java

Le moteur Guido est utilisable depuis un programme Java grâce à JNI (Java Native Interface) qui permet de faire un pont entre ces deux mondes. Les interfaces JNI ont un regain d'intérêt, notamment par l'utilité qu'elles peuvent avoir dans le développement Android. Pour utiliser du code natif avec Android, il est conseillé de développer des librairies et de les utiliser avec JNI plutôt que de développer une application entièrement native. Pour Guido, l'interface JNI était déjà existante. Le travail sur cette partie est donc une maintenance évolutive pour compléter l'existant avec les nouveautés et prendre en compte les modifications. Il ne peut pas y avoir de modification de structure en profondeur pour assurer une compatibilité descendante entre les versions. Nous allons voir tout d'abord ce qu'est JNI et quels sont ses concurrents. Nous allons ensuite voir ce qui a été réalisé puis quelques principes d'améliorations qui pourraient être mis en œuvre dans le cadre d'une refonte de l'interface JNI.

3.3.1 – JNI

La technologie JNI permet de faire appel à du code natif C/C++ directement depuis un programme en langage Java. L'appel à ce code natif est rendu très simple par l'utilisation du

mot clef « native ». Il faut également que la libraire native qui contient ce code ait été chargée par l'application java.

```
public native int findPageAt(guidodate date);
```

La méthode « findPageAt » peut être utilisée comme n'importe quelle autre méthode de la classe Java. L'utilisation de JNI est donc simple, mais la création de la colle entre JNI et le code natif est plus compliquée.

La suite des opérations à effectuer pour exposer l'API d'une librairie native avec JNI est illustrée Figure 30. Il faut commencer par créer les classes Java. Ensuite, l'outil « javah » permet de générer les fichiers de déclaration de fonctions de code natif qui seront appelées par les méthodes native Java.

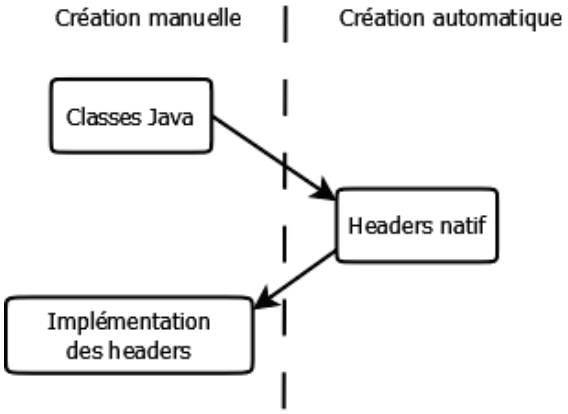


Figure 30 : Création d'une interface JNI

Il ne reste plus qu'à implémenter les fonctions en C correspondant aux headers générés. Ces fonctions C peuvent alors utiliser la ou les librairies C/C++ voulues. Elles doivent également faire les conversions de type nécessaire entre Java et C/C++ à l'aide des implémentations standard fournies par les kits de développement Java. Enfin, il est possible de lire et modifier les attributs de la classe java depuis le code natif.

La structure des appels de fonction du programme est explicitée Figure 31, avec en bleu le code Java et en rouge le code C/C++.

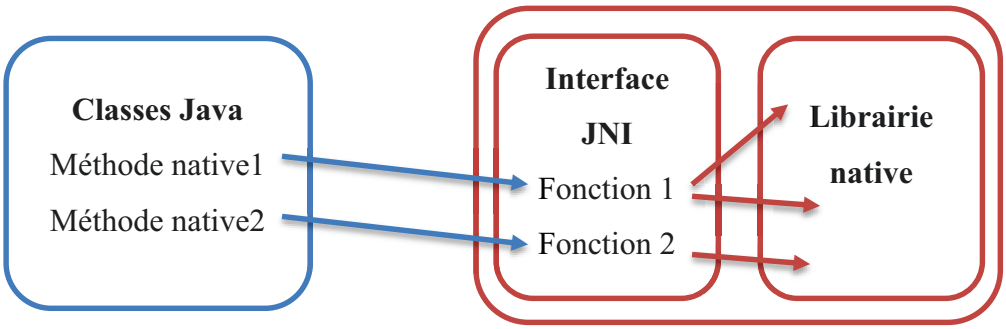


Figure 31 : Structure d'appel d'un programme utilisant JNI

Les méthodes « native » Java font appel aux fonctions de l'interface JNI définie par « javah ». L'implémentation qui en est faite peut faire appel à une ou plusieurs fonctions de la librairie native. La librairie native et l'interface JNI peuvent être regroupées dans une même librairie.

L'interface JNI est assez complexe et longue à implémenter. La boite à outil fourni par les kits de développement java est assez difficile à prendre en main. C'est un développement d'autant plus frustrant qu'il n'a aucune valeur ajoutée fonctionnelle pour le projet. La plupart du temps, ce sont des tâches systématiques. Prendre comme point de départ les fonctions de l'API native qui vont être exposées aurait paru être une meilleure solution que de partir des méthodes déclarées « native » dans la classe Java.

D'autres projets permettant de faire la passerelle entre Java et les libraires natives ont vu le jour pour limiter les coûts de développement. Les projets JNA et Bridj sont les plus couramment utilisés. Le problème des libraires générées par de tels projets est qu'elles sont généralement beaucoup moins performantes qu'avec JNI. Le projet JNA est environ 5 fois plus lent qu'une implémentation directe avec JNI. Le projet Bridj est quant à lui une réécriture de JNA pour permettre des optimisations, mais il reste environ 10 % plus que JNI d'après [Chafik, 2014].

En conclusion, nous pouvons dire que JNI reste un gage de performance. Les projets récents montrent que les performances s'approchent de celle de JNI. Avec le gain en temps de développement et en maintenance qu'ils proposent, l'utilisation d'un projet comme Bridj est certainement la meilleure solution pour créer une interface JNI aujourd'hui.

3.3.2 – Réalisation

L'interface C du moteur Guido est utilisée. Le principe de fonctionnement a été conservé. Chaque classe Java possède une méthode statique d'initialisation du JNI qui retrouve l'ensemble des identifiants des différents champs. Ces identifiants de champs sont utilisés pour la manipulation des attributs Java du côté natif. Des fonctions natives utilisent ces champs pour faire des conversions de type Java vers C et inversement. Pour chaque structure C utilisée dans le moteur Guido, une classe Java est créée. Les classes Java correspondent aux différents modules créés dans la bibliothèque C.

Les nouvelles fonctionnalités de Guido ont été ajoutées dans le JNI et d'anciennes ont été supprimées. Une autre modification a été apportée, car les classes Java du JNI utilisent un package de classe standard de Java pour faire des interfaces graphiques « `java.awt` ». Ce package n'existe pas dans la version Android de Java. Les classes de ce package sont utilisées pour délimiter des zones de l'écran afin retrouver la position graphique d'un intervalle de dates musicales. Ces fonctions sont utilisées pour faire le mapping, tel qu'il a été expliqué en paragraphe 1.2.2. Les classes « `java.awt` » sont donc remplacées par de nouvelles classes qui contiennent ces coordonnées. Par souci de compatibilité, les anciennes interfaces utilisant « `java.awt` » sont conservées.

Le JNI n'a donc été que peu modifié. Nous allons maintenant voir quelques principes d'amélioration qui pourraient être appliqués si une refonte en profondeur du JNI était entreprise.

3.3.3 – Principe d'améliorations

Comme cela a déjà été évoqué, l'interface JNI existante ne peut pas être complètement revue pour assurer une compatibilité entre les versions. La maintenance évolutive faite sur cette interface ne laisse donc que peu de liberté et plusieurs pistes d'amélioration ont été laissées de côté. Outre le fait que JNI est utilisé directement au lieu d'utiliser un outil tiers pour générer le code et améliorer la productivité et la maintenance, l'interface JNI actuelle possède plusieurs autres faiblesses dans son code Java.

La première amélioration à apporter serait le respect des conventions de nommage du langage Java. Java encourage les développeurs à créer des classes commençant par une majuscule et à utiliser le "lower camel case" pour les méthodes et attributs. Les classes

doivent également appartenir à un package et éviter d'être à la racine du dossier des fichiers source. Idéalement, le nom du package doit être porteur de sens avec généralement l'entreprise, le projet et le rôle du package dans le projet. Dans notre cas, ce pourrait être « `fr.grame.guido.jni` ». L'utilisation de nom de package précis permet d'éviter tout conflit entre des classes de même nom.

La seconde amélioration porterait sur la logique d'utilisation de la librairie qui n'est à mon avis pas usuelle pour un pur développeur Java. Son fonctionnement est calqué sur le fonctionnement de la libraire C, il faut donc explicitement libérer des ressources pour ne pas avoir de perte mémoire. Un programmeur Java ne se soucie pas de la gestion de la mémoire puisque c'est la machine virtuelle qui la gère pour lui. La libération des ressources est donc déconcertante et il y a de gros risques que des ressources ne soient pas libérées.

Il y aurait des moyens pour abstraire cette mécanique à l'utilisateur de la librairie java. En voici une avec des modifications en 3 points :

- ne pas pouvoir laisser manipuler les références des objets natifs à l'utilisateur
- libérer automatiquement la ressource native avant d'en créer une nouvelle
- utiliser la méthode « `finalize` », commune à tous les objets Java, pour libérer les dernières ressources utilisées lors de la destruction des objets Java. Cette méthode est appelée automatiquement par la machine virtuelle lors de la destruction des objets.

3.4 – Plateformes mobiles

Les applications existantes utilisant le moteur n'ont pas vocation à être portées sur les appareils mobiles. L'objectif est plutôt de mettre à disposition le moteur aux développeurs désirant l'utiliser dans leurs applications. Toutefois, l'application GuidoEditor, permettant de saisir des partitions tout en ayant un rendu temps réel, a été testée sur appareils mobiles. Étant une application Qt, il n'était pas difficile de la déployer. Son utilisation est fastidieuse sur mobile, mais elle a permis de faire subir des tests complémentaires au moteur.

3.4.1 – Android

De premiers tests de portage du moteur sous Android avaient préalablement été faits. Le travail avait donc été amorcé avant mon arrivée.

La première étape du portage est la compilation de la librairie pour Android en utilisant le kit de développement natif (NDK : Native Development Kit). Le kit de développement propose deux manières de procéder pour compiler un projet existant. La première est d'utiliser un outil propre à Android qui permet de décrire le projet (arborescence de fichiers et configuration de la compilation) à la manière d'un Makefile. Deux fichiers de configuration[3] sont utilisés et la commande « `ndk-build` » permet de lancer la compilation. La seconde solution est de faire créer par le kit de développement une chaine de compilation dans une version prédéfinie du NDK. Le compilateur Android est alors disponible en ligne de commande comme tous les autres compilateurs du système.

La première solution est la mieux intégrée à l'environnement de développement Android. Elle est également la plus simple à mettre en œuvre, bien qu'il faille décrire le projet dans des fichiers de configuration. De plus, elle est indépendante de la version du kit de développement utilisé et ne demande pas de configuration système pour pouvoir être utilisée. La seconde solution demande une meilleure connaissance d'Android au développeur souhaitant l'utiliser et plus de manipulations. Elle est moins adaptée pour la distribution des sources. C'est donc la première solution qui a été choisie.

Le moteur est configuré comme pour la compilation avec JavaScript. Les problèmes de calcul des métriques des polices sont les mêmes qu'avec JavaScript (il n'y a pas de primitives système), ce sont donc les métriques embarquées qui sont utilisées.

Deux libraires différentes du moteur peuvent être créées :

- sans interface JNI, elle est destinée au développeur souhaitant intégrer le moteur dans la partie native de son application Android. L'interface JNI est alors inutile.

[3] Pour plus de détails, voir l'Annexe 1

- avec l'interface JNI, elle est destinée au développeur souhaitant utiliser le moteur dans la partie Java de son application.

La seconde étape du portage sur Android est de mettre à disposition des classes Java pour le développement. Il y a bien sûr un package regroupant les classes Java pour le JNI adaptées à Android comme vu au chapitre précédent, mais il y a également deux autres packages pour le dessin des partitions :

- un package de commande de dessin générique : il permet le parcours d'un export binaire et le décodage des commandes et de leurs arguments. Ce package ne contient aucune directive réelle de dessin,
- un package de dessin pour Android : c'est une extension du package précédent qui dessine la partition dans un objet « `canvas` » spécifique à Android.

Une application de démonstration avec trois onglets a été créée. Le premier onglet permet de saisir le code Guido, le second affiche la partition avec un rendu SVG et le troisième affiche la partition avec le rendu dans un « `canvas` » à l'aide du package précédemment cité. Le rendu sur « `canvas` » oblige à embarquer la police Guido au format « `true type` » dans les ressources de l'application.

Voici en Figure 32 les trois onglets :

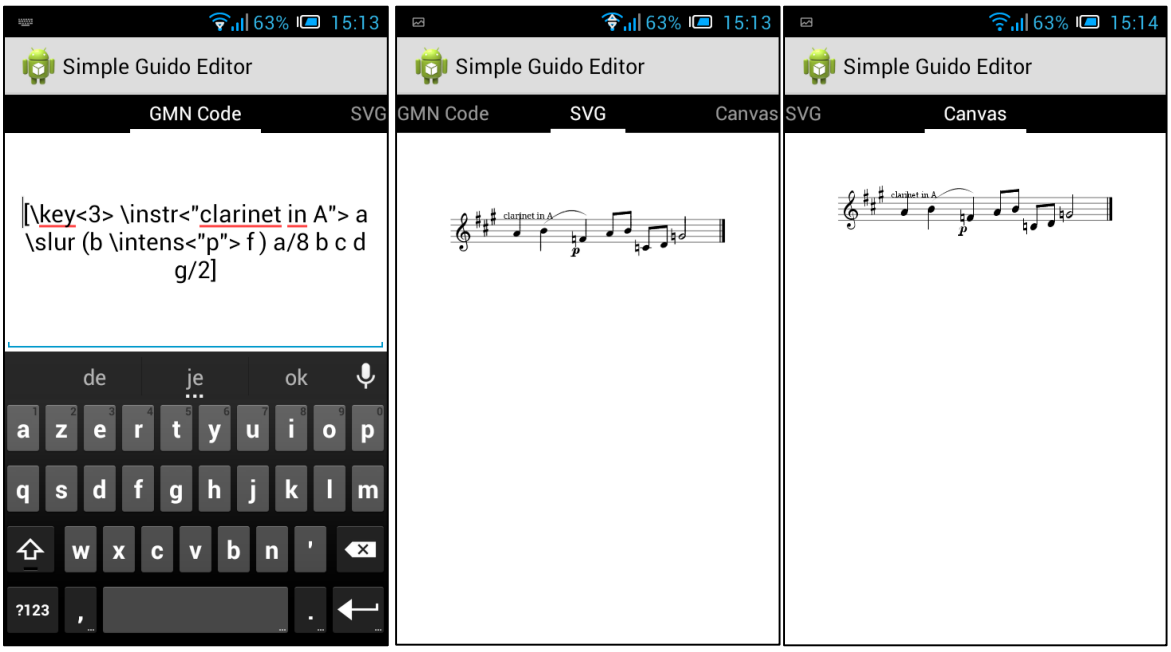


Figure 32 : Application de démonstration de Guido pour Android

La version du moteur Guido utilisé embarque directement l'interface native pour le fonctionnement de JNI.

Cette application a permis de révéler un problème d'anticrénelage[4] lors de l'utilisation de la solution « `canvas` » Android. Sur la version Android testée la désactivation de l'accélération matérielle dans le fichier de configuration de l'application règle le problème sans pour autant poser de problème de performance.

3.4.2 – iOS

Le moteur Guido est compilé pour iOS en utilisant l'environnement de développement Apple. Cette étape ne pose pas véritablement de problèmes, d'autant plus que le moteur comporte déjà les modifications nécessaires avec les versions JavaScript et Android. Tout comme pour ces versions, les métriques des polices de caractères sont embarquées.

La version iOS du moteur ne fournit pas d'application exemple ou de device graphique comme il peut y en avoir pour Android. Les applications iOS utilisent Objective C, langage qui permet d'intégrer directement des sources en langage C. Il n'y a donc pas de mécanisme comme JNI. Les devices graphiques existant peuvent également être utilisés.

Le manque de connaissance des applications iOS développée en Objective C avec le framework Apple ne me permettait pas de faire une application exemple de bonne qualité dans un temps correct. Ces efforts n'ont pas été entrepris, d'autant plus que l'utilisation du framework iOS ne paraissait pas nécessaire pour l'objectif final, le portage d'INScore.

L'application GuidoEditor est portée sous iOS. C'est un projet Qt qui permet de valider le déploiement d'application Qt sur iOS et le fonctionnement du moteur Guido. Toutefois, elle n'a pas vocation à être disponible sur le marché d'applications. Cette application a servi à faire des tests sur l'intégration de la police de caractères et des icônes dans un package. Elle

[4] Voir l'Annexe 2 pour la description de l'anticrénélage ou antialiasing

a également servi à faire des tests en parallèle de ceux faits sur INScore pour permettre d'isoler les problèmes du moteur.

Comme sur Android, le principal problème relevé est un problème d'anticrénelage. Ce problème n'est pas présent pour tous les modes d'affichage disponibles, mais seulement en utilisant le framework interne à Qt « Graphics View ». Ce framework, utilisé dans INScore, permet de gérer un ensemble d'items à l'intérieur d'une scène graphique. C'est une limitation présente dans la version iOS de Qt, qui n'a pas encore été corrigée.

Sur iOS, l'utilisation des « Graphics View » pose également un problème de performance. Le dessin d'une partition est beaucoup plus lent lorsque la partition est dessinée dans un item de la vue graphique que dans une image. Ce problème n'existe pas sous Android. Le tableau ci-dessous montre la différence de temps d'affichage de la même partition dans un item et dans une image. Les contextes d'exécution sont très similaires puisque les temps des autres phases sont presque identiques :

Phase	**Image**	**item**
Création de la représentation abstraite	62 ms	67 ms
Création de la représentation graphique	165 ms	149 ms
Dessin de la partition	171 ms	515 ms

Tableau 10 : Contre-performance sur iOS du dessin d'une partition Guido

Devant ce deuxième problème d'anticrénelage, la question de créer une méthode d'affichage commune avec un meilleur contrôle se pose. L'utilisation d'OpenGL et notamment de la version mobile OpenGL ES, disponible sur les deux plateformes, est donc envisagée.

3.4.3 – OpenGL ES, vers l'uniformisation du rendu ?

L'utilisation d'OpenGL (Open Graphics Library) et OpenGL ES (Open Graphics Library for Embedded System) pourrait être une solution pour avoir un rendu équivalent sur toutes les plateformes et pallier les problèmes d'anticrénelage rencontrés.

OpenGL est une interface de programmation disponible sur de nombreuses plateformes en étant optimisé pour le matériel. OpenGL est un standard proposé par l'entreprise Silicon Graphics qui met dans le domaine public l'avant-dernière version de la bibliothèque GL.

Elle permet de faire des graphismes 2D et 3D et propose un ensemble de primitives pour faire les transformations géométriques.

L'utilisation d'OpenGL peut donc permettre de faire un affichage performant et compatible avec de nombreuses plateformes. Malheureusement, la prise en main de cette bibliothèque est longue et fastidieuse. Une des difficultés supplémentaires dans notre cas est la gestion des polices de caractères, comme nous pouvons le constater en consultant [wikibooks, 2013]. OpenGL n'a pas de support direct des polices alors que leurs utilisations dans Guido est essentielle.

Deux techniques sont utilisées pour faire le rendu de caractères :

- créer des bitmaps ou des pixmap (c'est-à-dire des images) pour chaque caractère,
- créer des textures et appliquer la partie de la texture contenant le caractère désiré.

Comme le rendu de bitmaps ou de pixels est généralement plus lent dans la plupart des implémentations d'OpenGL que le mapping de texture, l'utilisation de textures est à privilégier.

L'utilisation de librairies externes existantes peut être envisagée, mais il faudrait que ces librairies soient facilement portables sur nos plateformes mobiles. Un premier inventaire des librairies les plus populaires a été fait :

- Freetype : permet d'utiliser les polices true type avec OpenGL. Elle est portable, mais utilise des pixmap au lieu des textures et est donc potentiellement moins performante.
- FTGL : permet de simplifier l'utilisation de Freetype par une surcouche.
- FNT library : utilise des textures à partir de polices en format Mark Kilgards. Il faut donc préalablement convertir la police. FNT fait partie d'un plus large projet appelé Portable Game Library et est portable.
- Freeglut : utilise des textures, mais n'est pas portable pour l'instant.

La création de sa propre librairie est également possible. Plusieurs sites spécialisés en donnent les principes. Un atlas de textures serait le meilleur compromis d'après [wikibooks, 2013]. Qt permet de simplifier la création d'une texture en fournissant des méthodes pour la gestion des polices. INScore étant dépendant de Qt, cette méthode pourrait être utilisée assez facilement.

L'utilisation d'OpenGL est un projet à part entière, mais une étude plus complète avec un prototype serait à considérer avec intérêt pour avoir un rendu uniforme quelle que soit la plateforme visée.

La migration du moteur Guido était une étape préalable à la migration d'INScore. Certains développements n'étaient pas nécessaires dans le cadre stricte de la migration d'INScore. Néanmoins, ils contribuent à la mise à disposition des partitions Guido par différents moyens. Les compositeurs ont alors un ensemble riche de possibilités pour créer leurs œuvres et mixer différentes approches de création de partition avec Guido. Comme nous avons pu le voir tout au long de ce chapitre, les développements sur Guido n'ont pas vocation à fournir des produits « clef en main », mais plutôt à mettre à disposition un ensemble d'outils, de démonstrations et de preuves de faisabilité permettant aux artistes de laisser libre cours à leur imagination. Nous allons maintenant voir le travail réalisé sur INScore pour la migration de l'application sur le web et sur les plateformes mobiles. Les évolutions qui permettent de tirer profit de ces nouvelles plateformes sont également présentées.

Chapitre 4

Travaux réalisés sur INScore

Le portage de l'application INScore sur le web était un des objectifs. Contrairement au moteur Guido, INScore n'est pas une librairie indépendante. L'intégration dans un service ne peut donc pas être faite directement. L'utilisation de JavaScript a été assez rapidement écartée, car le framework Qt dont dépend fortement INScore ne peut pas être utilisé facilement avec Emscripten. Un projet de portage de Qt existe, mais il est apparemment abandonné. Nous allons voir dans un premier temps les options qui ont été choisies pour une mise à disposition sur le web.

Le portage vers les appareils mobiles entraine de nouveaux besoins en communication entre les appareils. La mise en réseau de plusieurs instances d'INScore pour le travail collaboratif demande de nouvelles fonctionnalités. Les mécanismes de redirection des messages OSC ont été augmentés pour un meilleur contrôle. Cette mise en réseau a également révélé quelques faiblesses dans certaines configurations. Nous allons voir les adaptations qui ont été faites pour l'utilisation sur mobile et les extensions qui ont été apportées pour améliorer la mise en réseau.

4.1 – INScore Web

Contrairement à Guido, la version web d'INScore n'est pas un serveur web destiné à une mise à disposition en continu sur internet. Les aspects techniques ne permettent pas d'intégrer ce logiciel dans un serveur web sans une refonte du système graphique. L'intégration d'un serveur web dans le logiciel est le choix qui a été fait. Nous allons voir que cela a posé tout de même des problèmes tant techniques que pratiques pour pouvoir intégrer le serveur et l'utiliser dans les situations imaginées auparavant. Les deux types de

serveurs réalisés sont présentés, pour un support HTTP et pour un support web socket. Les spécificités du service web inhérentes à ces deux protocoles sont également détaillées.

4.1.1 – Création d'un Service Web

La création d'un service web d'INScore passe par la mise à disposition d'images de la partition à travers le réseau. L'interface graphique d'INScore est créée avec la partie « Graphics View » de Qt qui s'appuie elle-même sur la partie graphique du système d'exploitation de l'ordinateur. En conséquence, les ordinateurs ou les serveurs n'ayant pas de partie graphique ne peuvent pas exécuter INScore correctement. Or, un serveur web n'a généralement pas d'interface graphique, mais est administré en ligne de commande. L'intégration de la partie librairie d'INScore dans un serveur n'est donc pas viable pour rendre des images, puisque Qt a besoin de réellement faire le rendu dans une fenêtre pour pouvoir donner une image.

L'autre possibilité était d'intégrer le serveur web dans INScore. Nous avons vu qu'INScore gère un arbre d'objets, la racine étant l'application elle-même, puis des scènes et enfin un ensemble d'objets à l'intérieur des scènes. Le serveur, s'il est considéré comme un objet, peut être intégré à une scène. C'est l'option qui a été choisie. Un serveur permet de servir l'image de la scène à laquelle il appartient. Il est créé comme n'importe quel autre objet INScore par l'intermédiaire des messages OSC.

La solution retenue n'est donc pas un serveur qui permet de créer des partitions INScore, mais une partition INScore qui diffuse son contenu. Nous allons voir par la suite qu'une partition peut être entièrement pilotée à distance et qu'il est donc possible d'en créer une de toute pièce à distance à partir du moment où elle contient un serveur.

La diffusion d'une image ne suffit pas pour une partition qui se veut interactive. Les pages web n'avaient pas été conçues pour être interactives, mais de nombreuses évolutions permettent aujourd'hui de faire des interfaces riches.

L'interactivité au travers d'une page web

L'application INScore sur ordinateur permet deux types d'interaction :

- avec la souris, par clic et survol des objets pour lesquels un comportement a été ajouté pour ces actions,

- envoi de messages OSC, par glisser / déposer ou par réseau.

Les deux actions clic ou survol avec la souris sont gérées par deux méthodes dans l'API web qui permettent d'envoyer le type d'interaction et la position de la souris. À la réception par INScore de ces requêtes, il est possible de simuler des évènements de souris. Grâce à JavaScript, il est assez simple de récupérer la position d'un clic sur une image. Cette position est donnée en pixel par rapport au coin supérieur gauche de l'image. Ces informations sont donc faciles à exploiter.

Pour l'envoi des messages OSC, il n'y a pas de support HTTP du protocole OSC. Une méthode d'exécution de messages OSC a donc été ajoutée à l'API. Les messages OSC sont encapsulés dans le corps des requêtes à destination de la méthode d'exécution des messages du serveur. Toujours grâce à JavaScript, un glisser / déposer sur l'image dans le navigateur peut être mis en place. Le code JavaScript lit le contenu du fichier déposé sur l'image et crée la requête avec les messages OSC contenu dans le fichier.

Bien sûr, ces actions ont une influence sur la partition. Son aspect peut alors changer, même si ce n'est pas obligatoire. L'interactivité pose alors la question de la mise à jour graphique de la partition. Avec HTTP, nous avons vu qu'il n'est pas possible d'informer les clients du changement, car c'est un protocole non connecté. La solution est alors de demander des mises à jour régulières de l'image. Il peut donc y avoir un temps de latence entre la modification de la partition et sa mise à jour graphique ou au contraire, plusieurs requêtes inutiles. L'utilisation des web socket est ici avantageuse, car elle permet de notifier la modification au client et évite le trafic inutile.

Les actions de l'utilisateur, que ce soit une récupération d'image ou l'envoi de message OSC, posent une problématique technique. Les actions exécutées par le serveur ne sont pas synchrones avec la boucle de traitement de l'application. Nous allons voir comment elles sont intégrées au fonctionnement d'INScore.

Problématique technique

Nous avons vu en Figure 10 l'architecture du logiciel. L'ensemble des actions (actions utilisateur ou pilotage par réseau) est stocké dans une pile de messages. Une seconde boucle de traitement dépile et traite les messages à intervalle régulier. La mise à jour graphique est

également faite dans cette boucle de traitement. Le traitement des messages et la mise à jour graphique sont donc asynchrones pour rapport à l'arrivée des messages.

Un serveur web a lui aussi sa propre boucle de traitement. Le service a besoin de pouvoir récupérer l'image de la scène et d'exécuter les messages reçus. Les logs des messages de chaque requête d'un utilisateur web doivent être retournés dans la réponse. Lors de l'exécution d'une requête, le serveur fait appel à un objet implémentant les différentes procédures des requêtes possibles. Ces procédures doivent attendre un nouveau passage de la boucle de traitement d'INScore pour pouvoir être exécutées et récupérer le résultat. La Figure 33 montre le procédé utilisé, sachant que le serveur et la WebApi sont dans un thread différent de la boucle de traitement.

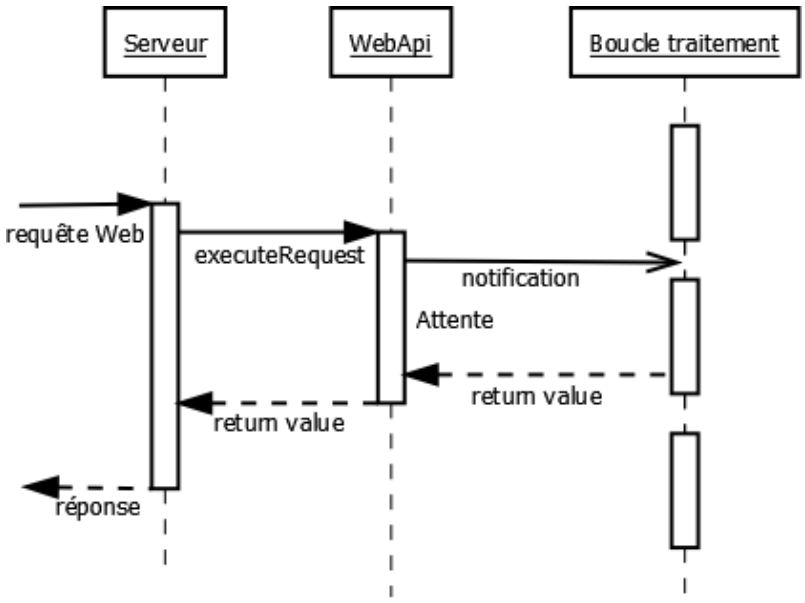


Figure 33 : Traitement des messages reçu par le web

Pour des questions de performance, il a été décidé que l'image de la scène n'est pas créée à chaque passage du traitement graphique, mais uniquement s'il y a eu une requête la demandant. La récupération de l'image doit donc attendre la prochaine exécution de la boucle de traitement. Dans le cas où la partition ne change pas et n'est pas rafraichie, l'image n'est pas créée. Un temps maximal d'attente a donc été défini, pour forcer la mise à jour graphique s'il arrive à expiration.

Lors de la soumission de messages par le web, la fin de leur traitement doit être attendue pour pouvoir récupérer les logs qu'ils ont générés. Pour être différenciés des autres messages, ils sont postés sur une nouvelle pile de messages. Lors du traitement de cette pile

de messages, les logs sont mis de côté pour pouvoir être renvoyés à l'utilisateur. Dans un contexte multi-utilisateur web, la pile de messages doit contenir uniquement les messages d'un utilisateur pour pouvoir isoler les logs. L'écriture et le traitement de la pile de messages dédiée au web doivent donc être exclusifs entre les utilisateurs.

La sécurité

La mise sur le web d'INScore offre de nouvelles possibilités, mais pose aussi des problèmes de sécurité et de droit d'accès comme le montre les cas d'utilisation Figure 19 et Figure 20. Pour avoir un système complet et flexible, tous les internautes ne devraient pas pouvoir exécuter tous les types d'actions, ce qui n'est pas le cas aujourd'hui. Il n'est pas possible de laisser l'ensemble des utilisateurs modifier la partition ou même supprimer et modifier le serveur web par lequel ils sont connectés. Les internautes devraient avoir des rôles différents, qui pourraient être résumés par le cas d'utilisation de la Figure 34 :

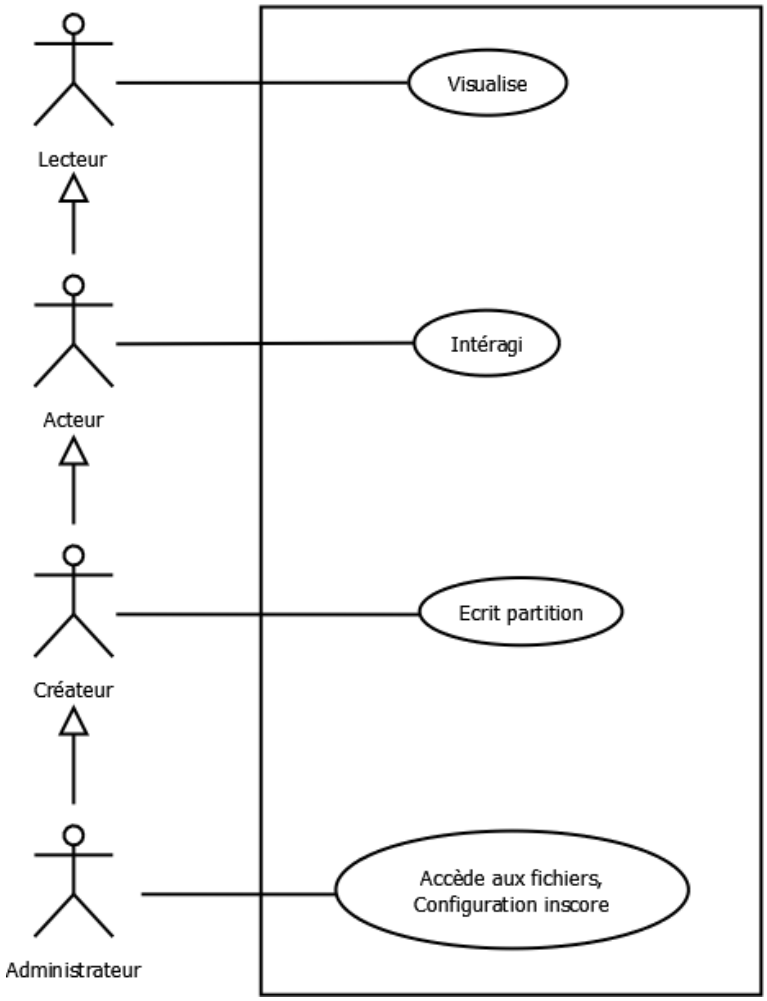


Figure 34 : Rôle des utilisateurs internautes

Le rôle de lecteur se limite à la récupération de l'image. Le rôle d'acteur ajoute les actions avec la souris, clic et survol. Le créateur peut envoyer des scripts de modifications de la partition par ajout, suppression et modification d'objets. Enfin, l'administrateur accède à l'ensemble des fonctionnalités.

De nombreuses commandes permettent de lire des fichiers sur le disque par l'intermédiaire d'objets construits à partir de ces fichiers. Il serait donc possible de lire n'importe quel fichier (autorisations système mis à part) et de récupérer leurs contenus. Il a donc été décidé de limiter l'accès par le web aux fichiers du dossier racine des documents de l'application. Cette fonctionnalité a pu être mise en place. Le concept de dossier racine existait déjà dans INScore et est configurable par message. L'utilisation du dossier racine permet également de prévenir l'écriture de fichiers à des endroits indésirables (par export ou sauvegarde des objets ou des scènes). De plus, une commande de récupération des fichiers dans le dossier racine est présente dans l'API web. Elle a été créée pour récupérer les sauvegardes ou les exports que l'utilisateur a fait.

Les objets construits à partir de fichiers locaux à la machine sont potentiellement dangereux, mais d'autres commandes au niveau de l'application le sont également. La modification du dossier racine de l'application évoqué précédemment devrait être réservée aux administrateurs. La création de serveur HTTP ou web socket, la redirection de messages reçus vers d'autres machines sont des exemples de commandes critiques pour la sécurité et la bonne marche de l'application.

Un internaute créateur de partition peut faire un ensemble d'actions, mais ces actions doivent être limitées. Les scripts envoyés ne peuvent pas contenir n'importe quel message et les sauvegardes ne peuvent pas être faites n'importe où. La Figure 35 résume les différentes actions d'un internaute créateur.

La mise en place d'une stratégie de sécurité demande beaucoup de travail. Les actions à faire pour mettre en place la stratégie de sécurité sont :

- affectation de rôle aux différentes commandes,
- créer un système de gestion des rôles des utilisateurs,
- mise en place d'un système d'authentification,
- mise en place de la vérification des autorisations.

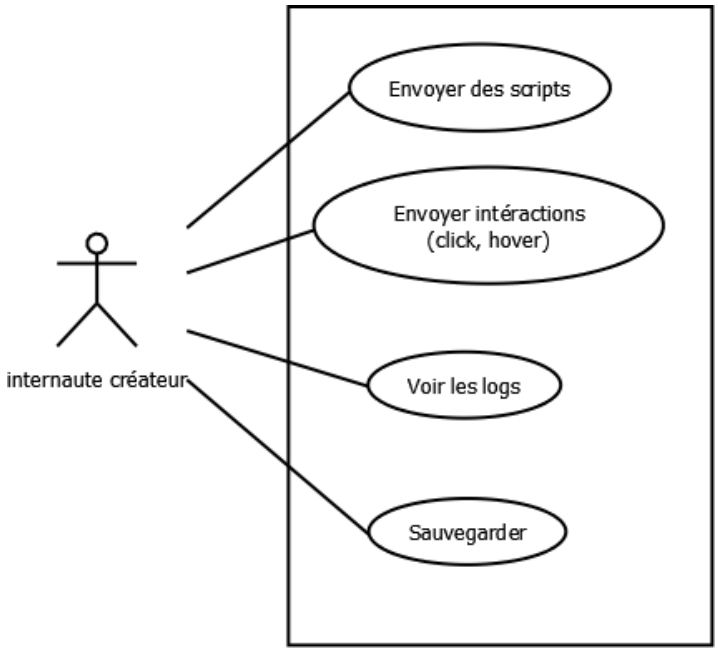


Figure 35 : Actions de l'internaute créateur de partitions

Les commandes et les rôles autorisés pouvant varier avec les besoins, un stockage de ces informations dans une base de données embarquée serait certainement une bonne solution.

Après avoir vu les différentes stratégies adoptées pour rendre disponible INScore sur le web, nous allons voir les spécificités des implémentations des serveurs HTTP et webSocket.

4.1.2 – Réalisation pour HTTP

Le serveur HTTP est basé sur la librairie Libmicrohttpd comme le serveur web Guido. Il n'est donc disponible que sur les systèmes Linux et Mac OS X. Ce composant a été architecturé sous forme de plugin et est donc optionnel au fonctionnement d'INScore.

Le service web est un service REST. Ce service étant particulier, il n'y a pas un respect strict des bonnes pratiques REST vues précédemment. Les réponses aux requêtes sont des objets JSON sauf pour l'image qui est renvoyée directement.

Le Tableau 11 ci-dessous décrit l'API complète du web service pour HTTP avec le type de requête (la méthode http utilisée), l'URI (ou le chemin de la requête) et enfin le contenu du corps de la requête.

Fonction	Type	URI	corps de requête
Version de l'écran	GET	serveurUrl/version	vide
Image	GET	/	vide
Fichier	GET	/path/du/fichier	vide
Script	POST	/	Script INScore
Clic de souris	POST	/click	Coordonnées x et y de la souris
Survol de souris	POST	/hover	Coordonnées x et y de la souris

Tableau 11 : API web d'INSCore avec le serveur HTTP

La récupération de l'image, contrairement au service Guido, respecte le principe de sûreté des requêtes GET et HEAD, puisque c'est l'algorithme de mise à jour graphique qui crée l'image. Le serveur envoie une notification et se contente de renvoyer l'image créée. La version de l'écran (une variable interne à INScore modifiée à chaque mise à jour graphique) est renvoyée avec l'image dans une entête HTTP spécifique « `X-Inscore-Version` ».

Avec HTTP, il est impossible pour le serveur d'informer ses clients d'une mise à jour. Les clients doivent donc périodiquement interroger le serveur. Pour éviter un trafic inutile en ramenant la même image, l'utilisateur a deux options :

- interroger le service de version de l'écran pour vérifier si un changement a eu lieu,
- utiliser la date de la dernière réponse.

La deuxième solution est possible par la mise en place d'un mécanisme standard d'HTTP. La spécification de HTTP 1.1 [Fielding, et al., 1999] au chapitre 13.3.3 explique comment mettre en place un mécanisme de validation (appelé « strong validator ») dans la requête en utilisant l'entête « If-Modified-Since ». Le client fait sa demande au serveur en lui donnant la date de la dernière image dans ce header. Le serveur répond alors avec un code 304 (« Not Modified ») si la partition n'a pas changée ou dans le cas contraire, retourne l'image avec la date de dernière modification dans le header « Last-Modified ». Les formats de date acceptés

par le serveur doivent être les trois donnés en exemple ci-dessous pour respecter la spécification [Fielding, et al., 1999] au chapitre 3.3.1 :

```
Sun, 06 Nov 1994 08:49:37 GMT  : RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT : RFC 850, obsoleted by RFC 1036
Sun Nov 6 08:49:37 1994        : ANSI C's asctime() format
```

Le serveur doit renvoyer une date au format RFC 822 (le premier de l'exemple).

Enfin, la dernière spécificité des réponses du serveur est de placer une directive « no-cache », pour que les intermédiaires entre le serveur et le client ne stockent pas l'image. Sans cette directive les intermédiaires pourrait resservir la même image sans réinterroger le serveur.

Les requêtes POST permettent de modifier la partition. L'idempotence des requêtes ne peut pas être garantie puisque l'action sur la partition dépend de ce qui a pu être défini auparavant. La partition n'est pas une ressource comme l'entendent [Richardson, et al., 2007] puisqu'elle contient un état des actions passées. Néanmoins, la syntaxe la plus simple et intuitive a été mise en place.

Il est fourni avec l'application des fichiers exemple permettant de contrôler INScore depuis une page HTML contenant le code JavaScript nécessaire à l'interactivité. L'utilisation de la librairie Libmicrohttpd peut être vue comme une limitation pour l'instant puisqu'elle a besoin d'être disponible sur chaque plateforme. Elle n'est pas disponible sur Windows, mais aussi sur les plateformes mobiles pour qui le travail de portage n'a pas été fait. L'utilisation d'un serveur web intégré à Qt serait peut-être une meilleure solution.

4.1.3 – Réalisation pour websocket

Le serveur utilisé pour l'implémentation web socket est un serveur interne à Qt. Il ne pose donc pas les mêmes problèmes que le serveur HTTP, puisqu'il est disponible sur toutes les plateformes. L'API exposée est la même, mais il a fallu trouver un moyen de créer les requêtes. Comme nous l'avions vu, ni les URI ni la notion de méthode comme HTTP ne peuvent être utilisés pour différencier les requêtes. Enfin, la communication étant bidirectionnelle, les réponses sont asynchrones. Il faut alors lier une réponse à une requête.

Pour créer ces requêtes, des objets JSON sont utilisés. Les objets encapsulent un identifiant de requête (« id »), la fonctionnalité utilisée (« method ») et les éventuelles données jointes à la requête par des champs additionnels, comme nous pouvons le voir dans le Tableau 12 :

Fonction	**Objet JSON**
Version de l'écran	`{ id : "4894", method : "version"}`
Image	`{ id : "145678", method : "image"}`
Fichier	`{ id : "4894", method : "file", path : "saveFolder/myScene.inscore"}`
Script	`{ id: "45612", method : "post", data : "/ITL/scene/rect set 1 1;"}`
Clic de souris	`{ id : 8954, method : "click", x : 152, y : 354 }`
Survol de souris	`{ id : 8954, method : "hover", x : 152, y : 354 }`

Tableau 12 : API web d'INScore avec le serveur web socket

Le serveur répond également par un objet JSON. Cet objet reprend l'identifiant de la requête et un champ « status » à OK ou ERROR. Un champ complémentaire contient le détail des erreurs.

Les retours binaires, comme l'image ou les fichiers, sont encodés en chaine de caractères base64 par le serveur et décodés par le code client. Il n'est en effet pas possible d'inclure des binaires dans des objets JSON.

Enfin, les web sockets permettent au serveur d'envoyer au client des notifications lorsqu'il y a des changements. Nous aurions également pu envoyer directement l'image. Nous avons préféré envoyer une notification, le client est alors libre de demander l'image ou non.

L'intégration de l'api web avec les web sockets a été assez simple, car elles sont plus adaptées à nos besoins. Du côté client, il y a également des avantages, car dans le cadre d'une page web dynamique, JavaScript est bien souvent utilisé. L'intégration des web sockets est simple et les objets JSON pratiques à manipuler. La principale difficulté est le décodage des chaines en base64 qui n'est pas encore très bien intégré à JavaScript, mais cela devrait changer très rapidement.

4.2 – Adaptation aux mobiles

INScore est un logiciel entièrement écrit en C++. Le portage de l'application sur mobile, en utilisant les possibilités des applications natives, doit se faire sans trop de problèmes. Ceci est d'autant plus vrai que le framework Qt est lui aussi multiplateforme.

Il reste tout de même une différence majeure, l'interface graphique. Nous allons voir dans un premier temps les problèmes liés aux spécificités de l'interface graphique des mobiles et tablettes. Nous allons ensuite voir comment les applications sont construites pour Android et iOS, puis les adaptations techniques qui ont été apportées. Enfin, un point est fait sur les limitations rencontrées.

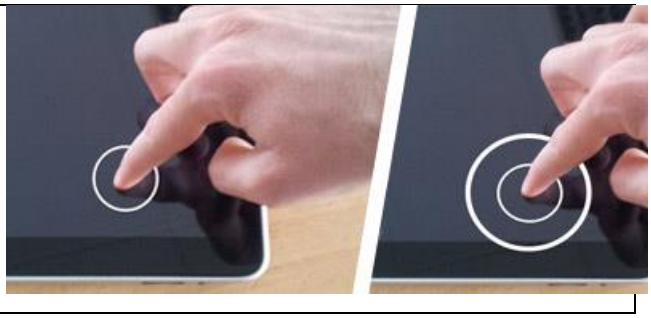
4.2.1 – Les problèmes d'interface graphique

Les interfaces graphiques des tablettes et téléphones nous sont maintenant familières. L'utilisation d'écrans tactiles, souvent de petite taille, a fait naitre de nouvelles conventions pour les interfaces graphiques.

Les applications sont en plein écran sur ce type de périphérique. Il est vrai que la taille des écrans ne permettrait que très difficilement de mettre plusieurs applications côte à côte. La fenêtre d'une application n'a donc pas de bordures.

Lorsque les applications ont plusieurs écrans, elles sont généralement organisées en onglets. La navigation entre les onglets se fait par des gestes tactiles ou par une pression sur l'intitulé de l'onglet. Également contrôlés par geste tactile, le zoom et le déplacement à l'intérieur d'une fenêtre sont devenus courants.

Voici avec le Tableau 13, une présentation des gestes tactiles les plus couramment utilisés :

Toucher (Tap)	Pression courte. Équivalent au simple clic de souris.	

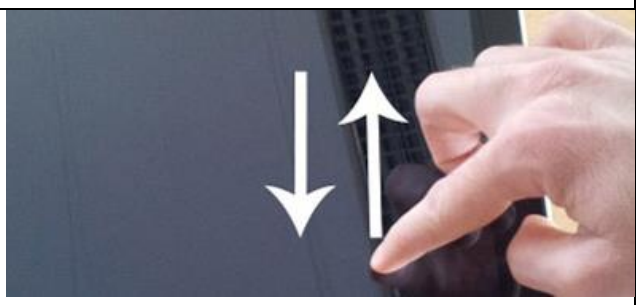
Presser (Tap and Hold)	Pression longue. Permet de saisir des objets ou d'ouvrir des menus contextuels.	
Defilement (Flick)	Défilement rapide en effleurant l'écran ou contrôlé en laissant le doigt appuyé.	
Etirer/ Pincer (Spread/ Pinch)	Zoom avant et zoom arrière.	

Tableau 13 : Quelques-uns des gestes pour écran tactile les plus courants

Le portage sans aménagement particulier d'une application Qt sur appareil mobile ne prend en compte aucune de ces particularités. L'application peut même ne pas être en plein écran. Les utilisateurs d'une telle application sont tout d'abord étonnés par l'aspect rustique de l'interface graphique qui ne respecte pas le thème d'affichage du périphérique. Ensuite, ils essayent généralement de zoomer et de changer d'onglet par les gestes. N'ayant pas ces fonctionnalités, l'application laisse une première impression négative.

4.2.2 – Construire une application

La compilation de l'application se fait à l'aide d'outils qui permettent de s'affranchir des spécificités des plateformes. Il positionne automatiquement des variables d'environnement pour utiliser les libraires et framework spécifique à la plateforme visée.

Pour les applications C/C++, il est courant d'utiliser CMake. La compilation d'INScore s'appuyait sur CMake, mais nous avons fait un portage vers QMake, fourni avec le framework Qt.

L'avantage de QMake sur CMake est d'avoir une syntaxe plus simple et plus concise. Dans le cadre de projet Qt, il simplifie beaucoup les choses. Le changement de plateforme est alors beaucoup plus simple. De plus, la compilation de projet Android avec CMake n'est pas supportée sans faire un gros travail de préparation.

La deuxième modification de l'application pour simplifier la compilation sur l'ensemble des plateformes est l'intégration des sources de plusieurs petites librairies dont dépend INScore. Ainsi, la librairie OSC (pour la communication) et le device graphiques Qt du moteur Guido ont été intégrés. Cela évite de fournir ces librairies compilées pour un ensemble de systèmes et processeurs différents. En effet, les tablettes et téléphones peuvent utiliser des processeurs ARM ou x86.

Enfin, l'application embarque un moteur JavaScript. Le moteur utilisé jusqu'ici était le moteur V8 de Google. Il aurait fallu fournir ce moteur à l'ensemble des plateformes visées. Nous avons donc changé de moteur. Notre choix s'est donc porté sur le moteur JavaScript intégré à Qt pour s'affranchir du problème de portage.

Quelques adaptations de code ont été nécessaires pour les plateformes iOS et Android. Le framework Qt crée automatiquement un enrobage pour les applications natives. Ainsi, la fonction « main », point d'entrée standard d'une application C/C++, doit être changée en « qtmain » sur iOS car Qt l'appelle à partir de code généré. Pour l'application Android, INScore est compilée sous forme de librairies natives. Le framework Qt crée à la compilations des classes Java qui utilisent ces librairies natives.

4.2.3 – Les adaptations

Les adaptations apportées à l'application sont de deux types. Il y a les adaptations spécifiques au système et les adaptations communes portant sur l'interface.

Spécificités de chaque plateforme

Sur Android comme sur iOS, les appareils sont souvent configurés pour une mise en veille automatique après une période d'inactivité. Il est possible de désactiver cette mise en veille en utilisant le framework de chaque appareil. Or, l'application est native, nous n'utilisons pas le code propre au kit de développement.

Pour Android, le framework Qt propose un module qui fait la passerelle entre l'application C/C++ et le framework Android. Il est alors possible d'appeler les fonctions Java depuis le code de notre application.

Sous iOS, les fichiers Objective C peuvent être mixés avec les sources C/C++. Il a donc été possible de modifier les paramètres à partir de code Objective C. Avec iOS, un second problème s'est posé. Les connexions réseau peuvent s'invalider pendant une mise en veille ou un passage en arrière-plan de l'application. Au retour sur l'application, elles sont inopérantes. Un écouteur d'évènement a été mis en place en Objective C pour arrêter les connexions réseau lors de la mise en veille, puis les redémarrer au retour sur l'application.

Le déploiement de fichiers de ressources sur les appareils est différent. Avec iOS, les applications sont installées et exécutées dans une boite. Les fichiers de ressources peuvent uniquement être déposés dans cette boite. Une configuration de l'application est nécessaire pour l'autoriser. Le dépôt de fichiers est nécessaire pour les utilisateurs et les développeurs pour qui un système de validation entre deux versions a été mis en place.

Adaptation de l'interface

INScore peut avoir plusieurs scènes. La scène est le conteneur de plus haut niveau d'éléments de la partition. Sur un ordinateur, les différentes scènes sont des fenêtres indépendantes. Pour mobile, les scènes sont intégrées dans des onglets. La taille des scènes est donc fixe sur mobile. Bien sûr, il n'est plus possible de visualiser plusieurs scènes en parallèle. Une commande par OSC permettant de sélectionner un onglet a été ajoutée, la sélection peut donc se faire à distance. Cette commande est également créée pour les versions sur ordinateur, elle place alors la scène au premier plan.

La seconde adaptation est de mettre en place une fonction de zoom et de déplacement à l'intérieur d'une scène. Ces fonctions n'existaient pas et ont donc tout d'abord été créées. Ensuite, un système de pilotage par geste tactile a été ajouté. Qt fournit un ensemble d'évènements et possède une classe de reconnaissance de gestes tactiles. Après avoir activé la capture des évènements et des gestes, il faut calculer le facteur de zoom ou la taille du déplacement.

Comme INScore peut entièrement être piloté depuis le réseau ou lui-même piloter d'autres logiciels par OSC, tous ces mouvements doivent être traduits en messages OSC. Pour ne pas

surcharger le réseau par des messages inutiles, un message OSC n'est envoyé que lorsqu'un évènement de fin de geste est reçu.

Le geste utilisé pour le zoom est le geste standard de pincement / éloignement avec deux doigts. Le déplacement à l'intérieur de la scène est également géré par l'application de deux doigts. Il est donc possible de modifier le zoom et de se déplacer dans la scène en même temps. Le déplacement se fait généralement avec un seul doigt, mais il s'est avéré que Qt génère des évènements à un doigt pendant le geste de pincement / étirement. Il est certainement très difficile d'isoler le type d'action qu'est en train de faire l'utilisateur si l'on désire avoir un défilement à un doigt et un zoom avec deux doigts.

C'est certainement une des raisons pour lesquelles la reconnaissance de geste dans Qt n'utilise que des gestes avec plusieurs doigts. Il est possible d'implémenter sa propre reconnaissance de gestes plutôt que d'utiliser celle fournie, mais l'effort de développement aurait été considérable pour un résultat qui reste incertain.

Enfin, un geste permet de changer d'onglet. C'est un défilement rapide, en effleurant l'écran. Le framework Qt implémente ce geste avec trois doigts.

Création d'un menu

Un menu s'affichant dans un onglet a été créé. Il permet d'avoir un minimum de contrôle sur l'application depuis l'appareil. Ce menu est également indispensable pour une validation de l'application lors de sa publication. Sans ce menu, seul un écran blanc représentant la scène vide est affiché.

Comme nous pouvons le voir sur la capture d'écran du menu en Figure 36, des informations sur les versions du logiciel sont affichées. Les fonctionnalités ajoutées par cet écran sont :

- le changement des ports d'écoute de l'application,
- le chargement d'une ressource par une URL,
- le chargement d'un fichier en local. Le glisser / déposer n'étant pas possible ce chargement est la seule possibilité pour l'application mobile en dehors du pilotage à distance.

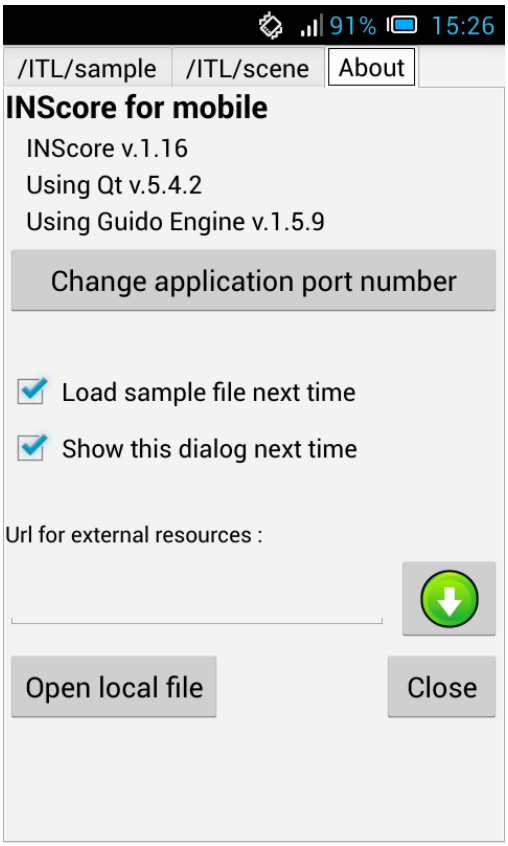


Figure 36 : Menu d'INScore pour appareil mobile

Les deux cases à cocher permettent de ne plus afficher automatiquement le menu lors de l'ouverture de l'application et de ne pas charger automatiquement l'exemple fourni. Ces préférences sont sauvegardées. L'affichage du menu est pilotable par OSC, ce qui est indispensable lorsque l'utilisateur a choisi de ne plus l'avoir par défaut. Ce menu est un nœud statique de l'application, tel que peut l'être la fenêtre de log par exemple.

Le style non natif de l'application est assez peu habituel sur ce type de plateforme, mais nous pouvons espérer que Qt améliore l'intégration des applications sur mobile. INScore possède les limitations soulevées lors du portage du moteur Guido. Sous iOS, l'anticrénelage ne fonctionne pas et les performances d'affichage des partitions Guido ne sont pas très bonnes. Cette dernière limitation n'est pas rédhibitoire, car elle n'est ressentie que lorsque les partitions sont vraiment de taille importante, ce qui est assez rare dans INScore.

Le portage de l'application sur mobile est donc une réussite. Le déploiement sur mobile Microsoft ne devrait poser que très peu de problèmes s'il est envisagé. Seuls les fichiers de construction de l'application devraient être à adapter.

4.2.4 – La consommation électrique

Les logiciels consomment de l'électricité. En ces temps de chasse au gaspillage énergétique, le logiciel passe lui aussi la barrière de l'économie verte avec la programmation verte [Ikonicoff, Avril 2015]. Le rendement énergétique du matériel est un enjeu depuis déjà quelques années, notamment pour les data center et les appareils mobiles fonctionnant sur batterie. Une nouvelle révolution verte de l'informatique se prépare avec le logiciel.

De nombreux logiciels ont et exécutent du code inutile. Les navigateurs internet par exemple, ont une activité sur les onglets en arrière-plan. L'utilisateur n'a pas conscience de l'activité de son ordinateur sur ces pages. Pourtant, ces onglets utilisent le processeur de la machine et consomment donc de l'électricité. Ce n'est pas un problème pour le fonctionnement des ordinateurs d'aujourd'hui qui ont une réserve de puissance, mais c'en est un du côté de la consommation globale des ordinateurs. Un onglet en arrière-plan est estimé à 100 milliwatts par Thierry Leboucq de la start-up Kaliterre. À l'échelle mondiale, le surcroit de consommation est énorme. Un autre exemple nous est donné par [Ikonicoff, Avril 2015] avec les moteurs de recherche. Ils renvoient des dizaines de pages de résultats, alors que l'internaute ne va utiliser dans la plupart des cas que les 4 ou 5 premiers résultats.

Bien entendu, l'engouement des entreprises pour le code vert n'est pas simplement écologique. Les limitations des appareils mobiles, et encore plus avec l'internet des objets, oblige à revoir la conception des logiciels.

Le logiciel INScore porté sur mobile pourrait faire l'objet d'une étude sur sa consommation. L'étude de l'impact d'un logiciel se fait de deux manières complémentaires :

- par des mesures dynamiques de la consommation : le projet PowerAPI de l'INRIA, par exemple, permet de faire des mesures instantanées de la consommation électrique. La difficulté est de faire la relation entre la consommation d'énergie et le code exécuté. Cette mise en relation peut être faite entre les mesures de consommation réalisées et un échantillonnage statistique du code exécuté créé à l'aide d'un outil de profiling. De tels travaux ont été réalisés par [Noureddine, et al., 2015].

- par une analyse statique de code : des outils tels que ceux développés par l'entreprise Kaliterre permettent de détecter les défauts et les patterns consommateurs du code d'un logiciel. Le développeur peut alors améliorer l'efficience de son logiciel.

Certaines libraires comme celle d'Intel permettent de mettre en place des compteurs dans le code et de les exploiter par la suite, mais elle ne contient pas d'intelligence. Il faut donc mettre en place une stratégie. Une étude sur INScore serait intéressante pour voir si une réelle amélioration pourrait être apportée.

4.3 – Mise en réseau

La version mobile d'INScore est particulièrement destinée à être pilotée à distance. L'utilisation d'un ensemble de périphériques permet de nouvelles utilisations, mais demande aussi quelques évolutions. Nous allons voir dans un premier temps le système de renvoi de messages d'INScore et ses évolutions. Ensuite le multicast est présenté, car il correspond aux besoins de la mise en réseau de plusieurs instance d'INScore. Enfin, un bilan est fait sur les limites constatées du protocole UDP en environnement sans fil.

4.3.1 – Mécanisme de forwarding

L'application INScore propose un système de renvoi (ou forward) des messages OSC reçus. Il permet de gérer une liste d'hôtes auxquels renvoyer les messages. Ce système a été étendu pour permettre un meilleur contrôle.

L'extension aux scènes

Le mécanisme de forward n'était possible qu'au niveau de l'application. Il a été étendu pour pouvoir configurer différemment le renvoi des messages de chaque scène.

Un des cas typiques d'utilisation, représenté en Figure 37 est l'envoi de partition séparée à chaque périphérique B, C et D alors que l'INScore A est utilisé pour piloter l'ensemble de la partition.

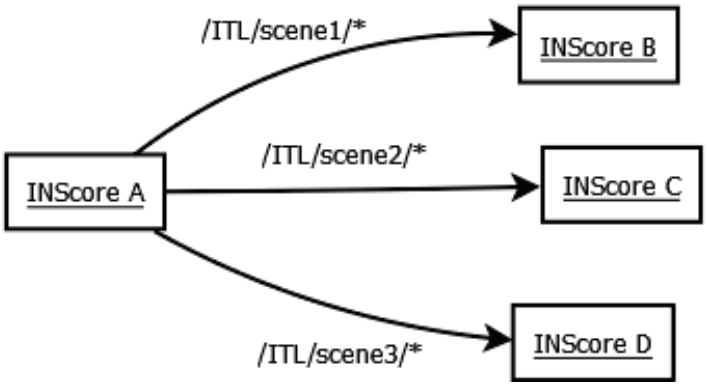


Figure 37 : Utilisation du renvoi de message configuré sur la scène

Les INScore B, C et D peuvent être utilisé par des musiciens en représentation. Le trafic réseau et le travail inutile sur les périphériques sont ainsi évités. La configuration permet de renvoyer les messages d'une scène à plusieurs hôtes.

Les filtres

Un système de filtre a été mis en place pour ne renvoyer que certains messages. Ces filtres sont gérés par une liste de messages ou de modèle de messages à accepter et à rejeter. Les modèles de message peuvent être créés à partir d'expressions régulières acceptées par OSC ou avec des commandes de message. Par exemple, le code ci-dessous permet de rejeter tous les messages commençant par « `/ITL/scene/line` » sauf ceux destinés à « `/ITL/scene/line2` » ou ceux utilisant la commande « `scale` ».

```
/ITL/filter reject /ITL/scene/line*;
/ITL/filter accept /ITL/scene/line2 scale;
```

Un cas typique d'utilisation est l'exécution d'un message contenant du JavaScript (voir Figure 38). Un INScore A exécute un script (message rouge) qui génère des messages (en bleu). Il est configuré pour renvoyer ses messages à B. Il renvoie alors le message contenant le script et les messages générés. L'INScore B exécute le message avec le script et génère les messages (en vert). Les messages « msg1 » et « msg2 » sont alors exécutés deux fois.

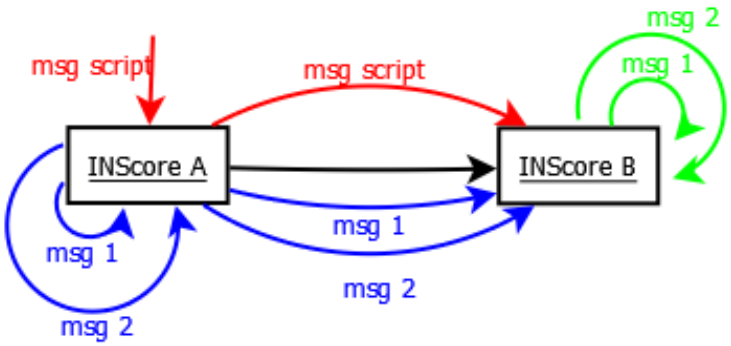


Figure 38 : Exécution d'un script JavaScript sans filtre

Avec le filtre, « msg1 » et « msg2 » ne sont plus renvoyés vers l'INScore B, seul le script est renvoyé (voir Figure 39). Son exécution sur B créé les messages 1 et 2 qui ne sont exécutés qu'une fois.

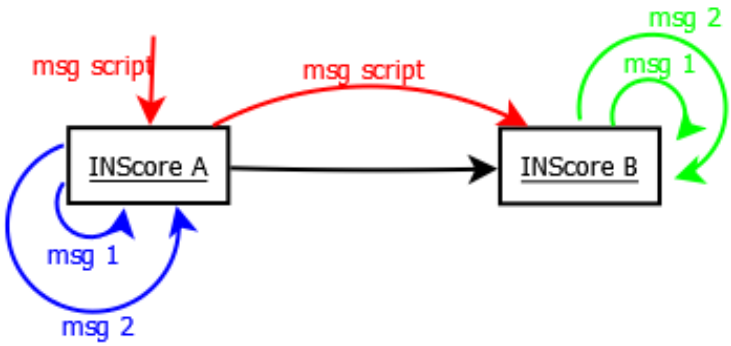


Figure 39 : Mise en place d'un filtre sur des messages INScore

Ce mécanisme de filtre est implémenté par un nœud statique au niveau de l'application et un nœud statique au niveau de chaque scène. Il est ainsi possible de paramétrer finement l'application.

La collaboration

Dans le but de créer des partitions de manière collaborative, la mise en réseau de plusieurs machines avec INScore utilisant le système de forwarding est une solution. Elle peut poser des problèmes de boucle dans les graphes créés. Un certain nombre de dispositions ont été prises pour casser les cas triviaux de boucles.

Le premier cas est la collaboration de deux instances d'INScore se renvoyant mutuellement les messages. Dans la Figure 40, les forward en place sont représentés en noir, le message en rouge. Nous pouvons voir que sans précaution particulière, une boucle est créée entre les deux instances d'INScore.

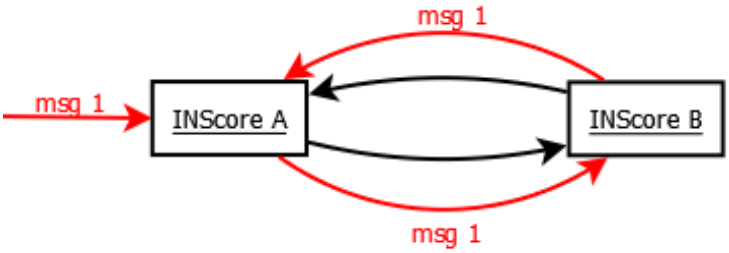


Figure 40 : Création d'une boucle infinie de renvoi de messages

Cette boucle est cassée en utilisant l'adresse IP de l'émetteur du message. Une vérification est faite pour que le message ne soit jamais renvoyé à son émetteur.

Avec cette amélioration, il est maintenant possible de créer une partition à plusieurs comme sur la Figure 41 :

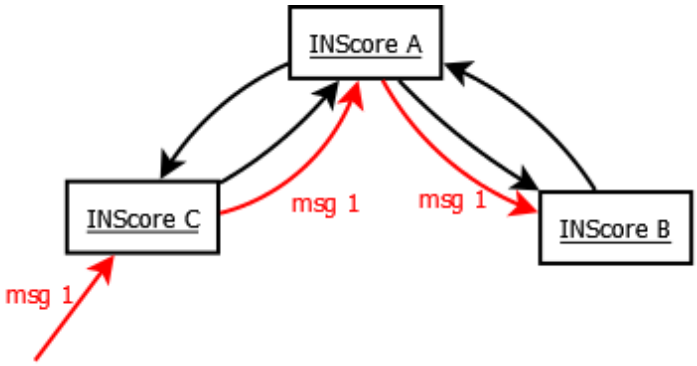


Figure 41 : Collaboration de 3 instances d'INScore

Il faudra tout de même rester vigilant, car la création d'une boucle entre trois instances ou plus ne sera, elle, pas détectée. C'est pourquoi il n'y a pas de renvoi de messages entre B et C. Les messages passent par l'intermédiaire de A et sont de toute façon partagés. Pour avoir un système collaboratif à n instances, une configuration basée sur le schéma précédent est suffisant.

La configuration de la Figure 42 est à éviter, car elle crée une boucle qui n'est pas détectée :

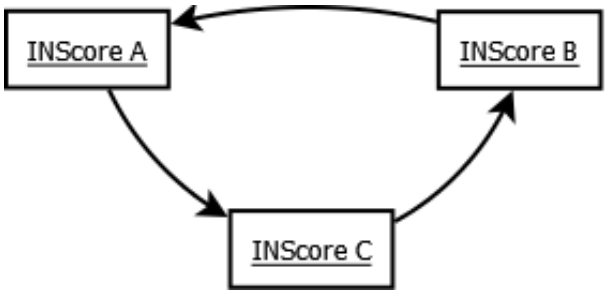


Figure 42 : Boucle infinie entre 3 instances d'INScore

Une extension d'OSC pour le mécanisme de forward d'INScore a été envisagée. Cette extension permettrait de garder le premier émetteur du message et donc de casser le cas de boucle précédent. En revanche, elle ne permet pas de gérer les boucles imbriquées, une réflexion plus complète sur l'ensemble de la problématique du renvoi devra être menée si une telle extension doit être implémentée.

Le broadcast (diffusion générale)

La librairie OSC a été étendue pour permettre l'utilisation de l'adresse de broadcast. Le broadcast permet de diffuser un paquet à l'ensemble des machines d'un sous-réseau (voir Figure 43).

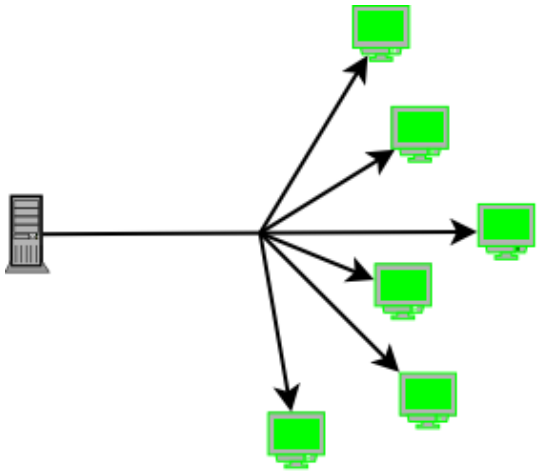


Figure 43 : Représentation du broadcast

Deux adresses de broadcast peuvent être distinguées :

- le broadcast limité `255.255.255.255` : le message est diffusé uniquement sur le segment de réseau. Les routeurs ne retransmettent pas ce message.
- le broadcast direct : le message est envoyé à toutes les machines d'un réseau. Par exemple pour le réseau `192.168.0.0` ayant pour masque de sous-réseau `255.255.0.0`, l'adresse de broadcast est `192.168.255.255`.

Son utilisation soulève de nouvelles questions sur le renvoi de message. Si un système de forwading a été mis en place entre des machines de ce réseau le paquet sera retransmis, alors que toutes les machines l'ont déjà reçu. Les messages reçus par broadcast ne doivent donc pas être retransmis par INScore. Le filtre mis en place utilise l'adresse de destination inscrite dans l'entête de contrôle IP pour identifier les messages à ne pas retransmettre. Le schéma

en Figure 44 montre que dans le premier cas l'INScore C reçoit deux fois le même message, alors que dans le second cas le message est filtré.

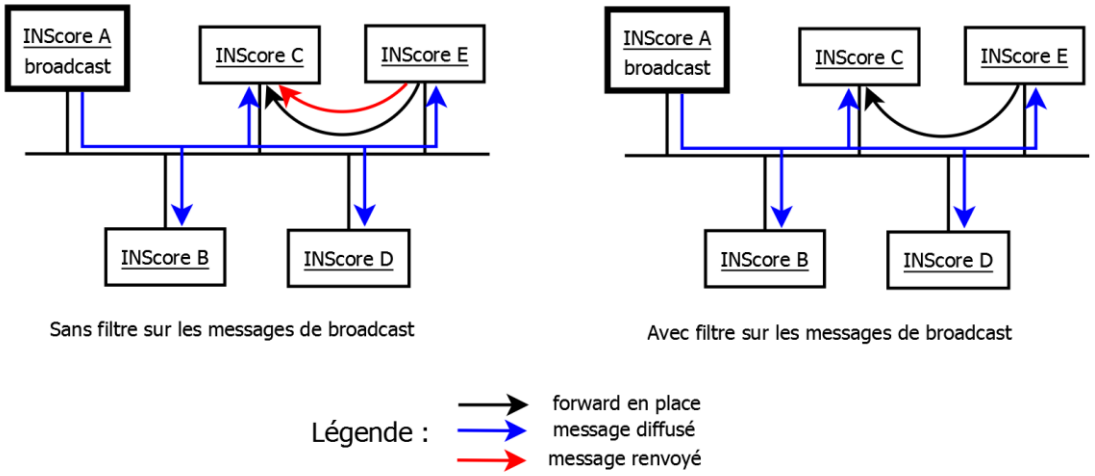


Figure 44 : Suppression du renvoi de messages reçus par broadcast

Le second problème est le renvoi de messages sur une adresse de broadcast. Dans le schéma de gauche de la Figure 45, l'INScore A renvoie les messages sur l'adresse de broadcast, il reçoit alors son propre message. Un autre filtre a été mis en place pour éviter la prise en compte de ce message par A.

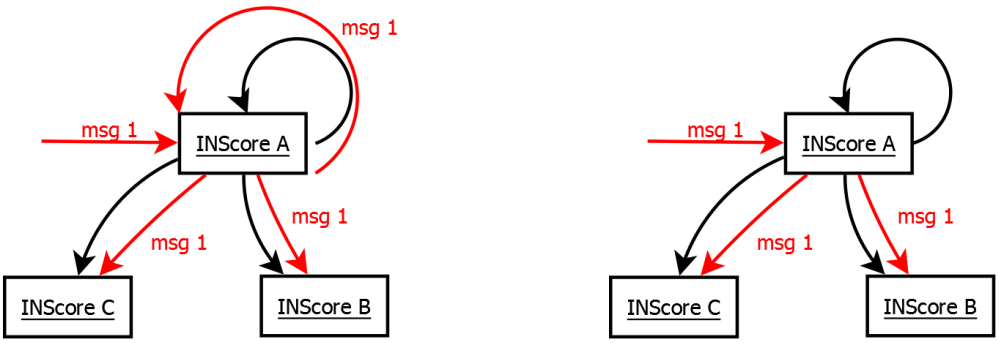


Figure 45 : Suppression de la prise en compte par l'émetteur d'un message envoyé sur une adresse de broadcast

Ces précautions prises, il reste le problème de l'inondation de messages indésirés dans le réseau par l'utilisation même du broadcast. Toutes les machines du réseau ne sont pas forcément concernées par les messages OSC. Elles peuvent ne pas participer à la partition collaborative. Une solution plus élégante pourrait certainement être mise en place avec le multicast.

4.3.3 – Le multicast

Le multicast permet de mettre en place une diffusion restreinte aux machines inscrites à la diffusion. Pour la mise en place du multicast sur un réseau IP, une adresse de multicast est tout d'abord définie. Les hôtes du réseau peuvent rejoindre la diffusion en utilisant le protocole IGMP (Internet Group Managemenet Protocol). Un arbre de diffusion est alors créé pour ce groupe comme le montre la Figure 46.

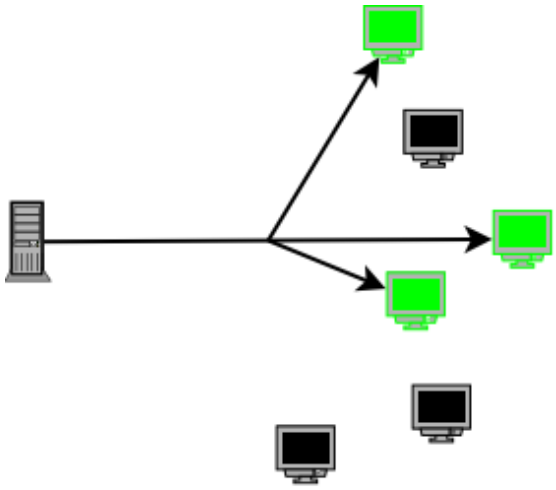


Figure 46 : Représentation du multicast

Le multicast est typiquement utilisé pour la diffusion de la télévision par les opérateurs de téléphonie. Lors du changement de chaine, le décodeur TV s'inscrit à un nouveau groupe de diffusion.

La spécification [Bhattacharyya, 2003] détaille 3 types de configuration réseaux multicast. Ces configurations sont liées aux trois topologies décrites par [Quinn, et al., 2001], qui donnent également des cas typiques d'utilisation :

- Any-Source Multicast (ASM) : le receveur rejoint le groupe en ne connaissant que l'adresse de multicast. Il reçoit les paquets de toutes les machines participant au multicast. C'est une topologie Many-To-Many ou Plusieurs à Plusieurs. Cette topologie permet de faire de la collaboration typique comme le chat.
- Source Specific Multicast : le receveur s'inscrit à une adresse et une source particulière. Il ne recevra que les paquets de cette source. C'est une topologie One-to-Many ou Un à Plusieurs. Les applications typiques sont la diffusion de flux audio / vidéo, la distribution de fichier, la publication d'annonce. Un cas d'utilisation dans

le domaine de la musique serait la relation professeur / élèves. Le professeur diffuse une partition pour l'ensemble de ses élèves.

- Source-Filtered Multicast (SFM) : le receveur s'inscrit à une adresse. Une liste de sources peut être ajoutée pour ne recevoir que les paquets provenant de celles-ci. Inversement, une liste d'hôtes peut être fournie pour ne pas recevoir leurs paquets. C'est une topologie Many-to-One ou Plusieurs à Un. Elle pourrait être utilisée pour une performance improvisée par des musiciens avec création en temps réel d'un conducteur. L'IETF pointe le problème de dimensionnement de cette architecture avec une possible implosion du receveur lorsque trop d'envoyeurs sont présents.

L'utilisation d'INScore étant relativement ouverte, ces trois topologies peuvent être envisagées.

Bien que l'utilisation du multicast semble parfaitement adaptée à l'utilisation collaborative d'INScore après la lecture de [Quinn, et al., 2001], elle n'est pas sans poser de problèmes comme le relève la spécification. L'allocation d'adresse est problématique avec l'architecture actuelle, car elle ne propose pas de solution pour éviter les collisions entre différentes applications.

Pour INScore, une implémentation possible serait de créer une nouvelle socket sur l'adresse de multicast. De nouveaux messages seraient nécessaires pour créer une adresse de multicast, la détruire, la rejoindre et la quitter.

4.3.4 – Les limites d'UDP

La librairie OSC utilise le protocole de transport UDP. Le passage à un réseau sans fil n'est pas sans problèmes, car il est beaucoup moins fiable qu'un réseau filaire.

UDP est un protocole de transport en mode non connecté qui n'a pas de vérification des paquets reçus. En s'appuyant sur [Tanenbaum, 2003] nous pouvons affirmer qu'UDP ne fait pas :

- de contrôle de flux,
- de contrôle d'erreur,
- de retransmission après réception d'un paquet erroné,
- et que l'ordre de réception n'est pas assuré.

Les appareils mobiles sont connectés aux réseaux par WiFi. L'expérimentation nous a confirmé que la fiabilité des liaisons WiFi est bien moindre que les liaisons filaires. [Tanenbaum, 2003] soulève également ce problème, notamment parce que les applications qui utilisent UDP attendent une grande fiabilité du réseau. Le passage d'un environnement où la perte de paquet et le désordonnancement sont théoriques mais peu probable à un environnement sans fil peut donc être difficile.

L'attente d'INScore vis-à-vis de la qualité du réseau est effectivement grande. L'ordre d'exécution des messages est souvent crucial. La non-exécution de messages peut entrainer des modifications visuelles importantes et des dysfonctionnements de l'interface créée.

Des applications de tests ont été réalisées. Elles n'ont pas permis d'identifier de problèmes logiciels lors de l'envoi de messages. La perte de messages intervient surtout lorsque beaucoup de messages sont envoyés dans une même boucle sans attente (entre 150 et 200 messages). La limite de la bande passante du réseau doit être atteinte.

4.3.5 – Bilan de la migration

Les nouveautés apportées à INScore, tant pour le portage sur les différentes plateformes que pour les évolutions permettant une pleine utilisabilité sur celles-ci, ont pu être menées à bien. Cependant, nous pouvons constater qu'il reste des questions en suspens et de nouvelles pistes à explorer. Les premiers cas d'utilisation réels permettront certainement d'identifier les voies à privilégier pour les prochaines évolutions.

La migration sur mobile ou sur le web d'une application n'est pas seulement une question technique, car les façons de l'utiliser et les attentes changent. Ces changements doivent avoir été anticipés pour permettre une bonne adoption par les utilisateurs. Ce type de portage a dû être réalisé pour de nombreuses applications, mais la spécificité de chacune d'entre-elles oblige à une réflexion unitaire.

Conclusion

La problématique de portage d'application sur le web existe depuis longtemps, mais ne peut pas recevoir de méthode standard tant les besoins sont différents. Ici, deux techniques complètement différentes ont été abordées avec Emscripten et les services web. Le monde JavaScript est en pleine évolution et proposera certainement d'autres avancées d'ici peu. Le successeur d'asm.js est d'ailleurs déjà prévu avec « WebAssembly ». Quant aux services web, nous avons pu voir que leur interopérabilité donnait une grande liberté à leurs utilisateurs, pouvant aller jusqu'à des utilisations combinées de services que les fournisseurs eux-mêmes n'avaient pas imaginé.

Pour INScore, les contextes d'utilisation sur le web sont nombreux. Lorsque des cas réels d'utilisation se présenteront, de nouveaux développements seront immanquablement nécessaires. En l'état actuel, une performance publique ne pourrait être envisagée. La mise en place d'un blocage de certaines fonctionnalités par le web sera le minimum requis pour éviter que des utilisateurs avertis et malintentionnés ne provoquent des dysfonctionnements.

Le portage sur mobile de l'application INScore est un succès. Les quelques limitations actuelles, tant au niveau du fonctionnement qu'esthétiques, seront certainement traitées par les futures versions du framework Qt. De premiers tests d'utilisation de partitions évolutives sur tablettes avec des musiciens ont été faits. Ces tests ont malheureusement été grevés par les pertes de paquets dus aux connexions sans fil. Ce problème reste entier et limite pour l'instant l'utilisation de l'application. Dans notre cas, un métronome et un curseur d'avancement à destination des musiciens étaient affichés sur la partition. La perte de paquets a entrainé des désynchronisations entre les appareils. Ce problème, lié au réseau lui-même, sera peut-être l'occasion de faire une évolution en proposant un mode connecté permettant la détection de perte de paquets.

L'extension d'une application à différentes plateformes rend sa maintenance plus difficile. Un soin particulier a été apporté à la construction des applications pour automatiser les tâches. Dans une petite structure où les ressources ne peuvent être augmentées, la pérennité de l'ensemble de ces versions dépendra du temps qu'elles prennent à être maintenue et testée.

Bibliographie

Android developer. 2015. Activities. *Android developer, documentation.* [En ligne] Disponible sur : http://developer.android.com/guide/components/activities.html. (Consulté le 18 06 2015)

—. 2015. Launch Checklist. *Essentials.* [En ligne] Disponible sur : http://developer.android.com/distribute/tools/launch-checklist.html. (Consulté le 17 06 2015)

Android source. 2015. ART and Dalvik. *Android source, documentation.* [En ligne] Disponible sur : https://source.android.com/devices/tech/dalvik/. (Consulté le 30 07 2015)

—. 2015. The Android Source Code. *Android source, documentation.* [En ligne] Disponible sur : https://source.android.com/source/index.html. (Consulté le 19 06 2015)

Apple Inc. 2015. About App Distribution. *iOS Developpeur Library, documentation.* [En ligne] Disponible sur : https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/AppDistributionGuide.pdf. (Consulté le 22 06 2015)

—. 2014. App Programming Guide for iOS. *iOS Developer Library, documentation.* [En ligne] Disponible sur : https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/iPhoneAppProgrammingGuide.pdf. (Consulté le 22 06 2015)

—. 2014. iOS Technology Overview. *iOS Developer Library, Technical documentation.* [En ligne] Disponible sur : https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iOSTechOverview.pdf. (Consulté le 22 06 2015)

Barrière, Jean-Baptiste. 2014. Miroirs Distants. *Biennales musique en scène.* [En ligne] Disponible sur : http://www.bmes-lyon.fr/miroirs-distants-cm.html. (Consulté le 01 07 2015)

Bhattacharyya, S. 2003. An Overview of Source-Specific Multicast (SSM). *Request for Comments: 3569.* [En ligne] Disponible sur : https://tools.ietf.org/html/rfc3569. (Consulté le 06 05 2015)

Bloomberg. 2005. Google Buys Android for Its Mobile Arsenal. *Bloomberg Business.* [En ligne] Disponible sur : http://www.bloomberg.com/bw/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal. (Consulté le 19 06 2015)

Carinola, Vincent. 2015. flux aeterna. [En ligne] Disponible sur : http://vr.carinola.free.fr/fluxaeterna/infos.html. (Consulté le 10 06 2015)

Chafik, Olivier. 2014. Bridj. *Google code.* [En ligne] Disponible sur : https://code.google.com/p/bridj/. (Consulté le 13 11 2014)

CNMAT. 2015. *Open Sound Control.* [En ligne] Disponible sur : http://opensoundcontrol.org/introduction-osc. (Consulté le 04 06 2015)

Cooney, Dominic. 2013. Shadow DOM 101. *HTML5 rocks.* [En ligne] Disponible sur : http://www.html5rocks.com/en/tutorials/webcomponents/shadowdom/. (Consulté le 10 12 2014)

Emscripten Contributors. 2014. Interacting with code. *Emscripten.* [En ligne] Disponible sur : http://kripken.github.io/emscripten-site/docs/porting/connecting_cpp_and_javascript/Interacting-with-code.html#interacting-with-code-call-javascript-from-native. (Consulté le 15 12 2014)

Fielding, R, et al. 1999. Hypertext Transfer Protocol - HTTP/1.1 . *Request for Comments: 2616.* [En ligne] Disponible sur : https://www.ietf.org/rfc/rfc2616.txt. (Consulté le 06 02 2015)

Flanagan, David. 2011. *Javascript, the definitive guide, sixth edition, chapitre 22.9.* O'REILLY, Sebastopol. 1078p.

Fober Dominique, Orlarey Yann et Letz Stéphane. 2012. INScore, an Environment for the Design of Live Music Scores. *Proceedings of the Linux Audio Conference 2012.* [En ligne] Disponible sur : http://www.grame.fr/ressources/publications/INScore-ID12-2.pdf. (Consulté le 04 06 2015)

Fober, Dominique. 2015. INScore, OSC Messages Reference. *Manuel utilisateur.* [En ligne] Disponible sur : http://inscore.sourceforge.net/OSCMsg.pdf. (Consulté le 21 05 2015)

Fober, Dominique, et al. 2015. Les nouveaux espaces de la notation musicale. *Journées d'Informatique Musicale, Improceeding.* [En ligne] Disponible sur : http://www.grame.fr/ressources/publications/GT-Notation-final.pdf. (Consulté le 15 06 2015)

Fober, Dominique, et al. 2010. Partitions musicales augmentées. *Article, Journées d'Informatique Musicale 2010.* [En ligne] Disponible sur : http://inscore.sourceforge.net/. (Consulté le 04 06 2015)

Gartner. 2014. Gartner Says Sales of Smartphones Grew 20 Percent in Third Quarter of 2014. *Gartner.* [En ligne] Disponible sur : http://www.gartner.com/newsroom/id/2944819. (Consulté le 19 06 2015)

—. 2014. Gartner Says Worldwide Tablet Sales Grew 68 Percent in 2013. *Gartner NewsRoom.* [En ligne] Disponible sur : http://www.gartner.com/newsroom/id/2674215. (Consulté le 22 06 2015)

Google. 2015. Closure Compiler. *Google Developers.* [En ligne] Disponible sur : https://developers.google.com/closure/compiler/?csw=1. (Consulté le 02 04 2015)

Grothoff, Christian. 2015. GNU Libmicrohttpd. *GNU Operating System.* [En ligne] Disponible sur : http://www.gnu.org/software/libmicrohttpd/. (Consulté le 28 07 2015)

Herman, David , Wagner, Luke et Zakai, Alon . 2014. Working Draft. *asm.js specifications.* [En ligne] Disponible sur : http://asmjs.org/spec/latest/. (Consulté le 17 04 2015)

Ikonicoff, Roman. Avril 2015. Les lignes de code passent au code vert. *Science & vie.* n°1171, p102-107.

LLVM Project. 2015. LLVM Language Reference Manual. *LLVM Compiler infrastructure.* [En ligne] Disponible sur : http://llvm.org/docs/LangRef.html. (Consulté le 16 06 2015)

Naroff, Steve. 2007. New LLVM C Front-end. *LLVM Developers' Meeting Proceedings.* [En ligne] Disponible sur : http://llvm.org/devmtg/2007-05/09-Naroff-CFE.pdf. (Consulté le 31 03 2015)

Noureddine, Adel , Rouvoy, Romain et Seinturie, Lionel. 2015. Monitoring Energy Hotspots in Software. *Journal of Automated Software Engineering.* [En ligne] Disponible sur : https://hal.inria.fr/hal-01069142/document. (Consulté le 24 07 2015)

Plouin, Guillaume. 2011. *Cloud Computing, une rutpture décisive pour l'informatique d'entreprise.* 2e edition. Dunod, Paris. 268 p..

Quinn, B et Almeroth, K. 2001. IP Multicast Applications : Challenges and Solutions. *Request for Comments: 3170.* [En ligne] Disponible sur : https://www.ietf.org/rfc/rfc3170.txt. (Consulté le 06 05 2015)

Renz, Kai. 2002. Algorithms and Data Structures for a Music Notation System based on GUIDO Music Notation. *Thèse, Université de technologie de Darmstadt, 163 p.* [En ligne] Disponible sur : http://sourceforge.net/p/guidolib/code/ci/master/tree/doc/kai_renz_diss.pdf.

Richardson, Leonard et Ruby, Sam. 2007. *Services Web RESTful - Des services web efficaces.* O'REILLY, Paris. 442 p.

Tanenbaum, Andrew. 2003. *Computer Networks.* 4th edition. PEARSON Education, ISBN 2-7440-7001-7.

The Qt Company. 2015. Qt Documentation, Signals & Slots. *Documentation technique.* [En ligne] Disponible sur : http://doc.qt.io/qt-5/signalsandslots.html. (Consulté le 30 03 2015)

—. 2015. Qt Documentation, Threads and QObjects. *Documentation technique.* [En ligne] Disponible sur : http://doc.qt.io/qt-5/threads-qobject.html. (Consulté le 30 03 2015)

—. 2015. Threads, Events and QObjects. *Qt Wiki.* [En ligne] Disponible sur : https://wiki.qt.io/Threads_Events_QObjects. (Consulté le 11 06 2015)

Vandette, bob. 2010. Java SE Embedded Performance Versus Android 2.2 . *Java SE Embedded, blog oracle.* [En ligne] Disponible sur : https://blogs.oracle.com/javaseembedded/entry/how_does_android_22s_performance_stack_up_against_java_se_embedded. (Consulté le 19 06 2015)

wikibooks. 2013. Modern OpenGL Tutorial Text Rendering 02. *OpenGL Programming.* [En ligne] Disponible sur : https://en.wikibooks.org/wiki/OpenGL_Programming/Modern_OpenGL_Tutorial_Text_Rendering_02. (Consulté le 17 12 2014)

Zakai, Alon. 2013. Emscripten: An LLVM-to-JavaScript Compiler. *Article.* [En ligne] Disponible sur : https://github.com/kripken/emscripten/blob/master/docs/paper.pdf?raw=true. (Consulté le 18 04 2015)

Table des annexes

Annexe 1

Fichiers de configuration pour « ndk-build »

Deux fichiers sont nécessaires pour la compilation de librairie native Android. Le premier, nommé « Android.mk » décrit le projet (répertoires des fichiers sources et options de compilation)

```
LOCAL_PATH := $(call my-dir)
SRC := $(LOCAL_PATH)/../../../../src/engine

include $(CLEAR_VARS)

LOCAL_MODULE := GUIDOEngine

# SRC var is a trick to use wildcard in android.mk
# if no wildcard, you have better to use relative path, conforming to android doc
LOCAL_SRC_FILES         := $(subst $(LOCAL_PATH)/,,$(wildcard $(SRC)/*/*.cpp))
LOCAL_EXPORT_C_INCLUDES := $(addprefix $(SRC)/, include devices lib)
LOCAL_C_INCLUDES        := $(subst $(LOCAL_PATH)/../,,$(wildcard $(SRC)/*/))
LOCAL_CPPFLAGS          := -Dandroid -frtti -DINDEPENDENTSVG

include $(BUILD_SHARED_LIBRARY)
```

Le second fichier, nommé « Application.mk » permet de configurer la compilation. Ici, la forme de la librairie standard C++ et la version du kit de développement sont précisés.

```
APP_STL := gnustl_shared
APP_PLATFORM := android -16
```

Annexe 2

L’anticrénelage ou antialiasing

L’anticrénelage permet de modifier les contours des formes en utilisant des pixels avec différents dégradés de couleur au lieu d’afficher les pixels bruts. Le résultat donne une apparence plus nette des contours et évite l’effet de créneaux.

Un des autres effets d’un bon filtre anticrénelage est d’éviter, dans certaines limites, la perte d’affichage d’une ligne horizontale ou verticale d’un pixel de largeur lors du zoom arrière sur une image.

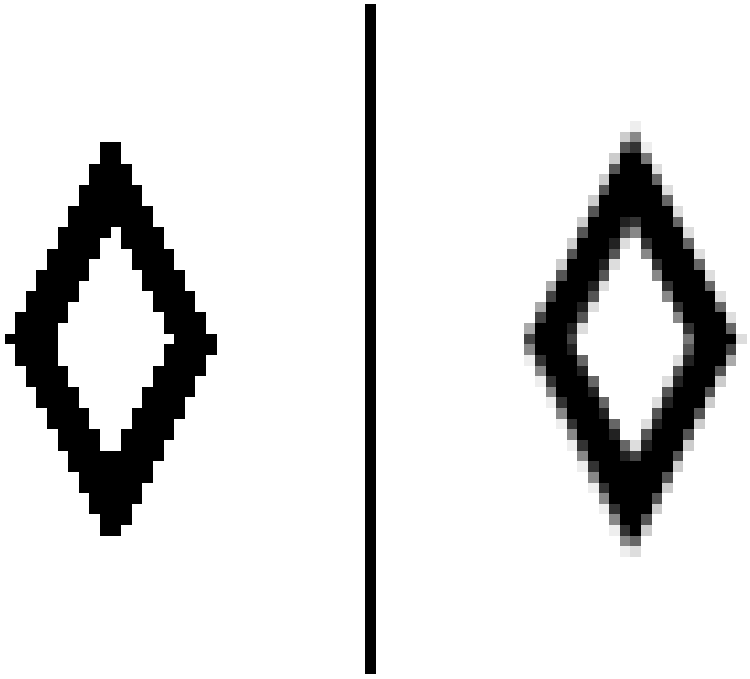


Figure 47 : Losange sans et avec anti-crénelage

Liste des figures

Liste des tableaux

Migration d'INScore pour plateformes mobiles et pour le web.

Mémoire d'Ingénieur C.N.A.M., Lyon 2015

RESUME

INScore est un environnement pour le design de partitions musicales augmentées. Ces partitions permettent l'extension de la notation à des objets graphiques arbitraires, la description de relations explicites entre les espaces graphiques et temporels, la représentation de l'interprétation au sein de la notation et l'interaction en temps réel.

Le portage sur plateformes mobiles et sur le web permet une ouverture à de nouveaux usages. Ces plateformes hétérogènes soulèvent à la fois des problèmes techniques et des questions de design, tant en terme de présentation que d'interaction avec la partition.

Mots clefs : web, mobile, Android, iOS, notation musicale, partition augmentée.

SUMMARY

INScore is an environment for the design of Interactive Augmented Music Scores. This score provides a musical notation extension for arbitrary graphic objects, describing explicit relationships between graphic space and time, the representation of interpretation in music notation and real-time interaction.

The porting to mobile platforms and the web allows an opening to new uses. These heterogeneous platforms raise both technical problems and questions of design, in terms of presentation as well as interaction with the score.

Key words : web, mobile, Android, iOS, musical notation, augmented score.