

# Sébastien Marchand: Tour d'horizon d'Objective-C/Cocoa avec SAM

par Sébastien Marchand ([Eliotis Services](#))

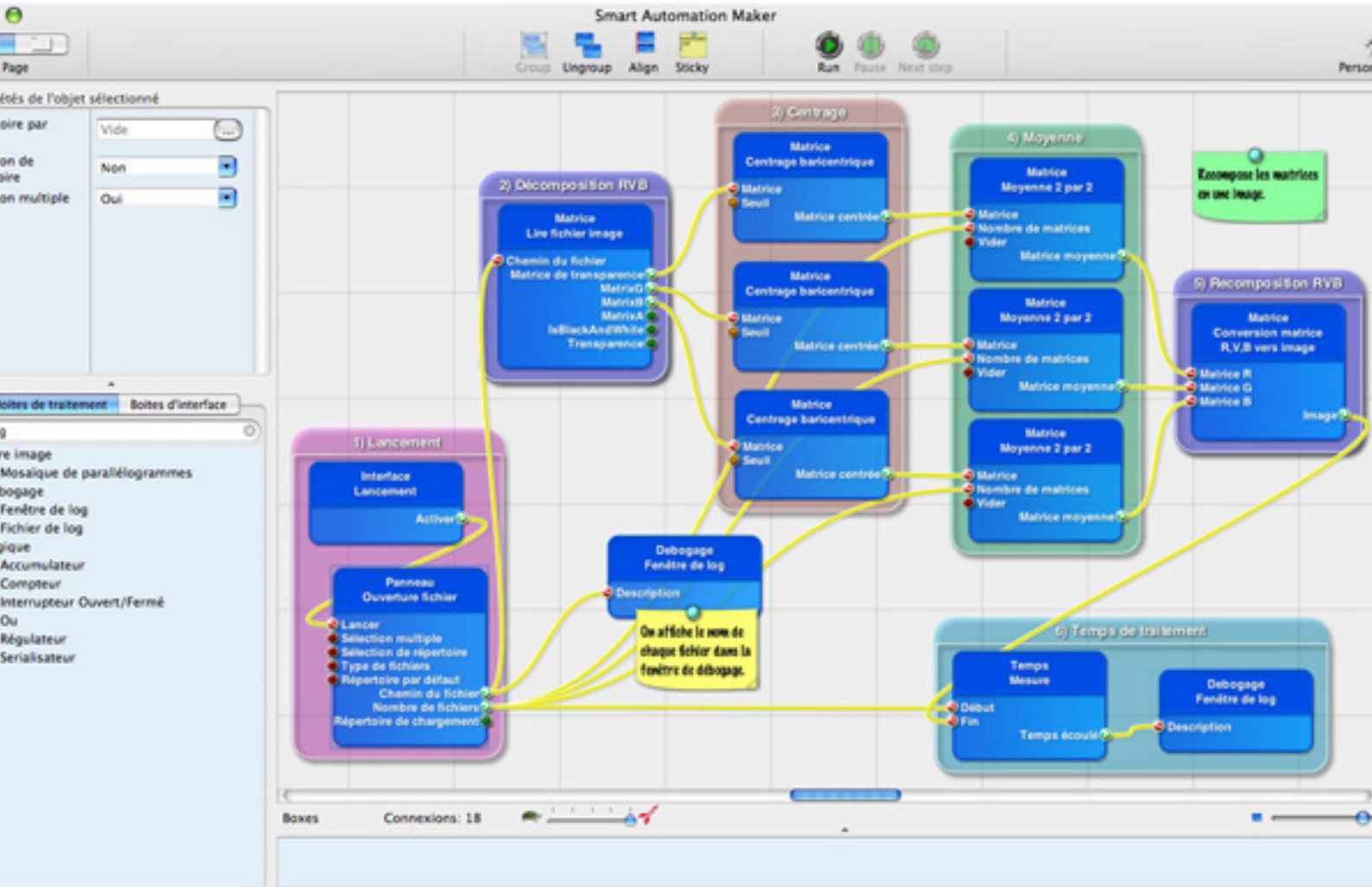
Date de publication : 25 Juillet 2007

Dernière mise à jour : 25 Juillet 2007

Développeur **Cocoa** depuis quelques temps déjà, j'ai eu l'occasion de constater que le langage **Objective-C** est assez peu connu et encore moins le framework **Cocoa** utilisé sous OSX. J'espère que ce petit tour d'horizon avec mon projet SAM (**S**mart **A**utomation **M**aker) vous mettra l'eau à la bouche et donnera envie à certains de tenter la découverte de **Cocoa**. C'est aussi l'occasion de discuter de ce couple encore méconnu alors n'hésitez pas.

- I - Qu'est-ce que SAM ?
- II - Présentation des technologies Objective-C/Cocoa employées pour ce projet
  - II-A - Dessins Vectoriels et bitmaps
  - II-B - Gestion des fichiers de ressource
  - II-C - Internationalisation
  - II-D - Gestion des plugins
  - II-E - Mutli-threading
  - II-F - Sérialisations/Copies
  - II-G - Capacités d'introspection et de mutation
  - II-H - Notifications
  - II-I - Core Image / Core Animation
- III - Conclusion
- IV - Liens

## I - Qu'est-ce que SAM ?

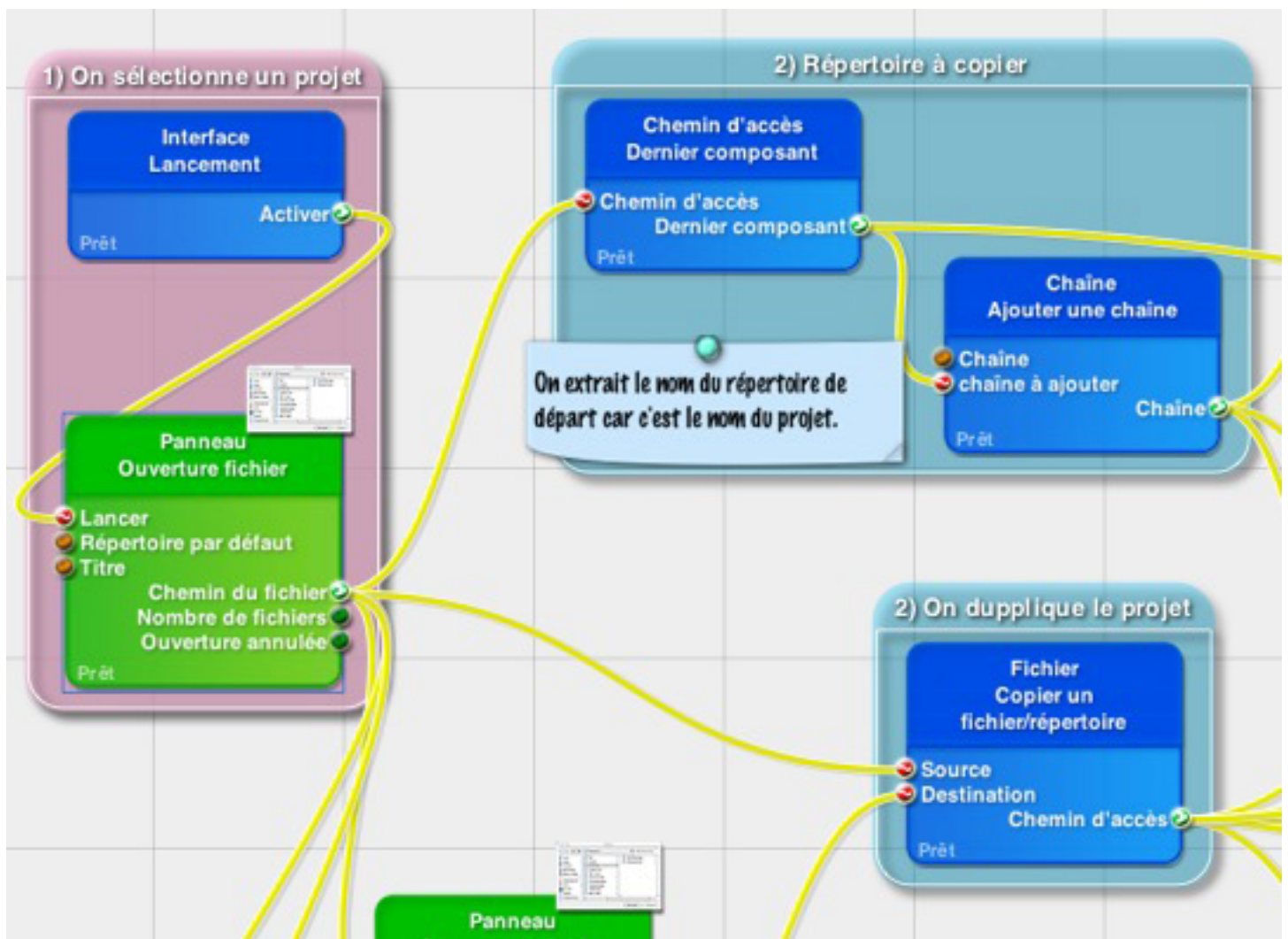


Smart Automation Maker est un logiciel hybride à mi-chemin entre **Quartz Composer (QC)**, **Interface Builder (IB)** et **Automator**. Il permet de développer des applications sous OS X de manière graphique en interconnectant des boîtes entre elles à la manière de QC (bien que la gestion des flux soit un peu différente). Il est possible de développer ses propres boîtes via un assistant. Ce projet est un produit commercial qui sera disponible prochainement.

## II - Présentation des technologies Objective-C/Cocoa employées pour ce projet

### II-A - Dessins Vectoriels et bitmaps

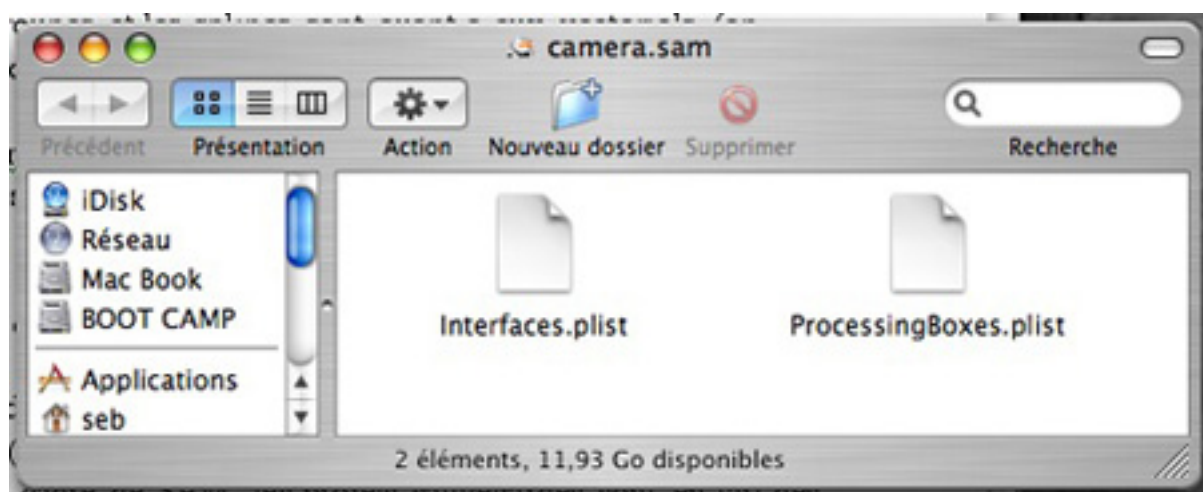
**Cocoa** permet une grande liberté en matière de dessin vectoriel. Il est ainsi possible de dessiner des objets vectoriels complexes (**NSBezierPath**), de dessiner des splines, de gérer des effets d'ombrage (**NSShadow**) ainsi que la transparence ou bien encore d'y mixer du texte (**NSString**) et des images bitmap (**NSImage**). OS X gérant d'entrée de jeux l'antialiasing le rendu est pour le moins convaincant



Dans l'exemple ci-dessus, les boîtes vertes et bleues sont dessinées sous forme de bitmap antialiasé en fonction du zoom. Les groupes et les splines sont quant à eux vectoriels (on notera au passage l'effet de transparence et d'ombrage sur les groupes). La classe **NSImage** utilisée ici pour les images bitmap est capable de lire une multitude de formats et même de nombreux types de fichiers RAW (format brut des appareils photos numériques). On saisit donc tout l'impact d'**iPhoto** sur la richesse de cette classe.

### II-B - Gestion des fichiers de ressource

L'une des particularités d'OS X vient du fait que les applications **Cocoa** sont en fait des répertoires (des Bundles) à part entière et non des binaires bruts. On peut ainsi y stocker toutes formes de données allant d'un simple fichier de configuration texte à des images ou du son. Cela s'applique aussi pour les frameworks (les « DLL » de **Cocoa**) ou bien tout fichier construit à partir de données hétéroclites. On peut ouvrir un paquet en faisant un clic droit sur le fichier -> Afficher le contenu du paquet...



Ci-dessus, dans le cadre de SAM, les projets sauvegardés contiennent en fait deux fichiers XML (l'un pour l'interface graphique et l'autre pour la partie traitement).

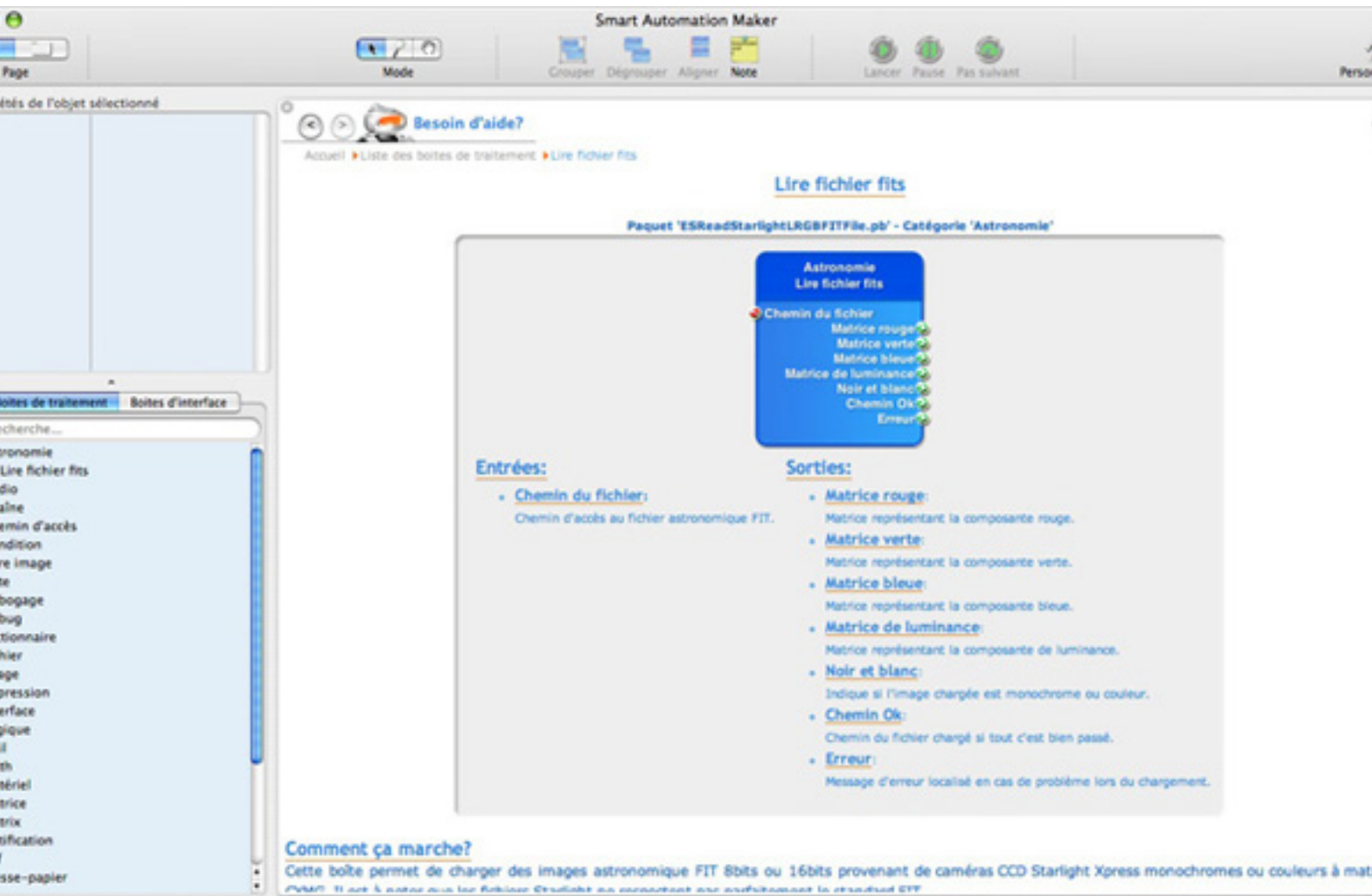
Ce principe est aussi utilisé pour la génération des auto-exécutables de SAM. L'application est alors constituée d'un bundle d'application "type" codé pour lire les fichiers XML dans ses ressources et les plugins des boîtes utilisées par le projet y sont aussi copiés pour un fonctionnement autonome.

## II-C - Internationalisation

OS X contrairement à Windows est 100% multilingue quelle que soit sa version. Cela se retrouve aussi au niveau des applications **Cocoa**. Les différentes traductions (fichier InfoPlist.strings pour le texte brut et fichier nib pour les interfaces graphiques générées par **Interface Builder**) sont alors stockées dans le répertoire de l'application comme des ressources à part entière. Dès lors, on comprend qu'il est très facile de traduire une application sans même disposer du projet d'origine. Ce fonctionnement est aussi exploité par SAM pour une traduction Français/Anglais (voir démo de création d'une boîte donnée en lien plus bas pour en voir l'application).

## II-D - Gestion des plugins

L'ensemble des boîtes de traitement ou d'interface de SAM sont autant de plugins (**NSBundle**) qui viennent enrichir les fonctionnalités du logiciel. **Cocoa** fournit toutes les interfaces nécessaires à leur chargement dynamique. A cela, s'ajoute toutes les possibilités de gestion des ressources et d'internationalisation déjà évoquées. Il en découle que chaque plugin intègre sa propre documentation multilingue. L'accès à la documentation se fait par une interface web (**WebView**) intégrée à l'application. Ici ce sont les technologies Web de **Safari** qui sont mises à contribution.



Besoin d'imprimer ? Pas de problème, l'accès à l'impression du système est disponible avec toutes les vues (terme employé pour définir une zone d'affichage dans l'interface). On peut même sauvegarder la page web en PDF (les API d'**Aperçu** sont passées par là...).

## II-E - Mutli-threading

Particularité de SAM, chaque boîte est indépendante afin d'exploiter au mieux les machines multi-coeurs et/ou multi-cpu. Il est possible d'exécuter toute méthode d'une classe (on conserve ainsi l'accès aux variables de la classe) en détachant un thread (NSThread). Les données échangées peuvent être « facilement » synchronisées (**NSLock**) pour sécuriser les accès concurrents.

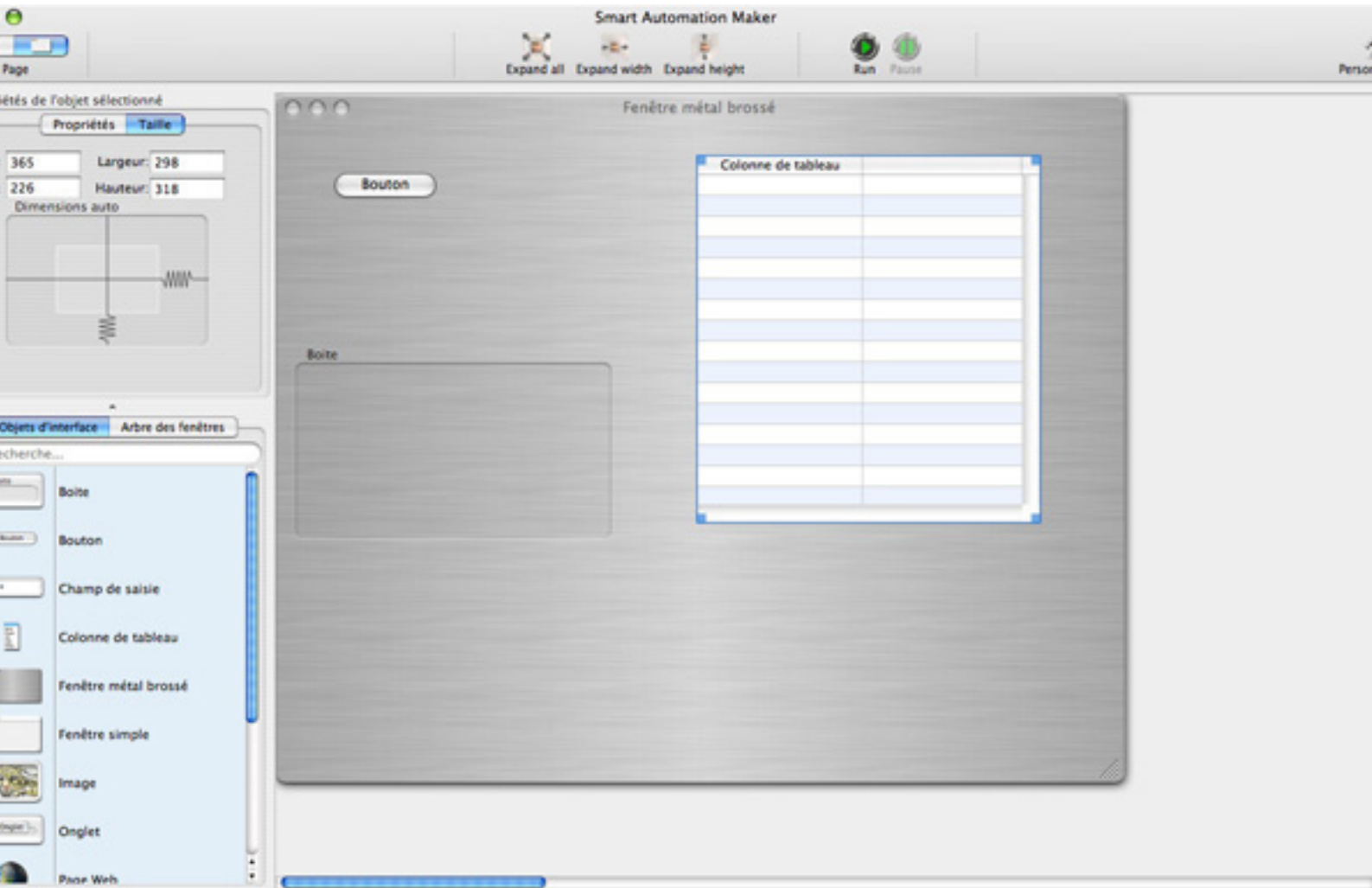
## II-F - Sérialisations/Copies

L'ensemble des classes usuelles se conforme aux protocoles de copie (**NSCopying/NSMutableCopying**) et de sérialisation (**NSCoding**). L'écriture et la duplication de données s'en trouvent simplifiées. SAM se base par exemple sur la sérialisation pour l'écriture des projets sur disque sous forme de fichiers XML plist. On retrouve ici le fonctionnement d'**iTunes** et d'**iPhoto** pour sauvegarder la structure de leur librairie.

## II-G - Capacités d'introspection et de mutation

Objective-C est un langage dynamique par définition. Il fournit ainsi de puissants mécanismes d'introspection mais aussi la possibilité d'enrichir des classes existantes (usage des Catégories) lors de l'exécution.

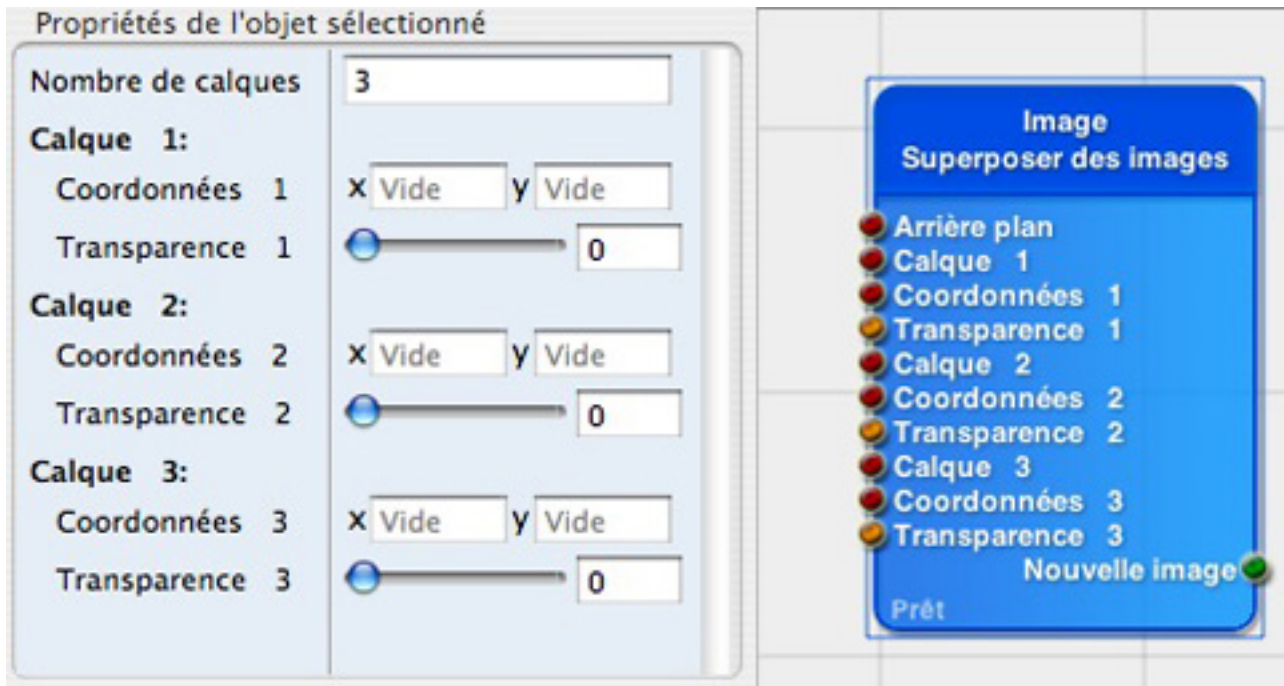
L'introspection est massivement utilisée dans l'éditeur d'interface de SAM afin d'être le plus ouvert possible à l'ajout de nouveaux composants graphiques inconnus.



Les catégories sont quant à elles utiles pour enrichir des classes comme des **NSMutableArray** ou bien des **NSMutableDictionary** afin d'y intégrer à la volée des sécurités (**NSLock**) pour l'accès concurrentiel des threads.

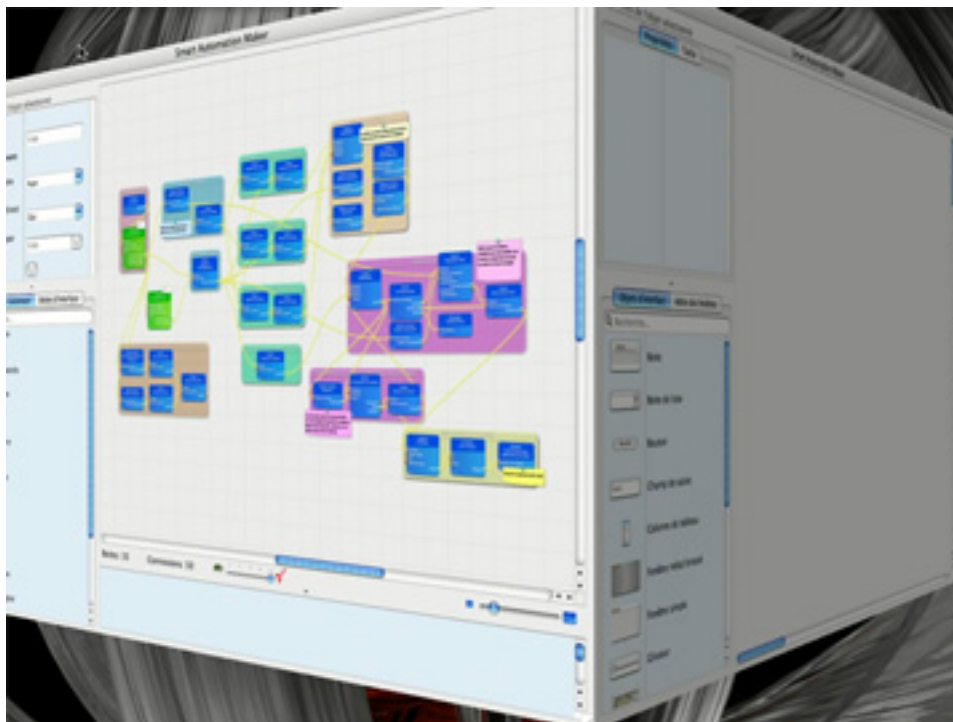
## II-H - Notifications

Les notifications permettent la communication inter-classes (**NSNotificationCenter**) mais aussi inter-thread et inter-process (**NSDistributedNotificationCenter**). Une boîte dont le nombre d'entrées est variable (l'utilisateur veut par exemple une boîte « superposer des images » avec 3 images au lieu de 2) peut ainsi se mettre à jour et demander à l'éditeur de propriétés de se mettre à jour en conséquence.

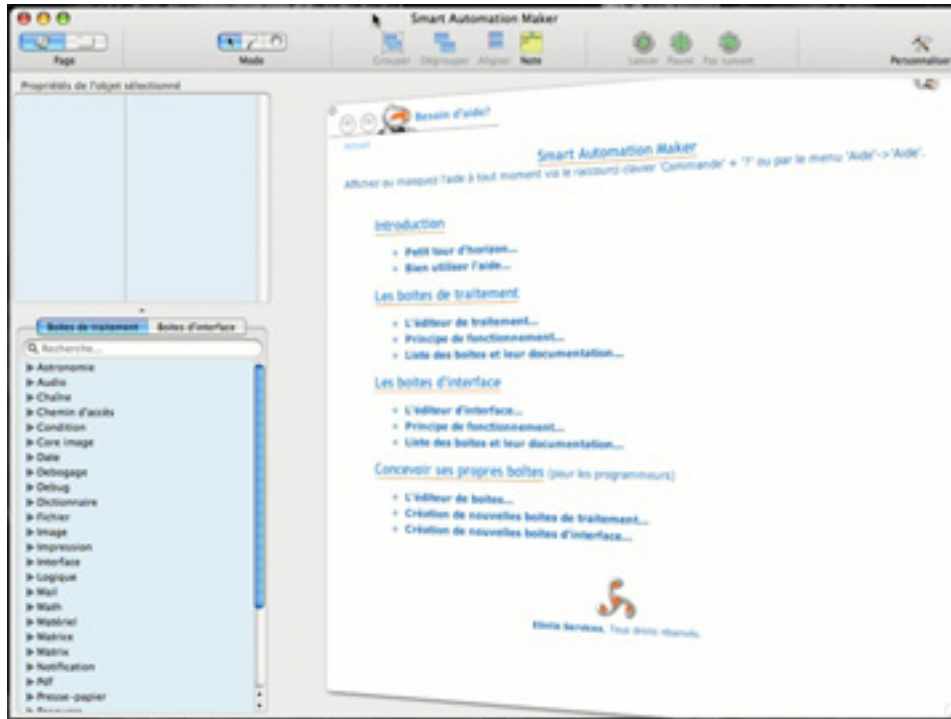


## II-I - Core Image / Core Animation

L'arrivée d'OS X Tiger en Avril 2005 a été l'occasion pour Apple de mettre à disposition des programmeurs de nouvelles fonctionnalités telles que Core Image et Core Animation. SAM intègre ces technologies aussi bien au niveau des boîtes de traitements que des effets visuels 3D de l'interface. Le passage de l'éditeur de traitement à l'éditeur d'interface se fait ainsi avec un effet de cube 3D aussi utilisé dans Tiger lors d'un changement rapide de compte utilisateur.



La documentation est quant à elle située au verso de chaque plan de travail qui se retourne par pivotement (on retrouve d'ailleurs cet effet de transition dans le logiciel **Keynote** d'Apple).



### III - Conclusion

Voilà pour une introduction non exhaustive au couple **Objective-C/Cocoa** sur un cas concret. J'espère avoir donné envie à ceux qui ne connaissent pas encore **Objective-C/Cocoa** de sauter le pas.

Pour plus d'informations sur SAM et le voir à l'oeuvre (vidéos de démo QuickTime), faire un tour sur : [www.sam.eliotis.com](http://www.sam.eliotis.com)

Les plus intéressés (personne ne dort?) pourront découvrir **la conception d'une boîte de traitement avec l'assistant**. On y voit notamment l'intégration avec Xcode ainsi que la gestion multilingue de **Cocoa**.

## IV - Liens

### Liens

- **Smart Automation Maker:** <http://www.sam.eliotis.com/>
- **Tutoriel Objective-C:** <http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf>
- **De C++ à Objective-C:** <http://pierre-chatelier.developpez.com/tutoriels/mac/objectivec/migration/>
- **À la découverte de QuartzComposer:** <http://gfx.developpez.com/tutoriel/mac/quartzcomposer/>

