Dominique Cunin, *Écrans mobiles : contexte technique et création artistique interactive,* in Revue Théorème n°29, *Mobiles, enjeux artistiques et esthétiques*, Laurence ALLARD, Laurent CRETON, Roger ODIN (éds.), 2018, Presse Sorbonne Nouvelle, pp. 73-88

Écrans mobiles : contexte technique et création artistique interactive

Résumé :

Les écrans mobiles sont des appareils avec lesquels les pratiques artistiques de l'interactivité ne peuvent se déployer qu'à l'aide de la programmation informatique. Si les ordinateurs proposent diverses formes d'environnements logiciels auteurs explicitement dédiés à la création d'œuvres interactives, les écrans mobiles requièrent la mise en œuvre de chaînes techniques *a priori* destinées aux spécialistes du développement informatique. Cependant, nous observons actuellement un tournant technologique qui repositionne la chaîne technique du Web en première ligne et qui déplace assez largement ces problématiques de mise en œuvre. Le navigateur Internet étant devenu le contexte de consultation des œuvres, il élargit considérablement les possibilités de diffusion. Cette réalité du contexte technique numérique contemporain a une incidence directe sur des projets comme *Mobilizing.js*, outil de programmation forgé à destination des créateurs souhaitant investir le potentiel de création et d'invention des écrans mobiles. Grâce à une certaine ubiquité du langage du Web, *JavaScript*, ce n'est plus seulement la mobilité des écrans qui peut être travaillée, mais aussi celle des œuvres elles-mêmes entre les différents appareils d'un paysage technologique toujours changeant.

Mots clef : Ecrans mobiles, interactivité, programmation pour les arts, technologies du Web, art des nouveaux médias.

Dominique Cunin, *Écrans mobiles : contexte technique et création artistique interactive,* in Revue Théorème n°29, *Mobiles, enjeux artistiques et esthétiques*, Laurence ALLARD, Laurent CRETON, Roger ODIN (éds.), 2018, Presse Sorbonne Nouvelle, pp. 73-88

Tout au long de son histoire, la création artistique a toujours entretenu une certaine relation avec la technique. Si elle a su développer des savoir-faire qui lui sont directement destinés, elle s'est aussi fréquemment emparée d'objets techniques conçus et inventés en dehors des pratiques artistiques. À l'ère du numérique, la question de la mise en œuvre technique dans la création artistique se repose de manière récurrente, au fil des évolutions des technologies qui imprègnent toujours plus notre quotidien. L'ordinateur, devenu protéiforme et omniprésent dans notre environnement et nos sociétés depuis plusieurs décennies, n'a pas échappé aux pratiques artistiques. Depuis les premiers ordinateurs, développés à des fins scientifiques, universitaires et militaires, jusqu'à l'informatique ubiquitaire aujourd'hui naissante, en passant par la micro-informatique personnelle, les artistes se sont confrontés aux machines numériques et ont cherché à exploiter leur potentiel de création.

Nous nous intéressons ici aux conditions techniques de la création artistique interactive sur les écrans mobiles. À travers notre expérience d'artiste et de développeur dans la conception et la création d'environnements logiciels auteurs, nous souhaitons montrer à quel point le contexte technique imposé par les industriels des technologies numériques contemporaines peut être exigeant lorsqu'il s'agit de le mettre en œuvre en tant qu'artiste. Nous proposons ainsi une forme d'état des lieux du contexte technique de la création artistique interactive sur les écrans mobiles.

Nous proposerons tout d'abord une définition des écrans mobiles et analyserons leurs chaînes techniques de création avant de préciser dans quel type de création artistique impliquant ces appareils nous nous sommes engagés. Le mise en regard des deux versions de *Mobilizing*, librairie logicielle de programmation dédiée à la création artistique interactive dont nous sommes l'un des auteurs, nous permettra de mettre en lumière l'instabilité constante des environnements techniques que nous propose l'industrie de l'informatique. Dans cette instabilité avec laquelle les différents acteurs du numérique doivent composer, l'omniprésence des navigateurs web et des technologies qu'ils mettent en œuvre apparaît comme une éventuelle solution qui, même si elle ne peut être parfaite, présente la promesse d'une standardisation et donc d'une mobilité accrue des projets à travers les plateformes, qu'elles soient mobiles ou non, actuelles ou futures.

Dominique Cunin, *Écrans mobiles : contexte technique et création artistique interactive,* in Revue Théorème n°29, *Mobiles, enjeux artistiques et esthétiques*, Laurence ALLARD, Laurent CRETON, Roger ODIN (éds.), 2018, Presse Sorbonne Nouvelle, pp. 73-88

Écrans mobiles : un contexte technique peu propice à la création ?

L'expression « écran mobile » désigne pour nous une catégorie de machines numériques résultant d'une hybridation entre les téléphones portables, les *personal digital assistant* (PDA) et les ordinateurs portables. Les *smartphones* et les tablettes numériques sont aujourd'hui les appareils les plus représentatifs de cette catégorie. Dans un travail de recherche antérieur[1], nous avons réuni un certain nombre d'indices concernant l'apparition de ces machines dans le paysage technologique contemporain. Dans le sillage d'Erkki Huhtamo, qui décèle un « désir de média mobile »[2] à la suite de ses recherches pour la constitution d'une « écranologie »[3], nous avons identifié un « désir d'écrans mobiles » dans l'histoire des technologies en y repérant à la fois des idées (et parfois des fictions d'anticipation célèbres[4]) et des appareils expérimentaux ou commerciaux[5]. Nous avons ainsi remarqué que les champs de la téléphonie mobile, des expérimentions en informatique ubiquitaire[6] et du jeu vidéo sur supports mobiles[7] sont autant de lieux de gestation de l'agglomération qui s'est opérée entre l'ordinateur et d'autres machines mobiles pour donner forme aux écrans mobiles contemporains que nous manipulons aujourd'hui au quotidien.

---

[1] Dominique Cunin, *Pratiques artistiques sur les écrans mobiles : création d'un langage de programmation*, thèse de doctorat en Esthétique, sciences et technologies des arts, spécialité Arts plastiques et photographie, soutenue le 1er décembre 2014. http://dominiquecunin.acronie.org/phd/These.pdf.zip

[2] Erkki Huhtamo, *Pockets of Plenty: An Archaeology of Mobile Media*, in « The Mobile Audience. Media Art and Mobile Technologies », édition Rieser Martin, Amsterdam/New York, NY, 2011, pp. 23-38. Version PDF: http://gebseng.com/media_archeology/reading_materials/Erkki_Huhtamo-Mobile_Media.pdf.

[3] Erkki Huhtamo, *Elements of Screenology: Toward an Archaeology of the Screen*, in ICONICS: International Studies of the Modern Image, Vol.7, pp.31-82, 2004, Tokyo: The Japan Society of Image Arts and Sciences. Pdf en ligne : http://gebseng.com/media_archeology/reading_materials/Erkki_Huhtamo-Elements_of_Screenology.pdf. Une page web antérieure à cet article (2001) mais portant le même titre est en ligne : http://wro01.wrocenter.pl/erkki/html/erkki_en.html.

[4] Voir à ce sujet Alan C. Kay, *A Personal Computer for Children of All Ages*, Proceedings of the ACM National Conference, Boston Aug. 1972, version en ligne : http://www.mprove.de/diplom/gui/kay72.html

[5] A titre d'exemple, nous pouvons évoquer le *Simon* d'IBM et le *Newton* d'Apple, commercialisés respectivement en 1992 et 1993.

[6] En particulier les travaux de Mark Weisser au Xerox Parc, Mark Weiser, *The Computer for the Twenty-First Century*, « Scientific American », pp. 94-104, Septembre 1991 (http://wiki.daimi.au.dk/pca/_files/weiser-orig.pdf) .

[7] OBANA Takashi, 「GAME&WATCH」のビデオゲーム史的視座 ― ルール・サウンド・インターフェイス, in Core Ethics (The bulletin of Graduate School of Core Ethics and Frontier Sciences) vol. 8, 2012, pp. 87-100 (version japonaise uniquement) URL : http://www.ritsumei.ac.jp/acd/gr/gsce/ce/2012/ot01.pdf

Les écrans mobiles sont une forme d'ordinateur dont le clavier physique a disparu pour laisser place à des interfaces virtuelles interactives par l'écran tactile, donnant ainsi forme à un écran transportable qui devient sa propre interface d'interaction. En effet, dans nos recherches et notre pratique artistique attachées aux écrans mobiles[8], il nous est clairement apparu que la caractéristique technique principale des écrans mobiles est que les nombreux capteurs qu'ils embarquent annulent le besoin de périphériques de capture externes dans la construction d'une interactivité. Accéléromètres, magnétomètres, gyroscope ou encore GPS font partie de ces capteurs qui permettent des interactivités que les ordinateurs ne permettent pas. Mais, malgré ce potentiel de création évident, force est de constater que les usages qui ont été prévus de ces appareils par leurs concepteurs et constructeurs relèvent majoritairement de la consultation. En effet, les écrans mobiles proposent des conditions de production de médias assez différentes de celles des ordinateurs à plusieurs égards. Pour nous en rendre compte, nous pouvons commencer par analyser leur chaîne de développement d'applications, qui, à l'inverse des situations de consultations, révèle la mise en œuvre nécessaire d'une machine tierce.

Prenons l'exemple du système d'exploitation iOS d'Apple. Pour réaliser une application sur cette plateforme et ainsi pouvoir proposer ou vendre son application sur le magasin en ligne afférent *AppStore*, il est indispensable d'utiliser une machine tierce. Les outils de développement dédiés à la programmation pour iOS sont disponibles sur Mac OS X, qui ne fonctionne officiellement que sur les ordinateurs *Macintosh*. Un logiciel dédié au développement d'applications pour Mac OS X et iOS doit être utilisé, *Xcode*[9]. iOS et son *kit de développement* (iOS SDK) étant des descendants directs du Mac OS X, ils reposent sur des briques logicielles semblables et sur les mêmes technologies. Ceci a pour conséquence que le langage de programmation qui doit être utilisé pour réaliser des applications est celui utilisé par Apple depuis de nombreuses années : *Objective-C*[10]. Lorsque cette chaîne de développement est accessible et qu'elle est mise en œuvre, les programmes créés à destination

[8] Nos travaux artistiques, généralement réalisés sous le nom *Acronie* et en collaboration avec Mayumi Okura, sont documentés à cette adresse : http://dominiquecunin.acronie.org.

[9] Xcode est un environnement de développement intégré qui facilite la création de programme reposant sur les composants logiciels fondamentaux des OS d'Apple. Il permet de développer, entre autres, des applications pour Mac OS X et iOS.

[10] Notons que *Swift*, un nouveau langage pouvant remplacer Objective-C, a été introduit par Apple dans ses chaînes de développement en 2014, soit 7 ans après la commercialisation de l'iPhone.

des iPhone et iPad peuvent être testés sur un simulateur intégré au SDK avant d'être testé sur un appareil physique.

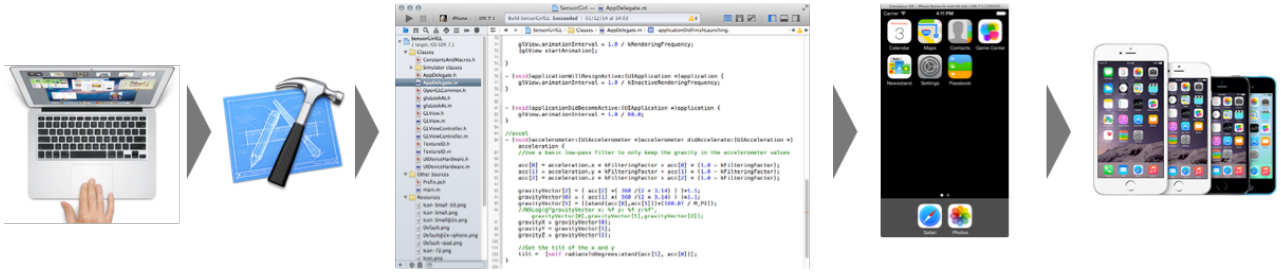

Figure 1 Schéma de la chaîne de développement sur iOS : un ordinateur Macintosh, usage de Xcode (iOS SDK), gestion et programmation du projet, test sur simulateur puis test sur appareil.

Cette chaîne de développement d'applications natives[11] est représentative du contexte technique de la création sur écrans mobiles. Qu'il s'agisse du système *Android* (Google) ou *Windows Phone* (Microsoft), nous constatons des chaînes similaires, impliquant nécessairement un ordinateur comme machine sur laquelle la programmation est effectuée avant de pouvoir tester l'application sur l'appareil lui-même. L'inconvénient principal de ces chaînes natives reste que chacune possédant ses outils et ses fondations logicielles, aucune interopérabilité entre les plateformes n'est possible sans un certain travail d'adaptation : une application développée pour *iOS* ne pourra en aucun cas être utilisée en l'état sur *Android*, qui utilise d'autres langages de programmation dans son SDK (Java et C++), il faut donc « porter »[12] le code source des programmes pour chaque plateforme que l'on souhaite supporter, ce qui implique un long travail d'écriture et de maintenance des programmes.

Cette chaîne de production logicielle, imposant l'usage de machines tierces (des ordinateurs), participe à la définition des écrans mobiles comme machines permettant principalement la consultation, puisqu'il n'est *a priori* pas possible de produire des applications directement sur ces machines. Aussi, ces environnements de développement sont peu accessibles aux non spécialistes, seuls des développeurs expérimentés peuvent

[11] Dans le champ de l'informatique, le terme « natif » est employé pour désigner des composants logiciels compilés spécifiquement pour l'appareil de destination, sans couche d'interprétation. Par exemple, la plus grande partie du code composant les pages d'un site internet repose sur des langages que le navigateur web interprète, alors que ce même navigateur web a été construit avec des langages pouvant être compilés en un code de bas niveau, pouvant être exécuté directement par l'ordinateur. Dans cet exemple, le navigateur est dit en « code natif ».

[12] C'est à dire reprendre le code source du programme pour l'adapter à un environnement autre que celui d'origine.

s'approprier ce genre d'outils, car ils ont été pensés à leur destination et non à celle des utilisateurs non programmeurs. Aussi, l'évolution extrêmement rapide des technologies des écrans mobiles, aussi bien dans leur versant matériel que logiciel, engendre une forme de course à la mise à jour, obligeant les développeurs à adapter leurs programmes d'une manière très régulière et soutenue aux nouvelles fonctionnalités disponibles. La création d'applications sur écrans mobile est donc une activité fortement spécialisée et très exigeante en compétences informatiques pratiques comme théoriques.

Suite à ce constat, si nous déplaçons notre point de vue pour adopter celui des artistes-plasticiens qui désirent investir les écrans mobiles dans leur pratique artistique, nous pouvons imaginer que le contexte technique que nous venons de décrire ne leur est pas facile d'accès, le niveau de compétence technique demandé étant relativement élevé. Mais il nous faut ici préciser le type de création auquel nous faisons référence. En effet, les écrans mobiles permettent certaines formes de création qui ne nécessitent pas la réalisation d'une application particulière. La production et l'édition de photographies et de vidéos à l'aide des capteurs photographiques intégrés (d'une qualité de plus en plus élevée) et d'applications préinstallées peuvent être le lieu d'une pratique à caractère artistique, amateure ou professionnelle. Les pratiques de l'image dessinée ou peinte peuvent aussi être investies[13], l'écran tactile multitouches proposant une nouvelle relation à la création de l'image, de nombreuses applications dédiées au dessin étant disponibles sur les magasins en ligne. La création musicale en *live* est également fortement investie grâce aux applications de synthèse sonore utilisant avantageusement l'écran multicouches et les capteurs embarqués. Cette situation découle de celle que l'on connaît sur les ordinateurs, avec lesquels la création et l'édition de « médias » (photographies, vidéos, textes, sons et musique, images de synthèse, etc.) sont très largement assistée par des logiciels spécialisés et répandus (*Photoshop*, *Illustrator*, *FinalCut*, *Ableton Live*, etc). Cette forme de création consiste donc principalement en la production de médias dont le contexte d'émergence original n'est pas numérique, mais qui ont été assimilés, numérisés pour devenir des « médias numériques », exploitables dans diverses chaînes techniques numériques et tirant parfois parti des particularités des écrans mobiles.

[13] David Hockney en a fait la preuve avec ses peintures réalisées sur iPad. http://www.wired.com/2013/11/hockney/ (consulté le 11/07/2015)

Mais ce champ de pratiques n'exploite qu'une faible partie des possibilités offertes par les appareils numériques qui sont aujourd'hui à notre disposition et, en particulier, il n'envisage pas la mise en œuvre de l'interactivité, pourtant supposée par le simple usage toutes ces machines-écrans qui répondent à nos sollicitations par l'intermédiaire d'interfaces (graphiques comme physiques). Durant les dernières décennies, la création artistique œuvrant avec l'interactivité a pris une telle envergure qu'elle a provoqué le développement de logiciels spécifiquement dédiés au travail de l'interactivité dans le champ des arts, travail qui ne peut s'opérer qu'à l'aide de la programmation informatique, quelle que soit sa forme. En effet, des outils comme *Max/MSP* ou *Processing* (parmi une multitude d'autres) tendent à faciliter et donc à favoriser les pratiques artistiques de l'interactivité. Mais cette culture et cette disponibilité des « outils auteurs »[14] font défaut sur les écrans mobiles, ce qui complique considérablement l'accès des utilisateurs (artistes ou non) à la création d'applications interactives sur ces supports. La création d'objets esthétiques interactifs mettant en œuvre les spécificités techniques de ces machines mobiles demande donc une pratique très exigeante, car il s'agit d'œuvrer avec la programmation informatique tout en tenant une conduite d'artiste afin d'en explorer le potentiel et de le mettre en œuvre (nous utilisons la figure de l'artiste en programmation pour désigner cette position). Mais cette situation s'est progressivement transformée : là où des environnements logiciels auteurs ont pris une grande importance dans la création interactive sur les ordinateurs « de bureau », les fortes évolutions des standards du web apportent aujourd'hui une réponse étonnement efficace et riche face aux difficultés de mise en œuvre de la programmation sur les écrans mobiles.

Les technologies du web à l'œuvre pour la création sur les écrans mobiles

Le *World Wide Web*, qui se déploie au-dessus du réseau informatique mondial *Internet*, est devenu une ressource que nous utilisons quotidiennement pour y puiser toutes formes d'informations ainsi que pour accéder à certaines modalités de communication via des « interfaces web » (messages électroniques, réseaux sociaux, etc.). Les technologies qui

[14] Mais aussi le manque d'environnements logiciels auteurs, qui ne sont pas uniquement des éditeurs de code sources facilitant la programmation dans un langage spécifique avec certaines librairies logicielles précises, mais qui proposent aussi différentes interfaces de programmation (parfois visuelle) pour produire des programmes (*Director*, *Flash*, *Unity*, etc.).

permettent la création de « contenus » sur le web prennent un intérêt bien particulier à l'ère des écrans mobiles, surtout depuis qu'elles prennent en compte les spécificités techniques de ces appareils. En effet, comme toute technologie numérique, les langages du web ont connu plusieurs paliers de redéfinition. Le HTML (*HyperText Markup Language*), langage de description des documents web fondés sur une syntaxe à balises, se stabilise progressivement vers sa version 5, *JavaScript*, langage de programmation de type *script* utilisé pour programmer le comportement du navigateur et, par conséquent, des éléments d'une page HTML, oscille entre ses versions 5 et 6, et CSS, langage spécialisé dans la définition de règles de mise en page dans des feuilles de styles pouvant s'inscrire les unes dans les autres en cascade (*Cascading Style Sheets*) en est à sa version 3. L'ensemble de ces langages connaissent des évolutions constantes et leur mise en œuvre dépends entièrement d'un élément central dans ce complexe logiciel, il s'agit de l'application qui interprète l'ensemble de ces codes et produit une représentation « lisible » et consultable des documents qu'ils décrivent : le navigateur Internet, ou navigateur web.

En tant qu'appareils fondamentalement connectés aux réseaux de communication (téléphonie, Internet, Bluetooth ad hoc, etc.), les écrans mobiles intègrent dans leurs applications de base des navigateurs web qui permettent de « naviguer sur le web », et donc de consulter des documents web. Il est donc possible de développer des sites web et de les consulter aussi bien sur son ordinateur que sur son écran mobile à partir du moment où un navigateur peut être exécuté. Le caractère multiplateforme est l'une des premières promesses qu'endosse la chaîne technique du web HTML 5 relativement aux chaînes de développement natives : « développer une fois, déployer partout »[15], principalement sous la forme de *webapp* (ou *Web Applications*), des documents web qui miment les applications natives. L'effort à fournir de la part du développeur est donc moindre, bien que, dans les faits, la gestion du support des différents navigateurs et des différentes plateformes matérielles peut s'avérer assez complexe. De plus, les technologies du web sont extrêmement répandues puisque la plus grande majorité des sites web sont construits avec le « triangle technique » HTML, CSS,

[15] *Develop Once, Deploy Everywhere* est une expression que l'on retrouve sur un très grand nombre de pages web traitant de HTML5 et de sa capacité à rendre compatible les pages web sur de nombreux supports, y compris mobile. On y trouve une référence au slogan de *Sun Microsystems*, concepteur du langage *Java* : *Write once, run anywhere*. Pour rappelle, le langage *Java* avait dès son origine pour ambition de permettre l'exécution d'un même programme sur toutes les plateforme pouvant exécuter une *Java Virtual Machine* (JVM).

JavaScript[16]. Le nombre de ressources en ligne concernant cette technologie est donc très important (tutoriels, cours, documentations, etc.), ce qui facilite grandement leur apprentissage.

Une autre caractéristique, qui provient de la nature même des technologies du web, vient renforcer cette aisance d'accès et d'apprentissage. Parce qu'elles sont interprétées par le navigateur, les sources des programmes ne peuvent être totalement masquées aux utilisateurs, comme c'est le cas dans les programmes informatiques compilés. Les codes sources des documents en cours de consultation peuvent donc être lus par les utilisateurs. Mieux encore, les navigateurs intègrent aujourd'hui des « outils développeurs » dont la vocation est de permettre le débogage des sources en temps réel, c'est à dire au moment même de leur consultation. Ces outils permettent donc un contrôle très précis des programmes, mais il permettent aussi d'étudier la structure et les sources des sites web que l'on consulte et, incidemment, d'apprendre de nouvelles méthodes et techniques grâce à leur analyse.

[16] Nous faisons ici l'impasse sur les langages déployés côté serveur (*PHP*, *Python*, *Perl*, *Tcl*, etc.), nous nous concentrons principalement sur le client web et ce qui est interprété par le navigateur depuis des pages statiques.

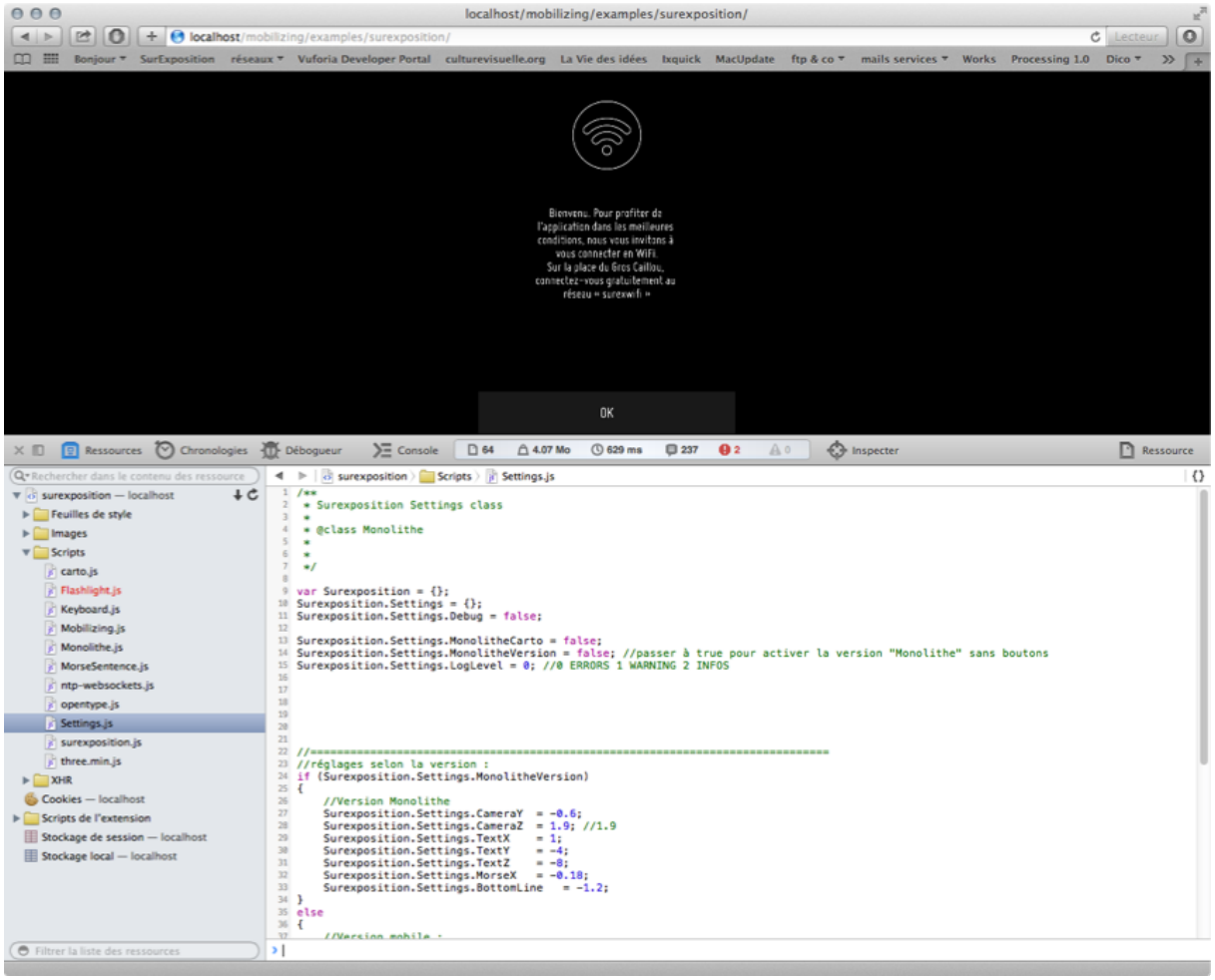

Figure 2 Capture d'écran d'une partie des outils développeurs du navigateur *Safari*, ici l'affichage des sources *JavaScript*.

L'écosystème technique du web prend une place de plus en plus grande dans le paysage technologique global. Les possibilités de prise en main par des non spécialistes grâce à l'accessibilité des sources, le grand nombre de ressources permettant l'apprentissage et une forme d'ubiquité du navigateur web jouent un rôle prépondérant dans cet état de fait. Mais il reste encore deux points qui méritent d'être évoqués concernant la place toujours plus importante qu'occupent ces technologies.

D'une part, parce que les ressources mises en œuvre dans les sites web sont aussi nombreuses que celles qu'un ordinateur peut traiter (images, vidéos, sons, textes, scripts, etc.), les navigateurs Internet sont devenus des sortes de reproductions logicielles des ordinateurs eux-mêmes, des systèmes d'exploitation de données conservées localement ou sur un serveur

distant, mais aussi, plus récemment, des composants matériels de l'ordinateur[17]. Devenus équivalents à des « machines virtuelles », les navigateurs sont aujourd'hui intégrés dans les systèmes d'exploitation sous la forme d'éléments de librairies logicielles, voire d'un *framework*[18], au même titre que d'autres composants logiciels de base inclus dans les kits de développement utilisés pour la création d'applications. Il est donc possible de les employer pour des tâches spécialisées en les utilisant comme composants d'affichage et de traitement de données dans des applications dont l'objectif diffère de la seule consultation de sites Internet. Cette dernière caractéristique revêt une importance particulière dans le champ de la création artistique sur les écrans mobiles, car elle permet l'extension des capacités du navigateur web par la création de ponts entre la couche native de l'OS hôte et le navigateur. En d'autres termes, lorsqu'un élément matériel de la machine sur laquelle le navigateur est exécuté n'est pas accessible par le navigateur, en particulier par l'intermédiaire de *JavaScript*, il est possible de créer un programme natif qui en ouvre l'accès et « l'expose » au contexte *JavaScript* en cours d'exécution dans le navigateur, pour ainsi étendre ses capacités. Cette technique n'est pas récente, elle est au cœur même des langages de programmation de type script, dont le principe est de permettre le contrôle d'un élément logiciel préexistant par une programmation interprétée, qu'il s'agisse d'un système d'exploitation complet (*AppleScript*) ou d'un logiciel applicatif quelconque (*JavaScript* dans la suite Adobe, *Lua* ou *Python* dans de nombreux logiciels d'image de synthèse, etc.), mais elle n'a été rendue officielle que tardivement sur les écrans mobiles[19]. Concrètement, cette technique d'extension du contexte *JavaScript* permet

[17] Nous faisons référence plus particulièrement ici à WebGL, implémentation d'OpenGL ES spécifiquement conçue pour les navigateurs web et qui permet l'accélération de l'affichage des contenus graphiques à l'aide d'un accès direct au processeur graphique de l'appareil par les scripts contenus dans les pages web. Cette accélération matérielle est actuellement possible sur toutes les plateformes, y compris mobiles. Les périphériques d'entrée vidéo, les cartes sons, les capteurs embarqués et tous les éléments matériels constituant la machine sont aussi concernés par cette évolution, du moins dans les spécifications de l'architecture Web.

[18] En programmation informatique, un framework est un kit de composants logiciels structurels qui sert à créer les fondations et les grandes lignes d'un environnement logiciel. Plus générique qu'une bibliothèque, un framework propose un cadre de travail au programmeur qui développe un logiciel particulier. Dans le cas de l'iPhone, il s'agit des composants fondamentaux du système d'exploitation qui pourront être utilisés par une application : fonctions fondamentales du système, gestion du son, de la vidéo, des images, etc. Parmi les nombreux *frameworks* disponibles, on trouve les *webView*, qui sont des navigateurs web sans interface graphique de contrôle mais disposant de toutes les fonctionnalités nécessaire à la consultation de pages web.

[19] En outre, *JavaScriptCore*, l'interpréteur et contexte d'exécution *JavaScript* d'Apple utilisé dans son navigateur *Safari*, n'est devenu publique et correctement documenté dans les SDK iOS et Mac OS X qu'en juin 2013, avec iOS 7.

d'ouvrir l'accès à de nouveaux capteurs embarqués dans des appareils de nouvelle génération alors même que leur mise en œuvre n'est pas prévue dans les spécifications du W3C[20] et sera donc impossible avec un navigateur standard.

D'autre part, on constate une très forte popularité du langage de programmation *JavaScript* qui tend à rendre la promotion des autres chaînes techniques plus difficile à faire dans le champ des pratiques artistiques de l'interactivité. En effet, dans un contexte où la plupart des acteurs de la scène des arts et technologies numériques orientent leurs développements vers les technologies du web[21], essentiellement parce qu'elles sont standardisées et répandues, ne pas adopter ces techniques pour en proposer d'autres potentiellement concurrentes n'a pas vraiment de sens. Cette popularité provient de différents facteurs. Les spécifications du HTML 5 ont beaucoup participé à cette popularité renouvelée (apparition de balise *vidéo* et *audio*, homogénéisation de l'interface de programmation, nouvelles possibilités de contrôle des éléments HTML via *JavaScript*, etc.). Mais c'est aussi l'apparition d'outils comme *Node.js*[22], plateforme de développement de serveur web entièrement programmable en *JavaScript*, qui accroît la notoriété de *JavaScript* comme langage de programmation « généraliste ». Grâce à des outils comme *Node.js* et à leur couplage avec les navigateurs web (y compris sur écrans mobiles), il est en quelque sorte envisageable de réaliser presque tous les types de projets interactifs à l'aide de *JavaScript*. Finalement, nous pouvons remarquer une forme d'uniformisation des syntaxes utilisées dans les langages de programmation. Comme nous l'avons noté plus haut, afin rendre le développement d'applications natives sur iOS plus attractif auprès de la communauté des développeurs, Apple a introduit un nouveau langage de programmation dans sa chaîne technique en 2014, *Swift*. Cette introduction, qui suit d'une année l'ouverture du *framework JavaScriptCore* dans les SDK d'Apple, peut être comprise comme une volonté d'ouverture afin de rendre la programmation d'applications natives plus accessible (mais on pourrait regretter qu'Apple n'est pas tout simplement choisi d'utiliser *JavaScript* comme langage de

[20] Le World Wide Web Consortium (http://www.w3.org) est un organisme de normalisation qui se donne pour mission de promouvoir et de standardiser les technologies du web.

[21] Nous pouvons prendre l'exemple du Projet CoSiMa, soutenu par l'ANR et auquel nous participons, qui vise à produire une plateforme logicielle pour l'interaction collaborative et collective sur la base des récents appareils mobiles et des technologies du web. http://cosima.ircam.fr. A une autre échelle, on peut aussi citer le portage en HTML5/JavaScript de *Processing* : *Processing.js* (http://processingjs.org)

[22] https://nodejs.org

programmation, comme l'a fait *Node.js*). L'exemple suivant, tiré du blog du développeur Mirco Zeiss, nous montre comment s'écrit la définition d'une classe basique en *EcamScript* 6 (ES 6, la prochaine version de *JavaScript* en cours d'adoption) et en langage *Swift* et met ainsi en avant le point auquel la syntaxe et certains concepts des deux langages peuvent être semblables.

```
//JavaScript ES 6
class Person {

  constructor(name) {
    this.name = name;
  }

  speak() {
    return 'Hey there, my name is ${this.name}';
  }

}

var me = new Person('Mirco');
me.speak();
     // Hey there, my name is Mirco


//Swift
class Person {

    var name: String

    init(name: String) {
       self.name = name
    }

    func speak() -> String {
       return "Hey there, my name is \(self.name)"
    }

}

var me = Person(name: "Mirco")
me.speak()
     // Hey there, my name is Mirco
```

(source : http://www.mircozeiss.com/swift-for-javascript-developers/, consulté le 13/07/2015)

Mobilizing vs Mobilizing.js : d'une mobilité à l'autre

Dans le cadre de notre pratique artistique et d'enseignements en école et université d'art, nous avons été confrontés aux difficultés techniques imposées par la chaîne de développement pour iOS (plus particulièrement, *Android* ayant été moins investi). C'est au cours de nos

expérimentations pratiques que nous avons imaginé la création d'un outil de prototypage pour les artistes désirant investir le riche potentiel de création interactive des écrans mobiles. Cet outil, intitulé *Mobilizing* et initié au sein du laboratoire de l'École Supérieure des Arts Décoratifs de Paris (EnsadLab), est devenu le centre de mes recherches artistiques et doctorales entre 2008 et 2013, ainsi qu'un vecteur d'enseignement de la programmation informatique pour les arts. *Mobilizing* était[23] un langage de programmation dont l'ambition était de permettre à des auteurs non spécialistes du développement informatique de réaliser eux-mêmes des applications sur écrans mobiles. Ainsi, *Mobilizing* permettait de programmer des applications interactives mettant en œuvre images, formes géométriques, textes, sons, séquences d'images animées et vidéos à l'aide d'un langage simple et intuitif. L'ensemble des capteurs disponibles dans les appareils mobiles étaient accessibles : accéléromètres, boussole numérique, capteur de proximité, écran tactile, récepteur GPS, etc. La seule plateforme supportée étant l'iOS, les applications créées à l'aide de *Mobilizing* pouvaient être exécutées sur iPod Touch, iPhone et iPad. Développé par Jonathan Tanant et moi-même, *Mobilizing* a été abandonné pour deux raisons principales. En premier lieu, étant un projet de recherche en esthétique, sciences et technologie des arts, *Mobilizing* n'avait pas vocation à prendre des dimensions industrielles et donc à se positionner comme un outil de référence dans le paysage des outils de création sur les écrans mobiles. La maintenance des sources de *Mobilizing* au fil des mises à jour de l'iOS a constitué une tâche suffisamment chronophage pour nous laisser supposer la quantité de travail induite par une diffusion plus large de l'outil et la gestion d'une communauté d'utilisateurs et nous n'avions pas les moyens temporels et financiers d'assumer une telle activité. En second lieu, le « tournant technologique » apporté progressivement par les techniques du web via HTML 5 et la prédominance grandissante de *JavaScript* nous ont poussés à reconsidérer notre projet, qui proposait un langage de script construit de zéro, avec une syntaxe mimant *Java* et *Processing,* mais qui n'avait pas l'efficacité des interpréteurs *JavaScript*. De plus, notre outil entrait de plus en plus en concurrence avec la chaîne technique du web, position qu'il n'était plus possible de tenir, pour les raisons que nous avons énoncées plus haut. La décision de récrire et de reconcevoir entièrement *Mobilizing* en *JavaScript* a donc été prise, donnant le point de départ de *Mobilizing.js*.

[23] Nous parlons au passé car la version 1.0 de *Mobilizing* n'est plus continuée, mais elle reste disponible à l'adresse suivante : http://dominiquecunin.acronie.org/mobilizing/

Si les deux projets que sont *Mobilizing* et *Mobilizing.js* partagent certaines caractéristiques, ils ne sont pas les mêmes, l'un n'étant pas le portage de son ancêtre dans un langage plus standard et connu. *Mobilizing.js* tente de profiter de l'expérience que nous avons accumulée lors de la recherche qu'a occasionnée *Mobilizing*, aussi bien dans le cadre de pratiques artistiques que dans celui d'enseignements. Entrer dans les détails techniques de l'implémentation de chaque projet pour en comparer les structures ne peut pas être fait ici, nous réservons pour d'autres textes la description technique précise de *Mobilizing.js* et des choix de design logiciel que nous avons décidés. Cependant, nous pouvons résumer les grandes caractéristiques des deux environnements pour les comparer et mieux percevoir les enjeux actuels de *Mobilizing.js*. La tableau ci-dessous met en regard les deux projets et leurs spécificités respectives.

Mobilizing (2008~2013)	***Mobilizing.js*** (2014~)
Langage de programmation pour l'expérimentation et la création d'œuvres interactives sur les écrans mobiles.	Librairie logicielle construite au-dessus des technologies du web pour la création d'œuvres interactives prenant en compte les supports mobiles.
Conçu à destination des artistes, des designers et des étudiants.	Conçu à destination des artistes et des designers et des étudiants.
Langage de script textuel « fait maison ».	*JavaScript*, possibilité d'ajout de surcouche en langage de programmation textuel ou visuel.
Librairie graphique 2D	Librairie graphique 3D temps réel, extensible par ajout de modules/plugin (système client-serveur, librairie audio, vidéo, etc.)
Ensemble de fonctions d'accès aux capteurs embarqués.	Ensemble de fonctions d'accès aux capteurs embarqués.
Compatibilité iOS exclusive	Compatibilité transversale avec tous les appareils qui contiennent un navigateur Web

Mobilizing désignait donc à la fois un langage de *script* et l'ensemble des outils logiciels qu'il permettait de mettre en œuvre sur la plateforme iOS. Les deux aspects les plus importants de ces outils étaient l'accès aux valeurs retournées par les capteurs embarqués et

un moteur graphique 2D. Le projet de *Mobilizing* reposait sur le constat que nous avons fait concernant la capacité des écrans mobiles à fournir des informations relatives à leur état physique propre à partir desquelles des interactivités peuvent être construites. C'est pourquoi l'usage des capteurs embarqués et leur mise en œuvre avec des objets graphiques en temps réel étaient le point central de nos préoccupations.

Mobilizing.js hérite de cette caractéristique et s'adresse sensiblement aux mêmes destinataires que *Mobilizing*, mais il tend à proposer plusieurs interfaces de programmation. La première reste celle de *JavaScript*, langage de script textuel, par-dessus laquelle d'autres interfaces peuvent être développées afin de permettre aux utilisateurs les plus novices d'apprendre la programmation en général, mais aussi afin de permettre des pratiques artistiques de la programmation différentes de celle proposée par les chaînes de développement classique (écriture, exécution, correction du programme dans des temps successifs et distincts les uns des autres). Entre autres, nous avons l'intention de produire un environnement de *live coding*[24] proposant une syntaxe simplifiée de *Mobilizing.js* et dont le principe, inspiré des outils historiques de cette mouvance de pratiques artistiques du numérique, serait d'écrire un programme pour en voir les effets immédiatement derrière les lignes de code, directement sur l'écran mobile. Ceci permettrait, d'une part, à des novices de se familiariser avec l'écriture de programmes dans l'objectif de produire des images interactives et, d'autre part, à des artistes de produire des performances *live* de programmation mettant en œuvre leurs univers graphiques et sonores. *Mobilizing* cherchait à situer et favoriser une exploration du potentiel esthétique de la mobilité par les spécificités techniques des écrans mobiles. Il s'agissait, en quelque sorte, de *mobiliser* les écrans mobiles dans les pratiques artistiques de l'interactivité. Tout en portant ce même projet, *Mobilizing.js* cherche à rendre « mobiles » des projets artistiques interactifs à travers les appareils

[24] Le *live coding* par des langages de programmation textuels est une pratique qui s'est consolidée au début des années 2000 avec l'augmentation de la puissance de calcul des ordinateurs et l'introduction de langage de scripts textuels permettant l'interprétation à la volée. Parmi les outils de programmation permettant cette pratique, nous pouvons citer *Fluxus*, un moteur de jeu vidéo en 3D temps réel scripté à la volé à l'aide du langage *Racket*, dérivé de *Scheme*, http://www.pawfal.org/fluxus/. A propos des pratiques artistiques œuvrant avec le *live coding*, consulter les textes suivants :
Nick Collins, *Live Coding of Consequence*, in Leonardo Volume 44, N° 3, juin 2011, pp. 207-211 PDF : http://users.sussex.ac.uk/~nc81/research/livecodingofconsequence.pdf;
Alex McLean, Dave Griffiths, Nick Collins et Geraint Wiggins, *Visualisation of live code*, Electronic Visualisation and the Arts (EVA 2010), 2010, PDF : http://www.bcs.org/upload/pdf/ewic_ev10_s2paper1.pdf.

numériques. Aujourd'hui, il nous semble que l'enjeu est de produire des outils de création *mobiles*, c'est pourquoi le noyau de *Mobilizing.js* est une librairie logicielle construite au-dessus des technologies du web permettant le déploiement sur les supports mobiles. Dans cette logique, si *Mobilizing.js* ne s'adresse pas exclusivement aux écrans mobiles, mais à toutes les formes d'ordinateur-écran (*smartWatch*, *smartTV*, consoles de jeux, etc.), nous imaginons qu'il puisse aussi être utilisé sur des appareils sans écrans, car à partir du moment où un interpréteur *JavaScript* est disponible, *Mobilizing.js* devrait pouvoir être utilisé (en dehors des fonctions graphiques qui n'auraient pas d'écran pour s'afficher).

Conclusion

La création d'œuvres interactives sur les supports numériques nécessite la mise en œuvre d'une forme de programmation informatique. Sans cela, la conception et la réalisation d'expériences esthétiques singulières utilisant les nombreuses capacités de captures des appareils mobiles, entres autres, ne peut être envisagée et « l'utilisateur » reste cantonné dans sa position de consommateur ou, au mieux, de producteur de médias. Parce qu'ils représentent un enjeu économique et technologique aussi vaste que crucial pour les constructeurs et les éditeurs de logiciels, les écrans mobiles subissent de nombreuses évolutions à un rythme très soutenu. Les conséquences de ces évolutions sur la création logicielle sont parfois lourdes, car lorsqu'un OS est mis à jour, son SDK l'ai aussi et si de nouvelles fonctionnalités font leurs apparitions et favorisent de nouvelles créations, certaines incompatibilités émergent également et remettent en cause les travaux réalisés antérieurement. La course à la mise à jour que l'on rencontre avec les chaînes de développement natif s'ajoute à la liste des divers obstacles que l'artiste non spécialiste de l'informatique rencontre et subit, dont le premier est probablement la maîtrise de la programmation informatique dans ses divers langages. Les technologies du web représentent aujourd'hui une alternative intéressante aux chaînes de développement natif, particulièrement à travers le langage *JavaScript*. En plus de permettre un développement unique pouvant être déployé sur de multiples plateformes, ces technologies sont abordables pour une grande communauté d'utilisateurs novices, car elles sont très largement diffusées et documentées.

Des projets comme *Espruino*[25], une forme de carte *Arduino* programmable en *JavaScript* et embarquant un interpréteur, tendent à confirmer plus encore l'ubiquité de *JavaScript* et l'intérêt de créer une librairie logicielle cohérente, dédiée à la création artistique interactive, qui deviendrait potentiellement elle-même ubiquitaire parce qu'elle profite de l'écosystème *JavaScript*. C'est sûr ces constats que nous avons basé notre choix technologique pour la création de *Mobilizing.js*, dans l'espoir que la stabilité apportée par *JavaScript*, toute relative soit-elle, persiste lors des prochains tournants technologiques qui arriveront inévitablement, tôt ou tard.

[25] http://www.espruino.com

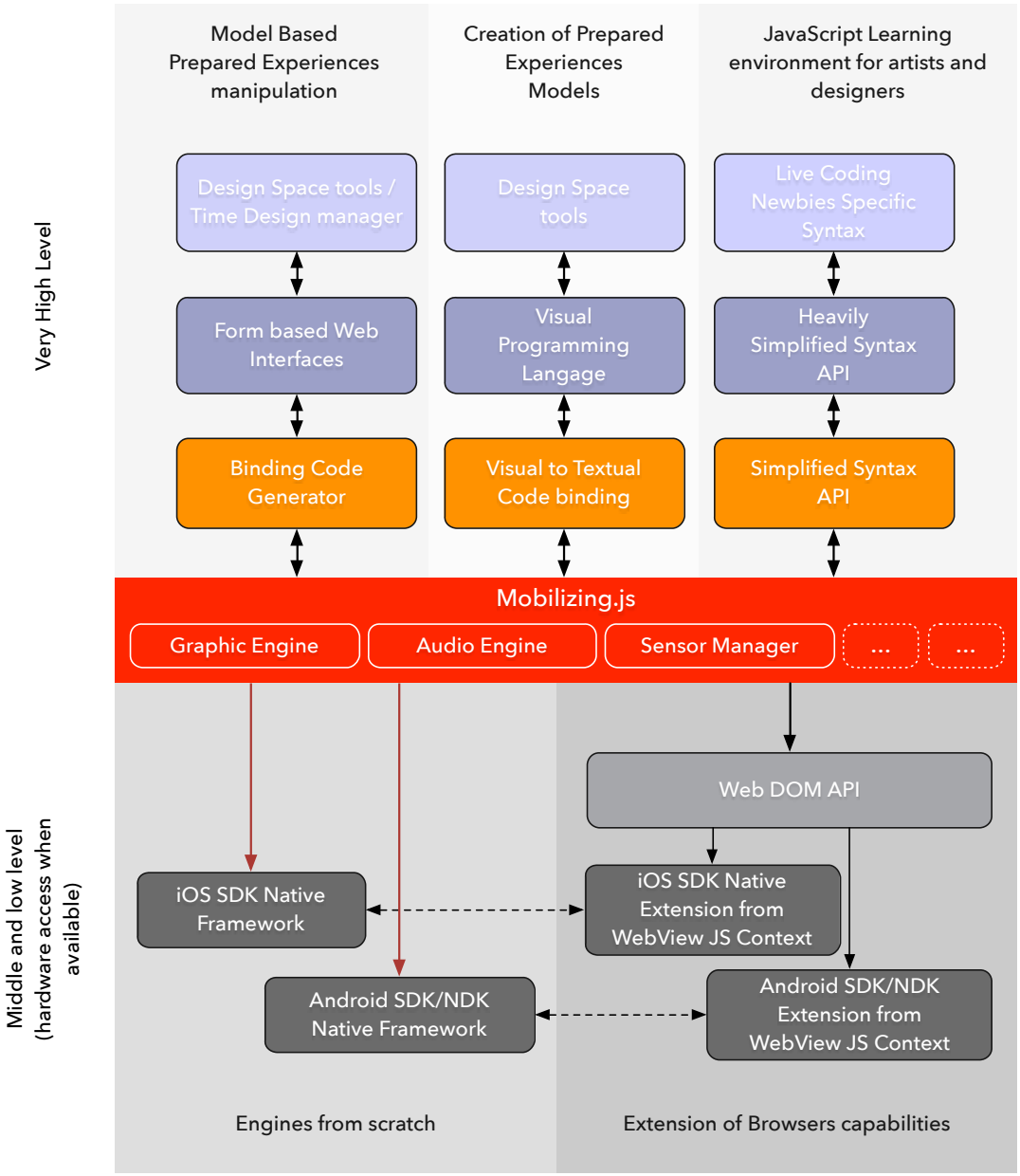

Figure 3 Représentation schématique des différentes couches constitutives de l'environnement logiciel de *Mobilizing.js*

À l'heure actuelle, *Mobilizing.js* est développé par Dominique Cunin et Jonathan Tanant, eux-mêmes auteurs de *Mobilizing*, avec l'aide de Oussama Mubarak. Il est porté par *EnsadLab* dans le cadre du projet CoSiMa, soutenu par l'ANR. Les projets artistiques de différents auteurs, dont Samuel Bianchini et Dominique Cunin, occasionnent l'intégration de

nouvelles fonctionnalités génériques. Le noyau de *Mobilizing.js* est open source. Le site officiel est http://www.mobilizing-js.net.