Silverlight et HTML 5

L'objet de cette note est de présenter les principales différences entre Silverlight et HTML 5 et d'expliciter en quoi **Silverlight apporte de nombreuses fonctionnalités complémentaires à HTML 5**.

A ce titre, il semble tout à fait justifié de considérer qu'**HTML 5 et Silverlight se complètent** et qu'il est donc tout à fait pertinent d'imaginer que **l'un et l'autre puissent cohabiter harmonieusement**.

> ***La stratégie de Microsoft est donc de continuer d'innover au travers de Silverlight tout en supportant HTML 5 dès que celui-ci sera standardisé par le W3C en permettant à HTML 5 et à Silverlight de cohabiter harmonieusement.***

1. HTML 5 n'est pas encore terminé

HTML 5 fera incontestablement franchir au Web une étape tout à fait significative. Pour autant, ce standard n'est pas encore terminé ; en effet, HTML 5 est actuellement une proposition de spécification en cours d'examen au sein du W3C.

Aujourd'hui, il y a de très belles démonstrations de certaines des fonctionnalités de HTML 5 qui démontrent que HTML 5 est tout à fait capable de créer des expériences utilisateurs dignes d'une application de type RIA (Rich Internet Application).

Cependant, la plupart des démonstrations de HTML 5 se focalisent autour de trois nouvelles fonctionnalités clés reposant sur les nouvelles balises <canvas>, <video> et <audio>. Pour autant, les spécifications de HTML 5 ne sauraient se limiter à ces fonctionnalités et sont, en fait, beaucoup plus larges que ce que laissent entrevoir ces quelques démonstrations. Ainsi que l'on peut le voir, le nombre de différences entre HTML 4 et HTML 5 est extrêmement significatif. Il n'est donc pas vraiment sérieux de vouloir analyser HTML 5 à la lumière de quelques démonstrations « sexy » pour en tirer la conclusion hâtive selon laquelle les *frameworks* RIA sont, d'ores et déjà morts. En effet, dans la réalité des choses, seule une petite fraction des spécifications HTML 5 est, d'ores et déjà implémentée par les différents navigateurs disponibles sur le marché.

Par ailleurs, les spécifications de HTML 5 sont encore en évolution et il y a encore eu des changements importants de ces spécifications ces dernières semaines. Certains ont même émis l'hypothèse selon laquelle il faudrait encore entre 3 et 5 ans pour implémenter l'ensemble des spécifications de HTML 5. Il semble cependant plus raisonnable de penser que certaines sociétés qui ont largement investi autour de HTML 5/WebKit[1] telles qu'Apple ou Google iront plus vite que cela. De même Microsoft qui a annoncé son implication très significative autour du support de la standardisation de HTML avec l'engagement direct du *test lead* d'IE9 au sein de la *task force* en charge de la standardisation des tests de HTML 5[2].

Pour autant, il semble nécessaire de finaliser d'abord les spécifications de HTML 5 avant de se livrer à des spéculations sur le temps qu'il faudra pour les implémenter toutes. Par ailleurs, il est certes intéressant de jouer avec quelques démonstrations et quelques fonctionnalités ; pour autant, il n'est guère prudent de commencer de déployer des fonctionnalités de HTML 5 avant que les

[1] WebKit est une bibliothèque de fonctions permettant aux développeurs d'intégrer facilement un moteur de rendu de pages Web dans leurs logiciels. Elle est disponible sous licence BSD et GNU LGPL. Originellement réservée au système d'exploitation Mac OS X (à partir de la version 10.3 Panther), elle a été portée vers Linux et Windows.

[2] http://blogs.msdn.com/ie/archive/2010/03/09/Working-with-the-HTML5-Community.aspx.

spécifications ne soient terminées. Ainsi, il est plus que probable que celles-ci seront implémentées en plusieurs phases et que, dans 2 ans environ, on verra les navigateurs disponibles sur le marché annoncer leur compatibilité totale avec HTML 5. Certains navigateurs tels que Chrome et Safari (qui sont fondés sur le WebKit) implémenteront probablement de manière progressive les fonctionnalités de HTML 5 au sein du WebKit, fonctionnalités qui seront ensuite incluses au sein de ces navigateurs. D'ici-là, il semble donc prudent de ne pas se laisser prendre au battage médiatique autour de HTML 5, même si, encore une fois, HTML 5 va permettre de faire franchir au Web une étape déterminante.

Il est important d'ailleurs de noter qu'il y a encore beaucoup de débats autour des spécifications de HTML 5. Ainsi, on trouvera au sein de cet article une description de certains des problèmes sous-jacents ; on notera en particulier les commentaires de Dorian Taylor à propos de l'importante dépendance de JavaScript dont HTML 5 est entaché et sur laquelle nous reviendrons plus loin.

2. Limitations des balises audio et vidéo

Un des ajouts les plus significatifs aux spécifications de HTML 5 consiste en l'apparition de balises permettant le support d'éléments audio et vidéo et qui permettent d'embarquer des fichiers multimédias directement au sein du balisage HTML. Le support de ces balises est, d'ores et déjà, présent au sein de la Platform Preview d'IE9.

Ceci permet d'utiliser les fichiers multimédias d'une manière très proche de la façon dont on utilise la balise <img> aujourd'hui. Il suffit de fournir l'emplacement de la ressource multimédia pour cela (des attributs additionnels permettent également de modifier le comportement des éléments multimédias, par exemple en mettant en œuvre le streaming). Ceci offre un moyen simple et standard d'améliorer une page HTML avec du contenu multimédia.

Exemple :

```
<video src= "movie.mp4" controls= "controls">
Votre navigateur ne supporte pas la balise video
</video>
```

Toutefois, une véritable expérience multimédia va bien au-delà du fait de pouvoir jouer un fichier audio ou vidéo. Voici, en effet, quelques limitations du support des éléments audio et vidéo au sein des spécifications HTML 5 ainsi qu'une comparaison avec Silverlight.

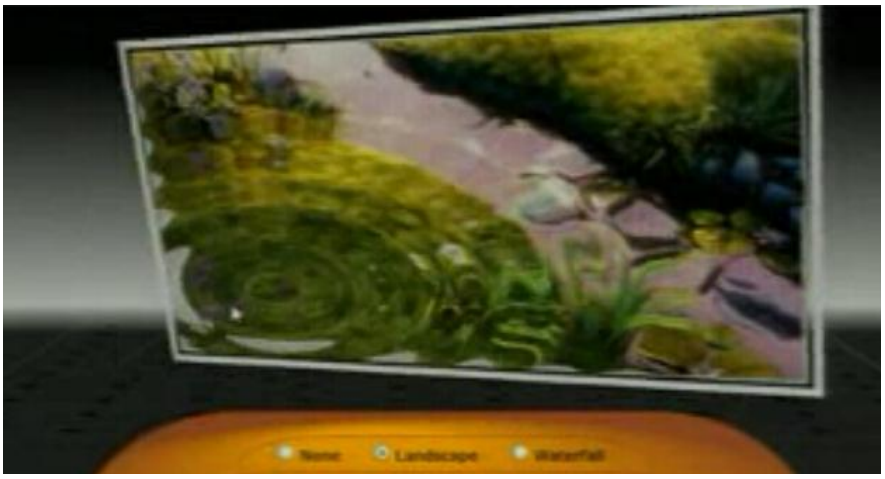
La manipulation et l'inspection en temps réel des fichiers audio n'est pas possible

Au sein de HTML 5, il est possible de placer des options additionnelles telle que *autoplay* ou *buffering* mais il n'est pas possible de manipuler les flux audio et de fournir vos propres modifications, par exemple grâce à un équaliseur. Silverlight (et Flash), quant à eux, le permettent. Ainsi, il semble peu probable que la balise audio de HTML 5 soit considérée comme suffisante pour les scénarios un peu avancés.

Limitation de la balise <video>

Au niveau de la vidéo, les mêmes limitations s'appliquent ; il est possible de placer des options telles que *autoplay* ou *buffering* mais les options un peu plus avancées font cruellement défaut. Ainsi, avec Silverlight, on dispose d'un large ensemble d'options telles que la possibilité d'appliquer directement des *shaders* sur la vidéo ou sur le streaming en HD 1080p. Ces options permettent la mise en place d'une expérience vidéo beaucoup plus riche.

Ainsi, une image de la vidéo HD Silverlight suivante sur laquelle une transformation 3D et un ombrage de pixel sont appliqués en temps réels (conférence Mix 2009) :



Pas de possibilités de type magnétoscope ou de type *smooth streaming*

Il est possible de créer un site de visualisation de vidéos tel que YouTube avec HTML 5. En fait, c'est déjà une expérience en cours par YouTube. Toutefois, si l'on veut créer une expérience riche telle que celle que l'on peut mettre en œuvre sur un magnétoscope numérique, il est nécessaire d'utiliser une technologie telle que Silverlight. En effet, la fonctionnalité de *smooth streaming* HD de Silverlight permet le déplacement image par image, la mise en pause de la vidéo, le rembobinage, la diminution de la vitesse ou du streaming quand la bande passante ralentit. Avec le *smooth streaming*, la *bufferisation* est minimisée et l'on peut passer à différentes portions de la vidéo en quasi temps-réel. Il est possible d'essayer le *IIS smooth streaming* Silverlight ici (on remarquera au passage que l'expérience est bien meilleure que celle que l'on obtient sur Youtube). Cette technologie de *smooth streaming* Silverlight a d'ailleurs été jugée suffisamment intéressante pour être utilisée lors des jeux olympiques et par Netfix pour fournir de la vidéo HD en 720p à des centaines de millions de consommateurs. C'est toute la différence entre le streaming dans un seul sens depuis un serveur vers un poste de travail (le téléchargement progressif de HTML 5 avec un simple « play, pause, stop ») et la mise en place d'un streaming dans les deux sens avec lequel l'utilisateur peut interagir (comme sur un magnétoscope). A notre sens, la richesse d'une telle expérience fait toute la différence car elle permet d'impliquer réellement l'utilisateur dans l'expérience ; et d'ailleurs les statistiques d'utilisation prouvent que la possibilité de mettre en place une telle expérience utilisateur multiplie par 3 le temps pendant lequel les utilisateurs regardent la vidéo sur un site.

Pas de protection de contenu/DRM au sein de la balise <video> de HTML 5

Ceci est problème très important quand il s'agit d'héberger des contenus sensibles ou soumis à des droits d'auteur et que vous ne voulez pas que les utilisateurs aient la possibilité de copier ou de sauvegarder ce contenu sur leur disque dur. Ainsi, il est difficile d'imaginer qu'un service de vidéo à la demande puisse diffuser un film récent que n'importe qui puisse ensuite copier sur son disque dur[3] ; tant qu'il n'y a pas eu une évolution de la balise <video> au sein de HTML 5, il est plus que probable que cette balise ne pourra être utilisée que pour des contenus non sensibles ou non

[3] Pour vous en convaincre, il vous suffit d'aller sur YouTube et de jouer une vidéo. Ensuite, vous allez dans le répertoire des fichiers temporaires de votre navigateur Internet et vous recherchez un fichier qui a été généré aux environs du moment où vous avez regardé la vidéo (celle-ci doit être parmi les plus gros fichiers). Ensuite, vous renommez le fichier en question pour lui donner l'extension « .flv » et le fichier est à vous… Ceci est rendu possible parce que le téléchargement progressif stocke temporairement le fichier vidéo sur votre disque dur. C'est ce qui rend possible la création d'outils très simples tels que downloadyoutubevideos.com…

protégés. Par comparaison, Silverlight supporte une large variété d'architectures de gestion de droits numériques et d'options de chiffrement.

Note : Silverlight n'est pas la seule technologie qui permette de faire du s*mooth streaming*. Ainsi Quicktime Streaming ou Flash Media Server d'Adobe sont deux exemples de technologies qui offrent des fonctionnalités de *smooth streaming adaptatif* et de magnétoscope numérique. On peut cependant noter que *Silverlight IIS Smooth Streaming* est un *add-on* gratuit de Windows Server 2008 IIS 7.x.

3. Les limitations de JavaScript

Si vous avez fait du développement Web, il est très probable que vous avez eu à utiliser JavaScript pour fournir un contenu dynamique au navigateur client. Or, JavaScript souffre d'un certain nombre de limitations dont on parle peu et qui seront encore exacerbées dans un contexte où la mise en œuvre de HTML 5 peut pousser à l'utiliser encore davantage.

JavaScript ne fournit pas de véritable parallélisme

Les postes de travail d'aujourd'hui, que ce soit des PC de bureau ou des PC portables mettent en œuvre de plus en plus de cœurs au sein des CPU (aujourd'hui 2 cœurs sur un portable ou 4 cœurs sur un PC de bureau sont très largement répandus). Même les futurs téléphones, tels que ceux permis par les prochaines versions de l'iPhone ou par le prochain Windows Phone 7 laissent présager de l'utilisation de plusieurs cœurs. Dans un tel cadre, il semble difficile de reposer sur un langage ou un *framework* qui ne peut pas tirer parti de tous les cœurs. Dans le cas de Silverlight et du .NET framework, il est possible de passer à l'échelle jusqu'à 8 cœurs. Or, ceci joue évidemment un rôle très important en termes de performance. On peut certes imaginer que les *Web Workers* pour JavaScript au sein des navigateurs Web puissent éventuellement fournir la possibilité d'utiliser plusieurs threads. Le problème, c'est que le JavaScript est une sorte d'hybride entre un langage impératif[4] et un langage déclaratif. Ainsi, la parallélisation des boucles en JavaScript n'est pas possible sans une réécriture. Par comparaison, en utilisant .NET, il est possible d'utiliser des langages fonctionnels tels que LINQ ou F# pour programmer facilement des tâches en parallèle.

JavaScript est à mi-chemin entre un langage fonctionnel et un langage orienté objet

JavaScript n'implémente que certains des paradigmes des langages fonctionnels et des langages orientés objet. Le gros intérêt de tout ce qui est fondé sur le .NET Framework et les notions de CLR (*Common Language Runtime*) et de DLR (*Dynamic Language Runtime*) c'est que le contenu dynamique fourni par Silverlight peut l'être en utilisant aussi bien par exemple C#, VB.NET, F# (basés sur la CLR) que Iron Python (langage dynamique basé sur la DLR). Ceci fournit une très grande flexibilité que JavaScript ne permet pas. Il est ainsi possible de choisir le langage correspondant à la tâche que l'on désire accomplir ou aux compétences des différents membres de l'équipe de développement.

JavaScript limite le partage de code

En effet, la question que l'on peut se poser est la suivante : « avez-vous du code correspondant à un composant serveur écrit en JavaScript ? » La réponse est très certainement non car on n'utilise pas vraiment de JavaScript sur les serveurs. Par contre, si vous utilisez un environnement de développement .NET, vous avez très probablement déjà des règles métiers ou des composants écrits dans un langage .NET et, ce qui est très avantageux dans ce cadre, c'est que vous pouvez réutiliser

[4] La programmation impérative est un paradigme de programmation qui décrit les opérations en termes de séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme (source Wikipedia).

certaines briques de ce code dans des environnements Silverlight client. Un autre exemple de partage de code autour de Silverlight est celui de WPF : dans la mesure où ces deux technologies permettent de nombreuses synergies, une large majorité de code peut être partagé entre Silverlight et WPF. Et, dans un futur très proche, il pourra en être de même dans des environnements Mobile.

Javascript n'est pas très performant

En dehors de quelques exceptions très minoritaires, une application bien écrite en JavaScript tournera toujours plus lentement qu'une application équivalente, bien écrite en Silverlight. Même si de très grands progrès ont été réalisés avec les dernières évolutions des moteurs de JavaScript (cf. les progrès réalisés avec Internet Explorer 9 dans sa Platform Preview.

Sécurité du code

JavaScript souffre du fait que le code peut être inspecté facilement (outils de développement fournis en standard avec IE8 ou accès à la visualisation des sources avec les versions antérieures ou les autres navigateurs). Silverlight souffre du même inconvénient, à ceci près que le code Silverlight est un peu plus difficile à inspecter. En effet, il faut généralement trouver les *assemblies* placées en cache et utiliser un outil tel que .NET Reflector pour les désassembler (Silverlight Spy est un autre outil qui permet d'inspecter le code Silverlight au fil de l'eau). Ainsi, si l'on désire effectuer un rétro-engineering d'une application Silverlight, il est nécessaire de passer à travers à travers davantage d'étapes.

Pour autant, s'il est nécessaire de déployer du code JavaScript ou Silverlight sur un client, il nous parait nécessaire de recommander l'utilisation d'un « obfuscateur » afin de rendre le travail de l'éventuel pirate un peu plus compliqué. Dans tous les cas, si le code qui doit s'exécuter contient de la propriété intellectuelle, nous recommandons de ne pas placer ce code du côté client.

4. Productivité

Nous définissons dans la suite la productivité comme l'ensemble des outils d'architecture et de conception, les *frameworks* et les outils de développement qui autorisent une meilleure efficacité dans la conception, le développement et le maintien en condition opérationnelle. HTML, CSS et JavaScript souffrent, à notre sens, d'un certain nombre de problèmes de productivité. C'est d'ailleurs la raison pour laquelle la plupart des applications de taille significative n'ont bien souvent pas été écrites directement en JavaScript ; ainsi, même si cela se sait peu, les Office Web Apps ou Google Wave ont été écrites en langage de haut niveau tels que C# ou Java pour être ensuite « compilées » en JavaScript.

Problèmes HTML et CSS

CSS est un langage qui permet d'enrichir HTML avec des styles. Cet ensemble fonctionne très bien en théorie. Dans la pratique, avec l'évolution du Web en direction d'une conception HTML/CSS plus fluide (c'est-à-dire, de facto, la généralisation des tables « div » avec CSS), les outils du Web ont des difficultés à suivre. De plus, le fait que certains navigateurs ne supportent pas toujours l'intégralité du standard CSS 2.x et ont conduit des sociétés comme Microsoft à soumettre des jeux de tests significatifs au W3C afin d'améliorer ce point ou encore à étendre de manière significative ce standard, ce qui n'est pas sans poser des problèmes de compatibilité entre les différents navigateurs. Ceci a d'ailleurs souvent conduit les développeurs à devoir se plonger manuellement dans l'écriture du CSS (ce n'est pas nécessairement une mauvaise chose en soit mais ce n'est guère productif). D'une manière générale, CSS n'est pas un langage très élégant et, par exemple, il est très difficile de travailler avec de grosses feuilles de calcul. Il suffit de regarder l'utilisation de CSS dans un environnement SharePoint 2007 pour s'en convaincre. C'est la raison pour laquelle des développements tels que le CSS Metaframeworks ont vu le jour afin de permettre la mise en place de fonctionnalités telles que l'héritage des styles CSS.

D'une manière générale, il y a souvent la tendance naturelle qui existe aujourd'hui dans le monde du développement Web à utiliser un certain nombre de *frameworks* Open Source pour pallier les insuffisances de JavaScript/CSS. Ainsi, on utilise jQuery pour améliorer le développement HTML DOM ou encore jLinq pour faire le pendant de .NET LINQ ou encore JS Charts afin de réaliser des graphiques en JavaScript. Au final, cela pose des problèmes de productivité et de maintenabilité :

- Les *frameworks* Open Source ou commerciaux s'intègrent rarement de manière simple au sein des outils de conception Web (certains *frameworks* comme jQuery qui sont très connus, sont toutefois capables de s'intégrer au sein de Visual Studio 2008/2010 – Microsoft vient d'ailleurs d'annoncer son intention de contribuer davantage à ce *framework* et d'en amplifier la prise en charge).
- Il est bien évidemment nécessaire de maintenir ces *frameworks* et il devient très difficile, dans le cas de grands projets, de jongler entre les différents *frameworks* complémentaires. Dans tous les cas, le suivi de ces *frameworks*, leur intégration dans l'environnement de développement, leur maintenance et leur intégration consomme pas mal de temps.
- Il n'y a pas vraiment d'outils de design purement HTML/CSS/JavaScript qui permette à un designer de concevoir son environnement graphique (*canvas*) et de pouvoir ensuite générer un résultat qui puisse être transmis au développeur ; ceci rend le processus de design des pages Web beaucoup plus difficile pour les non-développeurs que sont traditionnellement les designers.

Problèmes de compatibilité inter-navigateurs

Les problèmes de compatibilité inter-navigateurs sont toujours des problèmes complexes ; il suffit de voir la récente décision prise par Google d'abandonner le support d'Internet Explorer 6 pour s'en convaincre.

Développement de contrôles et des composants IHM

L'environnement et le support permettant de créer des composants IHM réutilisables en JavaScript est virtuellement inexistant. Cela nécessite dans la pratique l'implication des développeurs afin d'écrire du JavaScript qui génère du Dynamic HTML.

Intégration avec les technologies serveur

Travailler avec des technologies serveur telles qu'ASP.NET ou PDP ajoute un autre niveau de complexité à HTML. Ceci dégrade encore la productivité car il est nécessaire de gérer l'état local du client entre les appels serveur (cookies, cache local, etc.). En fait, même des outils comme Visual Studio n'arrivent pas à complètement masquer cette complexité pour un designer qui est bien souvent très loin de ces préoccupations d'intégration des technologies client et serveur.

Le *framework* Silverlight permet, quant à lui, de gérer cette problématique de productivité grâce à un ensemble cohérent d'outils qui permet de couvrir les besoins allant du designer au développeur tout en s'appuyant sur une riche fondation .NET.

5. HTML 5 ne cible que le Web

HTML 5 est un langage balisé pour le Web. Ainsi, il est nécessaire de disposer d'un navigateur pour interpréter le contenu HTML. Silverlight permet d'écrire une fois et de cibler le poste de travail, le Web et, maintenant, Windows Phone 7, Symbian et Android. Des fonctionnalités supplémentaires telles que le support du *multi-touch* permettent de tirer parti des nouveautés du hardware de manière native sans avoir à se reposer sur le navigateur.

Note : Il existe une technologie telle que Prism de Mozilla qui permet d'installer des sites HTML/JavaScript localement sur le poste de travail mais cette technologie ne semble pas très populaire et ne rencontre guère de succès commercial.

6. Support de l'accélération graphique (support GPU) ou du 3D

L'état actuel de la spécification HTML 5 ne permet pas explicitement de décharger le rendu graphique en direction du GPU. Il est cependant possible de commencer à supporter l'accélération graphique pour le rendu de HTML 5, de CSS 3 et de SVG 1.1 : c'est ce qui est d'ores et déjà disponible au sein de la Platform Preview d'IE9. Nous avons en effet revu le cœur d'Internet Explorer 9 de manière à ce qu'il utilise l'accélération matérielle fournie par le contrôleur graphique. Ainsi le moteur de rendu d'IE9 utilise le GPU pour l'ensemble des graphiques et texte présents dans les pages Web. IE9 envoie ainsi les calculs graphiques traditionnellement gérés par le CPU vers un composant plus rapide et plus spécialisé. De même, les balises <video> et <audio> bénéficient d'une telle accélération. Tout cela couplé au nouveau moteur JavaScript multi-cœurs, on se rend ainsi compte qu'IE9 tente de tirer le meilleur parti de la machine de l'utilisateur.

Pour se rendre compte de l'intérêt d'une telle accélération, il suffit d'utiliser un test appelé en anglais Flying Images qui est l'un des exemples de la suite d'IE9. Lorsque l'on lance le test Flying Images sur différent navigateurs, il est facile de constater qu'Internet Explorer 9 peut afficher des centaines d'images à pleine vitesse là ou d'autres navigateurs, Internet Explorer 8 inclus, arrivent bien plus vite à saturation complète.

D'une manière générale, la plupart des navigateurs du marché de permettent pas de disposer de ce type d'accélération puisqu'à notre connaissance, seule une *build* récente de Firefox 3.7a3 (version préliminaire) qui implémente un rendu au-dessus de Direct2D (technologie utilisée par IE9 pour l'accélération matérielle), permet de mettre en œuvre une telle accélération matérielle.

Silverlight, qui supporte de manière native la possibilité de profiter à plein des capacités d'accélération matérielle, apporte donc aux navigateurs autres qu'IE9, un réel plus en termes de performances.

Il y a également des plans pour supporter des technologies 3D dans le futur grâce à la technologie WebGL utilisant les *canvas* HTML 5. WebGL devrait ainsi fournir à HTML/JavaScript le support de la 3D, de l'accélération hardware et des *pixel shaders*. SilverLight (version 3), quant à lui, supporte déjà les projections 3D, l'accélération matérielle et Pixel Shader 2.0. De plus, dans un tel cadre, l'accélération GPU n'est pas limitée à juste un contrôle de la mise en page contrairement à HTML 5 qui ne supporte que les *canvas*. Il y a, par ailleurs, de nombreux moteurs 3D et physiques qui supportent de manière native Silverlight.

Un des aspects, à notre sens négatif, de l'implémentation WebGL est qu'elle nécessite toujours l'utilisation de JavaScript pour fournir un contenu dynamique au Canvas 3D. Ceci peut fonctionner sans problème pour réaliser des démonstrations simples mais il semble difficile d'imaginer de construire des scènes 3D interactives de cette façon. En effet, réaliser des calculs vectoriels 2D ou 3D ou faire du calcul matriciel en JavaScript n'est guère performant.

Par ailleurs, il faudra probablement un peu de temps avant que l'on ne dispose d'un large support de WebGL dans la mesure où ces spécifications sont encore en évolution. La prochaine version 3.7 de Firefox supportera sans doute WebGL dans des environnements de type Canvas 3D. D'autres navigateurs tels que Chrome, Safari offriront probablement également un tel support au travers de l'évolution du WebKit mais il faudra probablement encore une année ou deux dans la mesure où il y a encore des évolutions importantes en cours qu'il faudra implémenter et tester. Par ailleurs, dans la

mesure où WebGL est une évolution des *canvas* HTML 5, WebGL risque d'être à son tour influencé par les évolutions de la balise <canvas> de HTML 5.

7. Pas de support de webcam ou de microphone

Le support intégré au navigateur de tels périphériques constitue très certainement un élément important si l'on prend en considération l'extraordinaire développement des réseaux sociaux, de la collaboration et des applications multimédia. Malheureusement, les spécifications de HTML 5 ne prennent pas en compte ces types de périphérique. Il y a bien un plan pour ajouter le support de tels périphériques mais seulement après HTML 5 (5.01 ou 5.1). Par comparaison, Silverlight 4 supporte déjà ces deux types de périphériques.

8. Pas de modèle publication/souscription évolué

Aujourd'hui, le Web est fondé sur une architecture de type requête-réponse : vous naviguez depuis votre navigateur vers un URL et vous recevez une réponse. Une fois que cette réponse a été envoyée, le contenu est statique. La question qui se pose est donc de savoir ce qui se passe quand le serveur dispose d'un contenu nouveau ou est mis à jour. Il est alors possible d'exécuter du code JavaScript additionnel pour mettre à jour le contenu (appel Ajax) ; l'utilisateur peut aussi cliquer manuellement ou rafraichir le contenu. Ceci fonctionne mais reste lent et nécessite pas mal de code supplémentaire. Il serait donc préférable que le client puisse souscrire au contenu offert par le serveur ; de cette façon, à chaque fois que le contenu du serveur est modifié, une mise à jour automatique est envoyée pour mettre à jour les clients qui ont souscrit à cette notification. Silverlight offre un support élaboré d'un tel modèle qui s'exécute au-dessus du protocole net.tcp ainsi qu'en mode *WCF duplex polling*. Bien évidemment les performances d'un tel modèle sont très nettement supérieures. Il est possible de consulter un excellent exemple d'une telle mise en œuvre en environnement net.tcp ici.

En guise de conclusion

A la lecture de ces quelques éléments, il semble que HTML 5 soit loin de supporter tout ce qu'il est d'ores et déjà possible de réaliser à l'aide de Silverlight. Ainsi, les nouvelles fonctionnalités HTML telles que les balises *video, audio, canvas*, les *Web Workers*, WebGL, le support futur des balises *device* sont autant d'éléments qui manquent aujourd'hui à HTML et qui existent déjà en environnement Silverlight. Ceci, d'autant plus qu'il faudra très certainement encore un peu de temps pour que soit finalisées les spécifications de HTML 5, sans même parler de leur implémentation.

Toutes ces raisons nous semblent militer pour la mise en place d'une stratégie selon laquelle il nous parait pertinent de **continuer d'innover au travers de Silverlight tout en supportant HTML 5 dès que celui-ci sera standardisé par le W3C et en permettant à HTML 5 et à Silverlight de cohabiter harmonieusement**.