

Introduction au XUL à travers le framework Mozilla XPFE

par Maxime Alexandre ([homepage](#))

Date de publication : 10/08/2007

- I - Introduction
- II - Nom de code : XPFE
- III - La genèse
- IV - Un framework
- V - XPCOM API
- VI - Script Layer
- VII - UI Toolkit
- VIII - Applications / Extensions
- IX - Bilan
- X - Conclusion

I - Introduction

J'ai choisi de vous parler d'une technologie qui est à la fois innovante, multi plateforme, ouverte et gratuite : le framework Mozilla. Ce framework regroupe principalement le moteur Gecko, l'API XPCOM, le langage XUL, ainsi que plusieurs technologies standardisées intégrées par la communauté Mozilla. J'ajouterai mon avis (impartial, je l'espère) ainsi que mon retour d'expérience afin de vous donner une meilleure vision sur cette nouvelle technologie.

II - Nom de code : XPFE



Bienvenue dans le monde de XPFE (Cross Platform Front End), imaginé par la [Fondation Mozilla](#). Cette technologie est assez peu connue du public et des professionnels, du moins sous ce nom, car aujourd'hui tout le monde connaît Firefox, le navigateur Internet intégralement réalisé grâce à cette technologie.

Le but du framework, comme je vais vous le décrire dans la suite du document, est de fournir des outils de développement d'applications. Vous pouvez donc utiliser XPFE pour créer des logiciels avec comme ligne de mire l'utilisation des standards existants comme XML, RDF, HTML, CSS, JavaScript, etc. Autres attraits non négligeables, le code spécifique à écrire pour chaque plateforme est réduit au minimum et XPFE est disponible pour les systèmes Linux, Mac et Windows.

III - La genèse

Comme tout projet, XPFE a été conçu pour répondre à un besoin. Il faut se replacer dans le contexte original pour mieux comprendre. Entre 1995 et 1998, la guerre fait rage entre deux navigateurs Internet : Microsoft Internet Explorer et Netscape Communicator. Au fur et à mesure, Internet Explorer va lentement prendre la place du navigateur de Netscape. Cette progression est principalement due au succès grandissant du système d'exploitation de Microsoft : Windows.

Fin 1998, Netscape est à bout de souffle et se fait racheter par AOL (American Online). Internet Explorer représente alors 96% du marché des navigateurs. En réaction, des anciens employés de Netscape, motivés pour continuer l'aventure, décident d'ouvrir le code source de leur application fétiche à la communauté Open Source. Le projet est alors baptisé Mozilla et prend un nouveau visage.

 *Mozilla - contraction de "Mosaic " et "Godzilla" - représente le nom de code du projet Netscape Communicator.*

Ce groupe de personnes forme alors « The Mozilla Fondation » dont le but est d'apporter une identité au projet, représentant ainsi une des grandes communautés Open Source. L'idée est de développer un navigateur Internet (Firefox) et un client e-mail de nouvelle génération (Thunderbird).

Les objectifs sont nombreux, et le projet hérite de l'expérience de Netscape :

- Licence Open Source
- Multi plateforme
- Respect maximal des standards approuvés du W3C
- Réutilisation de code entre les différentes applications
- Langage d'interface de haut niveau

*

¹ <http://www.mozilla.org>

IV - Un framework

XPFE, comme son nom l'indique, permet de développer des applications multi plateformes. Il regroupe plusieurs sous langages et protocoles, chacun essayant de faire sa propre et unique tâche au mieux, dans la logique des services UNIX. Parcourons ce framework, des couches basses vers les couches de haut niveau.

Gecko


Gecko est le coeur du système. Il gère la partie graphique à présenter à l'utilisateur et intègre plusieurs fonctionnalités :

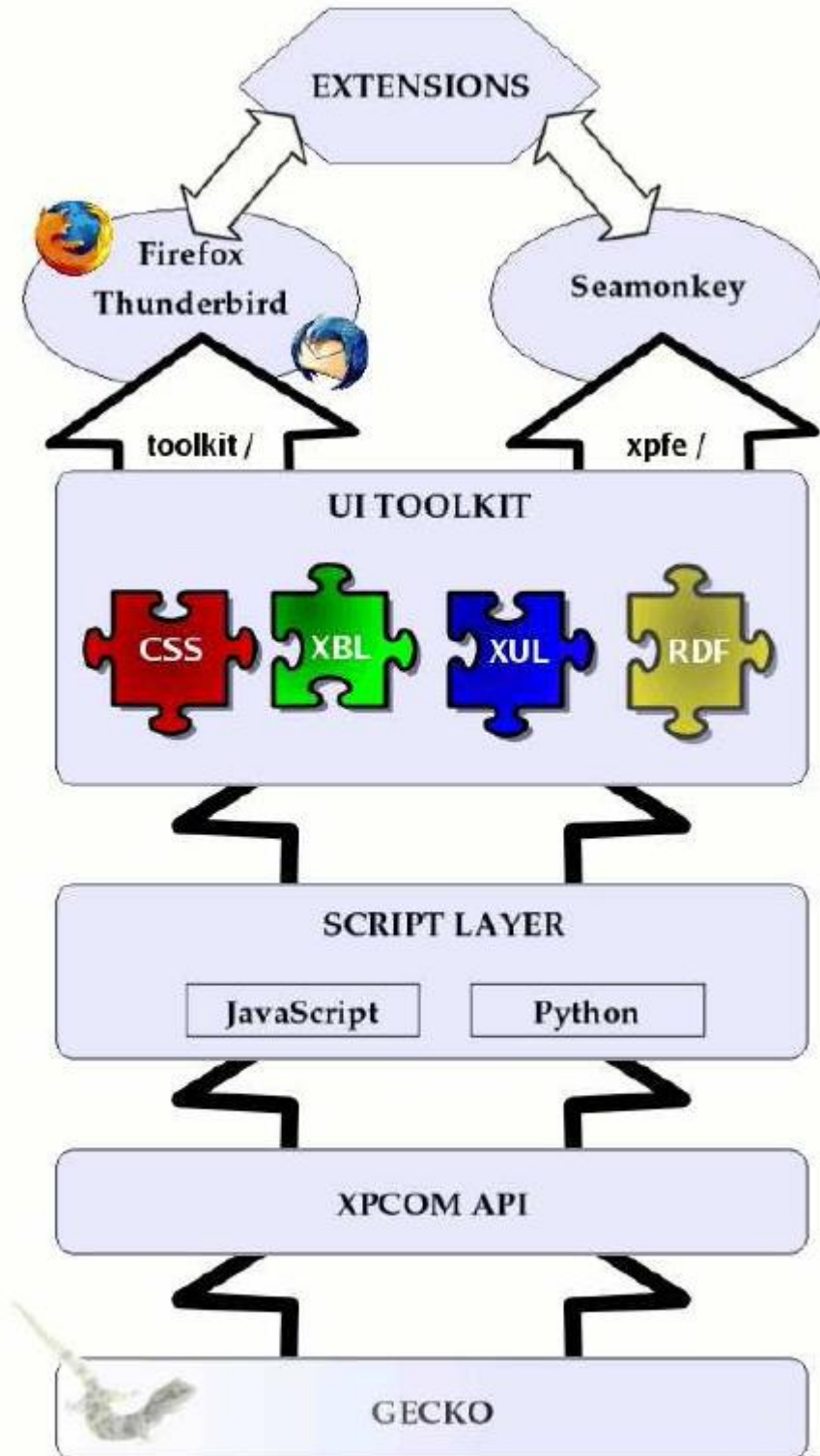
- le moteur de rendu
- le moteur DOM
- l'implémentation de CSS et la gestion des styles

Il a été conçu pour supporter les standards Internet du W3C. Il comprend donc, dans sa version actuelle 1.8.1 :

- HTML 4.0
- XUL
- CSS Niveau 1 (et support partiel des versions 2 et 3)
- JavaScript 1.7 (ECMAScript)
- DOM Niveau 1 et 2 (et support partiel du DOM 3)
- XML 1.0
- XHTML 1.1
- MathML
- XForms
- SVG
- RDF

Des efforts permanents sont réalisés afin d'intégrer les nouveaux standards dans les nouvelles versions. La prochaine version de Gecko (1.9) - qui sera d'ailleurs intégrée dans Firefox 3 - se verra plus performante et sera revue afin être mieux architecturée².

 *Gecko est une brique logicielle du framework de Mozilla, et à ce titre il peut être utilisé en tant que tel dans d'autres applications : Camino, Flock, SeaMonkey, XulRunner utilisent Gecko.*




*

² La roadmap de la version 1.9 est détaillée sur <http://www.mozilla.org/roadmap/gecko-1.9-roadmap.html>

V - XPCOM API

Il s'agit de la couche la plus complexe et la plus puissante du framework. XPCOM 3 (Cross Platform Component Object Model) intègre une gestion de bibliothèque de composants : chacun peut rajouter son propre composant ayant ses propres fonctionnalités, afin d'être utilisé au final dans l'application.

 Citons par exemple le composant *nsFile* qui permet de lire, écrire, modifier des fichiers sur le disque dur.


Mozilla met à disposition des utilisateurs du framework tous les composants qu'ils utilisent pour leurs applications : il en existe au total plus de 2 000.

Il n'y a donc plus qu'à écrire le code qui permet de choisir et d'utiliser ces « briques » logicielles déjà toutes prêtes : génial !

Pour les aficionados de Windows, Microsoft s'est lancé dans cette technologie depuis des années avec le COM, anciennement appelé OLE (Object Linking and Embedding).

Les possibilités de XPCOM sont :

- écrire du code multi plateforme : un même code source peut fonctionner sous Windows, Mac et Linux. Les composants « scriptés » (JavaScript et Python) fonctionneront nativement, les autres composants (C++ par exemple) nécessiteront d'être recompilés sous la plateforme à supporter. La portabilité du code est l'objectif principal de XPCOM.
- écrire du code réutilisable : on retrouve des fonctionnalités identiques entre différentes applications (lecture des périphériques, ouverture de fichiers, connexions réseau...), alors pourquoi ne pas les écrire une seule fois, puis ensuite les partager ? On y gagne à tous les coups : mises à jours partagées, gain de temps, séparations des composantes du logiciel...
- la séparation des fonctionnalités en composants de petite taille : on sait exactement ce que chaque composant est capable de faire et on peut donc, sans écrire une ligne de code, concevoir son application. Idéal pour les architectes !
- créer une passerelle de communication entre les différents langages supportant XPOM : JavaScript, C, C++, Python, Java. Un composant écrit dans un langage est disponible dans tous les autres !

 On peut imaginer un composant écrit en C++ implémentant un protocole réseau (tout simplement car la librairie n'est disponible que dans ce langage), et l'utiliser dans un autre programme écrit en Python

Voyons pour les plus curieux comment cela est possible !

Tout d'abord, il faut assimiler la conception de Composants et d'Interfaces.

Une **interface** est la partie publique du composant : elle présente ce que le composant contient et peut faire. Il s'agit ni plus ni moins de sa description. Cela s'intègre complètement dans la logique objet. Par exemple, je suis une voiture, j'ai une couleur, une taille, un moteur (attributs) et je sais rouler, freiner, tourner (méthodes). Cette description est écrite dans le langage de description IDL ([Interface Description Language](#)), utilisé entre autres avec le langage CORBA. Son unique but est de décrire une interface.


Concernant maintenant **l'implémentation**, on peut la faire dans plusieurs langages (C, C++, Javascript, Python). Le compilateur « xpidl » nous facilite cette tâche en nous fournissant tous les fichiers nécessaires depuis un fichier IDL : il n'y a plus qu'à écrire le code qui implémente les fonctionnalités décrites.

*

3 <http://www.mozilla.org/projects/xpcom/>

*

4 <http://www.mozilla.org/scriptable/xpidl/syntax.html>

 Dans cet exemple, Peugeot, Renault ou Porsche vont implémenter une voiture : ils vont ajouter des attributs (roues, volant, moteur#) et des actions seront possibles sur cette voiture (rouler, tourner, accélérer#). Pour utiliser la voiture, je n'ai pas forcément besoin de savoir comment elle est implémentée# ou plus prosaïquement être mécanicien.

Pour que tous ces composants puissent fonctionner sur toutes les plateformes, Mozilla a défini des types de variables au niveau IDL, et a implémenté un moteur comprenant l'implémentation de ces types pour chaque architecture et chaque langage. L'avantage est qu'une variable de type entier en C++ pourra être passée dans la couche XPCOM et sera comprise comme un type entier Python.

Exemple : Voici la description de l'interface « nsIToolbar » qui permet de créer une barre d'outils. On découvre que les composants qui l'implémentent peuvent ajouter et récupérer des éléments dans cette barre :

```
interface nsIToolbar : nsISupports {
    readonly attribute boolean visible;
    attribute int margin;
    attribute boolean horizontalLayout;
    attribute boolean lastItemIsRightJustified;
    attribute boolean wrapping;

    void AddItem(in nsIToolbarItem anItem, in int aLeftGap, in boolean stretchable);
    void InsertItemAt(in nsIToolbarItem anItem, in int aLeftGap, in boolean aStretchable, in int
anIndex);
    void GetItemAt(out nsIToolbarItem anItem, in int anIndex);
}
```


Une seule interface peut ainsi disposer de plusieurs composants, réalisés sous différents langages ou architectures. L'architecte logiciel peut donc librement décrire ce dont il a besoin, et les développeurs peuvent ensuite écrire les composants dans leur langage de prédilection.

Désormais, nous savons que nous pouvons afficher des pages XUL ou HTML avec des CSS (Gecko), et que l'on dispose des briques fonctionnelles (XPCOM). Reste à savoir comment faire le lien entre ces deux mondes. Comment puis-je appeler ces composants XPCOM dans mon application ?

VI - Script Layer

Comme son nom l'indique, cette couche est développée sous forme de scripts. Deux langages # de script bien sûr # sont utilisables : Javascript et Python.

Cette partie du framework est la plus accessible aux utilisateurs. Ne s'agissant que de scripts, aucune compilation n'est demandée à l'utilisateur# pour son plus grand bonheur.

 *Imaginons que je travaille sur une page XUL, et que je modifie son contenu (le texte d'un bouton par exemple) : je n'ai qu'à fermer cette fenêtre et la rouvrir pour que les changements soient pris en compte ! C'est exactement comme si je rechargeais une page d'un site Internet.*

Cette couche permet ainsi d'accéder, via des scripts, aux couches précédentes : Gecko et XPCOM.

Voici un exemple de la méthode pour appeler une composant XPCOM depuis Javascript. Ici, je déclare une variable « sound » qui va instancier le composant « @mozilla.org/sound;1 »

```
var sound = Components.classes["@mozilla.org/sound;1"].createInstance();
```

Sur cette instance, nous allons lui associer l'interface voulue : « Components.interfaces.nsILocalFile » qui correspond à l'interface développée par Mozilla.

```
sound.QueryInterface(Components.interfaces.nsILocalFile);
```

Désormais, je peux jouer un son depuis le Javascript, avec la méthode « play() » du composant. La variable « url » est récupérée d'après le composant « @mozilla.org/network/standard-url;1 »

```
url.spec = 'http://jslib.mozdev.org/test.wav';  
sound.play(url);
```

Génial ! Je peux jouer des sons dans mon application et de manière universelle# en 6 lignes de code ! Cela paraît tellement simple que je vous propose maintenant de comprendre comment associer des événements utilisateurs (souris, clavier#) à ces scripts.

VII - UI Toolkit

On dispose aujourd'hui de 3 interfaces possibles : Windows, Mac et Unix. Pour chacun de ces mondes, il faut à chaque fois au moins 3 ingénieurs pour développer une même fonctionnalité (exemple : afficher un bouton).

Mozilla propose une solution qui permet de décrire l'interface graphique de son application qui sera nativement compatible sur les 3 environnements.

De ce besoin est né le **langage XUL**. Il s'agit d'une description au format XML pour écrire des interfaces graphiques. D'où le nom de XUL pour XML User interface Language.

XUL permet donc de décrire des interfaces graphiques dans des fichiers textes, celles-ci pouvant contenir des widgets (boutons, barres, menus), des éléments textes et des éléments graphiques.

Afin de démystifier tout de suite le XUL, je vous propose un exemple de source en XUL

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window
title=#Find Files#
orient="horizontal"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
<button id="find-button" label="Find"/>
<button id="cancel-button" label="Cancel"/>
</window>
```

Une fois l'application lancée (vous pouvez essayer de créer un fichier texte « button.xul » avec l'exemple ci-dessus, et de l'ouvrir dans Firefox), vous verrez s'afficher à l'écran :



Maintenant que nous savons dessiner des widgets graphiques, il suffit de leur associer les actions possibles via le système d'événements.

Ici nous voyons un exemple de bouton qui réagit au clic via son attribut « oncommand » par l'appel à la méthode JavaScript « playSound() »

```
<button id="play-button" label="Play" oncommand=#playSound()#/>
```

Créer une interface graphique en XML représente de facto plusieurs avantages :

- 1 XML est un standard connu et reconnu
- 2 Un langage simple et intuitif
- 3 Pas de compilation car les fichiers sont interprétés à la volée. Le gain de temps est immédiat.
- 4 Les développeurs qui sont déjà formés au DHTML (Dynamic HTML) et à la création de site Internet en général peuvent se familiariser rapidement avec le XUL

Le XUL intègre aussi des technologies qui augmentent ses possibilités :

- 1 Une possible redéfinition des balises et des attributs via le langage **XBL** ([XML Binding Language](#))
- 2 Un système appelé « **overlays** » qui permet d'inclure plusieurs fichiers XUL dans une même fenêtre. C'est très puissant pour construire des menus en fonctions des extensions installées
- 3 Un système de représentation de données sous forme de graphe permettant d'échanger, de stocker et de représenter des données : **RDF** (Resource Description Framework). Idéal pour gérer des données de types favoris, profils utilisateurs, échanges d'information pour les mises à jour#

La mise en forme est aussi un aspect important à prendre en considération lors du développement d'interfaces graphiques. A la manière d'un site Internet, le standard utilisé est [CSS \(Cascading Style Sheets\)](#). Cela permet de rajouter des styles graphiques sur un ou plusieurs éléments de l'application. CSS est puissant car il permet de déporter tout l'aspect « design » dans des fichiers textes séparés ; les graphistes peuvent donc ne manipuler que ces fichiers pour travailler sur les aspects de taille et couleur de toute l'application.

Nous pourrions écrire un article entier sur le langage XUL, en attendant si vous voulez en savoir plus, vous pouvez consulter le tutorial de Mozilla [7](#).

*

5 <http://developer.mozilla.org/en/docs/XBL>

*

6 <http://www.w3.org/Style/CSS/>

*

7 http://developer.mozilla.org/en/docs/XUL_Tutorial

VIII - Applications / Extensions

Il s'agit de la dernière couche du framework, qui contient tout ce qu'il faut pour pouvoir exécuter l'application.

En plus de tout ce que nous avons vu, Mozilla a développé un espace virtuel de fichiers appelé **le chrome**, permettant de limiter la sécurité aux fichiers du projet.

Cela limite les problèmes de sécurité et ajoute diverses autres possibilités, comme la gestion de la localisation (langues), et la **gestion des extensions**. Cette dernière est très pratique à utiliser au niveau utilisateur (un simple glisser-déposer suffit à installer une extension), et permet de rajouter ou remplacer n'importe quel élément de l'application.

Voici des exemples de projets réalisés avec le framework de Mozilla qui commencent à sortir du lot :

- Nvu : un éditeur WYSIWYG pour développer son site Internet disponible sur <http://www.nvu.com>
- Songbird : un lecteur de musique web disponible sur <http://www.songbirdnest.com>
- Democracy player : un lecteur vidéo intégré « full web » disponible sur <http://www.getdemocracy.com>
- WengoLink : une application permettant de recevoir des appels audio & vidéo sur la plateforme Wengo disponible sur <http://www.wengo.com>

IX - Bilan

Nous allons dresser un habituel « pour et contre » de ce framework.

Inconvénients et problèmes du framework

- La documentation des différentes couches du projet manque souvent de recul et de profondeur, même si des efforts sont continuellement faits pour l'améliorer
- Le framework Mozilla est encore très jeune et manque de maturité
- Il reste énormément lié aux personnes et aux projets de la Mozilla Fondation
- La barrière technique pour rentrer dans le projet est assez haute. Il faut passer par de nombreuses étapes et astuces souvent non documentées avant de pouvoir avancer
- Manque d'outils adaptés, comme un IDE spécifique au XUL, pour ceux qui ne veulent pas trop rentrer dans la technique

Avantages du framework

- Multi plateforme
- Séparation du contenu, de la mise en forme et des processus métiers
- Très adapté au développement de RIA (Rich Internet Application)
- Pas de compilation au niveau XUL + couche de scripting = meilleure façon de développer des interfaces graphiques
- Plus de 2000 composants XPCOM disponibles
- Localisation facile
- Personnalisation et mise à jour facilitées par un puissant système d'extensions

X - Conclusion

Au final, comparé aux frameworks traditionnels comme peuvent l'être J2EE ou .Net, XPFE dévoile de toutes nouvelles façons de développer des applications. Son support natif sur Windows, Linux et Mac en fait déjà un concurrent sérieux. De plus, l'ajout de composants efficaces, tout en étant basé à 100% sur les standards du Web, fait du framework de Mozilla une solution très prometteuse. Sans oublier qu'il est complètement libre et gratuit.

Mais ce projet est-il aussi prometteur qu'il en a l'air ? Je mettrai en doute la réponse, car maîtriser la machine Mozilla n'est pas si simple. La découverte, puis l'utilisation avancée du framework nécessite un investissement personnel conséquent. Il faut souvent recourir à des astuces (connues ou peu connues) pour arriver à ses fins, signe révélateur du manque de maturité du projet.

Cependant si l'on regarde la qualité et le succès que peut avoir une application comme Firefox dans le monde de l'Internet, on se dit qu'il y a un large potentiel sous cette technologie. Mozilla propose de nouveaux réflexes de travail basés sur les standards et au final, cela vaut bien les quelques minutes de découverte passées à lire cet article.

Maxime ALEXANDRE, EDIS Consulting

