Professionnals I.T. Projects

Objet :	Étude technique : applications mobiles
Auteur :	P.I.T.P.
Destinataire :	Public
Date :	Avril 2013

Étude technique : applications mobiles

Ce document contient une rapide étude très orientée technique sur la question de l'application mobile. Les points abordés sont majoritairement techniques, il faudra juste en tenir compte pour pouvoir répondre aux besoins marketing. Les éléments présentés ici sont le fruit d'une veille rapide et de quelques pocs, c'est donc soumis à certaines réserves.

Applications mobiles et technologies

Point sur les applications HTML5

Commençons par une petite clarification des termes : l'application HTML5, dans le sens strict du terme, ne correspond pas du tout à nos besoins :

- Aucune interface avec le hardware, pas même pour des notifications push.
- Pas de présence sur les stores.

Ce que nous appelons applications HTML5 est en fait l'utilisation de technologies tierces pour produire des applications natives à partir de code HTML5 et / ou JavaScript. Cette distinction est transparente pour l'utilisateur, mais absolument pas neutre pour nous : nous n'avons pas la main sur la "finalité" du produit, ce qui soulève des questions sur :

- Les performances, dans la mesure où l'optimisation est aussi dépendante d'une connaissance très intime du middleware et de la plateforme de destination, et pourrait donc dans des cas complexes présenter une difficulté d'apprentissage supérieure aux seules solutions natives. On peut très raisonnablement supposer être en présence d'une technologie de type "Easy to learn, hard to master".
- L'obligation d'être "liée" à la technologie choisie, sous peine de devoir redévelopper l'application de zéro (et donc la pérennité dans le temps de ces solutions). Ce point est plus ou moins vrai selon les types de technologies, comme on l'abordera plus tard.

Les différents types de solution

Parmi ces solutions permettant d'utiliser du code HTML5 et / ou JavaScript pour produire du code natif, nous en avons relevés rapidement deux types parmi les noms les plus récurrents : Titanium et Phonegap. Il est très possible que d'autres alternatives moins connues existent. Précisons ici que les pocs se sont limités à la plateforme Android, étant moins familier de la plateforme iOS.
Principale inconnue à notre sens : connaître les réelles limitations techniques derrière chaque techno. Par exemple pour Phonegap, pour faire une notification push Android, il faut un plugin (certes, officiel). Quoi qu'il en soit chacune des technos semble très riche en plugins officiels et communautaires.

Nos poc a été identique sur les deux plateformes : une application Android capable de récupérer un flux RSS et de lister les titres d'articles.

Titanium

Titanium consiste en un SDK et un Studio. C'est un Framework JavaScript destiné à compiler du code Android ou iOS et selon ce que nous avons pu lire il générerait le code Java avant de le compiler en .apk : Nous ne sommes pas parvenu à trouver les sources Java concernées. L'interface ici est construite à partir de classe d'interface, comme pour du développement natif, et ne repose pas sur des pages HTML5.
Avantages :

- Bonne intégration du studio (basé sur Eclipse) : auto complétion, etc. Inconnue totale pour iOS, mais probable qu'il faille configurer XCode sur un mac.
- Licence Apache du SDK, peu de risque de voir la technologie s'évaporer...
- Production de code natif (à vérifier), donc plus facile à reprendre et à faire évoluer si on abandonne la technologie.
- Plus performant.

Inconvénients :

- Supporte "seulement" iOS et Android.
- Déplacement de la problématique : le gain de Titanium ne se joue que sur la syntaxe du langage, mais il faut apprendre les API. Le seul gain est le côté "code once, deploy everywhere", qui est toujours à prendre avec des pincettes, surtout quand l'everywhere ne représente que deux plateformes.
- L'interface n'est pas du HTML5, c'est donc un développeur qui doit intervenir sur l'application pour la couche présentation.
- Le poids ahurissant du .apk fournis : 9mo ! Il faut se rappeler que des mobiles sur le marché ne proposent que quelques centaines de mo d'espace, et que certaines applications simples ne dépassent pas 100ko...
- Autorisations demandées fantaisistes pour le besoin du poc, probablement la faute à une erreur de configuration de notre part, mais à vérifier.

Phonegap

Contrairement à Titanium, Phonegap est un wrapper HTML5 qui permet d'utiliser ses APIs JavaScripts favorites pour produire une application mobile native. Phonegap fournit une API Js minimale limitée aux interactions avec le hardware. Chacun utilisera la lib qu'il voudra pour son code métier. Phonegap ne génère aucun code natif.

Positif :

- Licence Apache, le SDK est la propriété de la fondation Apache.
- Poids de l'application : Positif en comparaison de Titanium : 2mo le .apk.
- Possibilité de réutiliser presque "As is" des morceaux de code desktop.
- Réel gain en termes de courbe d'apprentissage : jQuery Mobile est une très petite marche en comparaison de Java ou Objective C.
- Possibilité de faire travailler des intégrateurs pour travailler sur la couche présentation.

Négatif :

- Le seul builder est une propriété de Adobe qui ne fonctionne qu'en SaaS, avec hébergement des projets sur un repository (deux choix : github, ou Adobe). Il faut sans ça configurer un Eclipse avec les ADT (Android Developper Tool). Vu la dominante HTML5 / Javascript, Aptana est bien équipé pour être un IDE correct pour Phonegap.

Développements internes : Web Services

Quelles que soient les solutions choisies, on va devoir développer en interne des Web Services.
Quelques choix arrêtés :

- Architecture REST, car plus simple, très adaptée au protocole HTTP et plutôt légère.
- Format de sortie JSON, langage de choix pour des interfaces HTML5 / JS. JSON est aussi plus light que XML, et plus facile à générer en PHP.
- Authentification : à étudier. Très possible qu'une part de l'API soit publique pour simplifier et alléger le tout. REST étant stateless, probable qu'un système de token soit utilisé dans chaque requête "loggée".

Conclusions

Après avoir un peu étudié ces technos, nous les considèrons d'un moins mauvais oeil qu'initialement, surtout pour les solutions type wrapper comme phonegap :

- Courbe d'apprentissage initiale rapide
- Licence Apache plutôt rassurante.
- Bonne complémentarité des compétences : garantie de backups permanents dans l'équipe.

Si un jour on doit faire du natif pour des questions de limitations techniques ou de performances, on devra reprendre tout de zéro... ou trouver un moyen de mélanger les deux. Il est / sera peut-être possible de développer une application native qui embarque un wrapper, sous forme d'API…