Applications hybrides et adaptatives basées Apache Cordova

Ivan Madjarov

Applications mobiles adaptatives basées Apache Cordova

Ivan MADJAROV
Aix Marseille Université, CNRS LSIS UMR 7296,
IUT, Département Réseaux et Télécommunications,
Marseille, France
ivan.madjarov@univ-amu.fr

***Abstract*—Développer des applications multiplateformes pour les appareils mobiles est une tâche assez complexe car les différents systèmes qui les animent se sont rendus parfaitement incompatibles en termes de portage d'applications. Cordova, supporté par le groupe Apache, est une alternative de développement multiplateforme mobile se basant sur HTML5, CSS3 et JavaScript. Cordova est une forme de conteneur pour interfacer l'application Web avec les fonctionnalités natives de l'appareil mobile. Pour les étudiants R&T cette plateforme met en valeur les connaissances acquises dans les modules de développement Web et la programmation pour appareils mobiles.**

Keywords*—*Application mobile, Application adaptative, Adaptation de contenu.

I. INTRODUCTION

Les applications mobiles sont désormais omniprésentes en commençant par les smartphones et les tablettes, en passant par les montres intelligentes, et bientôt sur toute unité connectée. Le développement pour chaque plate-forme mobile s'avère une tâche complexe par la diversité des supports et langages impliquées et demande un effort de toute une équipe. Une nette simplification de cette tâche apporte la technique *Apache Cordova* [1] qui fournit un moyen de développer des applications mobiles en utilisant les technologies Web standard notamment HTML5, CSS3 et JavaScript.

Une application développée *Apache Cordova* s'installe comme une application native sans aucune limite pour les récents systèmes d'exploitation mobiles : iOS, Android, Windows Phone, Firefox OS, Ubuntu Phone. *Apache Cordova* fournit un ensemble de plugins qui donnent l'accès à la caméra de l'appareil mobile, le GPS, le système de fichiers, etc. La pérennité de l'environnement technologique est assurée par des mises à jour et le développement de nouveaux plugins pour tout nouvel appareil mobile. L'essentiel de la technique *Cordova* est une API JavaScript. Elle sert de "*wrapper*" (enveloppeur) pour le code natif pour rendre cohérente l'application avec les dispositifs mobiles. On peut considérer *Cordova* comme un conteneur d'application qui s'intègre avec *WebView*, comme montré sur la Figure 1, pour l'affichage de la page Web quel que soit son contenu (*jQuery, AngularJS*). Suivant le système d'exploitation *Cordova* s'adresse à la classe native Objective-C *UIWebView* sur iOS, ou à la classe native Android *android.webkit.WebView*, etc.

La technique *Cordova* est présentée dans le cadre du module *M4207C - Application informatique dédiée au R&T* pour la conception d'applications mobiles adaptatives. L'intérêt pour cette technique se justifie par la présence dans le PPN-R&T 2013 de modules de développement Web (*M1106 - Initiation au développement Web, M2105 - Web dynamique*) et le module *M4206C - Programmation sur appareils mobiles communicants* qui traite le développement d'applications Android pour smartphones et tablettes.

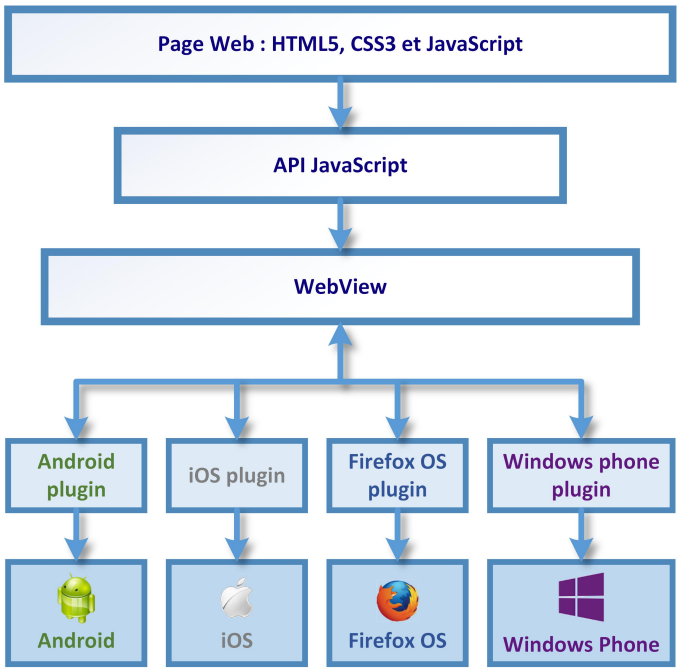

Fig. 1. Le processus de développement *Apache Cordova*

La Figure 1 illustre le principe relativement simple de création d'une application native basée *Cordova*. Une *Webview* traite les fichiers *HTML* en communication avec les APIs (accéléromètre, boussole, géolocalisation, caméra ou autres) par le biais du plugin correspondant au système d'exploitation de l'appareil mobile.

II. Installer, Configurer et Tester Cordova

A. *Installation de Cordova [7][8]*

Avant tout, il faut installer *NodeJS* [2]. *NodeJS* est une plateforme événementielle en JavaScript orientée vers les applications réseau avec une bibliothèque de serveur HTTP intégrée.

Cordova s'installe avec le *Node Package Manager* (npm) depuis l'invité de commandes (cmd) par le script suivant :

```
>npm install -g cordova@latest
```

L'étape suivante installe le SDK spécifique à la plateforme cible. Dans notre article nous allons cibler les smartphones et les tablettes sous Android. L'installation du SDK Android demande en premier l'installation du *Java Development Kit* [3]. Pour éviter les problèmes par la suite il faut créer une nouvelle variable d'environnement JAVA_HOME pointant vers la racine du dossier *C:\Program Files\Java\jdk1.8.0_65*. Après installation du *SDK Android* [4] il faut ajouter les dossiers *tools* et *platform-tools* dans le PATH des variables d'environnement. Une dernière étape est l'installation d'*Apache Ant* [5] et l'ajouter au PATH du système. *Ant* gère la compilation de l'application native. Les dernières versions de SDK Android comportent une version de *Ant*.

B. *Une Première Application*

On choisit un répertoire pour créer le projet "*bonjour*" avec la commande :

```
>cordova create bonjour
```

On accède au répertoire du projet :

```
>cd bonjour
```

On ajoute la plateforme cible :

```
>cordova platform add android --save
```

La configuration du projet est ainsi sauvegardée dans le fichier *bonjour/config.xml*. Le fichier *bonjour/www/index.html* contient un squelette d'application qu'on peut remplacer avec le code suivant :

```
<html>
<head>
<meta charset="utf-8" />
<meta name="viewport" content="initial-scale=1,
width=device-width" />
<title>Cordova Android</title>
<script type="text/javascript" src="cordova.js">
</script>
<script type="text/javascript">
var showMessageBox = function() {
document.write("<h2>Bonjour<br/>tout le
monde<br/>de Cordova</h2><img src='img/logo.png'
height='auto' width='auto'/>");
}
function init() {
document.addEventListener("deviceready",
showMessageBox, true);
}
</script>
</head>
<body onload="init();">
</body>
</html>
```

La compilation du projet :

```
>cordova build android
```

Pour tester l'application on peut lancer le simulateur intégré dans le SDK Android :

```
>cordova emulate android
```

ou sur un appareil mobile par connexion USB :

```
>cordova run android
```

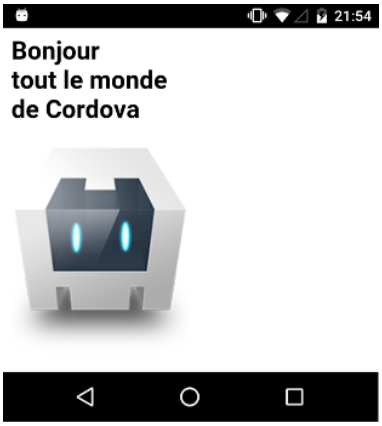

Fig. 2. Cordova: application hybride dialoguant avec le code natif du terminal Android

Les pilotes USB pour smartphone et tablette sont accessible à installation à partir du répertoire *android-sdk*.

C. *Structure et Contenu de l'Application Cordova*

A la création du projet "*bonjour*" l'arborescence présentée à la Figure 3 contient tous les éléments du projet.

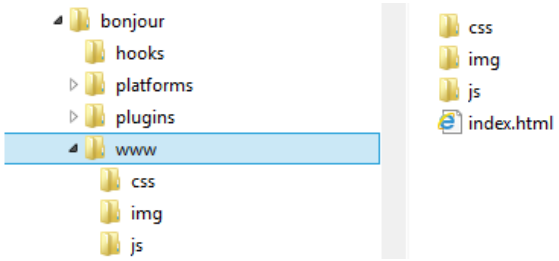

Fig. 3. Structure du projet "*bonjour*"

La balise <*meta*> dans la section <*head*> du fichier *index.html* présenté dans la section "*B*" spécifie le codage du document avec l'attribut *charset* et son adaptabilité vis-à-vis les différentes tailles et résolutions d'écrans mobiles avec l'attribut *viewport* [6].

La bibliothèque *JavaScript* nécessaire au fonctionnement de *Cordova* est introduite dans la section <*head*> avec la balise <*script*> :

```
<script type="text/javascript" src="cordova.js">
</script>
```

Les fonctionnalités de la bibliothèque ne sont pas disponibles immédiatement. La réception d'une notification est

nécessaire. Un appel à la fonction *init()* est effectué au chargement de la page *HTML* par l'évènement *onload* référencé par la balise *<body>*.

```
function init(){
    document.addEventListener("deviceready",
    showMessageBox, true);
}
```

La fonction *init()* installe l'écouteur d'évènement *document.addEventListener* de type *deviceready*. Le dispositif se déclenche en appelant la fonction *showMessageBox* lorsque *Cordova* est complètement chargée. Cet événement est essentiel à toute application. Il signale que l'API *Cordova* est chargée et prête au fonctionnement. La fonction *showMessageBox* fournit le résultat présenté à la Figure 2.

III. Les Plugins Cordova

Un plugin *Cordova* est un code d'extension (*add-on*) qui fournit l'interface JavaScript pour communiquer avec les composants natifs de l'appareil mobile. Ainsi, le plugin permet à l'application d'utiliser les capacités natives du périphérique au-delà de ce qui est disponible pour une application Web classique.

Pour ajouter la dernière version du plugin pour la caméra :

```
>cordova plugin add cordova-plugin-camera@latest
--save
```

Pour enlever le plugin de la caméra :

```
>cordova plugin rm cordova-plugin-camera --save
```

A. L'API Cordova Device

L'objet *Cordova Device* fournit des d'informations sur l'application et l'unité mobile. Pour utiliser cette API dans une application *Cordova* il faut installer le plugin *cordova-plugin-device*. Le code suivant fait appel à la fonction JavaScript *onBodyLoad()* lorsque l'événement *onload* est déclenché. Après notification du chargement de l'API une liste non-ordonnée compacte les informations dans la fonction *onDeviceReady()*. L'attribut *innerHTML* [12] remplace le contenu identifié par *devInfo* sur la page.

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=utf-8">
<meta name="viewport" content="width=device-
width, initial-scale=1.0" />
<link rel="stylesheet" href="css/w3.css">
<script src="cordova.js"></script>
<script>
function onBodyLoad() {
document.addEventListener("deviceready",
onDeviceReady, true);
}
function onDeviceReady() {
var tmpStr = '';
tmpStr += '<ul class="w3-ul">';
tmpStr += '<li> Cordova Version: ' +
device.cordova + '</li>';
tmpStr += '<li> Operating System: ' +
device.platform + '</li>';
tmpStr += '<li> OS Version: ' + device.version +
'</li>';
tmpStr += '<li> Device Model: ' + device.model +
'</li>';
tmpStr += '<li> Universally Unique Identifier: '
+ device.uuid + '</li>';
tmpStr += '</ul>';
document.getElementById('devInfo').innerHTML =
tmpStr;
}
</script>
</head>
<body onload="onBodyLoad()">
<h2 class="w3-container w3-center w3-teal">
Information sur <br/>l'API Cordova</h2>
<p class="w3-center">L'application fait appel à
cordova-plugin-device</p>
<p class="w3-center" id="devInfo">En attente
d'initialisation de l'API Cordova</p>
</body>
</html>
```

Le script affiche (Figure 4) des informations sur la version de Cordova (*device.cordova)*, le système d'exploitation (*device.platform*) et la version (*device.version*), le modèle de l'appareil (*device.model*), l'UUID associé à l'appareil (*device.uuid*).

Fig. 4. L'API *device* fournit des détails natifs sur l'appareil mobile

B. Application Stylée Adaptative

Les feuilles de style (*CSS3*) [9] permettent l'obtention d'une présentation élaborée et adaptative (*responsive*) à l'écran d'un appareil mobile. La norme *CSS3* déclare la mise en forme des pages *HTML* et exploite ainsi les avantages de la séparation du contenu et de la forme. La spécification *CSS3 Media Queries* définit les techniques pour adapter la présentation en fonction du périphérique utilisé [6]. Une aide précieuse pour la réalisation de belles applications mobiles est proposée par Adobe avec *Topcoat* [10]. C'est un ensemble de fichiers CSS *open source* prêts à l'utilisation. Un autre projet CSS est proposé sur le site de W3Schools [11]. *W3.CSS* est un *framework* compact, rapide et plus petit par rapport à d'autres du même type. Facile à apprendre, il est basé entièrement sur la norme CSS, (aucune bibliothèque *jQuery* ou *JavaScript* n'est nécessaire). L'application de la Figure 4 est réalisée avec la feuille de style *w3.css* [11].

C. *L'API Cordova Connect*

L'objet *Cordova Connect* fournit des informations sur la connexion réseau de l'appareil et la nature de cette connexion : WiFi, 4G, 3G, ou autre. Une application se sert de cette information pour déterminer le moment de transfert de données volumineux depuis ou vers un serveur, ainsi que pour la prise en compte de la précision d'une géolocalisation.

Pour utiliser l'objet *Cordova Connect* il faut ajouter le plugin correspondant dans le projet à développer :

```
>cordova plugin add cordova-plugin-network-
information@latest --save
```

La propriété *navigator.connection.type* de l'objet *Connection* renvoie l'information sur la connexion disponible pour l'application. A cet usage l'objet expose un ensemble de constantes pour la définition du type de connexion : *Connection.CELL_4G, Connection.WIFI, ...*

Dans la fonction *onDeviceReady()* on définit le tableau associatif `states` initialisé avec les différents éléments d'une connexion à identifier par le retour de l'instruction suivante :

```
var networkState = navigator.connection.type;
```

L'exemple présenté par son code et le résultat obtenu à la figure 5 montre une identification du type de connexion en cours orientée normalement au transfert de données.

```
<!DOCTYPE html>
<html>
<head>
<title>navigator connection type</title>
<script type="text/javascript" charset="utf-8"
src="cordova.js">
</script>
<link rel="stylesheet" href="css/w3.css">
<script type="text/javascript" charset="utf-8">
function onBodyLoad() {
document.addEventListener("deviceready",
onDeviceReady, false);
}
function onDeviceReady() {
var networkState = navigator.connection.type;
var states = {};
states[Connection.UNKNOWN]  = 'Inconnue';
states[Connection.ETHERNET] = 'Ethernet';
states[Connection.WIFI]     = 'WiFi';
states[Connection.CELL_2G]  = 'cellulaire 2G';
states[Connection.CELL_3G]  = 'cellulaire 3G';
states[Connection.CELL_4G]  = 'cellulaire 4G';
states[Connection.CELL]     = 'générique';
states[Connection.NONE]     = 'aucune';
document.getElementById('netInfo').innerHTML =
"Connexion "+states[networkState];
}
</script>
</head>
<body onload="onBodyLoad()">
<h2 class="w3-container w3-center w3-
teal">Cordova Connection API</h2>
<p class="w3-center">l'objet <b>Connection</b>
est accessible via<br />
<i>navigator.connection.type</i> pour afficher
<br />l'état de connexion de l'appareil.</p>
<hr />
<p class="w3-center" id="netInfo"></p>
</body>
</html>
```

L'affichage sur le terminal Android :

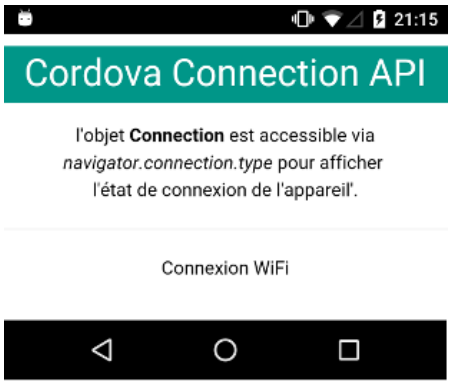

Fig. 5. L'API *connection* fournit des informations sur la nature de la connexion via la propriété *navigator.connection.type*.

D. *L'API Cordova Geolocation*

L'API Geolocation est basée sur la spécification de W3C [13]. L'objet *navigator.geolocation* permet l'accès aux données de localisation du capteur GPS (Système de Positionnement Global) de l'appareil. Ces données peuvent être déduites aussi des signaux du réseau mobile (les IDs cellulaires GSM/CDMA) ou Wi-Fi (l'adresse IP, RFID, les adresses MAC) ou Bluetooth.

Les données de géolocalisation sont considérées comme sensibles. Ces données révèlent l'endroit d'utilisation du GPS. Une application devrait afficher une notice avant d'accéder aux données par respect de confidentialité et ainsi permettre de recueillir l'autorisation de l'utilisateur de poursuivre.

Pour implémenter cette fonctionnalité il faut ajouter le plugin *cordova-plugin-geolocation* dans le projet Cordova prévu à cet usage. Pour une application Cordova-Android des commandes sont appliquées à la plate-forme cible en modifiant les paramètres de configuration spécifiques. A la compilation les autorisations nécessaires à la géolocalisation sont apportées au fichier : *platforms/android/res/xml/config.xml*

```
<feature name="Geolocation">
<param name="android-package"
value="org.apache.cordova.GeoBroker" />
</feature>
```

et au fichier : *platforms/android/AndroidManifest.xml*

```
<uses-permission android:name =
"android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name=
"android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name=
"android.permission.ACCESS_LOCATION_EXTRA_COMMAN
DS" />
```

Pour détecter la position de l'appareil on fait appel à la fonction asynchrone *getCurrentPosition(par1, [par2], [par3])* dans la fonction *onDeviceReady()* de l'application Cordova. La fonction renvoie la position de l'appareil sous la forme d'objet :

```
navigator.geolocation.getCurrentPosition
(onSuccess, onError);
```

La fonction (callback) *onSucces* est appelée si les mesures de géoposition sont transmises. Le cas contraire la fonction *onError* est appelée. La fonction *onSuccess* prend en paramètre l'objet de la localisation pour exposer les coordonnées à travers la propriété *coords*.

```
function onSuccess(pos) {
    var latitude = pos.coords.latitude;
    var longitude = pos.coords.longitude;
}
```

L'objet géolocalisation fournit des informations sur la mesure d'un ensemble de propriétés incluant : *Latitude, Longitude, Altitude, Accuracy, Altitude Accuracy, Heading, Speed, Timestamp*.

Le code suivant illustre l'implémentation de l'API *Geolocation* dans une application Cordova.

```
<!DOCTYPE html>
<html>
<head>
<title>Geolocation</title>
<link rel="stylesheet" href="css/w3.css">
<script type="text/javascript" charset="utf-8"
src="cordova.js"></script>
<script type="text/javascript" charset="utf-8">
function onBodyLoad() {
document.addEventListener("deviceready",
onDeviceReady, true);
}
function onDeviceReady() {
navigator.geolocation.getCurrentPosition
(onSuccess, onError);
}
function onSuccess(pos) {
document.getElementById('geoloc').innerHTML =
'Latitude: ' + pos.coords.latitude + '<br />' +
'Longitude: ' + pos.coords.longitude;
}
function onError(error) {
document.getElementById('geoloc').innerHTML
='code: ' + error.code + '<br />message: ' +
error.message;
}
</script>
</head>
<body onload="onBodyLoad()">
<h2 class="w3-container w3-center w3-teal">
Géolocalisation</h2>
<hr />
<p class="w3-center" id="geoloc">
A la recherche de la géolocalisation...</p>
</body>
</html>
```

L'affichage sur le terminal Android :

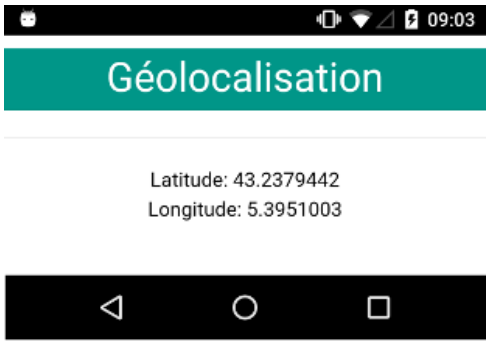

Fig. 6. L'API *geolocation* fournit des informations sur la localisation de l'appareil.

E. L'API Cordova Camera

L'API *Camera* peut prendre des photos avec la caméra de l'appareil mobile, ou peut choisir des images à partir d'une bibliothèque. Pour implémenter ses fonctionnalités il faut ajouter le plugin *cordova-plugin-camera* dans le projet. L'objet *navigator.camera* donne l'accès aux fonctions de la caméra.

La collecte et l'utilisation des images de la caméra d'un appareil mobile concernent la vie privée. Si l'utilisation de la caméra n'est pas apparente dans l'interface utilisateur l'application devrait afficher une notice avant d'accéder aux fonctionnalités de l'API par respect de la confidentialité et ainsi permettre de recueillir l'autorisation de l'utilisateur de poursuivre.

Dans le code on fait appel à la fonction *getPictures* pour ouvrir l'application par défaut de l'appareil mobile permettant à l'utilisateur de prendre des photos ou de consulter la galerie photos. Trois options sont disponibles :

```
navigator.camera.getPicture(onSuccess, onError,
[Options]);
```

- *Camera.PictureSourceType.CAMERA* active l'application caméra pour la prise de photos ;
- *Camera.PictureSourceType.PHOTOLIBRARY* ou *Camera.PictureSourceType.SAVEDPHOTOALBUM*, ouvre une boîte de dialogue pour accéder à une image existante.

La fonction passe au "callback" (*onSuccess*) l'image comme une chaine de caractères encodée en base64 ou l'URI du fichier image.

```
<!DOCTYPE html>
<html>
<head>
<title>Camera</title>
<meta http-equiv="Content-type"
content="text/html; charset=utf-8">
<meta name="viewport" id="viewport"
content="width=device-width, height=device-
height, initial-scale=1.0, maximum-scale=1.0,
user-scalable=no;" />
<script type="text/javascript" charset="utf-8"
src="cordova.js"></script>
<script type="text/javascript" charset="utf-8">
function onBodyLoad() {
document.addEventListener("deviceready",
onDeviceReady, false);
```

```
}
function onDeviceReady() {
navigator.camera.getPicture(onSuccess, onFail, {
quality: 25, destinationType:
Camera.DestinationType.DATA_URL });
}
function onSuccess(imageData) {
var image = document.getElementById('monImage');
image.style.display = 'block';
image.src = "" +
imageData;
}
function onFail(message) {
alert('Failed because: ' + message);
}
</script>
</head>
<body onload="onBodyLoad()">
<h1>Camera</h1>
<p>Cette application utilise l'API camera de
Cordova pour prendre des photos avec l'appareil
mobile.</p>
<img
style="display:block;width:10.000em;height:12.50
0em;" id="monImage" src="" />
</body>
</html>
```

Pour éviter les problèmes de mémoire il faut mieux privilégier la destination *Camera.destinationType.FILE_URI* au lieu de *DATA_URL* à cause d'une résolution des photos assez bonne pour les appareils récents. Les photos sélectionnées de la galerie de l'appareil mobile ne sont pas réduites en taille et qualité, même si un paramètre est spécifié.

IV. CONCLUSION

Le développement d'applications mobiles devient une tâche récurrente pour une multitude de plateformes. Trois solutions, plus ou moins complexes se présentent pour les environnements mobiles :

1. Une application *native* est développée pour une seule plateforme avec des outils qui lui sont propres. Le langage de développement est spécifique au système d'exploitation. La distribution et la mise à jour se fait par l'intermédiaire d'un store dédié.

2. Une *webapp* est développée avec les technologies web HTML5, CSS3 et JavaScript. L'application est distribuée via un serveur web et est exécutée via le navigateur web du mobile. Contrairement aux applications natives, une *webapp* est développée une seule fois pour être exécutée sur n'importe quelle plateforme mobile via Internet. C'est une économie importante de temps et du coût de développement. L'inconvénient majeur toujours mis en avant est l'absence d'accès aux fonctions natives du mobile, ce qui réduit le champ fonctionnel d'une *webapp*.

3. Une application *hybride* est un mix des deux premières solutions. Le développement combine des éléments web sous forme de *webapp* et des éléments de l'application *native* en compilant une exécutable compatible avec le système d'exploitation (Figure 1). Les plateformes *Phonegap* et *Cordova* permettent de créer une application indépendante à partir de pages web et l'utilisation des fonctions natives de l'appareil mobile. L'approche *hybride* permet de mutualiser le développement sur plusieurs systèmes d'exploitation. Développer en *hybride* contribue à l'optimisation du temps et du coût de développement.

REFERENCES

[1] Apache Cordova, https://cordova.apache.org/

[2] NodeJS, https://nodejs.org/

[3] Java Development Kit (jdk), http://www.oracle.com/technetwork/java/javase/downloads/index.html

[4] SDK Android, http://developer.android.com/sdk/index.html

[5] Apache Ant, http://ant.apache.org/bindownload.cgi

[6] Ivan Madjarov, Arnaud Février, Responsive Web Design: une approche pour concevoir des sites Web adaptatifs et une occasion d'inciter les étudiants à consulter des cours responsives, Les cahiers pédagogiques R&T, Actes du 3ème Workshop, pp. 245-250, Saint Pierre de La Réunion, 17-20 novembre 2014.

[7] Installing Cordova and SDKs on OS X, https://evothings.com/doc/build/cordova-install-osx.html.

[8] Installing Cordova on Windows, https://evothings.com/doc/build/cordova-install-windows.html.

[9] W3C, Cascading Style Sheets, https://www.w3.org/Style/CSS/

[10] Topcoat, http://topcoat.io.

[11] W3.CSS, http://www.w3schools.com/w3css/default.asp

[12] MDN, https://developer.mozilla.org/fr/docs/Web/API/Element/innerHTML.

[13] Apache Corodva Geolocation, https://cordova.apache.org/docs/