UNIVERSITÉ CATHOLIQUE DE LOUVAIN

FACULTÉ DES SCIENCES APPLIQUÉES

DÉPARTEMENT D'INGÉNIERIE INFORMATIQUE

Adaptation du « front-end » d'un web service en fonction du contexte d'utilisation

Concept et prototype

Promoteur : **Jean VANDERDONCKT**
Lecteurs : **Hildeberto MENDONÇA**
& **Adrien COYETTE**

Mémoire présenté en vue de l'obtention du grade de master 60 crédits en sciences informatiques, par **François DEBANDE** & **Quentin PONCELET**

Louvain-la-Neuve
Année académique 2010-2011

Remerciements

Nous tenions à remercier Mr Vanderdonckt de nous avoir apporté son soutien sur ce sujet intéressant qu'il nous a proposé.

Nous remercions également Mr Coyette et Mr Mendonça d'avoir accepté le rôle de lecteur de ce mémoire.

Table des matières

Chapitre 1 : Introduction

Ce premier chapitre va vous permettre de découvrir l'envergure du travail réalisé durant cette dernière année universitaire. Nous allons y décrire le problème abordé, le contexte dans lequel il se situe et ensuite, l'approche utilisée pour le résoudre.

1.1. Contexte du problème

Grâce aux nouvelles technologies de réseau sans fil et les progrès au niveau de la miniaturisation des matériels, nous nous dirigeons incontestablement vers un monde où l'utilisateur devient mobile. De nos jours, il n'est plus rare de croiser dans la rue des personnes surfant sur Internet avec un Smartphone ou, plus récemment, une tablette. Selon une étude parue dans la presse belge et réalisée en février 2011 par ANT Research [ANT 2011], un belge sur cinq serait en possession d'un Smartphone et trois pourcents de la population serait également en possession d'une tablette. Et, selon l'étude prévisionnelle réalisée par Morgan Stanley (voir Figure 1), le nombre d'utilisateurs d'Internet mobile dépassera, en 2014, le nombre d'utilisateur se connectant via leur ordinateur fixe.

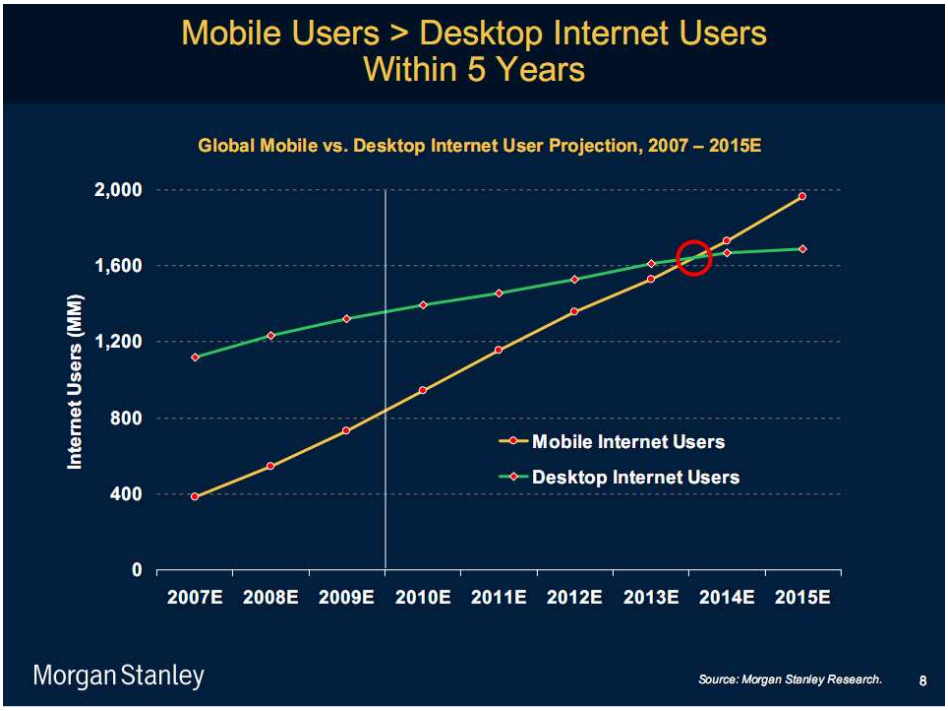

Figure 1 : Mobile vs Fixe (Source : [MEE 2010])

Mais si ces nouvelles technologies sont attrayantes pour l'utilisateur, elles représentent de nouveaux défis en termes d'interaction entre l'homme et la machine.

Le contexte d'utilisation (défini par l'utilisateur, la plateforme et l'environnement) devient varié, changeant et imprévisible [GAN 2007].

Il convient donc de développer des applications accessibles sur la majorité des plateformes (allant de la montre aux écrans géants), selon les choix de l'utilisateur et de l'environnement dans lequel il se trouve. Ceci pose plusieurs problèmes dans le sens où, avant l'avènement du web mobile, la navigation des pages web qui était conçue uniquement pour des ordinateurs de bureau, n'était pas prévue pour des appareils mobiles aux caractéristiques particulières, comme de plus petits écrans, moins de bande passante, des processeurs lents et de petites mémoires [OHN 2008] [SCH 2001]. De par notre expérience personnelle, nous remarquons également que les utilisateurs ne vont peut-être pas systématiquement accéder aux mêmes genres de pages web que celles qu'ils consultent chez eux, sur leur ordinateur. Du fait de leur plateforme mobile, ils privilégieront d'avantage l'accès direct à l'information désirée.
Il devient donc de plus en plus pertinent d'adapter l'information au contexte d'utilisation.

1.2. Description du problème

L'adaptation du Web au contexte d'utilisation est devenue une préoccupation majeure depuis l'avènement du web mobile. Ainsi, il existe actuellement plusieurs solutions permettant d'adapter statiquement une page web aux nouvelles plateformes mobiles, alors qu'elle était réalisée à la base pour les ordinateurs de bureaux.
À notre connaissance, aucune de ces solutions ne propose de créer un front-end adapté dynamiquement aux données renvoyées par un web service par rapport au contexte d'un utilisateur. Ce contexte d'utilisation ayant été standardisé récemment par le **W**orld **W**ide **W**eb **C**onsortium (**W3C**) sous le nom de « **D**elivery **C**ontext **O**ntology » [CAN 2009]. Suivant le contexte actuel, il semblerait pourtant important de répondre à ce problème encore non résolu.

Nous sommes conscients de la complexité de cette problématique, notamment à cause de la nouveauté de cette standardisation par le W3C, et donc de son manque de maturité. Mais c'est ce qui fait toute l'originalité d'un tel projet.

1.2.1. L'objectif central

Le but de ce mémoire est de conceptualiser une nouvelle manière d'aborder la création et l'adaptation du front-end d'un web service en fonction du contexte de l'utilisateur, de par la conception d'un prototype.

Pour atteindre cet objectif, nous aurons principalement besoin d'étudier les concepts liés au « **D**elivery **C**ontext **O**ntology », au fonctionnement des web services ainsi qu'aux différentes catégories d'utilisateurs. Notre prototype sera en effet un serveur proxy situé entre l'utilisateur et le web service.

1.3. Méthodologie de travail

Afin d'atteindre notre objectif, nous nous sommes fixés la feuille de route suivante :

1. recherche de documentation sur le sujet traité ;
2. recherche des web services ;
3. sélection d'un web service compatible avec nos critères de recherches ;
4. choix d'implémentation ;
5. réalisation du prototype pour un seul web service ;
6. analyse des problèmes rencontrés et recherche de solutions ;
7. introduction d'un second web service ;
8. introduction de la combinaison des deux web services précédents ;
9. réalisation du prototype final ;
10. évaluation de celui-ci.

1.4. Plan du mémoire

Nous commencerons ce travail par l'état de l'art des concepts sur lesquels nous baserons par la suite le développement théorique de notre propre solution. Nous mettrons alors celle-ci en pratique et l'évaluerons. Nous terminerons par tirer les conclusions de l'ensemble de ce mémoire et de ce que nous vous avons proposez.

Chapitre 2 : État de l'art

Cette démarche préliminaire de l'état de l'art va nous permettre de poser la base théorique nécessaire à la compréhension de l'implémentation de notre prototype. Nous commencerons tout d'abord par définir ce qu'est un web service ainsi que ses différents modes de fonctionnement. Nous détaillerons ensuite le concept du « contexte d'utilisation » à travers le standard « Delivery Context Ontology » du W3C. Enfin, nous découvrirons les solutions existantes dans le monde de l'adaptation du web vers le mobile, desquelles nous pourrons retirer de nouveaux concepts intéressants pour l'implémentation de notre propre solution.

2.1. Un web service

Les web services permettent l'accès à des web-applications via Internet, quelque soit la plateforme et le langage de l'utilisateur [WEB 2011]. Lorsque l'on parle d'utilisateur, il est utile de préciser que les web services ne possèdent pas d'interface graphique pour afficher les données. Ainsi, ils délivrent des données qui devront être interprétées par des applications qui se chargeront de les rendre compréhensible et accessible pour l'utilisateur.

Ces web services peuvent être utilisés aussi bien pour faire appel à des applications de manière régulière (prévisions météorologiques, conversions de données monétaires ou de métriques,...) que pour aider les utilisateurs à résoudre des problèmes d'interopérabilité entre machines équipées de différentes plateformes, et permettre ainsi un échange fiable des données.

Afin de garantir l'accès aux web-applications, les web services utilisent le langage **XML** (**E**xtensible **M**arkup **L**anguage) [BRA 2008] qui favorise l'échange des données entre toutes les machines des utilisateurs indépendamment de leur plateforme, du langage de leur application, etc.

Parmi tous les web services, nous pouvons distinguer deux grandes catégories :

- **les web services « WSDL »**, utilisant les standards *SOAP*, *UDDI* et *WSDL* ;
- **les web services « RESTful »**, appelé ainsi car ils respectent les principes *REST* énoncés par Roy Fielding [FIE 2000].

Dans la suite de cette section, nous allons décrire ces deux catégories plus en détail.

2.1.1. WSDL

Comme nous l'avons vu dans la section précédente, les web services permettent l'accès à des web-applications, que l'on pourrait comparer à des fonctions appelée selon un format spécifié par le web service et qui renvoient un résultat dans un format également défini par ce dernier.

Cette définition des formats de données d'entrée et de sortie, ainsi que des méthodes disponibles, le protocole de transfert et la manière d'accéder à un web service, sont rassemblés dans un fichier de type *WSDL*. Grâce à ce fichier édicté par le web service lors de la création ou la modification des web-applications, l'utilisateur sait exactement comment formuler sa requête et où l'envoyer pour qu'elle atteigne le web service capable d'y répondre correctement.

Pour obtenir ce fichier *WSDL* lié à un web service particulier, l'utilisateur peut, s'il sait où le trouver, interroger directement le web service si ce dernier le permet. L'utilisateur va alors effectuer les différentes étapes illustrées dans la Figure 2.

Recommandé par le **W3C** (**W**orld **W**ide **W**eb **C**onsortium), le **WSDL** (**W**eb **S**ervices **D**escription **L**anguage) est un langage basé sur le XML [W3S 2011a] [CHR 2001] qui permet de décrire des web services grâce aux informations susmentionnées.
Un document WSDL est donc un fichier XML contenant plusieurs éléments qui vont décrire le web service :

- la balise **types** **(1)** qui contient les types de données utilisés par le web service ;
- les balises **message** **(2)** qui vont décrire les différents éléments utilisés par une opération exécutée par le web service. On peut les comparer à des arguments pour des fonctions ;
- les balises **portType** **(3)** qui vont décrire les différentes opérations qui peuvent être exécutées par le web service. On peut les comparer à des fonctions que l'on peut appeler ;
- les balises **binding** **(4)** qui vont décrire, pour chaque port, les paramètres d'entrée/sortie des messages ainsi que le protocole de communication ;
- la balise **service** **(5)** contient les ports définis dans les balises **binding**.

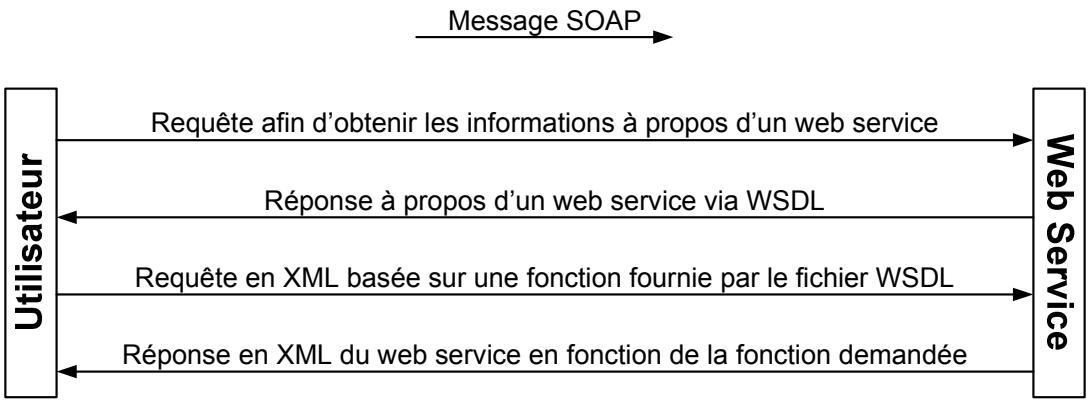

Figure 2 : Fonctionnement WSDL

Le Fichier 1 nous propose un exemple pratique, provenant du W3C :

```
Afin d'illustrer les différents éléments cités ci-dessus, le
<?xml version="1.0"?>
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
          xmlns:xsd1="http://example.com/stockquote/schema"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types> (1)
       <schema targetNamespace="http://example.com/stockquote/schema"
       xmlns="http://www.w3.org/2000/10/XMLSchema">
           <complexType name="TimePeriod">
              <all>
                  <element name="startTime" type="xsd:timeInstant"/>
                  <element name="endTime" type="xsd:timeInstant"/>
              </all>
           </complexType>
           <complexType name="ArrayOfFloat">
              <complexContent>
                  <restriction base="soapenc:Array">
                      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:float[]"/>
                  </restriction>
              </complexContent>
           </complexType>
       </schema>
    </types>
    <message name="GetTradePricesInput"> (2)
        <part name="tickerSymbol" element="xsd:string"/>
        <part name="timePeriod" element="xsd1:TimePeriod"/>
    </message>
```

```
    <message name="GetTradePricesOutput"> (2)
        <part name="result" type="xsd1:ArrayOfFloat"/>
        <part name="frequency" type="xsd:float"/>
    </message>
    <portType name="StockQuotePortType"> (3)
        <operation name="GetLastTradePrice" parameterOrder="tickerSymbol timePeriod
       frequency">
           <input message="tns:GetTradePricesInput"/>
           <output message="tns:GetTradePricesOutput"/>
        </operation>
    </portType>
    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType"> (4)
        <soap:binding style="rpc"
       transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetTradePrices">
           <soap:operation soapAction="http://example.com/GetTradePrices"/>
           <input>
               <soap:body use="encoded" namespace="http://example.com/stockquote"
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
           </input>
           <output>
               <soap:body use="encoded" namespace="http://example.com/stockquote"
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
           </output>
        </operation>>
    </binding>
    <service name="StockQuoteService"> (5)
        <documentation> My first service </documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
           <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>
</definitions>
```

Fichier 1 : Exemple d'un fichier WSDL [CHR 2001]

Dans le cas où l'utilisateur ne connait pas le web service ou que ce dernier ne permet pas de récupérer directement le fichier WSDL, il existe l'**UDDI** (**U**niversal **D**escription, **D**iscovery and **I**ntegration) [CLE 2004] qui est une spécification définie par l'**OASIS** (**O**rganization for the **A**dvancement of **S**tructured **I**nformation **S**tandards) et qui peut être comparée à un annuaire contenant toutes les informations utiles pour l'accès et l'utilisation des web services.

Elle se compose de 4 structures :

- **« BusinessEntity »** qui fournit une description de chaque organisation qui a ajouté son web service dans l'annuaire ;
- **« BusinessService »** qui permet de réaliser un groupement logique de web services. A ce stade, les web services sont également décrits mais de manière non technique ;
- **« BindingTemplate »** dont chaque structure de ce type représente un web service. C'est à ce niveau que l'on a accès aux coordonnées du web service représenté.
- **« Technical model »** ou **« tModel »**, qui donne accès à la description technique d'un web service, typiquement, son fichier WSDL.

Grâce à UDDI, l'utilisateur est capable de trouver un web service et d'obtenir son fichier WSDL, sans avoir besoin de le connaitre au préalable.

La Figure 3 ci-dessous, nous montre les différentes étapes nécessaires pour que l'utilisateur puisse obtenir le fichier WSDL d'un web service grâce à UDDI.

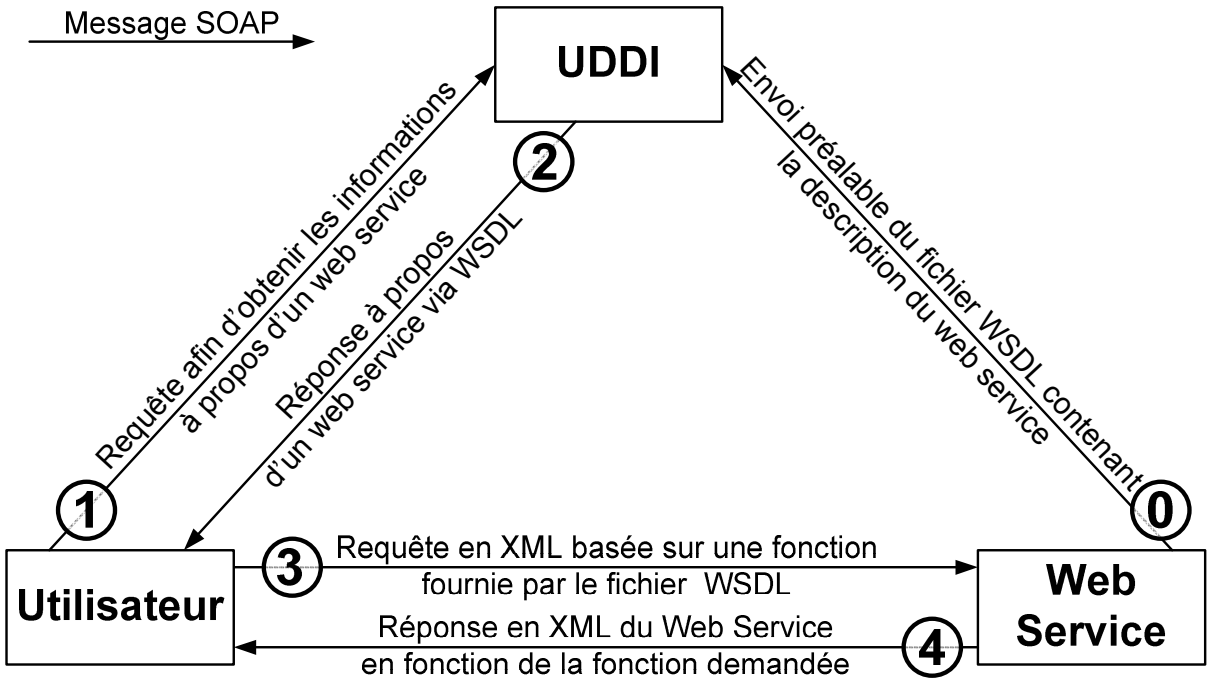

Figure 3 : Fonctionnement WSDL avec UDDI

A ce niveau, on se rend compte que, quel que soit la manière utilisée (avec ou sans UDDI) pour obtenir le fichier WSDL fourni par le web service, l'utilisateur est dépendant des informations contenues dans celui-ci pour le format à utiliser. De plus, il ne peut pas vraiment choisir les données qu'il désire en retour, du fait que la fonction appelée renvoie une réponse prédéfinie par le web service.
Il est donc parfois nécessaire à l'utilisateur de faire plusieurs appels, à plusieurs fonctions, pour recevoir toutes les informations désirées.
L'utilisateur est également dépendant du bon vouloir de l'organisation qui fournit le web service car c'est elle qui édicte les différentes fonctions que l'on peut utiliser et libre à elle de les supprimer/modifier arbitrairement. L'utilisateur peut ainsi se retrouver avec une application non fonctionnelle simplement à cause d'une modification de fonction.

Vous aurez pu constater dans les Figure 2 et Figure 3, que les web services emploient le protocole SOAP pour transporter les différents messages échangés entre les différentes entités. **SOAP** (**S**imple **O**bject **A**ccess **P**rotocol) est un protocole utilisé pour la communication via Internet entre différentes machines. Recommandé par le W3C, ce protocole est basé sur le langage XML [GUD 2007]. Typiquement, un message SOAP, comme illustré par la Figure 4 ci-dessous, est un fichier XML qui contient plusieurs éléments [W3S 2011b] :

- **« Envelope »**, qui permet d'identifier le fichier XML comme un message SOAP. Cet élément est la racine du message SOAP ;
- **« Header »**, qui contient les informations spécifiques à l'application (authentification, sécurité,…). Premier élément contenu dans la racine ;
- **« Body »**, qui contient les informations sur les appels et les réponses. Deuxième élément contenu dans la racine ;
- « **Fault »** qui contient les informations sur les erreurs. Cet élément optionnel est contenu dans le « **Body** ».

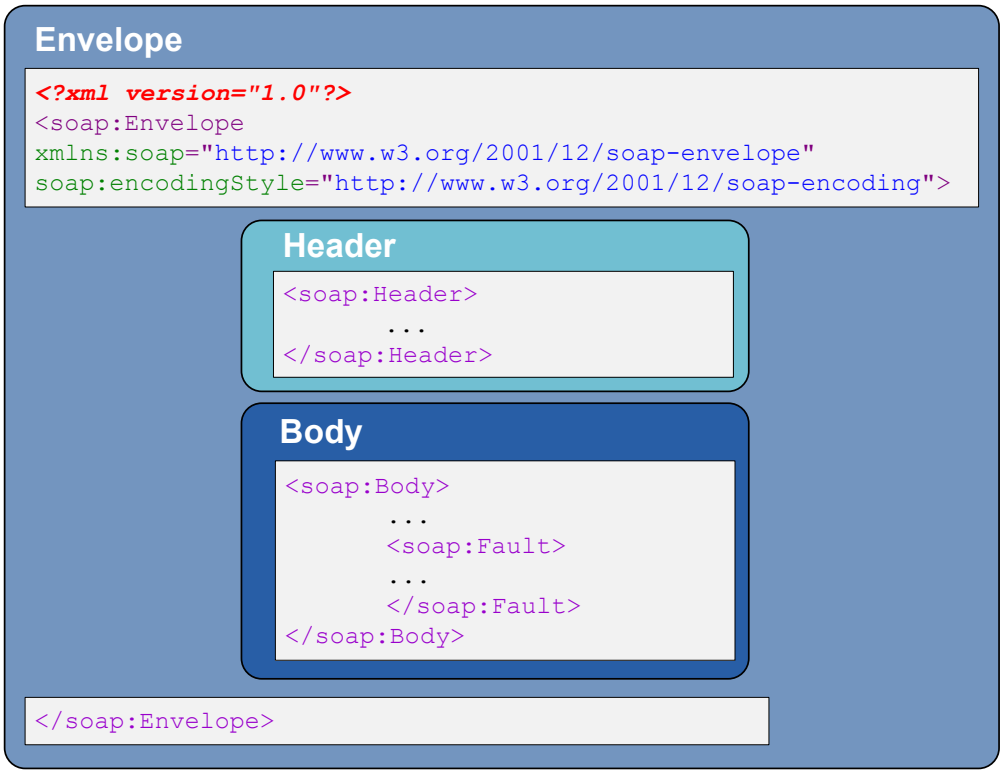

Figure 4 : Exemple d'un message SOAP

Ainsi formé, les messages SOAP sont transportés principalement via le protocole HTTP [W3S 2011b], bien que SOAP ne soit pas lié à un protocole particulier. Un avantage lié à cette encapsulation est qu'elle permet à ces messages de passer plus facilement à travers les firewalls, vu qu'ils laissent généralement passer les requêtes HTTP. SOAP est également indépendant de la plateforme et du langage utilisé. Il garantit ainsi, l'interopérabilité entre les différentes machines à partir du moment où elles sont capables de formuler et de comprendre des messages SOAP [VAN 2004].

Dans le cadre des web services utilisant le mode WSDL, il est utilisé pour communiquer le fichier WSDL aux différents acteurs ainsi que pour transmettre la requête au web service.

2.1.2. REST

REST (**R**epresentational **S**tate **T**ransfer) est, comme le décrit Roy Fielding dans sa thèse [FIE 2000], un style architectural exploitant les technologies et les protocoles du Web.

A la différence du mode WDSL où l'utilisateur doit faire appel aux différentes fonctions fournies par le web service, en mode REST, il identifie les ressources intéressantes et y accède directement via le protocole http, comme nous le montre la Figure 5 ci-dessous.

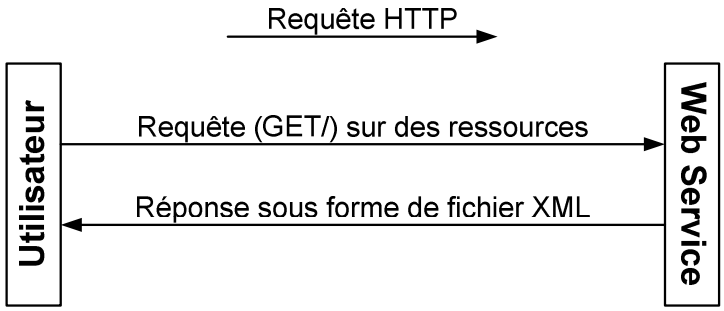

Figure 5 : Fonctionnement REST

Contrairement à WSDL, REST n'est pas un standard et il n'existe donc pas de spécifications précises à respecter. Il convient donc à l'utilisateur de maitriser REST avant de développer une application ou un web service utilisant ce style [FIG 2006].

Voici quelques points à respecter pour avoir une architecture REST [ROD 2008] [RIC 2007] :

- On doit utiliser les méthodes HTTP d'une manière compatible avec celle définie par le protocole. Il nous suffit donc d'établir une correspondance entre les opérations à effectuer sur le serveur et les méthodes http :
  - lorsque l'on veut créer une ressource, on doit utiliser la méthode POST ;
  - lors de la récupération d'une ressource, on doit utiliser la méthode GET ;
  - pour modifier ou mettre à jour une ressource, on doit utiliser la méthode PUT ;
  - afin de supprimer une ressource, on doit utiliser la méthode DELETE.

- la communication entre une application utilisateur et un web service doit être sans état, c'est-à-dire que l'application utilisateur doit envoyer, dans sa requête, toutes les informations nécessaires pour que le web service formule directement sa réponse et vice-versa. Ceci permet de minimiser le travail du coté web service qui pourra ainsi, répondre à un plus grand nombre de requêtes [WIK 2011] ;
- utilisation d'**URI** (**U**niform **R**esource **I**dentifier) dont la structure doit être prévisible et simple à comprendre afin d'accéder facilement aux ressources disponibles. Une solution est de représenter une URI comme un chemin à travers un arbre représentant les différents dossiers ;
- cette utilisation d'URI permet également d'utiliser plus facilement un système de cache qui nous permettrait par la suite de réutiliser les données précédemment récupérées sur le web service car à chaque URI, correspond une ressource particulière. Ainsi, on peut réduire les appels au web service ;
- les types MIME doivent être supportés par le web service même si le XML est plus répandu.

Un web service respectant ces différents principes REST est donc dit RESTful.

2.2. Delivery Context Ontology

Basé sur le standard W3C **W**eb **O**ntology **L**anguage (**OWL**), le fichier **D**elivery **C**ontext **O**ntology (**DCO**) permet de fournir des informations, dans un modèle formel, sur l'environnement dans lequel un appareil va interagir avec Internet ou d'autres services [CAN 2009]. L'utilisation d'une ontologie va permettre de représenter un ensemble de concepts par un modèle de donnée ainsi que les relations entre ces concepts.

Ce fichier DCO est divisé en quatre parties spécifiques [TIM 2009] :

- la partie **« Environnement »** qui fournit des informations à propos de la localisation de la machine ainsi que des informations concernant le réseau ;
- la partie **« Hardware »** qui, comme son nom l'indique, contient des informations concernant le matériel présent dans la machine. On peut citer le CPU, la mémoire ou l'écran ;
- la partie **« Mesure »** qui définit tous les termes en relation avec les unités de mesures et propose également les moyens pour les convertir ;
- la partie **« Software »** qui définit les formats utilisables ainsi que d'autres informations concernant les logiciels présents qui sont susceptibles d'interagir avec le web service comme l'OS ou les web browsers.

Comme vous pouvez le constater, cette ontologie fournit le contexte d'utilisation dans lequel l'utilisateur interagit avec une plateforme particulière dans un environnement particulier en vue d'effectuer une tâche interactive. Ce fichier DCO contient une importante source d'informations particulièrement utile lorsque l'on désire réaliser des applications dépendantes du contexte.

2.2.1. Envoi du DCO

L'envoi du fichier DCO n'est actuellement pas défini par le W3C. Cependant, certaines pistes ont été envisagées lors de réflexions précédant la mise en place du DCO :

- **CC/PP** (**C**omposite **C**apability/**P**reference **P**rofiles) [KLY 2004] [W3C 2004] est une recommandation du W3C, datant de 2004, qui a pour objectif de fournir un format permettant à chaque machine de diffuser ses capacités à un serveur Web. Ainsi, un profil CC/PP décrit les capacités d'une machine de même que les références de l'utilisateur de celle-ci. Ces deux points permettent de définir correctement le contexte d'utilisation afin de permettre l'échange de ce profil, une solution mettant en place une extension du protocole HTTP a été proposée mais, en pratique, elle n'a pas été largement mise en œuvre [GIM 2006].

- **UAProf** (**U**ser **A**gent **P**rofile) est basé sur le format CC/PP. Il a été développé, par le WAP Forum et par l'Open Mobile Alliance, plus spécifiquement pour la description de téléphones mobiles [JER 2009].
  A la différence de CC/PP, UAProf utilise le header du protocole HTTP pour transmettre un lien vers le fichier contenant le contexte d'utilisation [GIM 2006]. Cependant, si l'on n'a pas encore interagi avec le matériel, on ne peut pas avoir accès à son UAProf. De plus, tous les appareils n'envoient pas de fichier UAProf.
  Une solution à ce problème consiste, pour peu que l'on ait réussi à déterminer l'appareil utilisé, à demander le profil à WURFL qui est une base de données open-source répertoriant les fichiers UAProf des différents appareils mobiles disponibles.

Grâce à ces deux prédécesseurs du fichier DCO, nous pouvons émettre l'hypothèse que, dans l'avenir, le transfert du DCO se déroulera probablement via le protocole HTTP. Mais vu l'évolution rapide du domaine, cela reste une supposition personnelle.

2.3. Les solutions existantes

Dans cette section, nous allons présenter quelques solutions existantes dans le monde de l'adaptation du web vers le mobile, qu'elles se retrouvent sous la forme d'un package complet prêt à être utilisé, ou bien simplement un prototype, un concept, une idée. Dans un premier temps, nous avons trouvé les candidats et les avons analysés (avantages, inconvénients, limites). Nous les avons ensuite comparés, afin d'en retenir les concepts les plus pertinents, sur lesquels nous pourrions nous baser pour commencer notre propre solution.

2.3.1. Classification théorique

Selon Timothy Bickmore [BIC 1999], il est possible de classifier les différentes approches existantes pour l'adaptation du web en mobile. Il propose pour cela cinq catégories :

11. **'Device-specific authoring'** : il existe plusieurs versions du site web, une spécifique pour chaque type d'appareil cible (par exemple : une version PC, une version Smartphone et une version Tablet) ;
12. **'Multiple-device authoring'** : il n'y a qu'une seule version du site, avec des sous-versions pour les différents appareils cibles (comme par exemple l'utilisation de différentes feuilles de style CSS en HTML) ;
13. **'Client-side navigation'** : l'adaptation est réalisée directement par le client (des barres de défilement ou la possibilité d'un zoom sur la zone d'affichage du document en sont des exemples) ;
14. **'Page filtering'** : permet aux utilisateurs de voir uniquement les parties d'une page qui les intéressent ; l'adaptation est obtenue par un filtre du contenu.
15. **'Automatic re-authoring'** : une version du site existe, celle-ci est ensuite automatiquement adaptée pour l'appareil cible (par exemple, réduire toutes les images JPEG de 50% quand il s'agit d'un Smartphone) ;

Cette dernière catégorie **'Automatic re-authoring'** peut être séparée en deux sous catégories :

a. **'Transducing'** : la structure originale est préservée et les éléments qu'elle contient sont adaptés ;
b. **'Transforming'** : en plus des éléments qu'elle contient, toute la structure est transformée et adaptée.

Chacune de ces approches a ses avantages et inconvénients, le mieux étant de trouver un compromis.

‘Device-specific authoring’ est généralement la manière d'obtenir les meilleurs résultats, en raison de l'implication directe des concepteurs, mais limite l'accès de l'utilisateur à un plus faible ensemble de pages disponibles sur le web.

‘Multiple-device authoring’ a besoin de beaucoup plus de travail manuel au niveau de la conception que la création d'une plateforme unique, de type « ordinateur de bureau », tandis qu’il faut moins d'effort pour la création d’un site spécifique à un appareil.

‘Client-side navigation’ permettra de bien fonctionner si une bonne série de techniques de visualisation peuvent être développées, mais exige que le site soit envoyé à la machine cliente en une seule fois, afin de ne pas perdre de la bande passante, ni de la mémoire.

‘Automatic re-authoring’ ne nécessite pas de travail supplémentaire de la part du concepteur du site pour que toutes les pages soient accessibles. Les pages « re-authored » exigent moins de bande passante et de mémoire que les pages d’origine. Si cette technique peut produire des pages « re-authored » lisibles, navigables et esthétiques, sans perte d'information, alors elle est supérieure aux autres approches.

‘Page filtering’ retourne uniquement une petite partie d'une page qui est facilement navigable. Cette approche est idéale dans les situations où l'utilisateur suit de près une page particulière, dont la mise en page est fixe, mais dont le contenu évolue, vu qu’il peut ajuster ses filtres en fonction du format de la page. Inversement, en appliquant des filtres aux pages que l'utilisateur n'a jamais vus auparavant, cela ne peut pas produire les résultats souhaités.

Dans la suite de cette section, vous pourrez trouver une explication des quelques solutions appartenant toutes à la catégorie **‘Automatic re-authoring Transforming’** que nous avons découvert. Des exemples, un schéma de fonctionnement ou une capture d’écran seront fournis pour des explications plus détaillées.

2.3.2. Description des solutions pratiques

M-Links [SCH 2001] [HIL 2003] est un système de proxy middleware créé en 2001. Il va permettre aux utilisateurs de téléphones mobiles d'obtenir une liste de liens tirés de la page web qu'ils auraient traditionnellement visitée avec leur ordinateur de bureau. Ils pourront ensuite parcourir cette liste de la même manière que s'ils parcourraient l'arborescence de dossiers se trouvant en local sur leur machine (Figure 6 - **a**), pour localiser un fichier par exemple.

Quand l'utilisateur trouve un lien, il lui est possible d'appeler un service ' TOOLS ' (analogue au clic droit de sa souris sur un document en local), afin d'utiliser le même principe de menu contextuel (Figure 6 - **b**) et ainsi avoir la possibilité d'une liste d'actions à réaliser sur le lien courant.

La Figure 7 montre les différents liens et interactions entre les serveurs web où se trouvent les pages originales, le serveur dédié au service M-Link (où s'opère la transformation de cette page) et l'appareil mobile surfant sur le web.

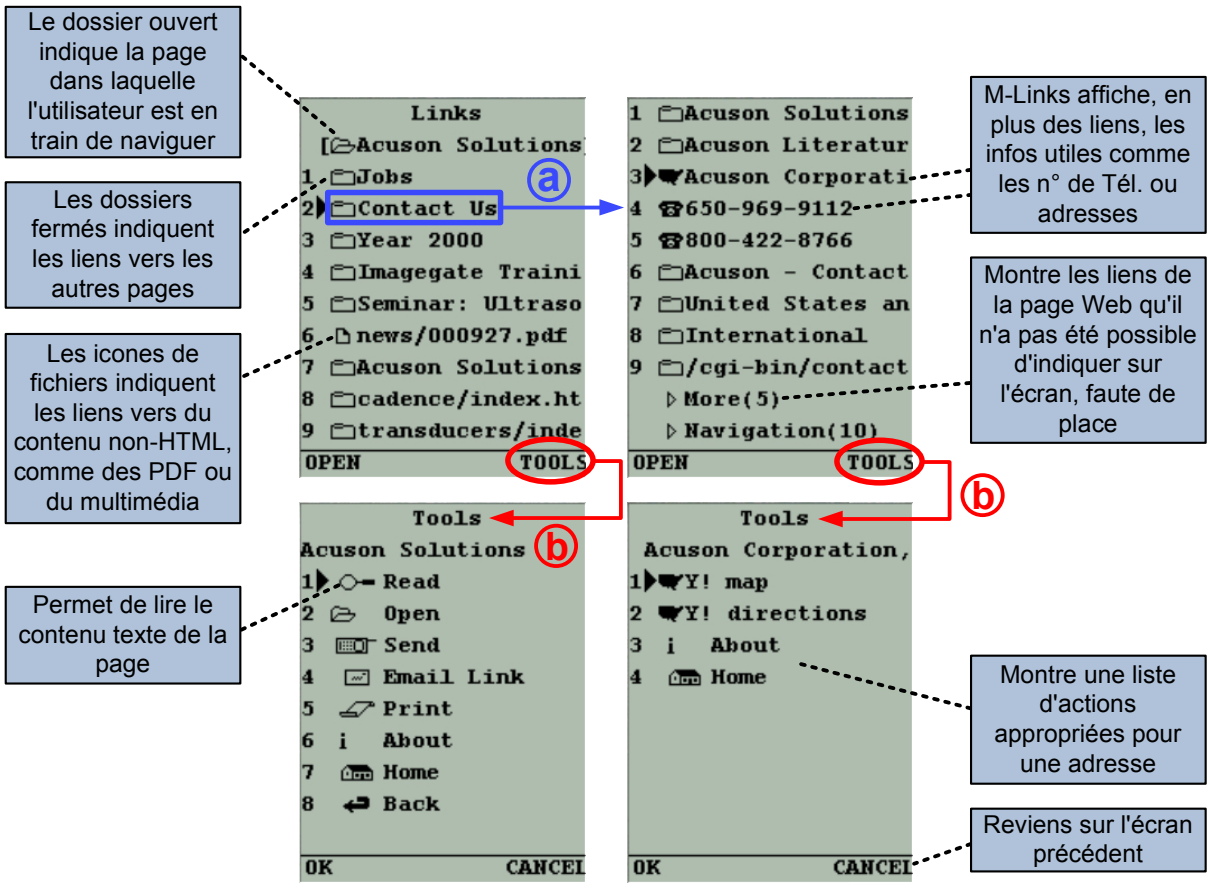

Figure 6 : Utilisation de M-Links sur un téléphone web Neopoint 1000 (Inspiré de [HIL 2003])

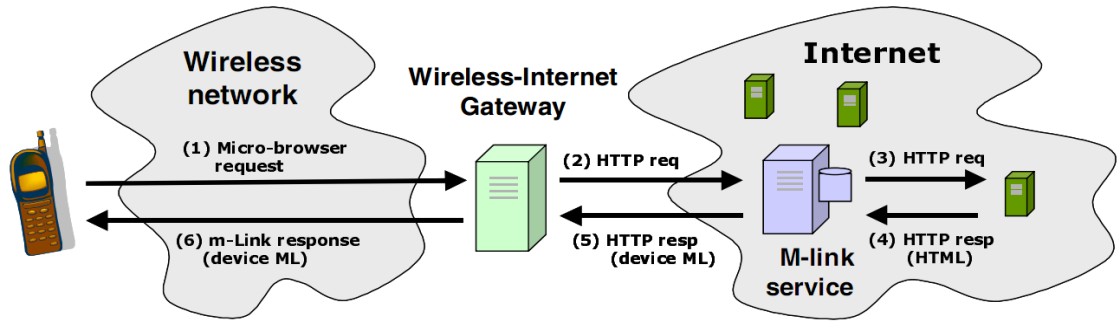

Figure 7 : Liens et interaction (Source : [SCH 2001])

La Figure 8 détaille quant à elle l'architecture de M-Links. Le **« User interface generator »** convertit toutes les informations dans une forme appropriée pour l'appareil. Le **« Service manager »** gère la liste des différents services applicables sur la page (enregistrement, impression, lecture, ...). Et le **« Link engine »** qui s'occupe de toute la gestion des liens.

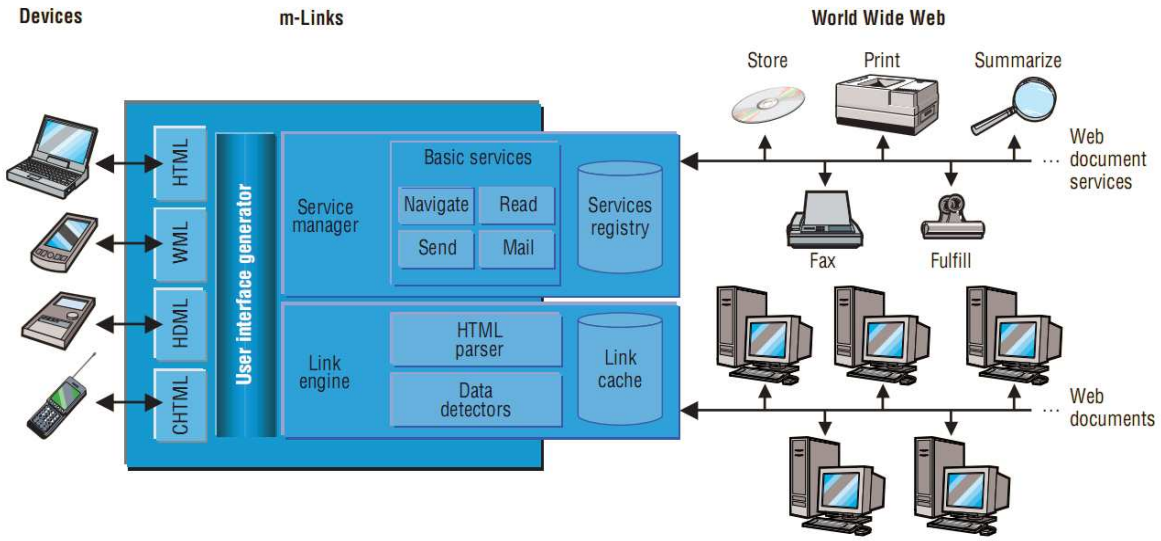

Figure 8 : Architecture M-Links (Source: [SCH 2002])

M-Links est à la fois simple à utiliser et puissant. Sa simplicité provient de l'interface de navigation : séparer les liens du contenu de la page rend la navigation aussi simple que la sélection d'un lien à partir d'une liste. Sa puissance provient de l'interface d'action : parce que les utilisateurs peuvent appliquer différents services basés sur le Web à n'importe quel lien, ils peuvent faire plus avec le contenu d'une page que de simplement le lire sur leur téléphone.

Sur les quelques téléphones portables qui ne sont pas encore adaptés pour gérer du contenu d'un certain type (comme les documents PDF, musiques MP3, vidéos MPEG,…), un utilisateur de M-Links pourra toujours « faire quelque chose » avec n'importe quel contenu qu'il trouvera. Par exemple, l'utilisateur qui navigue et trouve un fichier PDF, peut appliquer le service d'envoi d'e-mail à ce PDF (ou à son URL) pour une utilisation future sur son PC de bureau. [HIL 2011]
M-Links a aussi des inconvénients : lors de feedbacks entre les programmeurs et les partenaires utilisateurs du système, pour certains d'entre eux, la navigation se révélait éprouvante lors de la recherche du contenu souhaité. Ils ont souvent eu des difficultés à « relocaliser » le contenu qu'ils avaient pu situer avec leur ordinateur de bureau, sur le téléphone. Cette expérience est à la base de l'ajout d'une nouvelle fonctionnalité de filtrage : « Montrez-moi tous les liens que j'ai consulté auparavant ».
La majorité des utilisateurs interrogés se sentaient également « frustrés » d'avoir à s'enregistrer via le clavier de leur téléphone, alors qu'ils voudraient pouvoir créer leur compte à partir de leur ordinateur de bureau.

Skweezer [SKW 2011] a été créé en 2001 et est un service de retraitement de pages, qui formate et compresse le contenu de ces pages, en vue d'une utilisation par des appareils mobiles. Les pages Web qui sont consultées, via l'interface du service, par l'utilisateur sur son appareil mobile, transitent tout d'abord par des serveurs dédiés qui appliquent à ces pages un traitement « amincissant », pour ensuite être redirigées, dans leur version épurée, vers l'utilisateur. Plusieurs fonctionnalités sont disponibles :

- Le système intègre une reconnaissance du type d'appareil mobile dans l'optique d'une adaptation aux téléphones à faible mémoire interne.
- Il intègre également la détection des pages déjà optimisées sur lesquelles seules des transformations d'image sont appliquées ou encore le passage automatique vers l'équivalent mobile d'une page lorsqu'il existe.
- Si l'affichage des images est activé, il réduit le poids des images, sans les redimensionner.

- Il propose un outil de « Highlighting text » (surligne tous les termes demandés dans le texte de la page) ou le « Find-in-Page Search™ » (qui recherche séquentiellement un terme et affiche le reste de la page à partir de cet endroit. Tout le début de la page jusqu'au terme recherché n'est donc pas affiché).
- Il permet d'activer ou non la pagination, qui fractionne la page web visitée en plusieurs parties ; il est possible de passer à la page suivante, afin de parcourir ainsi l'entièreté de la page en plusieurs fois.
- Il est également possible pour l'utilisateur de s 'enregistrer et garder ainsi quelques informations à son sujet (langue préférée, activation des images, activation de la pagination, taille maximum (en Ko) que la page peut atteindre, création de signets sur sa page d'accueil pour les URL favorites).
- Il est multilingue : il propose sa propre interface dans plusieurs langues et permet la traduction des pages demandées.
- Il intègre aussi un répertoire reprenant quelques liens, classés par catégories, dans lesquels il est possible de rajouter ses propres liens.
- Il donne enfin des informations sur la page qu'il vient de remettre en forme, comme la taille, ou le pourcentage « d'amincissement ».

Il utilise la technologie REST et propose une API pour les personnes désirant intégrer ce système pour les liens internes et externes de leur site web. Le format d'URL à utiliser est de la forme :

`http://www.skweezer.com/s.aspx?q=www.uclouvain.be/epl&i=1&h=UCL&l=fr`

- **`q`** l'URL de la page à adapter ;
- **`i`** pour les images : `1` = avec, `0`=sans ;
- **`f`** le(s) mot(s) donné(s) au *Find-in-Page Search™* ;
- **`h`** le(s) mot(s) à surligner dans la page ;
- **`l`** le langage du bas de page de Skweezer.

L'avantage de Skweezer est qu'il fonctionne avec toutes les URL et qu'il enregistre, en quelque sorte, le « contexte » de l'utilisateur, puisqu'il lui est permis de choisir certains paramètres, comme par exemple le taille des pages pour la pagination lorsqu'elle est activée. Il vise aussi à réduire le temps de téléchargement de la page, en réduisant la taille de son contenu, en termes d'octets.

Par contre, il est possible de relever plusieurs inconvénients :

- aucune réorganisation de mise en page (texte brut, aligné à gauche) ;
- ne prend pas les CSS d'origines en compte ;
- il ne supporte pas la transformation de long texte, ne gère pas les tableaux.
- la traduction de la page est plutôt lente, voir ne fonctionne pas correctement ;
- certaines fonctionnalités ne sont pas achevées ;
- bugs ou messages d'erreurs du serveur.

Mowser [MOW 2011a] est un moteur d'adaptation de contenu Web pour téléphones mobiles créé en 2007. Il se veut différent des autres systèmes d'adaptation de par son côté complètement ouvert sur la façon dont les pages sont converties, afin de permettre aux éditeurs de prédire quels changements vont se produire, et les aider à créer leurs pages en conséquence.
Mowser peut modifier les images, le formatage du texte et/ou certains aspects fonctionnels de ces pages.
Si la version mobile d'un site existe déjà, Mowser respecte un standard (utilisé par Digg.com et Google Mobile Search) qui consiste à insérer un lien d'en-tête spécifique, qui prévient les adaptateurs de la redirection vers la version mobile, et de ne pas lancer le processus d'adaptation comme pour une autre page.
Son processus d'adaptation contient 7 étapes [MOW 2011b] :

1. **chargement de la page** et **suppression des scripts** ;
2. **analyse de l'en-tête** (titre de la page, lien vers la version mobile, type d'encodage,…) ;
3. **analyse des liens** (réécriture d'attributs comme *href* et *src*, les formulaires en méthodes POST, ou d'autres liens, pour les rediriger ensuite vers le proxy de Mowser. Comme les liens, les images de plus de 150 pixels sont également converties) ;
4. **suppression des balises superflues** (comme *style*, *link*, *object*, *applet*, *embed*, *iframe*, *basefont*, *map*,…) ;

5. **suppression des attributs non désirés** (ceux qui peuvent perturber le formatage, comme rel, style, align, width, heigth, bgcolor, border, size, onselect, onclick, onmouseover, onfocus, cellspacing, valign, background, …) ;
6. **conversion des balises de « bloc »** (transformation des *tables*, *tr*, *td*, *col*, *tfoot*, *colgroup*, *nobr*, *thead*, *th*, … en *div*) ;
7. **hachage de la page** (division de la page si celle-ci s'avère trop lourde).

Il utilise la technologie REST, et le format d'URL à utiliser est de la forme :

`http://mowser.com/web/www.uclouvain.be`

L'avantage principal de Mowser est l'utilisation d'une grande police et d'un grand interligne, ce qui permet d'aérer le contenu afin de le rendre lisible sans avoir besoin d'utiliser le zoom sur l'appareil mobile. L'inconvénient à cela est que le contenu prend alors plus de place, et qu'il est nécessaire de défiler beaucoup vers le bas pour tout voir. Un autre inconvénient est que Mowser ne redimensionne que les images supérieures à 150 pixels. Il fonctionne pour tous les sites, mais ne gère pas les longs textes et les tableaux. Le texte affiché est assez brut, sans mise en forme, puisqu'il ignore les couleurs et propriétés CSS mais préserve quelques balises HTML pour la mise en forme du texte, comme <*b*> (**gras**), <*i*> (*italique*), <*u*> (souligné).
Il vise à réduire l'utilisation verticale de l'écran, mais s'avère être (pour certains types d'appareils) au détriment d'effectuer beaucoup de défilement horizontal.
Il reste quelques bugs (admis par les développeurs de Mowser) que ce soit dans l'étape 3 ou 7. Il ne gère pas les *iframe*, ni les feuilles de style. Nous avons aussi relevé un problème d'encodage, qui survient pour certains sites, et parfois certaines parties d'une même page, comme par exemple dans la requête pour le site http://www.uclouvain.be, à un endroit, on peut voir un problème d'apostrophe (« Lâ€™UCL a conclu… ») alors que plus loin dans la page, on peut voir une apostrophe qui est bien affichée (« Des chercheurs de L'UCL … »).

Customizable Two-dimensional Semantic Redesign [PAT 2010] est une solution proposée comme un nouvel outil pour l'adaptation « desktop-to-mobile », basé sur un algorithme qui permet, dans un premier temps, d'adapter les éléments basiques :

- les images sont rétrécies tout en gardant leurs proportions ;
- certains éléments sont remplacés par d'autres, sémantiquement équivalents, mais prenant moins d'espace sur l'écran (par exemple, une liste peut être remplacée par un menu déroulant) ;
- les longs textes sont réduits, et affichés en entier uniquement sur demande ;
- les images et textes dans un tableau sont redimensionnés.

Après ces transformations basiques, la partie de l'algorithme s'occupant du fractionnement de pages est activée ou non, en fonction du coût global qui a été recalculé (le fractionnement de pages s'occupe lui de placer, dans une autre page, une partie de tableau trop grande par rapport à la largeur de l'écran, ou de la partie d'un long texte réduit qu'il est possible d'accéder à la demande de l'utilisateur.)

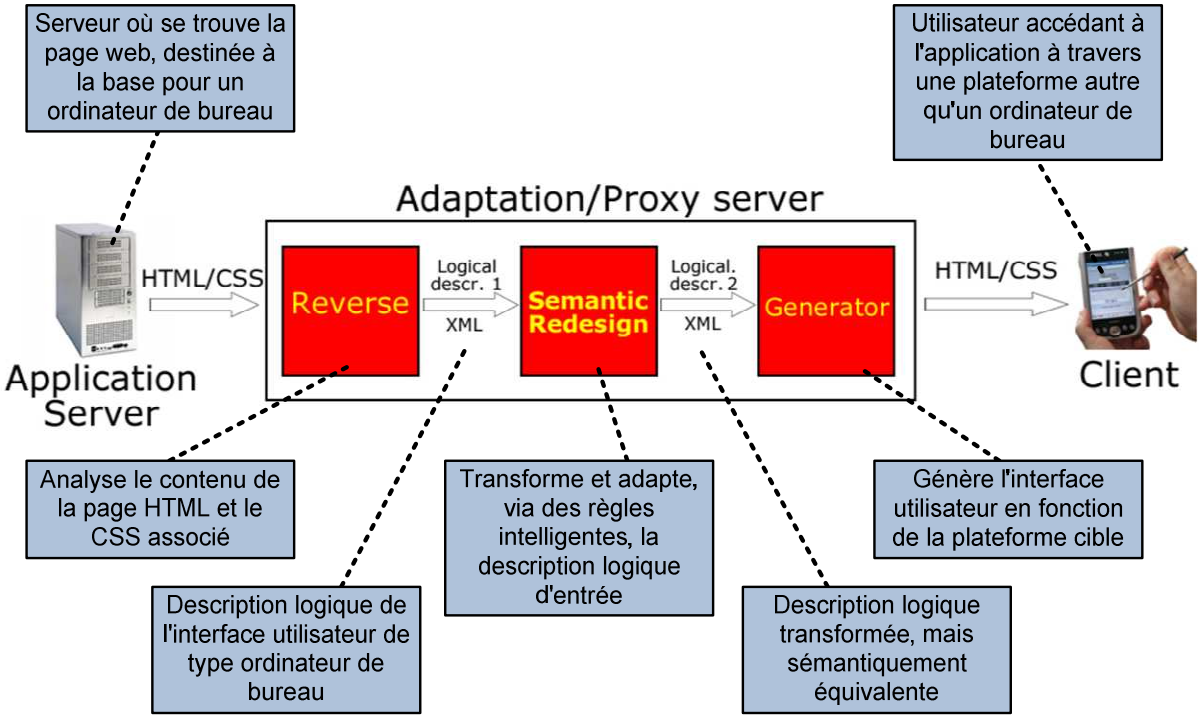

Figure 9 : Schéma de fonctionnement du Semantic Redisign (Inspiré de [PAT 2010])

L'avantage de « Customizable Two-dimensional Semantic Redesign » est qu'il respecte un minimum le style de base du site (de par les feuilles de style CSS intégrées dans le processus d'adaptation), et que les éléments d'interface peuvent être remplacés par d'autres éléments, sémantiquement équivalentes, mais nécessitant moins d'espace sur l'écran. L'utilisateur a la possibilité de configurer le processus d'adaptation en modifiant les paramètres de personnalisation, comme la taille minimum et maximum du texte, la police de celui-ci, la transformation des boutons radios ou listes déroulantes, la hauteur et largeur minimum et maximum des images,...

GoMobi [GOM 2011] a été la première plateforme de mobilisation de contenu au monde. C'est un mélange de système de gestion de contenu traditionnel et de constructeur de site web mobile pratiquement automatique. Cette plateforme permet de créer facilement un site, directement en « version mobile ».
Il propose plusieurs fonctionnalités « mobiles-friendly » ce qui le rend facile d'utilisation pour les clients désirant accéder aux informations recherchées quand ils en ont le plus besoin :

- **calendrier** : les utilisateurs peuvent publier leurs évènements, comme des ventes ou des conférences, via un lien XML depuis Google Calendar ;
- **nous trouver** : donne la direction et les distances séparant l'utilisateur d'une entreprise depuis son emplacement actuel via GPS ;
- **gestion de la configuration multilingue** : soit directement à la création du site dans une des langues proposées, ou en permettant aux visiteurs de traduire dans la langue désirée ;
- **contactez-nous** : téléphone, adresse, Facebook, heures d'ouvertures ;
- **gestion de photos** par l'usage d'une galerie ;
- **news** sous format RSS ;
- **lien** vers la version « complète » du site existant ;
- **laisser un message.**

L'avantage de cette solution goMobi est qu'elle est très rapide et facile à utiliser, car assez intuitive pour mettre en place cette liste de fonctionnalités. Très « user-friendly », il fonctionne sur tous les types d'appareils mobiles, allant des téléphones les plus basiques aux Smartphones les plus développés.
Par contre, il est impossible d'entrer du code personnel, comme du JavaScript ou PHP, vu que c'est le système qui gère lui-même la création de la page. Il n'y a pas plus de fonctionnalité que ce qui se trouve déjà fait ; il n'est pas possible d'implémenter la sienne.

mobiReady [MOB 2011] n'est pas en soit une solution d'adaptation du web vers le mobile, mais mérite sa place dans cette section, puisqu'elle pose une question toute simple aux développeurs de site Web : « *Est-ce que votre site est prêt pour le mobile ?* ». C'est en effet intéressant de se demander si un site web classique, qui n'aurait pas forcément été développé avec une optique mobile, peut être consulté de manière efficace sur un appareil mobile.
mobiReady permet de se donner une idée quant à cette perspective, en proposant un rapport d'un ensemble de tests, comme le calcul de sa taille ou le temps nécessaire à son affichage. Il reflète l'état général de préparation du site avec un score exprimé sur une échelle de 1 à 5. Il donne également des conseils aux développeurs pour améliorer les différents aspects à prendre en compte.

Le navigateur installé sur le matériel pourrait lui-même directement interpréter les pages web que l'utilisateur visite. Évidemment, de telles adaptations ne sont pas nécessairement plus rapides (vu qu'il s'agit d'une charge supplémentaire) qu'une navigation et un affichage direct.
Des techniques existent déjà, comme le « Small Screen Rendering » sur le mini browser Opera, qui reformate la page en une seule colonne verticale de sorte qu'il suffit uniquement de faire défiler de haut en bas (et ainsi éviter le défilement horizontal).

Les longues listes sont réduites automatiquement par une fonction de « pliage de contenu » : un signe (+) est affiché à côté du contenu plié, et un clic permet de le déplier et l'afficher [OPE 2011]. Le navigateur Safari de l'iPhone redimensionne automatiquement la page Web à la taille de l'écran et permet à l'utilisateur de zoomer et dézoomer par de simples gestes via l'interface tactile. Teasheark est quant à lui en coopération avec des serveurs qui lui sont dédiés. Ils transcodent et précompilent en partie les sites Web, et transmettent ensuite les résultats à l'appareil mobile [TEA 2011].

Ils sont nombreux les navigateurs (Firefox for mobile, Android browser, Teashark, Opera Mini, Skyfire Mobile Browser, Links, Web Os Browser, uZard Web, Nokia Series 40 Browser,…) à permettre ces mêmes spécificités, ou à en proposer d'autres.

Cette liste non exhaustive rappelle la véritable « guerre des navigateurs » qui se joue depuis quelques années sur les plateformes de bureau, et qui se joue inévitablement à l'heure actuelle dans le domaine du mobile.

2.3.3. Comparaison de ces approches

Dans la Figure 10 se retrouve une comparaison entre Mowser et Skweezer. Nous avons visité le site http://www.uclouvain.be avec une tablette « Samsung Galaxy Tab », et avons réalisé des captures d'écran de l'affichage rendu par les deux solutions. On remarquera immédiatement la différence de taille de police et d'interligne. Même si la lecture est plus aisée, sans besoin de réaliser un zoom, cela oblige Mowser à afficher en plusieurs fois (suivant le défilement vertical de la page vers le bas) ce que Skweezer affiche en un seul écran.

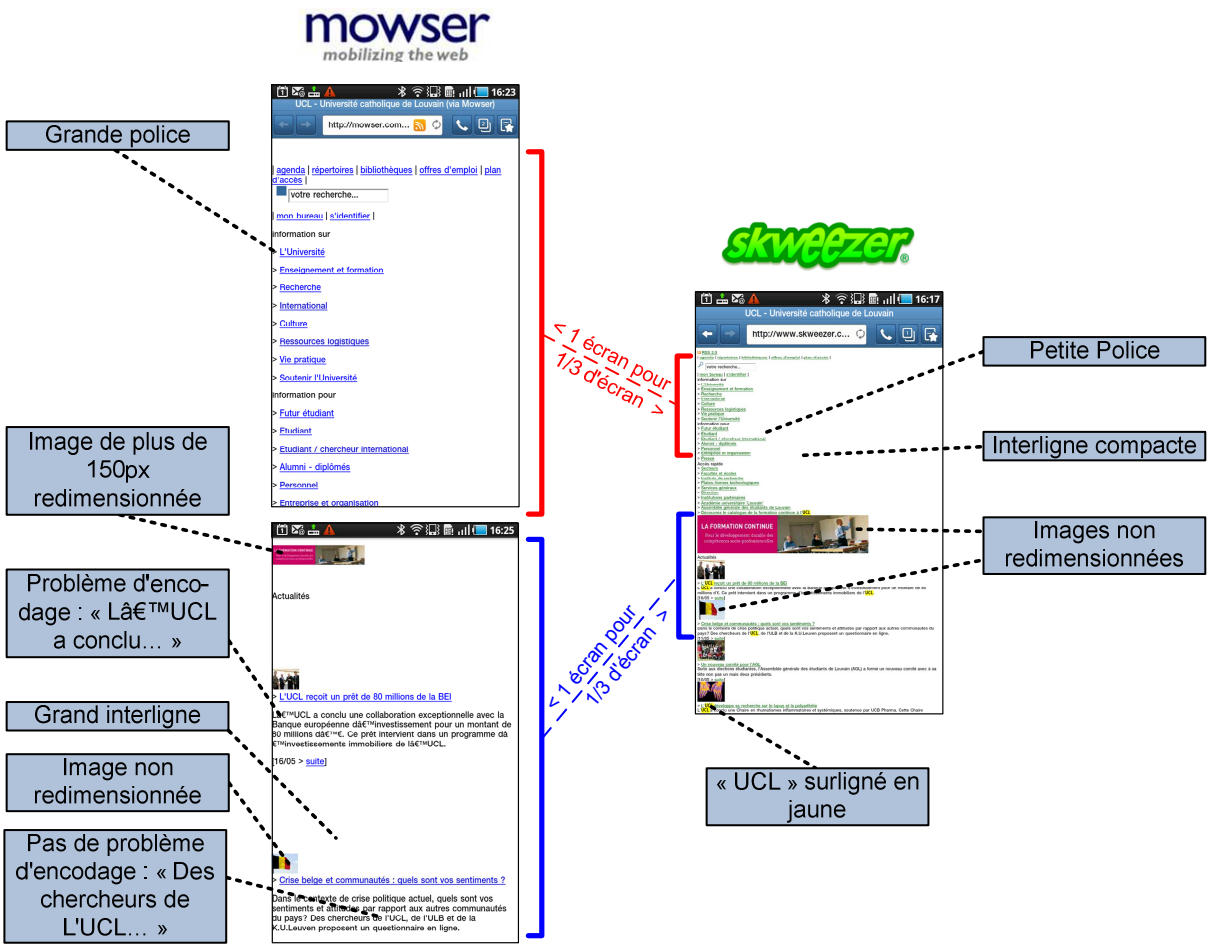

Figure 10 : Mowser VS Skweezer – Sur une tablette « Samsung Galaxy Tab »

Sur les Figure 11 et Figure 12, le prototype « Semantic Redisign » est ajoutée à la comparaison. Cette solution s'avère être la plus flexible des trois, puisqu'elle garde une certaine cohérence avec le style d'origine, gagne de la place en transformant sémantiquement certains éléments et affiche correctement les tableaux, puisqu'elle gère ceux-ci, en les fractionnant s'ils sont trop grands.

S'il y a de longs textes, ils seront entièrement affichés par Mowser et Skweezer, même s'ils ne sont pas forcément pertinents, tandis que Semantic Redisign proposera automatiquement un fractionnement des pages, en proposant de lire l'entièreté du texte via un lien à visiter.

Les trois solutions s'accordent sur le point de la gestion du contenu plus spécial, comme le Flash ou les applets Java, qui ne sont adaptés par aucune d'entre elles.

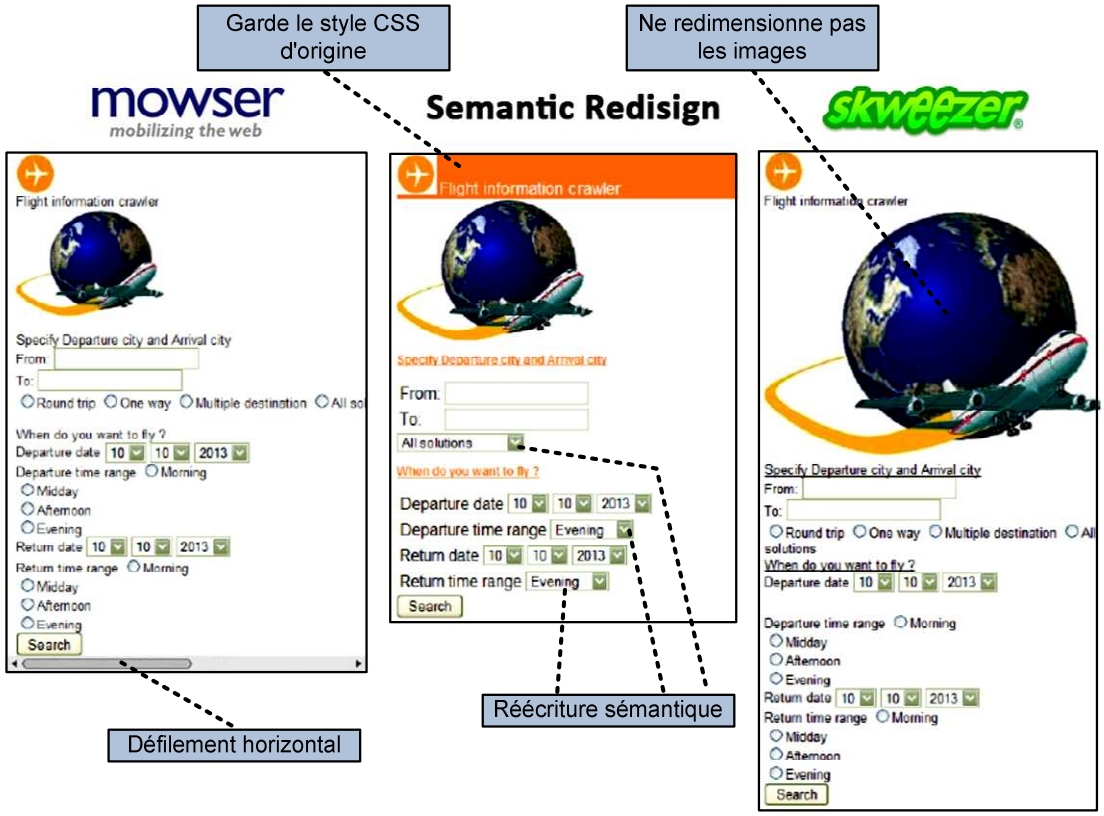

Figure 11 : Comparaison des solutions - Requête (Inspiré de [PAT 2010])

Notez sur la Figure 12 que Skweezer affiche deux informations intéressantes sur la page : sa taille, ainsi que le pourcentage de compression par rapport à la page d'origine.

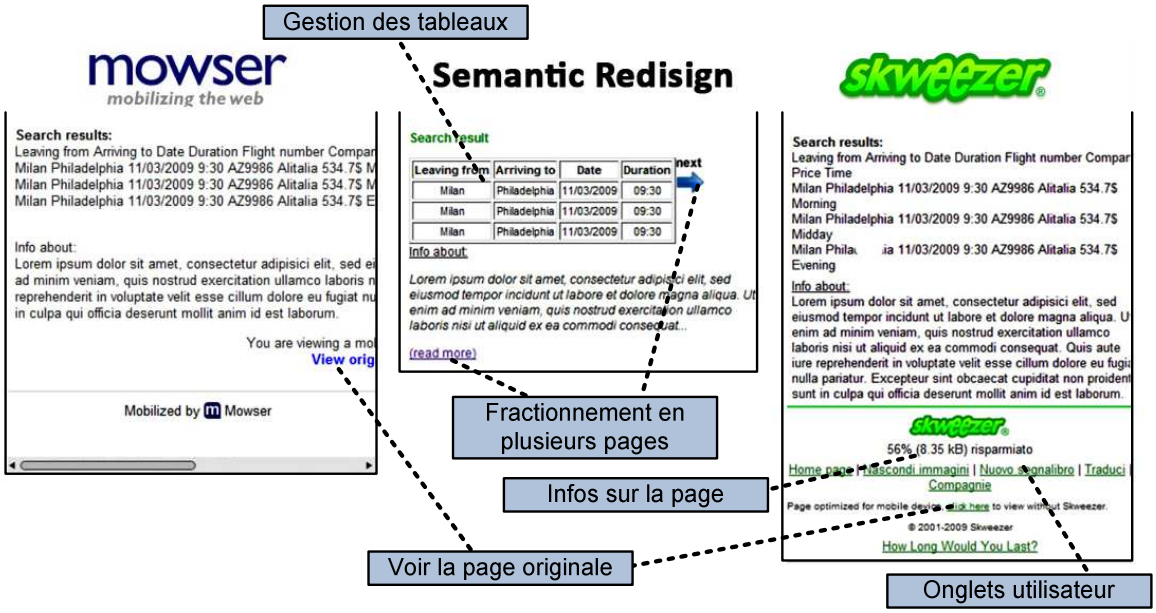

Figure 12 : Comparaison des solutions - Résultats (Inspiré de [PAT 2010])

Dans le Tableau 1 ci-dessous, se retrouvent les cinq solutions d'adaptation que nous avons développées dans la section précédente.

Légende ✗ : Non ✓ : Oui ? : Inconnu N.A : Non Applicable

	M-Links	mowser	Semantic Redesign	skweezer	goMobi
Taille de la page reçue connue	?	?	?	✓	?
Redimensionne les images	N.A	✓	✓	✗	N.A
Style d'origine conservé	✗	✗	✓	✗	N.A
Personnalisation des paramètres d'adaptation	?	✗	✓	✗	N.A
Gestion des tableaux	✗	✗	✓	✗	N.A
Gestion des longs textes	✗	✗	✓	✗	N.A
Fractionnement en plusieurs pages	✗	✗	✓	✓	N.A
Défilement	Vertical	Horizontal	Vertical	Vertical	Vertical
Signets pour URL favoris	?	✗	✗	✓	?
Espace utilisateur	✓	✓	✗	✓	?
Réduis la qualité des images	N.A	✓	?	✓	N.A
Police	Petite	Grande	En proportion de l'originale	Petite	Grande
Gratuit	?	✓	✓	✓	✗
Gestion de contenu Flash	✗	✗	✗	✗	✓

Tableau 1 : Comparatif des solutions

Comme vous venez de le voir, il est possible de comparer les trois solutions que sont Mowser, Semantic Redesign et Swkeezer, étant donné qu'elles sont basées sur le même concept d'adaptation de page. Par contre, ce n'est pas l'objectif visé par goMobi, puisqu'il se veut être une solution de création de pages directement adaptées au web mobile. Certains critères sont donc « Non Applicable »  . Lorsqu'il ne nous est pas possible de répondre au critère, nous indiquons « Inconnu »  .

2.4. Les concepts retenus

Les concepts les plus intéressants retirés des solutions proposées dans la section précédente, vont nous permettre de préparer une base de travail pour l'élaboration de notre prototype sur laquelle nous pourrons réfléchir. Toutes les idées ne seront pas forcément développées, mais elles ont l'avantage d'alimenter les discussions concernant nos choix d'implémentation. Nous retiendrons donc :

- le fractionnement de pages ;
- l'affichage de statistiques sur la page (taille, temps d'affichage,…) ;
- le type d'architecture « proxy middleware » de M-Links ;
- le choix d'afficher ou non les images (économie de bande passante) ;
- l'enregistrement des utilisateurs (pour du paramétrage personnalisé) ;
- éviter l'aspect « brut » de l'information en utilisant des styles ;
- la taille de la police d'écriture assez grande ;
- la personnalisation des paramètres d'adaptation ;
- le redimensionnement des images.

2.5. Conclusion

Ce chapitre fut l'occasion de découvrir l'intérêt du fichier DCO détaillant le contexte d'utilisation, mais aussi l'envoi de ce fichier qui n'est pas encore géré au niveau du W3C, du fait de sa standardisation récente. Nous avons pu également entrevoir deux modes de communication différents des web services. Plusieurs solutions dans le monde de l'adaptation du web vers le mobile ont aussi été relevées, avec chacune leurs forces et leurs faiblesses. Et bien que la majorité des solutions présentées consiste à adapter une page web, nous avons pu mettre en évidence plusieurs concepts qui nous seront utiles.

Grâce à cette base de travail, nous pouvons désormais réfléchir au sujet de nos propres choix d'implémentation. Dans le chapitre suivant, nous allons décrire ces différents choix qui nous ont permis d'arriver à notre solution ainsi que la manière dont elle fonctionne concrètement.

Chapitre 3 : Notre solution

Faire l'état de l'art dans le chapitre précédent nous a permis de définir les bases à partir desquelles nous allons appuyer notre travail de réflexion sur les choix que nous aurons à prendre durant l'implémentation de notre prototype.
Dans le chapitre qui nous occupe à présent, nous allons nous concentrer sur la description de notre solution, en détaillant son schéma de fonctionnement et l'analyse de son architecture. Nous aborderons également la réflexion sur le choix du langage de programmation utilisé pour l'implémentation. Nous définirons ensuite les différentes règles d'adaptation nécessaires au bon déroulement du processus d'adaptation. Nous reprendrons enfin l'ensemble des étapes importantes sous forme de diagrammes de séquence.

3.1. Schéma de notre solution

La Figure 13 montre le schéma général du fonctionnement de notre solution. Elle vous présente, en quatre grandes étapes théoriques, la manière dont le client va recevoir le résultat de sa requête :

(1) le client envoie au Proxy son fichier DCO et sa requête ;

(2) en fonction du fichier DCO, le Proxy génère une requête personnalisée qu'il envoie au web service ;

(3) le web service renvoie sa réponse contenue dans un fichier XML ;

(4) le Proxy envoie la réponse au client via une page HTML qu'il a générée.

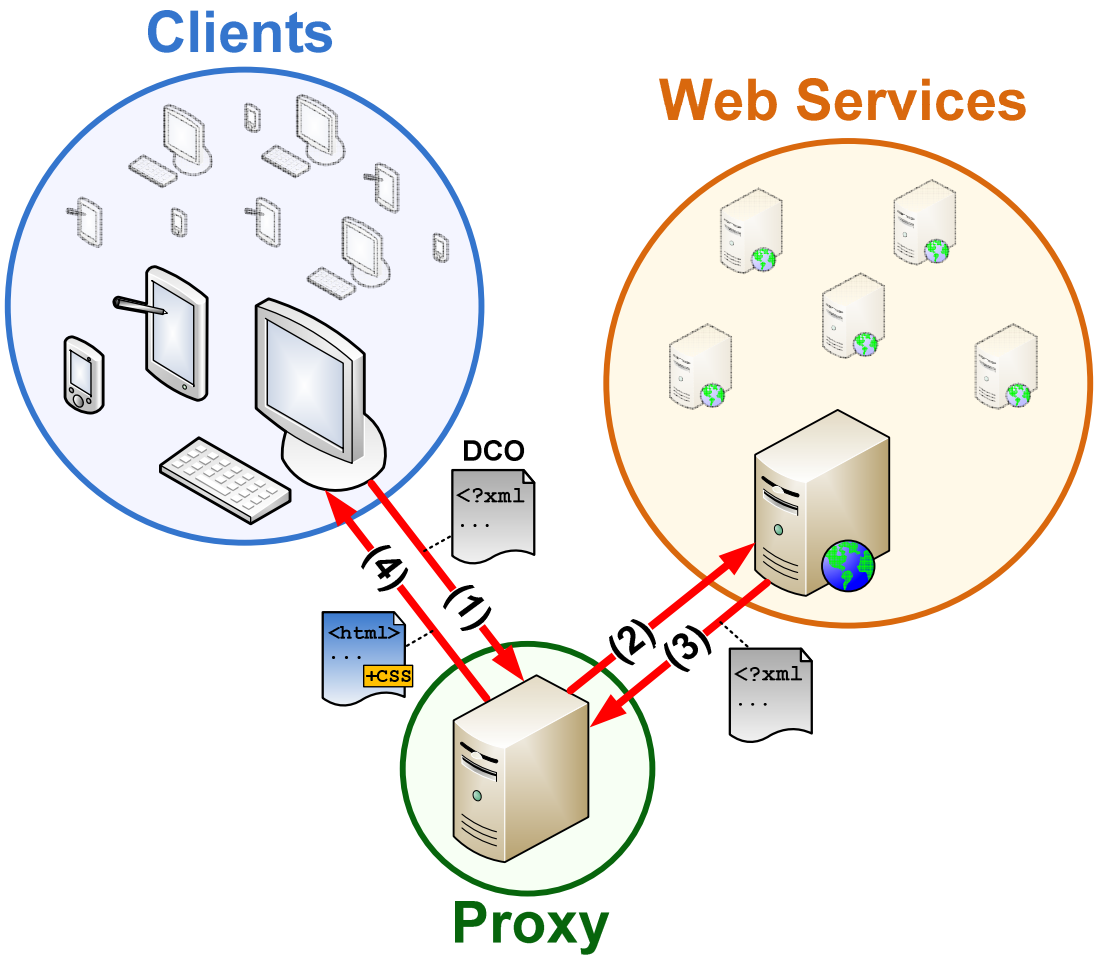

Figure 13 : Schéma général du fonctionnement de notre solution

Les points suivants permettront de passer à la loupe les différentes parties qui composent notre solution.

3.1.1. Clients

Nous n'avons considéré que trois catégories de clients (Figure 14) : les Smartphones, les tablettes/Pocket PC et les PC de bureau. Nous avons laissé de côté les matériels aux tailles trop extrêmes, comme les écrans géants (grandes télévisions numériques ou autres projecteurs) ainsi que les montres (du moins, celles qui permettent à un utilisateur de surfer sur le Web).

A ce niveau, il est bon de faire la distinction entre un *client*, un *utilisateur* et un *matériel*. L'*utilisateur* est la personne physique qui se trouve derrière le *matériel* qu'il utilise pour surfer ; ils forment à eux deux un *client*. Par exemple, lorsque nous parlerons d'un *client Smartphone*, il s'agira bien d'un *utilisateur* derrière un *matériel* de type Smartphone. Cette distinction est importante, puisqu'un utilisateur peut avoir plusieurs matériels en sa possession, et les utiliser indépendamment pour surfer sur la toile.

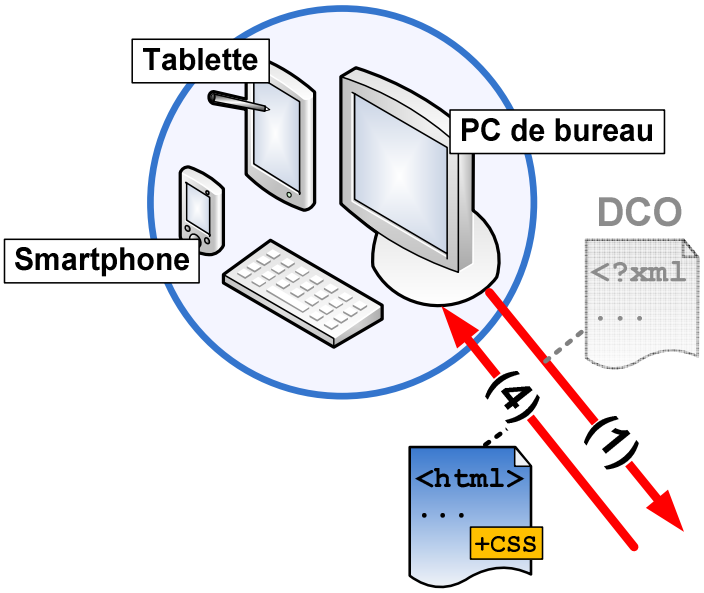

Figure 14 : Zoom sur les Clients

Comme énoncé dans la section 2.2.1, l'envoi du fichier DCO n'ayant pas encore été développé ni standardisé par le W3C, nous avons simulé cet envoi depuis le client vers le Proxy (Figure 14–**(1)**). En théorie, le client envoie son fichier DCO en même temps qu'il s'identifie sur le Proxy.

En pratique, le client se connecte sur le Proxy, et choisit, via la première page, la catégorie dans laquelle il se trouve. Le Proxy utilisera pour la suite, le fichier DCO correspondant, qu'il a déjà en sa possession. Il est bien entendu évident que dans l'optique d'une utilisation future de ce prototype et en fonction de l'évolution des recherches par le W3C, cette façon de faire ne correspondra pas à la réalité (vu que tous les matériels possèderont, théoriquement, un fichier DCO qui leur est propre).

Après l'envoi de sa requête, le client en attend le résultat, qui lui sera renvoyé sous la forme d'une page HTML directement affichable dans son navigateur (Figure 14–**(4)**). Cette page sera personnalisée au niveau du contenu de l'information, mais aussi de son affichage (via le CSS attaché).

3.1.2. Proxy

Comme indiqué dans la Figure 15, notre Proxy est composé de deux parties distinctes : le **noyau** (incluant la « Gestion DCO » et la « Gestion Utilisateurs ») ainsi qu'un ensemble de **modules** permettant de communiquer avec différents web services.

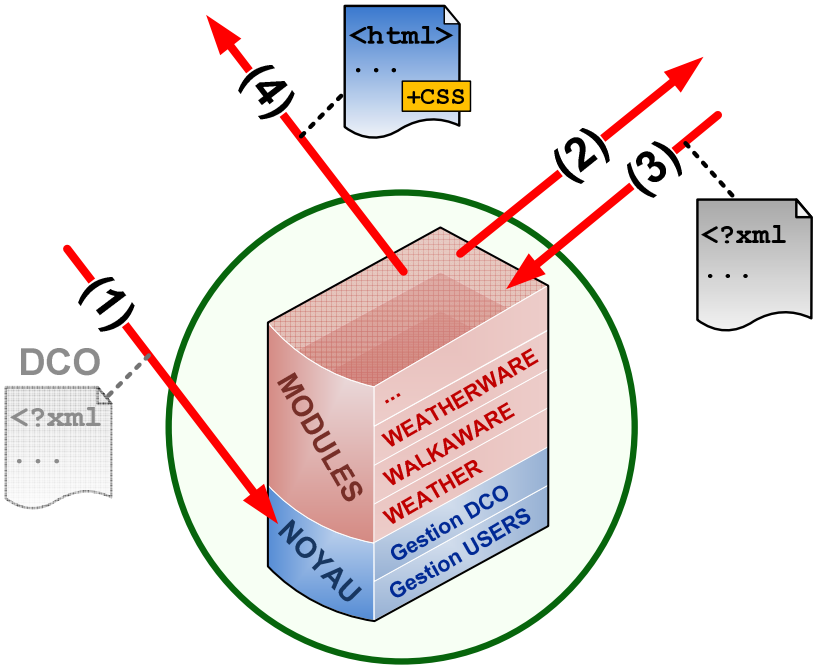

Figure 15 : Zoom sur le Proxy

Le noyau renferme donc la partie « Gestion des utilisateurs », qui s'occupe de tout ce qui est connexion et déconnexion des utilisateurs, ou encore la création d'un nouveau compte s'il n'est pas encore inscrit. Il est ainsi possible d'identifier de manière unique un utilisateur et de faire le lien avec le matériel qu'il utilise. Cet accès est important dans l'optique d'un affichage personnalisé en fonction des préférences de l'utilisateur, étant donné que les modules n'ont pas d'accès direct aux données contenues dans la base de données du noyau.

La partie « Gestion DCO » s'occupe quant à elle de réceptionner le fichier DCO « *envoyé par le client* » (Figure 15 - **(1)**). Elle va, grâce à un ensemble de règles d'adaptation propres au noyau, transformer ce DCO en un fichier XML simplifié (Fichier 2), qui sera ensuite transmis au module sélectionné par le client via le cache du module (le cache correspond à un dossier présent à la racine de chaque module).

```
<?xml version="1.0"?>
<Data>
  <Context> (a)
    <Platform>Smartphone</Platform>
    <DisplayStyle>Text</DisplayStyle>
    <InformationLevel>Small</InformationLevel>
    <Orientation>Column</Orientation>
    <BatteryLevel>Good</BatteryLevel>
    <Background>Bright</Background>
    <Colors>Bright</Colors>
  </Context>
  <Hardware> (b)
    <Display>
      <height>800</height>
      <width>480</width>
    </Display>
    <Battery>
      <batteryBeingCharged>false</batteryBeingCharged>
      <batteryLevel>75</batteryLevel>
    </Battery>
  </Hardware>
</Data>
```

Fichier 2 : Exemple de fichier XML représentant un contexte

L'avantage de ce fichier est d'être beaucoup plus petit et simple que le DCO, puisque ne s'y retrouvent que les informations que nous avons considérées comme intéressantes et pertinentes.

C'est un choix d'implémentation de notre part, justifié par la taille jugée trop importante que peut atteindre le fichier DCO (plus de 500 Ko). C'est également une facilité pour les modules qui pourront utiliser un contexte « simplifié » et ne pas perdre du temps à faire des recherches dans l'entièreté du fichier DCO (vu que ces recherches auront déjà été réalisées par le noyau). Notez l'importance de cette étape, puisque c'est autour de ce contexte que pourront être implémentés les différents modules.
Vous trouverez plus d'informations sur les règles d'adaptation à la section 3.4.

Les modules vont pouvoir être implémentés indépendamment du noyau par d'autres développeurs. Le seul lien entre le noyau et un module est le fichier XML que le module reçoit en entrée. Comme ce fichier XML est nommé selon l'identification unique de l'utilisateur, c'est lui qui va donner la notion de « contexte » définie par le noyau (Fichier 2 - **(a)**) et propre au client qui utilise actuellement le module. Il est intéressant de remarquer, dans ce fichier XML, la présence d'une partie « Hardware » (Fichier 2 - **(b)**) reprenant les spécificités matérielles de la plateforme du client. Nous avons décidé de fournir ces informations également importantes dans l'optique où le développeur du module désire les utiliser pour affiner l'adaptation de son interface graphique voir même, créer un nouveau contexte personnalisé différent de celui proposé par le noyau. Dans l'ordre, le module réalise les étapes suivantes :

1. récupérer dans son cache, le fichier XML mis à disposition par le noyau ;
2. fournir au client la possibilité de choisir les différents paramètres ;
3. créer la requête en fonction des paramètres sélectionnés et l'envoyer vers son web service (Figure 15 - **(2)**) ;
4. récupérer la réponse provenant du web service (Figure 15 - **(3)**) ;
5. générer un affichage adapté en fonction du contexte ;
6. envoyer au client le fichier HTML (associé à son CSS) correspondant à l'affichage final (Figure 15 - **(4)**).

Notez que lors des étapes 2 et 5, le module va appliquer les règles d'adaptation qui lui sont propres (voir section 3.4).

Le développeur du module reste libre de ses choix, mais nous proposons tout de même une marche à suivre-type pour l'implémentation d'un module :

1. se documenter sur le web service ;
2. identifier les ressources disponibles ;
3. construire une requête adéquate ;
4. récupérer les informations reçues via le XML de réponse du web service ;
5. préparer l'affichage adéquat en fonction des objectifs fixés par le module.

3.1.3. Requête WSDL

Il est utile de préciser que seul le web service NDFD possède une version WSDL. La suite de cette section sera donc uniquement consacrée à ce dernier.

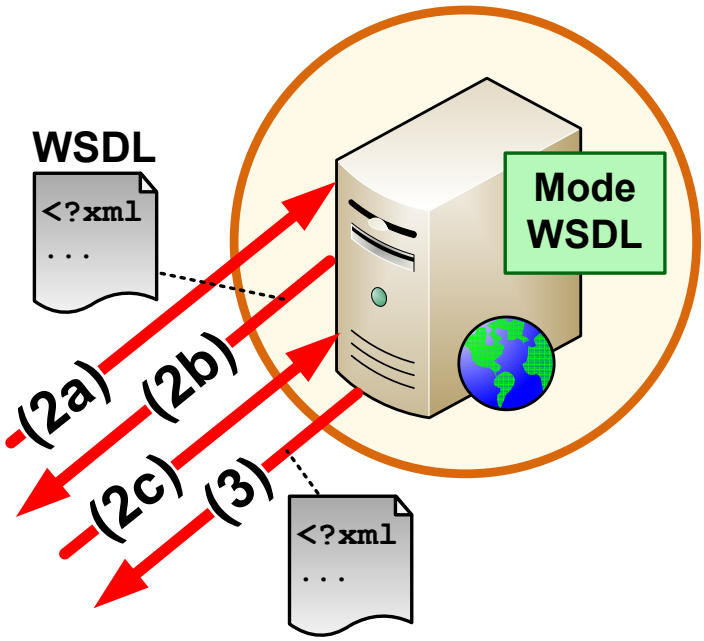

Figure 16 : Zoom sur le mode WSDL

Lorsque l'on veut réaliser une requête WSDL, il est dans un premier temps, nécessaire de créer un nouveau client SOAP (voir Code 1) qui sera chargé de récupérer le fichier WSDL à l'adresse indiquée dans la documentation fournie par le web service (Figure 16 – **(2a)** et **(2b)**).

```
$soapclient = new SoapClient
('http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl');
```

Code 1 : Création du client SOAP et récupération du fichier WSDL

Une fois ce fichier récupéré, nous allons pouvoir construire notre requête en adéquation avec une fonction proposée par le web service (Code 2).

```
$result = $soapclient->__soapCall('NDFDgen', $parameters);
```

Code 2 : Requête WSDL au web service NDFD

Pour ce faire, nous allons donc envoyer au web service, via le client SOAP, la fonction choisie ainsi qu'un tableau contenant l'ensemble des paramètres nécessaires à son exécution (Figure 16 - **(2c)**).
La réponse envoyée par celui-ci est contenue dans un fichier XML (Figure 16 – **(3)**). Il nous faudra donc ensuite utiliser un parseur pour obtenir les informations utiles à la suite de l'exécution.

3.1.4. Requête REST

Avant de réaliser notre requête REST, il sera nécessaire d'aller chercher, manuellement, la documentation utile à la création de cette dernière. En effet, il n'existe pas de mécanisme pré-requête, comme pour WSDL, permettant de récupérer, via un fichier XML, toutes les informations nous permettant de la réaliser correctement.

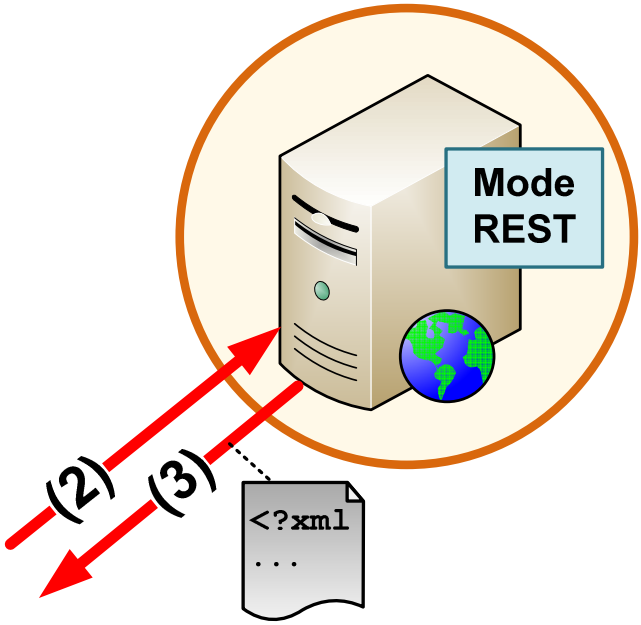

Figure 17 : Zoom sur le mode REST

D'un autre coté, lorsque l'on désire réaliser notre requête (Figure 17 – **(2)**), il n'est plus nécessaire de créer un client SOAP.

```
$req_URL =
"http://www.weather.gov/forecasts/xml/sample_products/browser_interface/ndfdXMLclient.php?lat=
".$coord[0]."&lon=".$coord[1]."&product=time-series&begin=".$start_time->format('Y-m-d
H:i:s')."&end=".$end_time->format('Y-m-d H:i:s');
foreach($tab_param as $param => $value) {
       if($value){
         $req_URL = $req_URL."&".$param."=".$param;
       }
}
return file_get_contents($req_URL);
```

Code 3 : Requête REST au web service NDFD

```
$req_URL =
"http://www.walkaware.com/geo/Search?login=ftpn&idl=4&lat=".$coord[0]."&lng=".$coord[1]."&dist
ance=".$radius."&poi=true&pc=";
foreach($tab_param as $param) {
       $req_URL = $req_URL.$param.";";
}
return file_get_contents($req_URL);
```

Code 4 : Requête REST au web service WalkAware

Remarquons ici que contrairement à la version WSDL, nous accédons directement aux ressources et donc les paramètres ne dépendent plus d'une fonction définie par le web service mais uniquement des ressources auxquelles nous souhaitons accéder.
Tout comme pour la version WSDL, la réponse est contenue dans un fichier XML (Figure 17 – **(3)**) qu'il faudra donner au parseur afin d'en extraire les différentes informations.

3.1.5. Comparatif des deux modes

Le premier avantage va pour REST, pour qui il n'est plus nécessaire d'utiliser le protocole SOAP. Notez que l'on pourrait imaginer une solution consistant à encapsuler une requête REST dans un message SOAP, mais cela n'apporterait qu'une complexité supplémentaire inutile. Nous avons ainsi économisé :

- **du temps**, parce qu'il ne faut plus demander le fichier WSDL au web service avant de réaliser la requête, elle peut être formulée directement ;
- **de la bande passante**, vu qu'on économise le coût de transfert du fichier WSDL, dont la taille peut varier selon les fonctions proposées par le web service.

Le second avantage qui ressort est également en faveur de REST : nous pouvons directement choisir les ressources qui nous intéressent, contrairement à WSDL, où nous restons dépendant des fonctions disponibles ainsi que des paramètres à fournir. De par ces choix opérés par le web service, il se peut que la requête imposée par la version WSDL contienne des informations qui ne nous sont pas utiles et/ou que nous ayons besoin de réaliser plusieurs requêtes à des fonctions différentes, afin de récupérer l'entièreté des informations nécessaires au fonctionnement de l'application, mais aussi d'autres informations inutiles prises au passage. Dans la version REST, réaliser plusieurs requêtes est également possible (par exemple, si l'on a besoin du résultat d'une requête afin de pouvoir accéder à d'autres ressources). Mais étant donné que nous sommes maître du choix des ressources, nous ne demandons que les informations qui nous sont utiles pour la suite. Nous économisons ainsi également de la bande passante.

Notre solution devant être utilisée également par des Smartphones et des tablettes pouvant se connecter via une connexion EDGE/3G, nous avons donc privilégié la solution REST pour les gains en bande passante.

3.1.6. Web services

La particularité de notre solution est que pour chaque nouveau web service que l'on veut consulter, il est nécessaire d'implémenter un nouveau module sur le Proxy car un module est dépendant de la sémantique propre au web service avec lequel il interagit. Dans la forme actuelle, le Proxy n'utilise que deux web services bien distincts.

NDFD [NAT 2011] (**N**ational **D**igital **F**orecast **D**atabase) est le nom du web service météo lié au module « Weather » de la Figure 15.
Ce service va rechercher, via la base de données du National Weather Service des États-Unis, les prévisions météorologiques d'un endroit donné.
Nous avons sélectionné ce web service car il répondait précisément à nos critères de recherche :

- il fournit des informations météorologiques variées qui nous permettront de travailler avec plusieurs paramètres ;
- il fonctionne en mode REST et WSDL. Ce critère est important dans la mesure où nous devions choisir le meilleur mode pour notre application ;
- il fourni une documentation publique et détaillée ;
- il est gratuit.

C'est avec ce web service que nous avons débuté l'implémentation du noyau et du module « Weather » de notre Proxy.

WalkAware est une application web qui permet de créer des circuits touristiques, sur lesquels il est possible d'ajouter des points d'intérêts multimédia (photos, vidéos, enregistrements audio), et de les télécharger sur son GPS ou son Smartphone [WAL 2011].

Ce web service est lié au module « WalkAware » de la Figure 15. Ce web service nous a permis de découvrir l'utilisation de Google maps, via l'affichage des différents points d'intérêts sélectionnés ainsi que la création de nos propres circuits générés à partir de ceux-ci.
Enfin, vous aurez remarqué sur la Figure 15 qu'un troisième module s'y retrouve. Nous l'avons baptisé « Weathaware », mot-valise provenant de la contraction de « Weather » et « WalkAware », car il représente la combinaison de ces deux modules. C'est en effet une possibilité de notre système, d'implémenter un module qui utilise plusieurs web services.

3.2. Langage de programmation

«Il n'y a que deux sortes de langages de programmation : ceux dont les gens disent toujours du mal et ceux que personne n'utilise.» [STR 2011]

Bjarne Stroustrup Informaticien, écrivain et professeur en sciences informatiques ; auteur du C++

Un des points initiaux abordés dès nos débuts dans ce projet, était le langage de programmation à utiliser pour implémenter le prototype. La première question qui nous est venue à l'esprit était le choix entre une programmation « côté client » ou « côté serveur ».

3.2.1. Côté client

Avantages

- Alléger le serveur, dans l'optique où une multitude d'utilisateurs se connectent en même temps.
- Prise en compte, en temps réel, des modifications du contexte.

Inconvénients

- Au niveau des Smartphones et des tablettes, la surcharge de travail demandé à leur faible processeur pourrait diminuer leur autonomie.
- Le bon déroulement du programme dépend entièrement du navigateur du client.
- Exemple avec JavaScript : un inconvénient majeur est que certains utilisateurs ne l'activent pas dans leur navigateur.

3.2.2. Côté serveur

Avantages

- Alléger le client, vu que c'est le serveur qui travaille.
- Plus aucune trace du code serveur dans le résultat html, ce qui implique un gain en sécurisation du code.
- Plusieurs possibilités d'interactions avec d'autres systèmes ou applications, comme les bases de données par exemple.

Inconvénients

- Surcharge si le nombre de clients est important.
- Besoin de recharger la page pour prendre en compte de nouveau paramètres.

Devant travailler avec des Smartphones et des tablettes, notre choix s'est porté sur le « côté serveur » pour les avantages qu'il procure sur ces matériels.

3.2.3. Comparaison des langages disponibles

De par ce premier critère de sélection, les candidats que nous avons retenus sont :

- Ruby - http://www.ruby-lang.org/fr/
- Ruby on Rails - http://rubyonrails.org/
- PHP - http://www.php.net/
- Python - http://www.python.org/
- ASP.net - http://www.asp.net/

Dans le Tableau 2 ci-dessous, se retrouvent confrontés ces langages de programmation que nous avons comparés sur base de plusieurs critères.

	Ruby	RAILS	python	php	Microsoft ASP.net
Utilisation par le web service sélectionné	-	-	-	+	-
Facilité de développement	-	-	+	+	-
Temps de développement estimé	-	-	+	+	-
Intégration native de SOAP	+	+	+	+	+
Affinité avec le XML	+	+	+	+	+
Gratuit	+	+	+	+	+
« À la mode »	-	+	+	+	-

Tableau 2 : Comparaison des langages de programmation

Il est clair que la majorité de ces critères peuvent paraître subjectifs car ils dépendent de l'appréciation de chacun. Néanmoins, ils illustrent parfaitement les interrogations auxquelles nous avons dû faire face lors du choix du langage.

Nous pouvons constater que PHP et Python sont les deux langages qui se démarquent des autres. Ceci provient certainement du fait que nous avons dû les utiliser lors de projets antérieurs. Le principal critère qui a fait pencher la balance en faveur du PHP provient du web service NDFD qui propose des exemples d'utilisation de ce service écrits en PHP.

3.3. Analyse de l'architecture

Lors de la réalisation du prototype, le choix du type d'architecture s'est immédiatement posé. Le développement de ce prototype n'étant pas l'apanage de l'un ou l'autre, il nous fallait trouver une architecture qui nous permettait de travailler efficacement et séparément sur les différentes parties du projet.

Notre choix s'est porté vers un pattern de type **M**odèle-**V**ue-**C**ontrôleur :

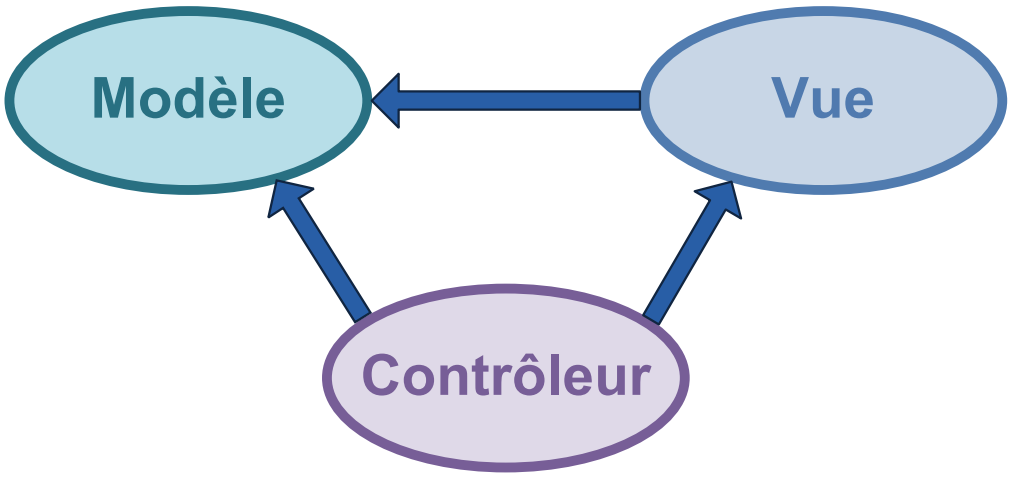

Figure 18 : Pattern Modèle-Vue-Contrôleur

Ce pattern requière une découpe de l'application en trois entités distinctes [PHP 2009].

Le **modèle** est l'entité qui assure la gestion des données dans la base de données (accès, modification, création, suppression). Elle est sollicitée par la vue qui se contente simplement d'afficher, au besoin, les différentes données et le contrôleur.
Ainsi, dans notre solution, le modèle du noyau contient la base de données SQL nous permettant d'enregistrer les différents clients.
Au niveau du modèle des modules, il contient les différentes classes que nous utiliserons et la fonction qui va récupérer les informations utiles dans le cache. Le cache contenant le fichier XML envoyé par le noyau.

La **vue** est l'entité qui comprend toute l'interface visible pour le client. Elle interagit principalement avec le modèle. Attention, si la vue affiche les données, elle n'effectue aucun traitement de ces dernières. Dans le noyau, la vue contient les pages HTML utilisées pour la connexion des clients ainsi que pour le choix du module. Dans les modules, la vue contient les pages HTML utilisées pour la configuration des paramètres ainsi que les fichiers CSS nécessaires pour l'affichage final.

Le **contrôleur** est l'entité qui va réagir à une interaction du client et faire le lien entre la vue et le modèle. Il va interpréter ces interactions et déclencher les actions correspondantes : vérification, modifications des données, changement de la vue, etc.
Le contrôleur du noyau va participer à l'exécution des actions concernant :

- la connexion et la déconnexion des clients ;
- la récupération et l'analyse du fichier DCO ;
- la création du fichier XML résumant le fichier DCO dans le cache du module ;
- le choix du module.

Le contrôleur des modules va quant à lui :

- analyser le fichier XML contenu dans le cache et générer les classes correspondantes ;
- réaliser les requêtes vers le web service correspondant en fonction de la configuration choisie par le client ;
- générer le CSS qui va être utilisé pour l'affichage de la page finale.

Grâce à la clarté de l'architecture imposée par ce modèle, chaque entité étant bien définie, la répartition des différentes parties à développer nous a été facilitée. Notre volonté à développer ensemble nous imposait également une certaine rigueur dans les spécifications des différentes fonctions pour ne pas devoir modifier celles déjà existantes afin qu'elles s'adaptent à une nouvelle. L'utilisation de ce pattern nous a également facilité la tâche dans le sens où les modifications et les corrections apportées à une entité n'affectent pas les autres tant que les spécifications sont respectées.

La principale difficulté résultant de l'utilisation de ce pattern fut la mise en place d'un système d'index afin de faire interagir correctement chaque entité avec les autres.
Nous pouvons remarquer qu'avec la double casquette de développeur du Proxy et de développeur des modules, l'architecture choisie est identique pour tous. Mais dans le cadre d'une application ouverte à d'autres développeurs de module, chacun d'entre eux peut choisir l'architecture qu'il souhaite et il se pourrait que seul le noyau ait une architecture de type MVC. C'est également pour cette raison que nous avons mis le cache à part dans les modules.

3.4. Règles d’adaptation

Cette section va énoncer les différentes règles d'adaptation auxquelles nous avons réfléchi dans l’optique de créer un contexte simplifié utilisable facilement par les modules. Cette liste de règles est non-exhaustive car nous les avons choisi avant tout parce qu’elles ont été implémenté à partir des éléments se trouvant dans le fichier DCO. Ainsi, par exemple, pour distinguer les trois catégories de matériel utilisé (voir section 3.1.1), nous nous sommes basés sur la hauteur et la largeur de l’écran dont les valeurs sont disponibles dans le fichier DCO. Ces règles ont également été sélectionnées pour leur simplicité, qui doit permettre aux développeurs de module de les assimiler rapidement et ainsi de les inciter à s’approprier le concept de contexte en créant eux-mêmes leurs propres règles d’adaptation. En agissant de la sorte, nous continuions dans notre objectif premier qui est l’adaptation du contexte d’utilisation et espérions encourager les développeurs de module à poursuivre dans la voie que nous avons tracée.

Afin de préserver l'indépendance entre le développeur du Proxy et le développeur de module, nous avons joué le jeu de chacun d'entre eux et c'est pourquoi des règles ont été définies pour le noyau ainsi que pour chaque module.

Nous avons également défini des tables de décisions qui nous ont aidés au niveau de l'implémentation proprement dite lorsque plusieurs règles étaient combinées.

3.4.1. Les règles du noyau

Ces règles d'adaptation ainsi que la table de décision qui en découle, ont pour principale fonction, la création d'un contexte simplifié, modélisé par un fichier XML, qui sera envoyé aux modules. Ces règles sont classées par ordre décroissant d'importance.

- Les préférences de l'utilisateur sont toujours prioritaires par rapport aux paramètres prédéfinis automatiquement.
- L'utilisateur peut choisir entre trois niveaux de détails des informations à recevoir :
  - Bas
  - Intermédiaire
  - Élevé
- Un appareil dont la résolution est comprise entre 480x854 et inférieur est considéré comme un Smartphone.
- Un appareil dont la résolution est comprise entre 480x854 et 1280x800 est considéré comme une tablette/Pocket PC.
- Un appareil dont la résolution est supérieure à 1280x800 est considéré comme un PC.
- Si la hauteur est plus grande ou égal à la largeur, on privilégie l'affichage en colonne.
- Si la largeur est plus grande que la hauteur, on privilégie l'affichage en ligne.
- Le style d'affichage varie en fonction de la plateforme sélectionnée :
  - Textuel sur un Smartphone
  - Graphique sur une tablette/Pocket PC
  - Graphique sur un PC
- L'affichage varie en fonction de l'orientation de l'appareil :
  - En colonne si l'orientation est verticale
  - En ligne si l'orientation est horizontale

- La quantité d'information reçue varie en fonction de la plateforme sélectionnée :
  - Peu d'information sur un Smartphone
  - Peu d'information sur une tablette/Pocket PC
  - Beaucoup d'informations sur un PC
- L'état de la batterie varie selon le niveau de charge de celle-ci :
  - Si le niveau est compris entre 100% et 20%, l'état est égal à Good
  - Si le niveau est inférieur à 20%, l'état est égal à Low
- Les échanges appareil/Proxy varie en fonction de l'état de la batterie :
  - Minimisation des échanges appareil/Proxy si le niveau de la batterie est Low
  - Maximisation des échanges appareil/Proxy si le niveau de la batterie est Good
- L'état de l'intensité du signal réseau varie selon le niveau de celui-ci :
  - Si le niveau est compris entre 100% et 20%, le signal est égal à Good
  - Si le niveau est inférieur à 20%, le signal est égal à Low
- Les échanges appareil/proxy varie en fonction de l'intensité du signal réseau :
  - Minimisation des échanges appareil/proxy si le signal est Good
  - Maximisation des échanges appareil/proxy si le signal est fort
- Le style d'affichage varie en fonction de l'état de la batterie :
  - Fond et couleurs sombres si le niveau de la batterie est Low
  - Fond et couleurs vives si le niveau de la batterie est Good
- Le contraste d'affichage varie en fonction du jour et de la nuit :
  - Fond clair et couleurs sombres pour le jour
  - Fond sombre et couleurs claires pour la nuit

A partir de ces règles, nous avons défini la table de décision présentée dans le Tableau 3.

Conditions																											
Résolution <= 480x854	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F
480x854 < Résolution < 1280x800	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F
Résolution >= 1280x800	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T
Hauteur <= Largeur	T	T	T	F	F	F	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F	T	T	T
Niveau État Batterie < 20%	T	T	T	T	T	T	F	F	F	T	T	T	T	T	T	F	F	F	T	T	T	T	T	T	T	T	T
Niveau Signal Réseau < 20%	T	T	T	T	T	T	-	-	-	F	F	F	T	T	T	-	-	-	F	F	F	T	T	T	F	F	F
Période = Jour	T	T	T	T	T	T	-	-	-	T	T	T	F	F	F	-	-	-	T	T	T	F	F	F	F	F	F
Actions																											
Plateforme = Smartphone	X			X			X			X			X			X			X			X			X		
Plateforme = tablette/Pocket PC		X			X			X			X			X			X			X			X			X	
Plateforme = PC			X			X			X			X			X			X			X			X			X
Affichage = Texte	X			X			X			X			X			X			X			X			X		
Affichage = Graphique		X	X		X	X		X	X		X	X		X	X		X	X		X	X		X	X		X	X
Orientation = Ligne	X	X	X				X	X	X	X	X	X	X	X	X										X	X	X
Orientation = Colonne				X	X	X										X	X	X	X	X	X	X	X	X			
Quantité d'information = Peu							X	X	X							X	X	X									
Quantité d'information = Beaucoup	X	X	X	X	X	X				X	X	X	X	X	X				X	X	X	X	X	X	X	X	X
Échange appareil/Proxy = minimal							X	X	X	X	X	X				X	X	X	X	X	X				X	X	X
Échange appareil/Proxy = maximal	X	X	X	X	X	X							X	X	X							X	X	X			
Fond = sombre							X	X	X				X	X	X	X	X	X				X	X	X	X	X	X
Fond = clair	X	X	X	X	X	X				X	X	X							X	X	X						
Couleurs = sombre	X	X	X	X	X	X	X	X	X	X	X	X				X	X	X	X	X	X						
Couleurs = clair													X	X	X							X	X	X	X	X	X

Tableau 3 : Table de décision du noyau

Le Code 5 ci-dessous présente le résultat obtenu au niveau de l'implémentation grâce aux règles d'adaptations et à la table de décision du noyau.

```
if($width <= 480 AND $height <= 854) {
   $new_platform = $dom->createElement('Platform', 'Smartphone');
   $new_display_style = $dom->createElement('DisplayStyle', 'Text');
   $new_information = $dom->createElement('InformationLevel', 'Small');
   }
elseif($width <= 1280 AND $height <= 800) {
   $new_platform = $dom->createElement('Platform', 'Tablet');
   $new_display_style = $dom->createElement('DisplayStyle', 'Graphic');
   $new_information = $dom->createElement('InformationLevel', 'Small');
   }
else {
   $new_platform = $dom->createElement('Platform', 'PC');
   $new_display_style = $dom->createElement('DisplayStyle', 'Graphic');
   $new_information = $dom->createElement('InformationLevel', 'Big');
   $new_battery = $dom->createElement('Battery', 'Good');
   }
```

```
if($height >= $width) {
   $new_orientation = $dom->createElement('Orientation', 'Column');
   }
else {
   $new_orientation = $dom->createElement('Orientation', 'Line');
   }
if($batteryBeingCharged == TRUE OR $batteryLevel >= 20) {
   $new_battery = $dom->createElement('BatteryLevel', 'Good');
   $new_background = $dom->createElement('Background', 'Bright');
   $new_colors = $dom->createElement('Colors', 'Bright');
   }
else {
   $new_battery = $dom->createElement('BatteryLevel', 'Low');
   $new_background = $dom->createElement('Background', 'Dark');
   $new_colors = $dom->createElement('Colors', 'Dark');
   }
```

Code 5 : Utilisation des règles d'adaptation dans le noyau

3.4.2. Les règles du module « Weather »

Les règles d'adaptation de ce module sont également classées par ordre décroissant d'importance. En effet, nous avons décidé de privilégier un affichage adapté au contexte plutôt qu'un contenu adapté au contexte d'utilisation.

- Le nombre de jours de prévisions varie en fonction de la plateforme sélectionnée :
  - Nombre de jours de prévisions = 2 sur un Smartphone
  - Nombre de jours de prévisions = 4 sur une tablette/Pocket PC
  - Nombre de jours de prévisions = 6 sur un PC
- Le nombre de paramètres sélectionnés varie en fonction de la plateforme sélectionnée :
  - Nombre de paramètres sélectionnés = 3 sur un Smartphone
  - Nombre de paramètres sélectionnés = 5 sur une tablette/Pocket PC
  - Nombre de paramètres sélectionnés = 9 sur un PC
- La taille du texte affiché varie en fonction de la plateforme sélectionnée :
  - Grande sur un Smartphone
  - Grande sur une tablette/Pocket PC
  - Petite sur un PC

- La taille des images affichées varie en fonction de la plateforme sélectionnée :
  - Petite sur un Smartphone
  - Moyenne sur une tablette/Pocket PC
  - Grande sur un PC
- Le choix des paramètres dépend de la localisation de l'utilisateur :
  - Vent et hauteur des vagues si on se situe au littoral
  - Vent et quantité de précipitation si on se situe en plaine
  - Quantité de neige tombée si on se situe en région montagneuse
- Les unités de mesures dépendent de la localisation de l'utilisateur :
  - Système impérial pour les USA, Libéria, Birmanie
  - Système métrique pour le reste du monde

A partir de ces règles, nous avons défini la table de décision présentée dans le Tableau 4.

Conditions																		
Plateforme = Smartphone	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F
Plateforme = tablette/Pocket PC	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F
Plateforme = PC	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T	F	F	T
Localisation = Littoral	T	T	T	F	F	F	F	F	F	T	T	T	F	F	F	F	F	F
Localisation = Plaine	F	F	F	T	T	T	F	F	F	F	F	F	T	T	T	F	F	F
Localisation = Montagnes	F	F	F	F	F	F	T	T	T	F	F	F	F	F	F	T	T	T
Localisation = USA/Libéria/Birmanie	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F
Actions																		
Nombre de jours de prévisions = 2	X			X			X			X			X			X		
Nombre de jours de prévisions = 4		X			X			X			X			X			X	
Nombre de jours de prévisions = 6			X			X			X			X			X			X
Nombre de paramètres sélectionnés = 3	X			X			X			X			X			X		
Nombre de paramètres sélectionnés = 5		X			X			X			X			X			X	
Nombre de paramètres sélectionnés = 9			X			X			X			X			X			X
Taille du texte = Grande	X	X		X	X		X	X		X	X		X	X		X	X	
Taille du texte = Petite			X			X			X			X			X			X
Taille des icones = Petite	X			X			X			X			X			X		
Taille des icones = Moyenne		X			X			X			X			X			X	
Taille des icones = Grande			X			X			X			X			X			X
Paramètres + Vent	X	X	X	X	X	X				X	X	X	X	X	X			
Paramètres + Hauteurs des vagues	X	X	X							X	X	X						
Paramètres + Qt. de précipitation				X	X	X							X	X	X			
Paramètres + Qt. de neige tombée							X	X	X							X	X	X
Système impérial	X	X	X	X	X	X	X	X	X									
Système métrique										X	X	X	X	X	X	X	X	X

Tableau 4 : Table de décision du module « Weather »

Afin d'appliquer ces règles dans le module « Weather », nous avons implémenté une fonction « CSS Generator » qui va créer un fichier CSS personnalisé en fonction du contexte d'utilisation grâce à la fonction définie dans le Code 6 épuré ci-dessous.

```
function css_content() {
  $bckgrcolor = "#F0F8FF";
  $textcolor = "#000000";
  $heading1color = "#6495ED";
  $heading3color = "#66CCFF";
  $widthform = "100%";
  $formbgcolor = "#CCCCFF";
  $platform = unserialize($_SESSION['Context'])->getPlatform();
  $background = unserialize($_SESSION['Context'])->getBackground();
switch ($platform) {
  case "PC" :
    $iconsize = "110";
    $heading1size = "18";
    $heading3size = "14";
    $widthform = "60%";
    $paddingfield = "20px";
    break;
  case "Tablet" :
    if ($background == "Dark")
    {
      $bckgrcolor = "#000000";
      $textcolor = "#FFFFFF";
      $heading1color = "#FFFFFF";
      $heading3color = "#FFFFFF";
      $formbgcolor = "#000000";
    }
    $iconsize = "90";
    $heading1size = "20";
    $heading3size = "13";
    $widthform = "95%";
    $paddingfield = "8px";
    break;
  case "Smartphone" :
    if ($background == "Dark") {
      $bckgrcolor = "#000000";
      $textcolor = "#FFFFFF";
      $heading1color = "#FFFFFF";
      $heading3color = "#FFFFFF";
      $formbgcolor = "#000000";
    }
    $iconsize = "150";
    $heading1size = "14";
    $heading3size = "12";
    $widthform = "90%";
    $paddingfield = "5px";
    break;
}
```

```
$CSSContent = "
body {
background-color: ".$bckgrcolor.";
font-family: Verdana ;
color : ".$textcolor.";
}
div#heading h1{
text-align : center;
font-size : ".$heading1size."pt;
color: ".$heading1color.";
".$marginh1."
}
div#heading h3{
text-align : center;
font-size : ".$heading3size."pt;
color: ".$heading3color.";
".$marginh3."
}
.icon {
width : ".$iconsize."px;
height : ".$iconsize."px;
display: block;
margin: 0 auto;
}
form {
background-color:".$formbgcolor.";
width:".$widthform.";
font-family : calibri;
border:1px solid ".$textcolor.";
margin : auto;
padding : 5px;
}";
return $CSSContent;
}
```

Code 6 : Fonction principale du « CSS Generator » : « CSS Content »

La fonction « CSS Content » renvoie donc le contenu de la feuille de style personnalisée en fonction du contexte d'utilisation. Le fichier CSS est nommé de façon unique via l'identifiant de l'utilisateur. Ainsi, le module sait à qui il doit appliquer cette feuille de style, étant donné qu'elle lui sera entièrement dédiée et personnalisée.

3.4.3. Les règles du module « WalkAware » :

Pour les raisons énoncées dans la section 3.4.2 ci-dessus, les règles d'adaptation de ce module sont également classées par ordre décroissant d'importance.

- La taille du texte affiché varie en fonction de la plateforme sélectionnée :
  - Grande sur un Smartphone
  - Grande sur une tablette/Pocket PC
  - Petite sur un PC
- La taille des images affichées varie en fonction de la plateforme sélectionnée :
  - Petite sur un Smartphone
  - Moyenne sur une tablette/Pocket PC
  - Grande sur un PC
- La distance de recherche des Points d'intérêts varie selon la plateforme sélectionnée :
  - 20 km sur un Smartphone
  - 50 km sur une tablette/Pocket PC
  - 100 km sur un PC
- L'affichage de la Google maps varie selon la plateforme sélectionnée :
  - Petite sur un Smartphone
  - Moyenne sur une tablette/Pocket PC
  - Grande sur un PC
- Les unités de mesures dépendent de la localisation de l'utilisateur :
  - Système impérial pour les USA, Libéria, Birmanie
  - Système métrique pour le reste du monde

A partir de ces règles, nous avons défini la table de décision présentée dans le Tableau 5.

Conditions						
Plateforme = Smartphone	T	F	F	T	F	F
Plateforme = tablette/Pocket PC	F	T	F	F	T	F
Plateforme = PC	F	F	T	F	F	T
Localisation = USA/Libéria/Birmanie	T	T	T	F	F	F
Actions						
Taille du texte = Grande	X	X		X	X	
Taille du texte = Petite			X			X
Taille des icones = Petite	X			X		
Taille des icones = Moyenne		X			X	
Taille des icones = Grande			X			X
Taille Google maps = Petite	X			X		
Taille Google maps = Moyenne		X			X	
Taille Google maps = Grande			X			X
Distance de recherche = 20 km	X			X		
Distance de recherche = 50 km		X			X	
Distance de recherche = 100 km			X			X
Système impérial	X	X	X			
Système métrique				X	X	X

Tableau 5 : Table de décision du module « WalkAware »

3.4.4. Les règles du module « Weathaware » :

Ce module étant la combinaison des deux présentés aux sections 3.4.2 et 3.4.3 ci-dessus, les règles d'adaptation de ce module sont classées par ordre décroissant d'importance.

- Le nombre de paramètres sélectionnés, pour « Weather », varie en fonction de la plateforme sélectionnée :
  - Nombre de paramètres sélectionnés = 1 sur un Smartphone
  - Nombre de paramètres sélectionnés = 2 sur une tablette/Pocket PC
  - Nombre de paramètres sélectionnés = 4 sur un PC
- La taille du texte affiché varie en fonction de la plateforme sélectionnée :
  - Grande sur un Smartphone
  - Grande sur une tablette/Pocket PC
  - Petite sur un PC

- La taille des images affichées varie en fonction de la plateforme sélectionnée :
  - Petite sur un Smartphone
  - Moyenne sur une tablette/Pocket PC
  - Grande sur un PC
- L'affichage de la Google maps varie selon la plateforme sélectionnée :
  - Petite sur un Smartphone
  - Moyenne sur une tablette/Pocket PC
  - Grande sur un PC
- Les unités de mesures dépendent de la localisation de l'utilisateur :
  - Système impérial pour les USA, Libéria, Birmanie
  - Système métrique pour le reste du monde

A partir de ces règles, nous avons défini la table de décision présentée dans le Tableau 6.

Conditions						
Plateforme = Smartphone	T	F	F	T	F	F
Plateforme = tablette/Pocket PC	F	T	F	F	T	F
Plateforme = PC	F	F	T	F	F	T
Localisation = USA/Libéria/Birmanie	T	T	T	F	F	F
Actions						
Nombre de paramètres « Weather » sélectionnés = 1	X			X		
Nombre de paramètres « Weather » sélectionnés = 2		X			X	
Nombre de paramètres « Weather » sélectionnés = 4			X			X
Taille du texte = Grande	X	X		X	X	
Taille du texte = Petite			X			X
Taille des icones = Petite	X			X		
Taille des icones = Moyenne		X			X	
Taille des icones = Grande			X			X
Taille Google maps = Petite	X			X		
Taille Google maps = Moyenne		X			X	
Taille Google maps = Grande			X			X
Système impérial	X	X	X			
Système métrique				X	X	X

Tableau 6 : Table de décision du module «Weathaware»

3.5. Diagrammes

À la lumière des sections précédentes, voici différents scénarios exposant l'ensemble des étapes importantes (définies par le diagramme d'activité) exécutées lors de l'utilisation du Proxy par le client, et ce, pour chacun des trois modules implémentés.

3.5.1. Diagramme d'activité

Nous allons modéliser l'entièreté du processus en commençant par l'authentification et en terminant par la remise au client du fichier HTML correspondant à l'affichage des résultats (Fin).

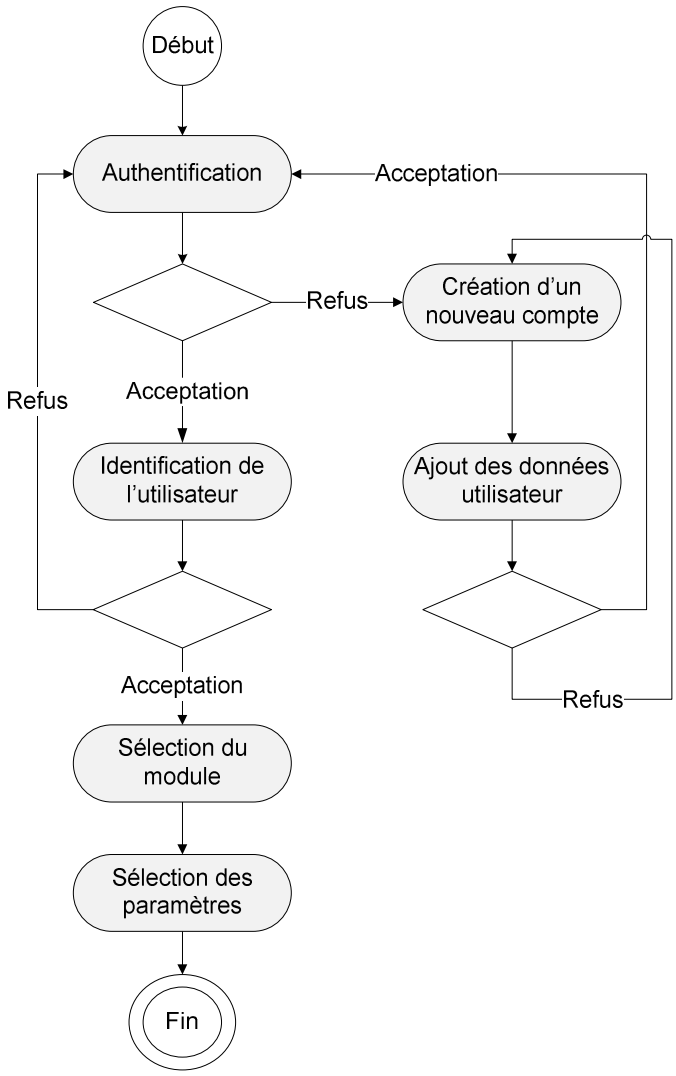

Figure 19 : Diagramme d'activité du Proxy

Le cheminement est assez linéaire du fait que peu de choix sont présentés au client et que l'essentiel des choix proposés correspondent à des paramètres, regroupés à l'intérieur de la dernière tâche, qu'il peut faire varier ou non selon ses préférences.

3.5.2. Diagrammes de séquence

Nous pouvons constater dans les Figure 20, Figure 21 et Figure 22 suivantes, le rôle restreint mais néanmoins indispensable du noyau. Il va en effet recevoir le fichier DCO en provenance de l'utilisateur, puis le traiter en fonction des règles d'adaptation énoncées dans la section précédente, afin de fournir aux différents modules un fichier XML représentant le contexte d'utilisation.

Le module « Weather », représenté dans la Figure 20 ci-dessous, est le module qui à un fonctionnement standard dans le sens où il traite le fichier XML en provenance du noyau, effectue sa requête au web service, traite et affiche la réponse.

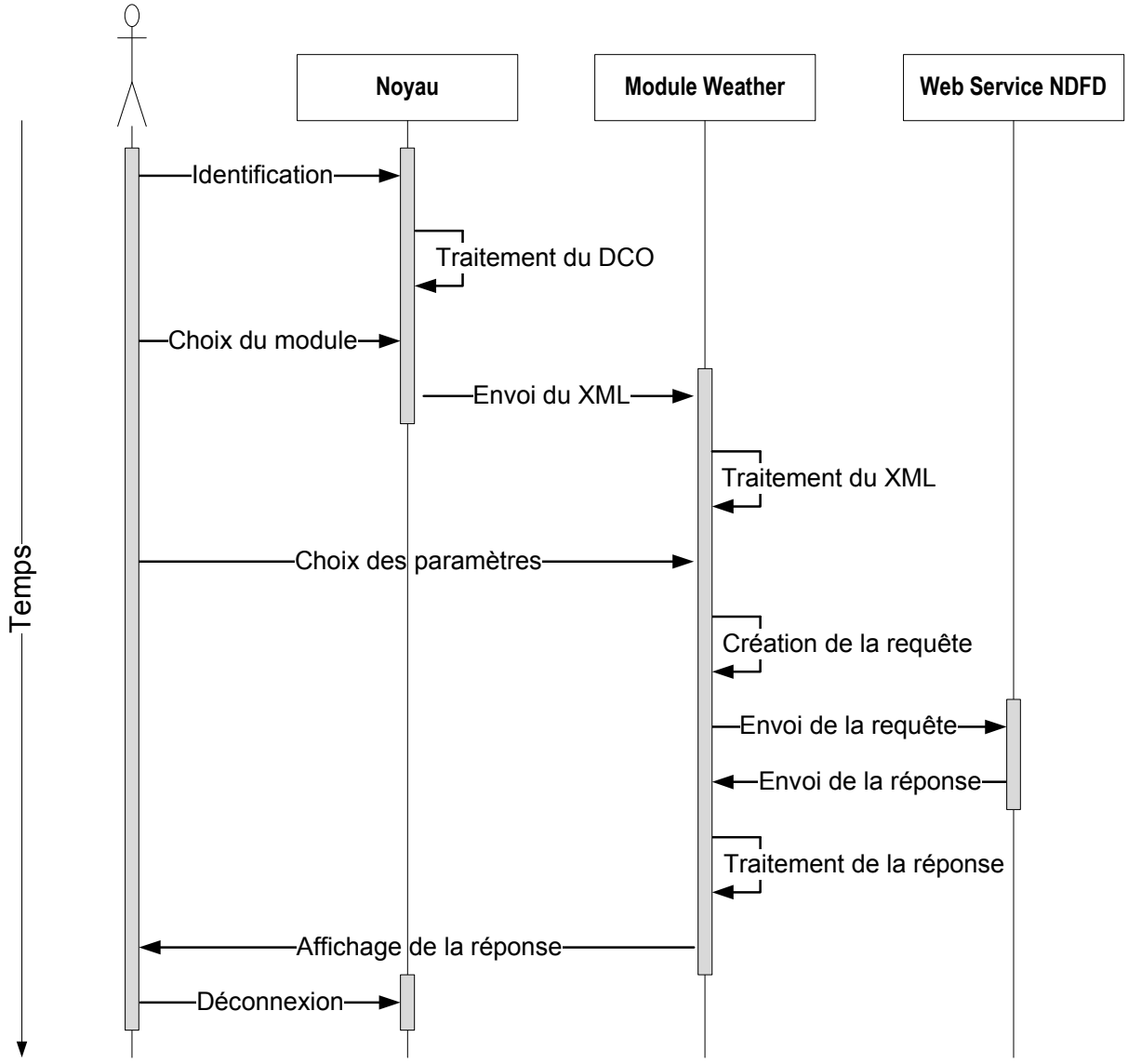

Figure 20 : Diagramme de séquence pour le module « Weather »

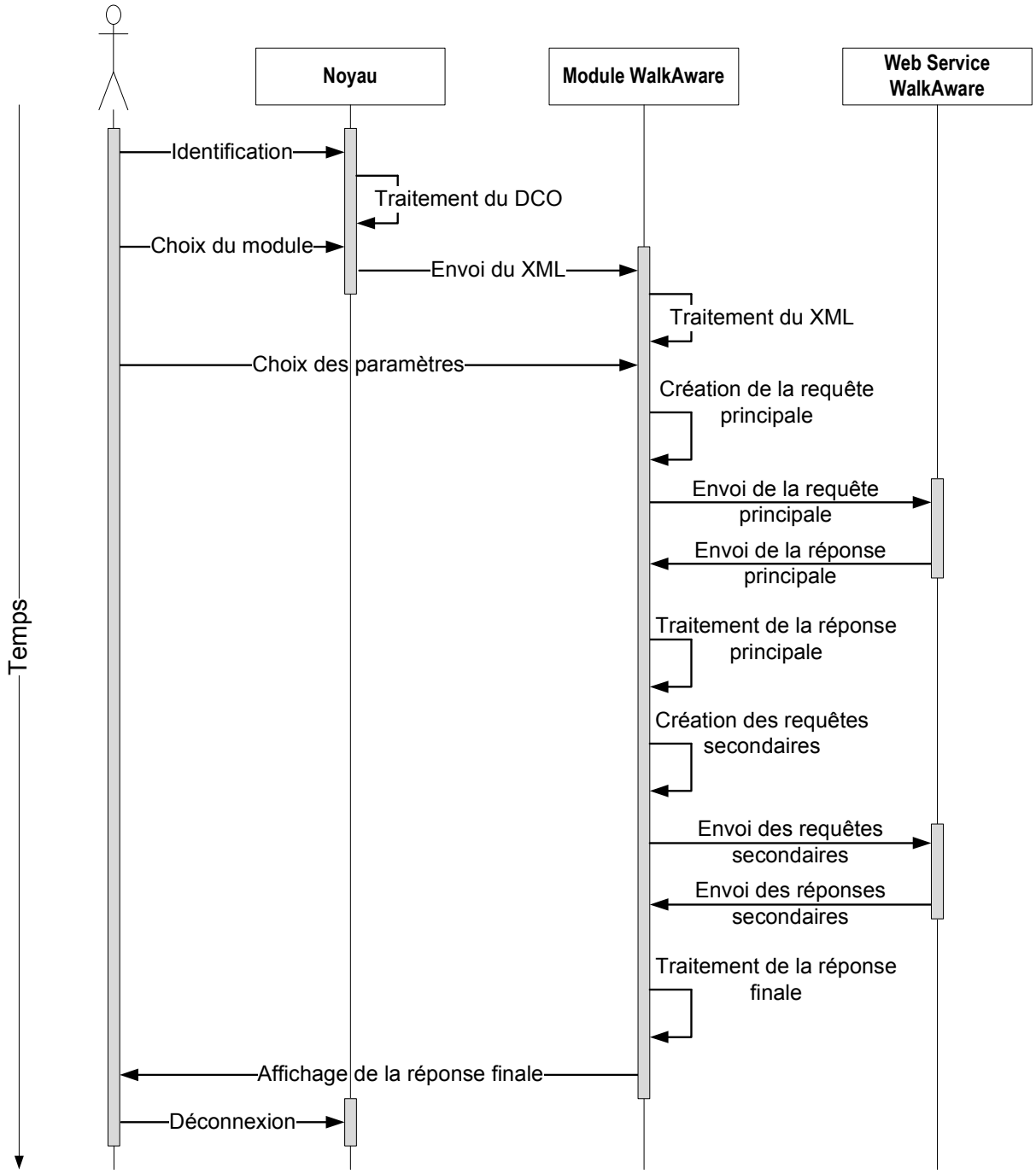

Figure 21 : Diagramme de séquence pour le module « WalkAware »

Dans le module «WalkAware», représenté dans la Figure 21, nous voyons qu'il réalise deux requêtes au web service. Cela s'explique par le fait que la seconde requête consiste à accéder à des ressources qui dépendent de la première requête. Il est donc nécessaire de traiter la première avant d'envoyer la seconde.

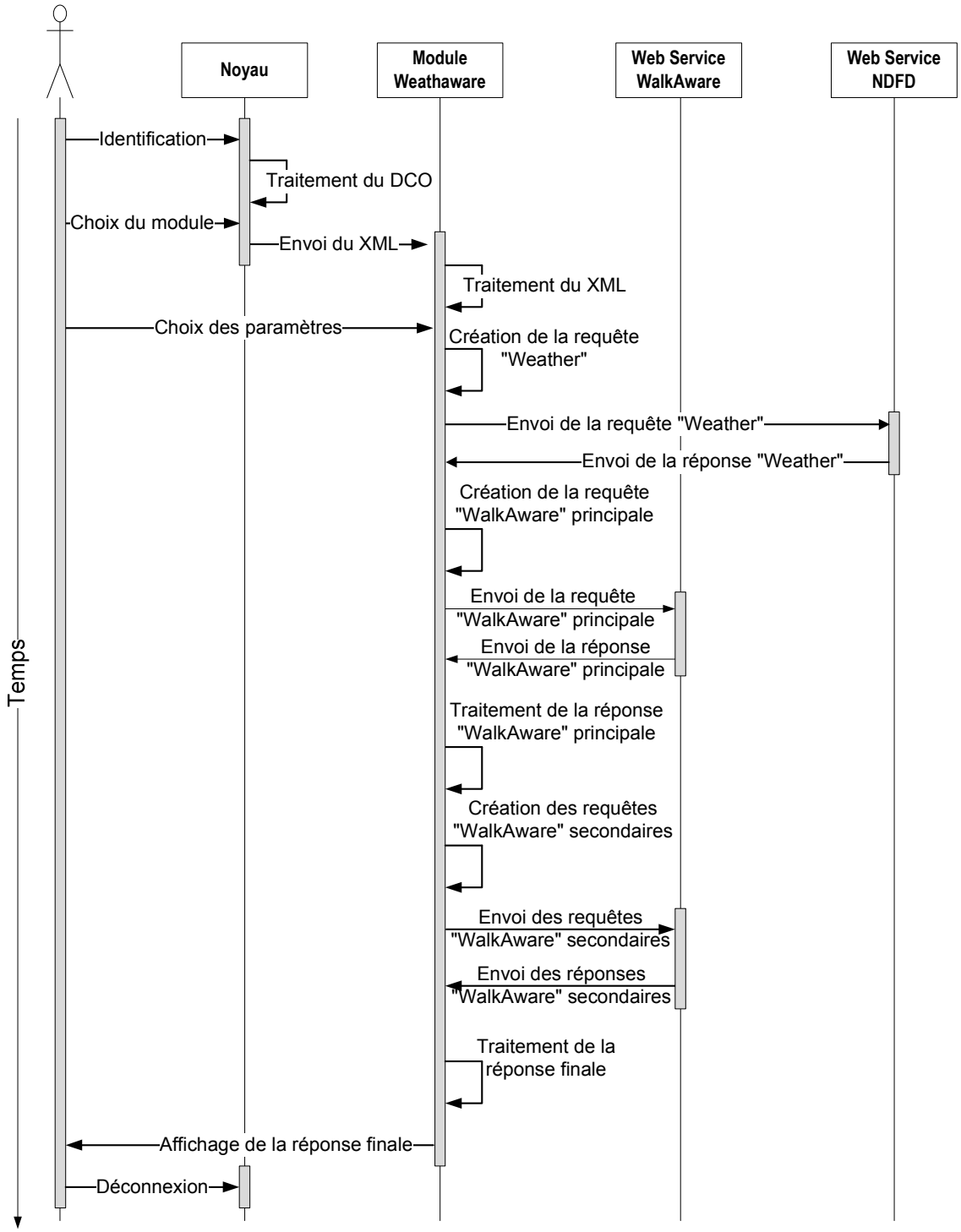

Figure 22 : Diagramme de séquence pour le module «Weathaware»

Le module «Weathaware», représenté par la Figure 22, est simplement la mise en commun des deux modules précédents. Ceci explique le nombre de requêtes aux différents web services qu'il doit effectuer.

3.6. Conclusion

De par sa particularité à intégrer le contexte d'utilisation du client via son fichier DCO, de permettre l'ajout de nouveaux modules assortis sémantiquement à un ou plusieurs web services particuliers, d'alimenter la création d'une interface graphique par l'ajout de règles d'adaptation, vous aurez remarqué l'aspect « *inédit* » de la solution que nous proposons ici pour aborder l'adaptation du front-end d'un web service. Et même s'il est possible de développer un nouveau module en mode WSDL, nous avons privilégié le mode REST, principalement pour les gains en bande passante, parfois cruciale pour un appareil mobile.

Au chapitre suivant, nous vous présenterons trois études de cas pratiques mettant en application notre solution. Nous relèverons les avantages et inconvénients d'utiliser une telle solution, et réaliserons différents tests de performances.

Chapitre 4 : Évaluation de notre solution

Dans le chapitre précédent, nous avons détaillé théoriquement notre solution, ainsi que les choix d'implémentation nécessaires à l'élaboration de celle-ci. Nous allons montrer ici trois études de cas pratiques avec lesquelles nous avons appliqué notre solution. Les avantages et inconvénients d'utiliser une telle solution seront également relevés, ainsi que différents tests de performances pour son évaluation.

4.1. Études de cas

Nous allons, par le biais de trois études de cas, décrire le fonctionnement pratique de notre implémentation. Celles-ci possédant une partie commune dédiée au noyau, elle sera décrite en premier lieu. Chaque module sera ensuite détaillé individuellement, puisque chaque étude de cas correspond respectivement à un module implémenté sur notre proxy : « Weather », « WalkAware » et « Weathaware ». Pour chaque étude, nous utiliserons des captures d'écran ainsi que quelques exemples extraits code source. Ceci afin de détailler au mieux le front-end visible par l'utilisateur, mais aussi la partie invisible sur laquelle il s'appuie. Notez que ces portions de code peuvent avoir été modifiées pour plus de clarté et d'expressivité. Les captures d'écran pour les Smartphones et les tablettes, ont été respectivement réalisées avec un « HTC Desire » et une « Samsung Galaxy Tab ».

Afin de gérer les différents cas de figures existants, nous allons utiliser plusieurs fichiers DCO contenant des contextes différents (voir Figure 23).

Figure 23 : Choix du fichier DCO sur Smartphone

4.1.1. Le noyau

Suite au choix du fichier DCO, l'application le traite avec les règles d'adaptation propres au noyau et redirige le client vers la page d'authentification (voir Figure 24 et Figure 25). Cette page est adaptée au contexte d'utilisation, principalement grâce aux règles suivantes :

- un appareil dont la résolution est comprise entre 480x854 et inférieur est considéré comme un Smartphone ;
- un appareil dont la résolution est comprise entre 480x854 et 1280x800 est considéré comme une tablette/Pocket PC ;
- un appareil dont la résolution est supérieure à 1280x800 est considéré comme un PC.

Remarquez la différence de taille des caractères et les dimensions du formulaire d'authentification, entre la tablette et le PC, du fait de la règle :

- la taille du texte affiché varie en fonction de la plateforme sélectionnée :
  - grande sur un Smartphone ;
  - grande sur une tablette/Pocket PC ;
  - petite sur un PC.

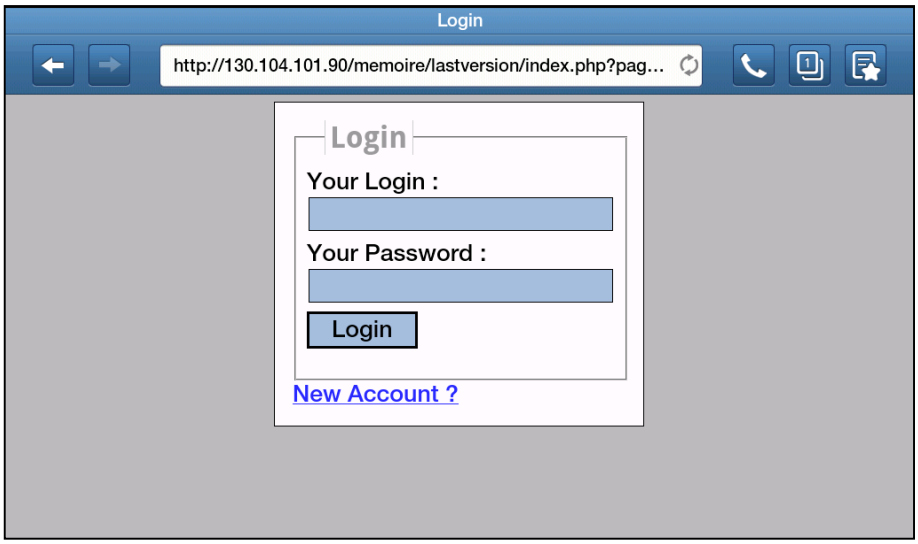

Figure 24 : Authentification sur tablette/Pocket PC

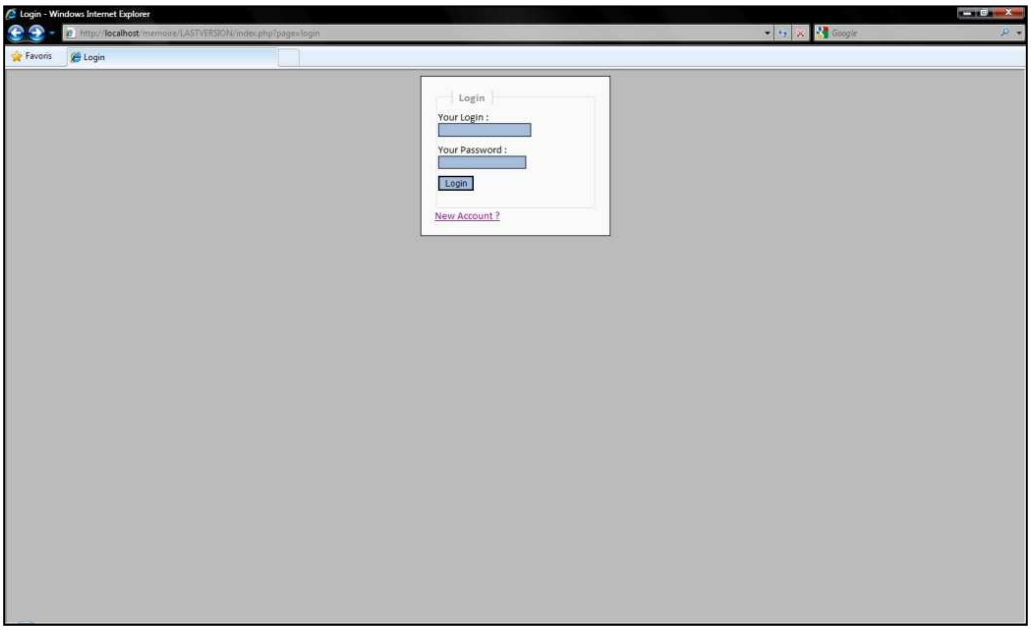

Figure 25 : Authentification sur PC

S'il ne possède pas de compte, l'utilisateur pourra accéder à une page d'enregistrement (voir Figure 26).

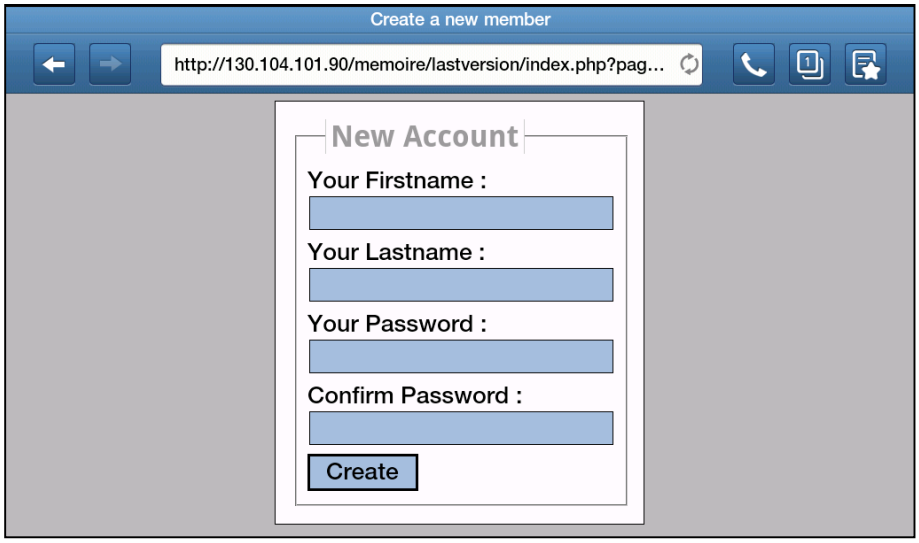

Figure 26 : Création d'un compte sur tablette/Pocket PC

Par la suite, nous lui proposerons de choisir un module parmi ceux présent sur le Proxy (voir Figure 27). Une fois son choix effectué, le noyau passe la main au module sélectionné.

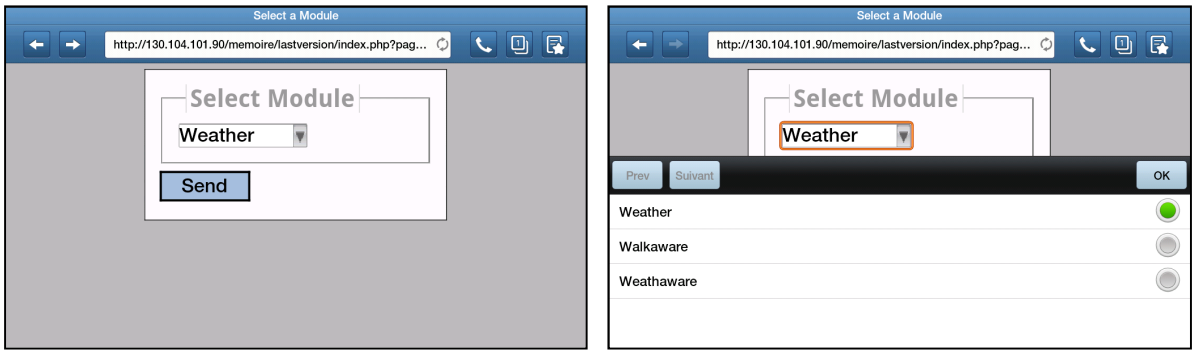

Figure 27 : Choix du module sur tablette/Pocket PC

Vous pouvez remarquer en Figure 27 ci-dessus, la manière dont sont gérées les listes déroulantes par le navigateur lui-même. C'est de là que nous avons décidé d'utiliser des listes déroulantes lorsque les utilisateurs de Smartphones et tablettes ont plusieurs choix proposés.

4.1.2. Le module « Weather »

Ce module est particulièrement intéressant au niveau de l'adaptation, puisque le web service correspondant (NDFD) nous propose plusieurs paramètres différents, qui vont nous permettre de mettre en pratique l'expression des changements par rapport au contexte d'utilisation. Ainsi, en fonction du contexte et en adéquation avec les règles d'adaptation de ce module, une page de configuration adaptée est affichée au client (voir Figure 28, Figure 29 et Figure 30) où les différents paramètres sont présélectionnés selon les deux règles suivantes :

- le nombre de paramètres sélectionnés varie en fonction de la plateforme sélectionnée :
  - nombre de paramètres sélectionnés = 3 sur un Smartphone ;
  - nombre de paramètres sélectionnés = 5 sur une tablette/Pocket PC ;
  - nombre de paramètres sélectionnés = 9 sur un PC ;

- le nombre de jours de prévisions varie en fonction de la plateforme sélectionnée :
  - nombre de jours de prévisions = 2 sur un Smartphone ;
  - nombre de jours de prévisions = 4 sur une tablette/Pocket PC ;
  - nombre de jours de prévisions = 6 sur un PC.

La façon dont nous avons géré la présélection du nombre de jour par rapport à la plateforme est relevée par le Code 7.

```
<?php if($platform == "PC")  { ?>
 <input type="radio" name="days" value="1">1 day</input>
 <input type="radio" name="days" value="2">2 days</input>
 <input type="radio" name="days" value="3">3 days</input>
 <input type="radio" name="days" value="4">4 days</input>
 <input type="radio" name="days" value="5">5 days</input>
 <input type="radio" name="days" value="6" checked="checked">6 days</input>
<?php } else { ?>
 <select name="days">
  <option value="1">1 day</option>
  <option value="2" <?php if($platform == "Smartphone") {echo 'selected';}?> >2 days</option>
  <option value="3">3 days</option>
  <option value="4" <?php if($platform == "Tablet") {echo 'selected';}?>>4 days</option>
  <option value="5">5 days</option>
  <option value="6">6 days</option>
</select>
<?php } ?>
```

Code 7 : Affichage et présélection des jours en fonction de la plateforme

Figure 28 : Configuration de « Weather » sur PC

Tous ces paramètres ne sont sélectionnés que par défaut, puisque le client garde la possibilité de modifier les paramètres à sa convenance, afin de respecter la règle :

- les préférences de l'utilisateur sont toujours prioritaires par rapport aux paramètres prédéfinis automatiquement.

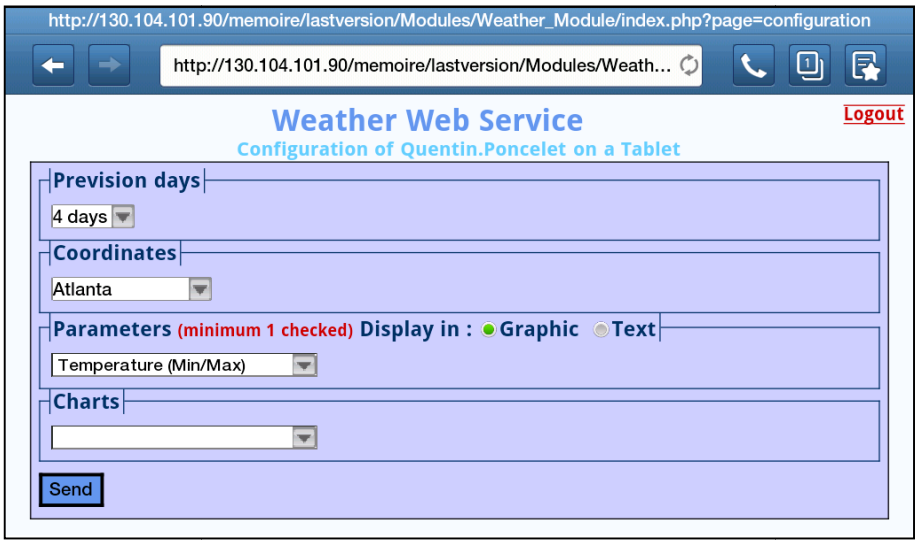

Figure 29 : Configuration de « Weather » sur tablette/Pocket PC

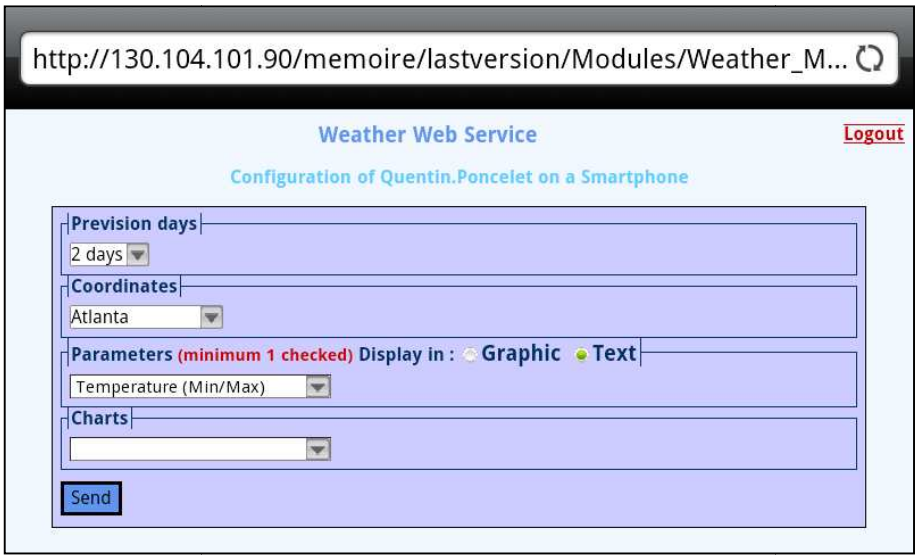

Figure 30 : Configuration de « Weather » sur Smartphone

Vous remarquerez une différence d'affichage entre le PC et les deux autres appareils, au niveau du choix du nombre de jour de prévision, ainsi que la liste des paramètres à sélectionner.

Sémantiquement, les deux types d'affichages restent les mêmes (d'un côté une simple liste de cases à cocher et de l'autre une liste déroulante de formulaire), mais cela a l'avantage de gagner de la place pour les deux écrans aux plus petites résolutions. Vous retrouverez un exemple de cette différenciation dans le Code 9 mais aussi dans le Code 7 et Code 9. Dans le Code 9, vous pourrez également remarquer la gestion du style d'affichage en mode graphique ou textuel (ici, le choix d'afficher les graphiques de température, de précipitation et de couverture nuageuse). Ceci permet de respecter la règle :

- le style d'affichage varie en fonction de la plateforme sélectionnée :
  - textuel sur un Smartphone ;
  - graphique sur une tablette/Pocket PC ;
  - graphique sur un PC.

```
<?php if($platform == "PC") {?>
<ul>
  <li><input type="checkbox" checked />Temperature (Min/Max)</li>
  <li><input type="checkbox" checked />Weather Icons</label></li>
  <li><input type="checkbox" checked />Liquid Precipitation Amount</li>
  <li><input type="checkbox" checked />Wind (Speed/Direction)</li>
  <li><input type="checkbox" checked />Cloud Cover Amount</li>
  <li><input type="checkbox" checked />Wave Height</li>
  <li><input type="checkbox" checked />Snowfall Amount</li>
</ul>
<?php }
else { ?>
<select name="my_param[]" multiple>
  <option selected>Temperature (Min/Max)</option>
  <option selected>Weather Icons</option>
  <option>Liquid Precipitation Amount</option>
  <option <?php if($platform == "Tablet") {
              echo 'selected';}?> >Wind</option>
  <option>Cloud Cover Amount</option>
  <option>Wave Height</option>
  <option>Snowfall Amount</option>
</select>
<?php } ?>
```

Code 8 : Deux affichages en liste sémantiquement identiques

```php
<?php if($DisplayStyle == "Graphic" AND $InformationLevel == "Big") {?>
<ul>
  <li><input type="checkbox" value="temp" checked />Temperature (Min/Max)</li>
  <li><input type="checkbox" value="lpa" checked />Liquid Precipitation Amount</li>
  <li><input type="checkbox" value="cca" checked />Cloud Cover Amount</li>
</ul>
<?php }
else { ?>
<select name="charts[]" multiple>
  <option value="temp">Temperature (Min/Max)</option>
  <option value="lpa">Liquid Precipitation Amount</option>
  <option value="cca">Cloud Cover Amount</option>
</select>
<?php } ?>
```

Code 9 : Deux affichages en liste sémantiquement identiques

Lorsque l'utilisateur a sélectionné la configuration qu'il désire, le front-end final adapté au contexte d'utilisation est alors généré. Nous pouvons remarquer sur les Figure 31 et Figure 32 l'affichage en mode graphique pour le PC et la tablette. L'affichage en mode textuel avec icones adaptées spécifique à l'utilisation d'un Smartphone, est visible à la Figure 33.

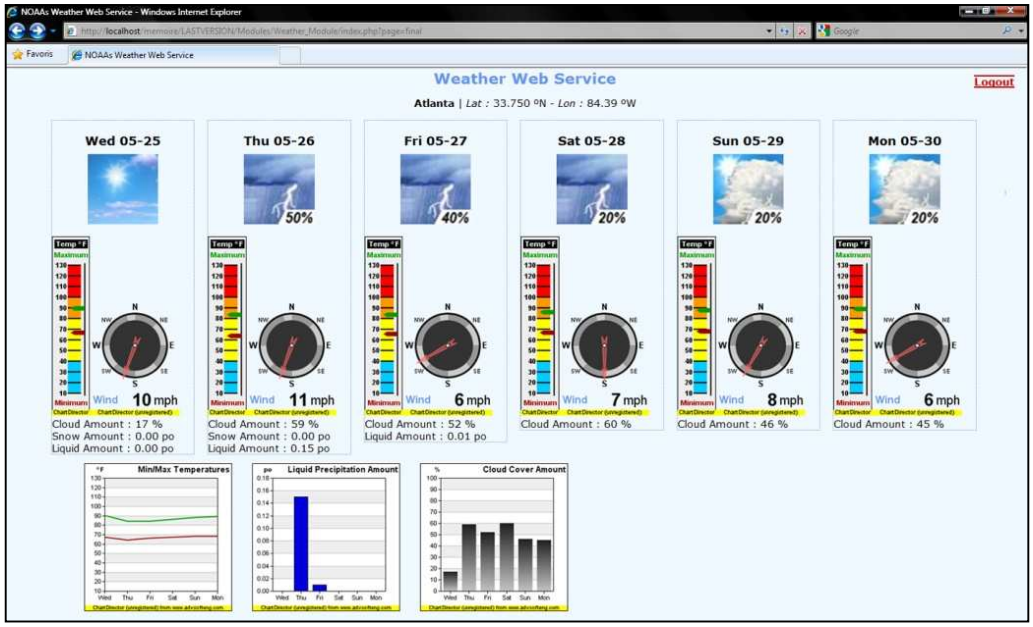

Figure 31 : Affichage final « Weather » sur PC

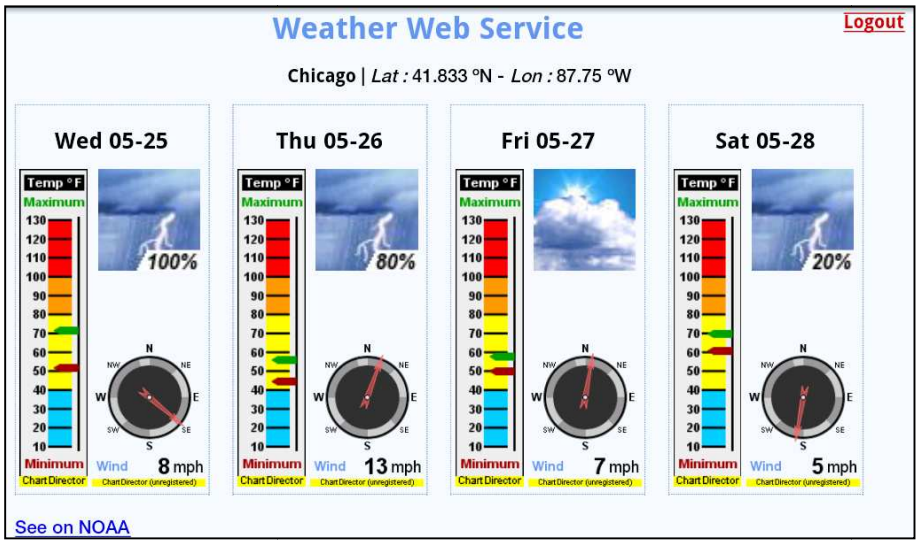

Figure 32 : Affichage final de « Weather » sur tablette/Pocket PC

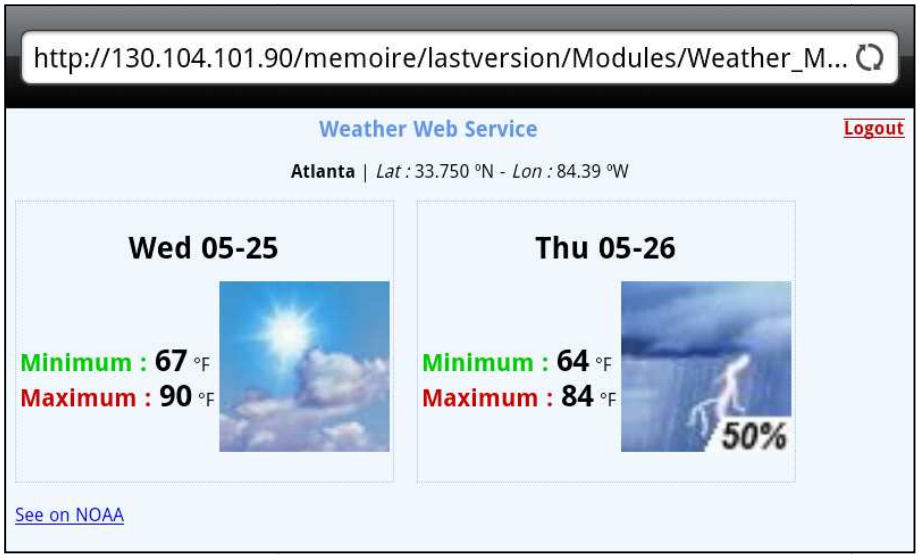

Figure 33 : Affichage final « Weather » sur Smartphone

En Figure 34 et Figure 35, vous trouverez l'affichage de la page de configuration et la page finale sur un Smartphone et une tablette dont les batteries sont faibles, selon les règles proposées par le noyau et appliquées dans les trois modules :

- l'état de la batterie varie selon le niveau de charge de celle-ci :
  - si le niveau est compris entre 100% et 20%, l'état est égal à Good ;
  - si le niveau est inférieur à 20%, l'état est égal à Low ;

- le style d'affichage varie en fonction de l'état de la batterie :
  - fond et couleurs sombres si le niveau de la batterie est Low ;
  - fond et couleurs vives si le niveau de la batterie est Good ;
- le contraste d'affichage varie en fonction du jour et de la nuit :
  - fond clair et couleurs sombres pour le jour ;
  - fond sombre et couleurs claires pour la nuit.

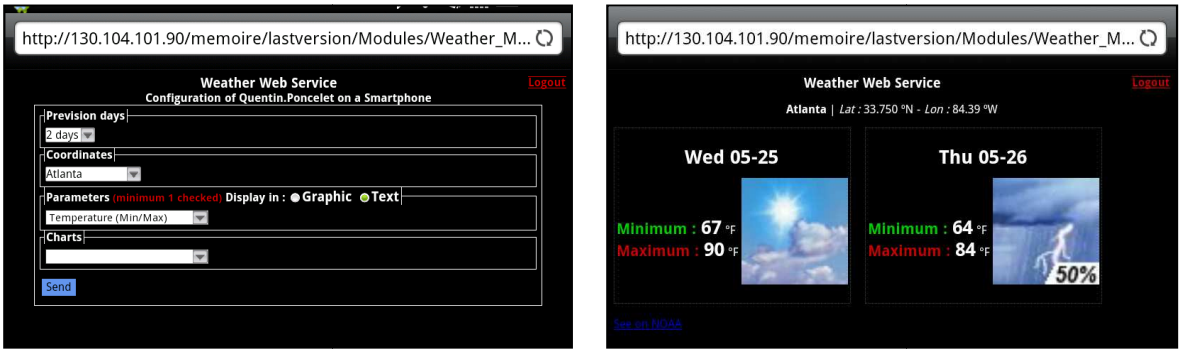

Figure 34 : Affichage Configuration et Finale de « Weather » sur Smartphone Batterie faible

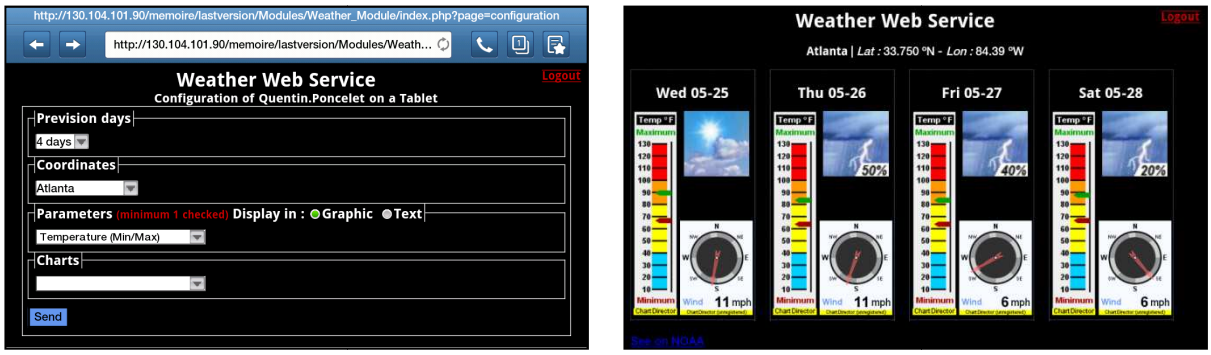

Figure 35 : Affichage Configuration et Finale de « Weather » sur tablette/Pocket PC Batterie faible

4.1.3. Le module « WalkAware »

Les clients sont ici aussi redirigés vers une page de configuration adaptée. Les Figure 36 et Figure 37 montrent respectivement cette page de configuration sur un Smartphone avec et sans une batterie faible. Les mêmes remarques formulées pour le module précédent peuvent l'être ici, au niveau des règles d'adaptation du contraste par rapport au niveau de charge des batteries, ainsi que l'affichage des paramètres en une liste de cases à cocher pour le PC et une liste déroulante de formulaire pour les appareils mobiles.

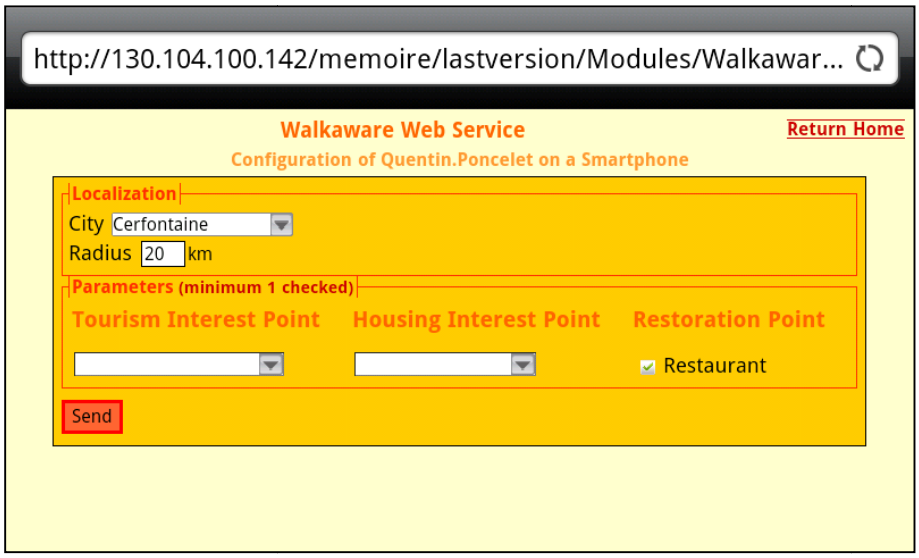

Figure 36 : Configuration « WalkAware » sur Smartphone

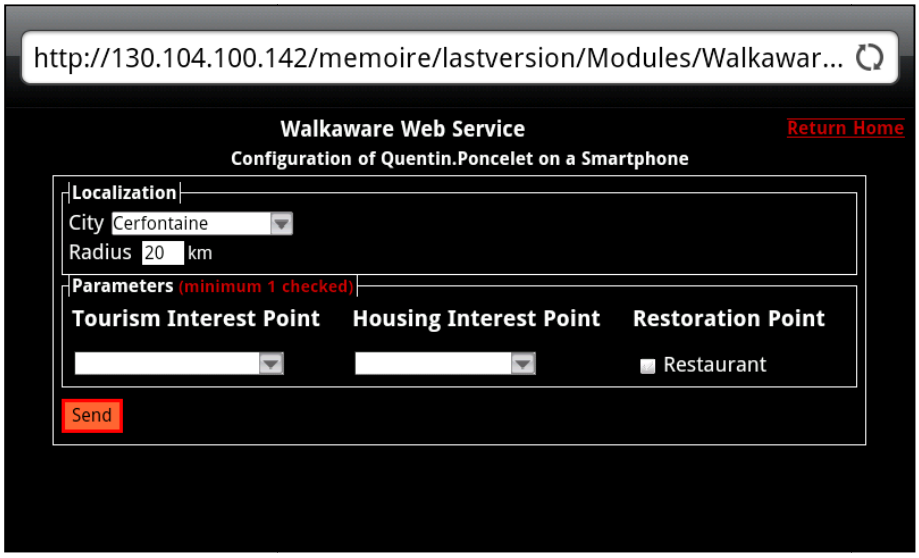

Figure 37 : Configuration « WalkAware » sur Smartphone Batterie faible

De la même manière, les Figure 38 et Figure 39 présentent cette page pour une tablette et une tablette avec sa batterie faible. Remarquez l'adaptation du rayon de recherche par rapport à la ville sélectionnée, d'après le respect de la règle ci-après.

- La distance de recherche des Points d'intérêts varie selon la plateforme sélectionnée :
    - 20 km sur un Smartphone ;
    - 50 km sur une tablette/Pocket PC ;
    - 100 km sur un PC.

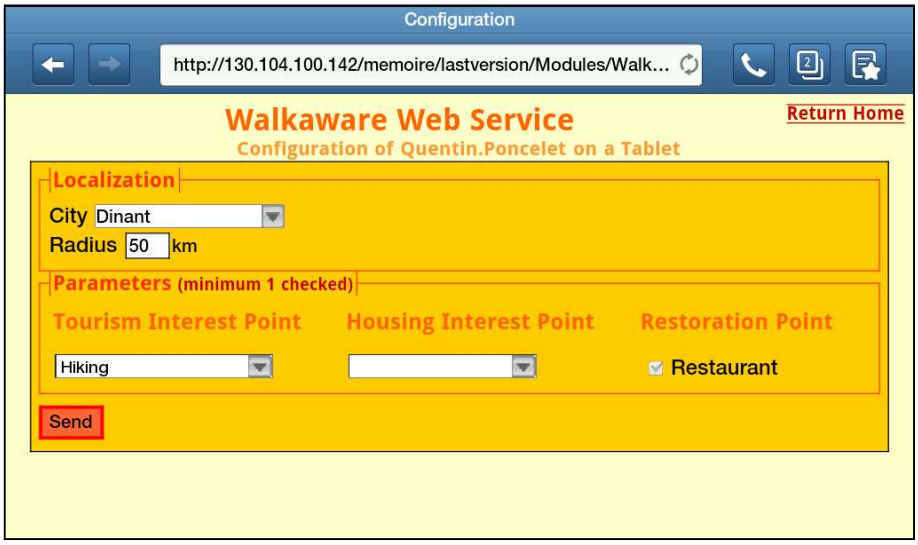

Figure 38 : Configuration « WalkAware » sur tablette/Pocket PC

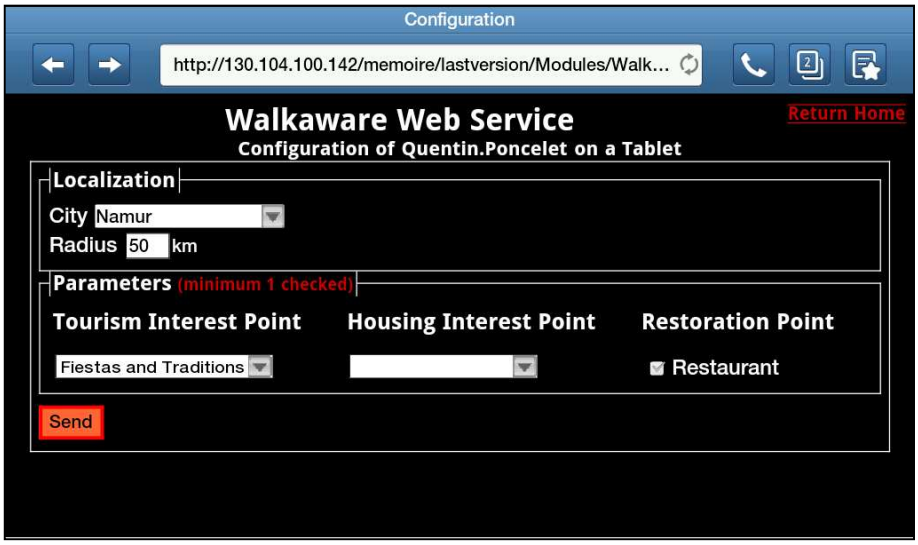

Figure 39 : Configuration « WalkAware » sur tablette/Pocket PC Batterie faible

La page de configuration de la Figure 40 est finalement celle qui est générée et affichée pour un PC.

Figure 40 : Configuration de « WalkAware » sur PC

Ce module « WalkAware » diffère du module précédent dans la mesure où l'affichage du résultat final nécessite l'appel à un autre web service : Google maps. Vous remarquerez par les Figure 41, Figure 42 et Figure 43, l'adaptation de la place que prend la carte de Google proportionnellement au reste de la page (où se retrouvent aussi un panneau de « Customization » de la ballade, et l'itinéraire textuel de la ballade nouvellement créée), selon le respect de la règle :

- l'affichage de la Google maps varie selon la plateforme sélectionnée :
  - petite sur un Smartphone ;
  - moyenne sur une tablette/Pocket PC ;
  - grande sur un PC.

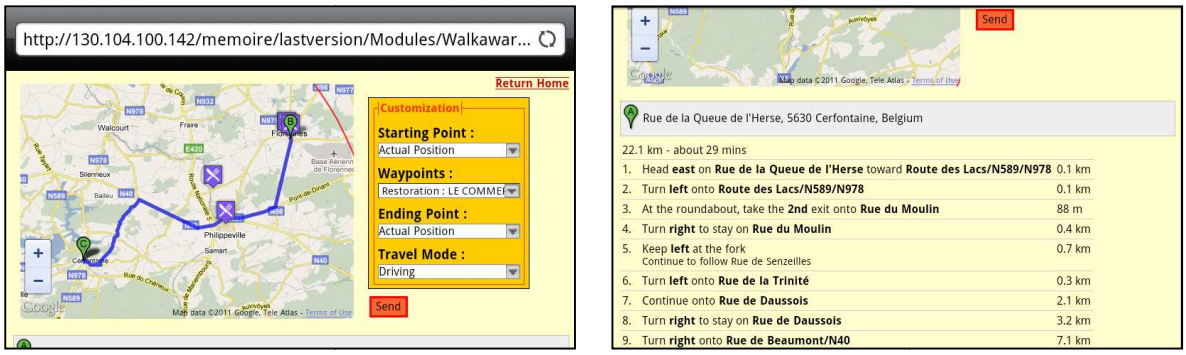

Figure 41 : Affichage final de « WalkAware » sur Smartphone

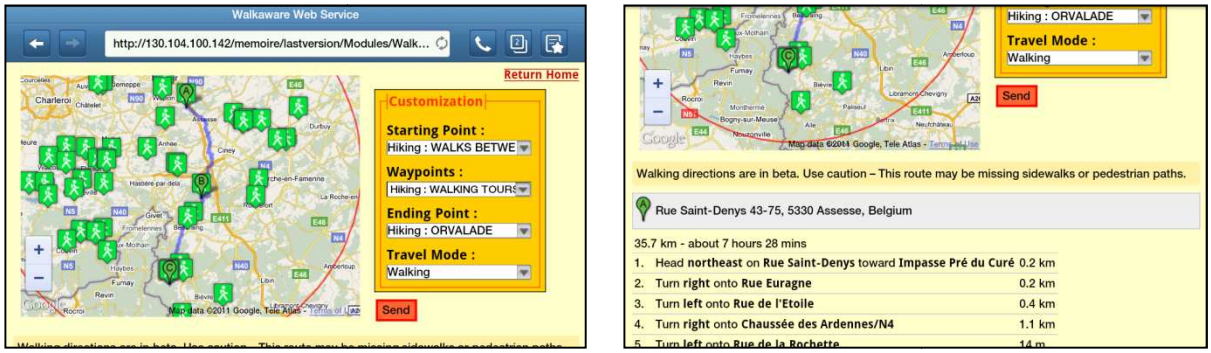

Figure 42 : Affichage final de « WalkAware » sur tablette/Pocket PC

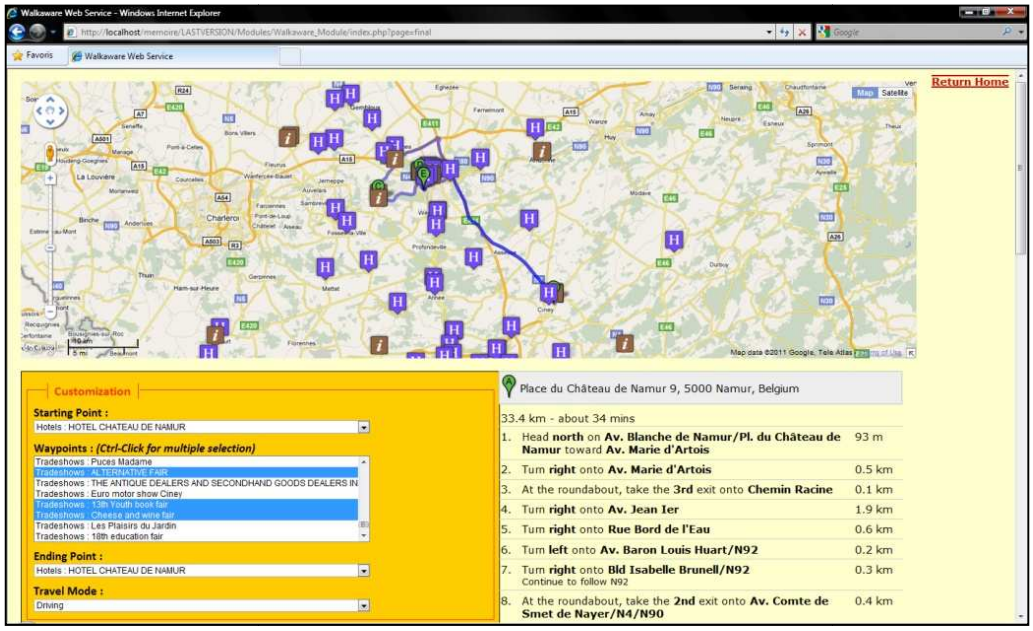

Figure 43 : Affichage final de « WalkAware » sur PC

Se retrouvent en Figure 44 et Figure 45, l'affichage de la page finale avec Smartphone et tablette à batterie faible.

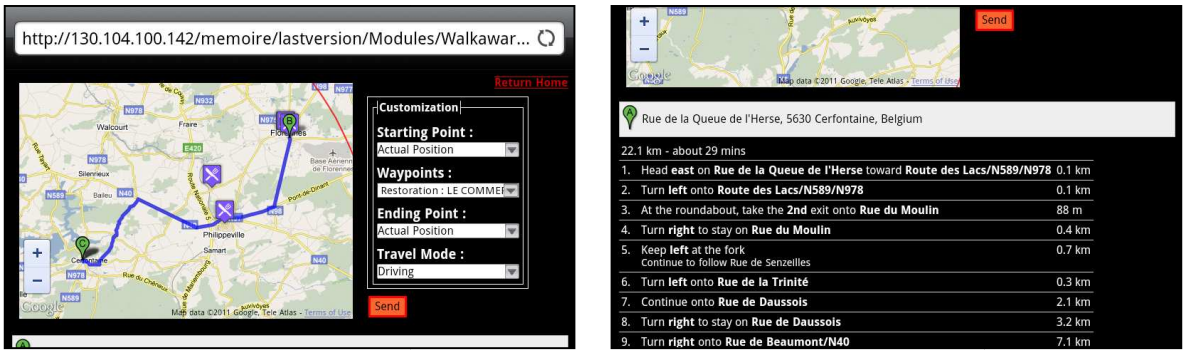

Figure 44 : Affichage final de « WalkAware » sur Smartphone Batterie faible

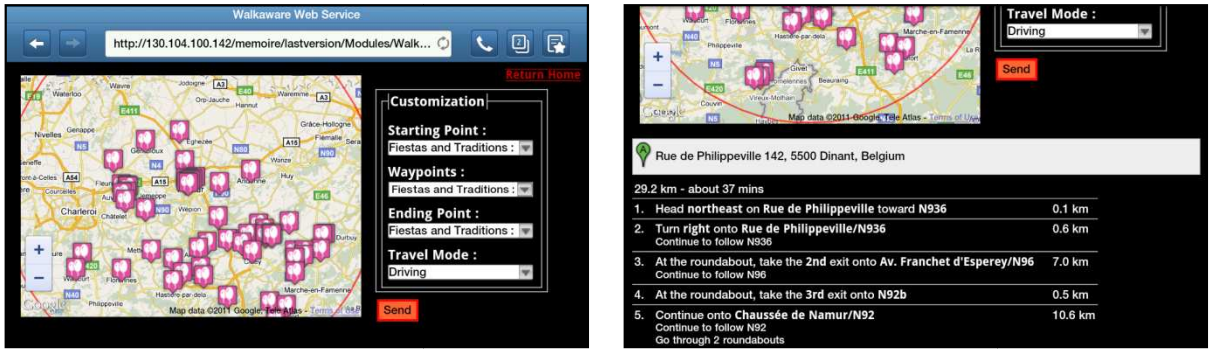

Figure 45 : Affichage final de « WalkAware » sur tablette/Pocket PC Batterie faible

4.1.4. Le module « Weathaware »

Ce module étant la combinaison des modules « Weather » et « WalkAware », tous les commentaires réalisés précédemment pour ces deux modules restent valables ici.

Les Figure 46, Figure 47 et Figure 48 affiche les adaptations de la page de configuration du module.

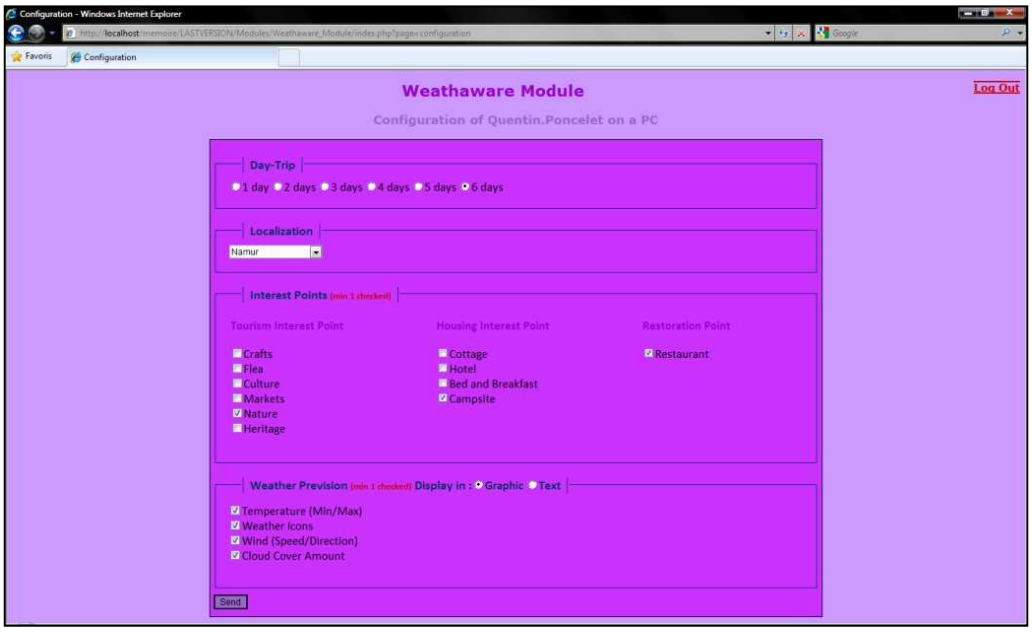

Figure 46 : Configuration de « Weathaware » sur PC

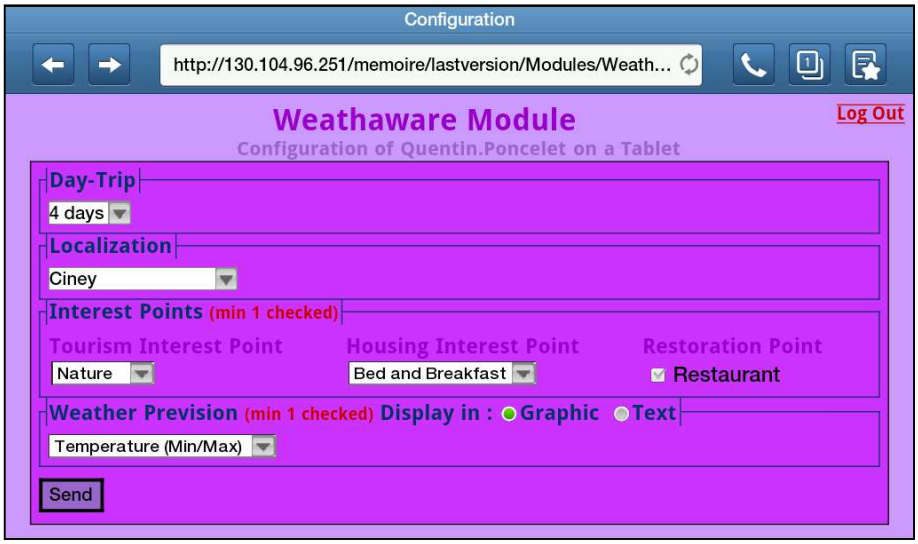

Figure 47 : Configuration de « Weathaware » sur tablette/Pocket PC

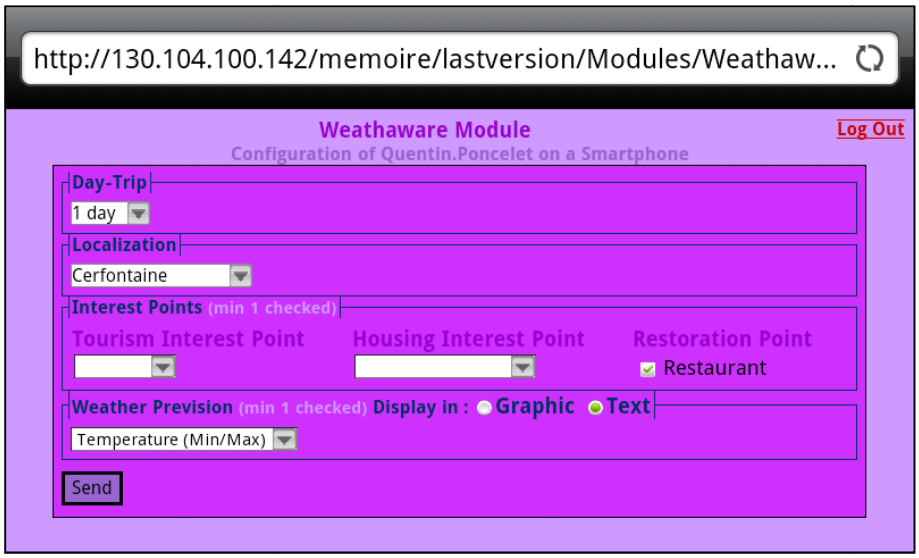

Figure 48 : Configuration de « Weathaware » sur Smartphone

Les Figure 49 et Figure 50 affiche respectivement l'adaptation de la page finale du module sur un PC et une tablette.

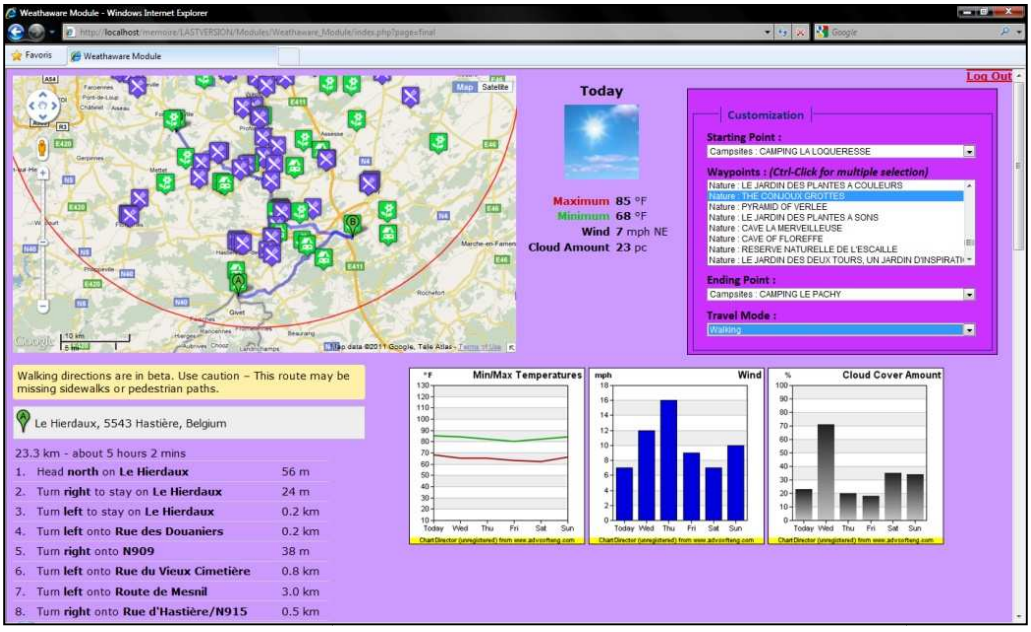

Figure 49 : Affichage final de « Weathaware » sur PC

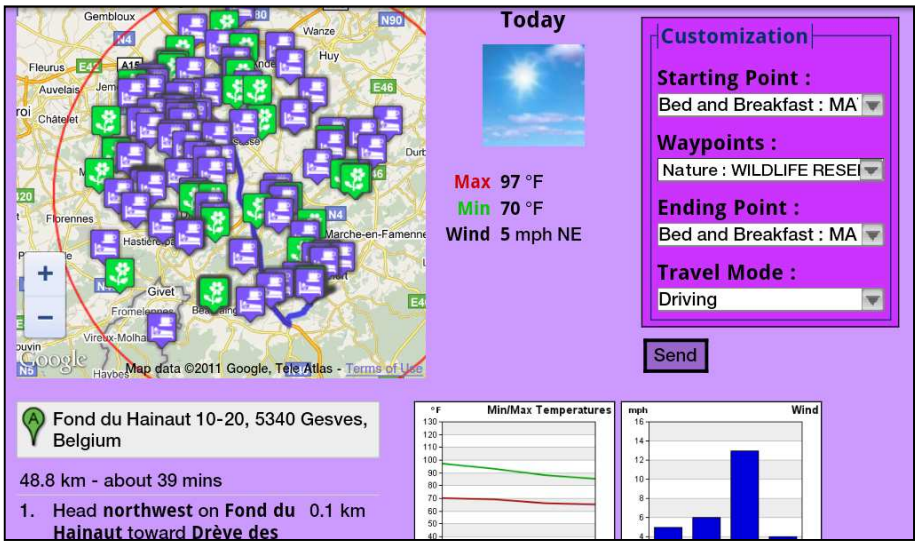

Figure 50 : Affichage final de « Weathaware » sur tablette/Pocket PC

Remarquez sur la Figure 50 la largeur du panneau de « Customization » qui ne permet pas de voir l'entièreté des informations contenues dans la liste déroulante. Cela est compensé (voir Figure 51) par la façon dont la tablette gère les listes déroulantes, pour un affichage des informations sur l'entièreté de la largeur de l'écran.

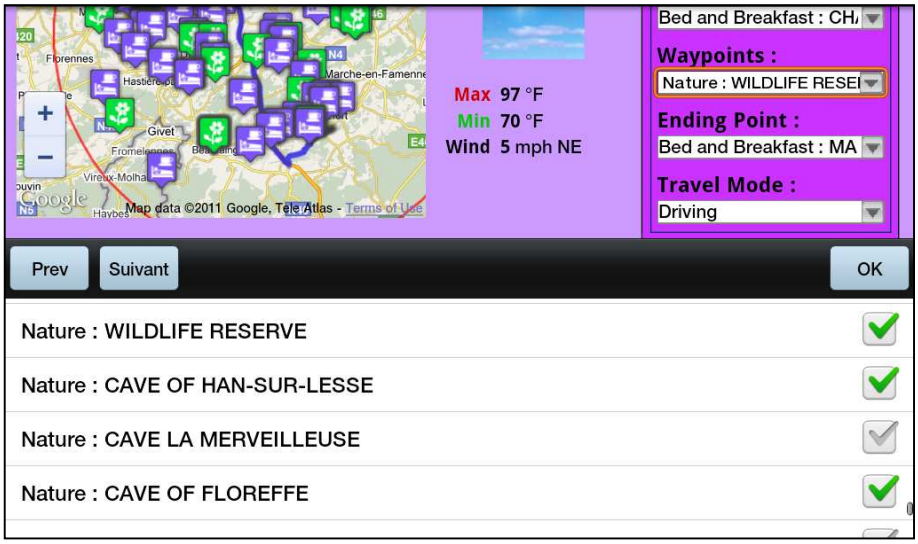

Figure 51 : Liste déroulante dans l'affichage final de « Weathaware » sur tablette/Pocket PC

En Figure 52, se retrouve l'affichage en mode texte des jours de prévisions durant la ballade, ainsi que l'itinéraire à suivre, sur un Smartphone.

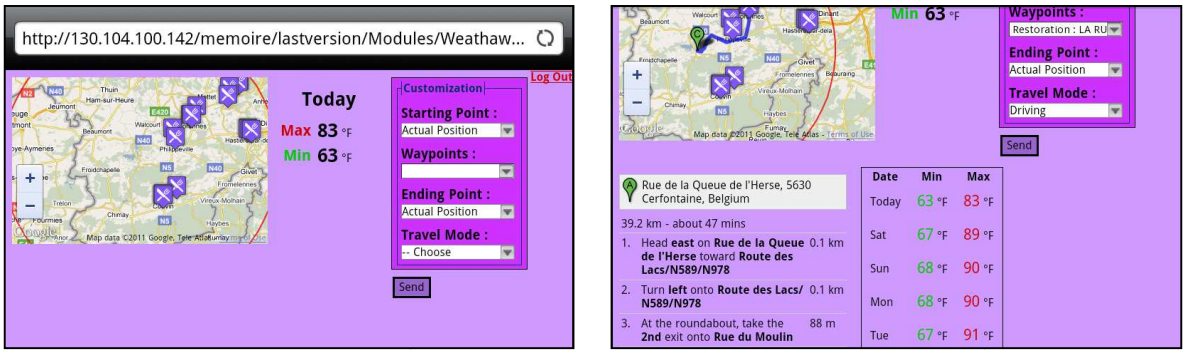

Figure 52 : Affichage final de « Weathaware » sur Smartphone

Ce module adapte également l'affichage pour des tablettes et Smartphones avec une batterie faible. Il n'est pas opportun de remettre ici les captures d'écran, étant donné qu'elles ne diffèrent pas de leurs homologues que vous avez pu plus haut.

4.2. Les avantages et forces

Le principal avantage de notre solution est qu'elle propose une **optimisation « pré-requête »** qui, dés la connexion du client, prend en charge le contexte d'utilisation et emploie ce dernier afin de créer une requête optimale. Cela diffère de la majorité des solutions existantes présentées (voir la section 2.3) qui proposent une optimisation « post-requête ». Cette optimisation minimise le temps dévolu à la requête de même que les échanges entre le Proxy et le web service interrogé. Elle permet ainsi d'augmenter la réactivité de l'application. Dans la même optique de minimisation du temps et des échanges de données, nous avons utilisé le mode REST (section 3.1.4) qui permet des gains significatifs en temps et en transfert de données par rapport au mode WSDL.

Un autre avantage provient du fait que nous proposons aux développeurs de modules, un **fichier XML résumant le fichier DCO**. Ainsi, le noyau ne doit pas transférer l'entièreté du fichier DCO aux modules et ne s'occupe qu'une seule et unique fois de cette gestion du « contexte ». Ceci à pour objectif de faciliter la création des modules car ceux-ci ne doivent pas gérer le fichier DCO et n'ont plus qu'à utiliser ce contexte simplifié comme entrée. Nous épargnons, grâce à cette solution, le coût de transfert proxy-module inhérent à ce fichier.
Dans le même ordre d'idée, le « CSS Generator » (section 3.4.1) produit un **fichier CSS le plus concis** et **optimal** possible, envoyé ensuite au client.

Nous proposons également de laisser, au développeur de module, le **choix d'utiliser ou non « notre vision » du contexte simplifié** puisque nous avons inclus, dans le fichier XML, une section « Hardware » contenant les données brutes fournies par le fichier DCO et utilisées dans le calcul de notre contexte. Cette vision permet une certaine liberté dans la création du module car il n'y a pas de contrainte posée par le noyau excepté le format du fichier XML.

Du fait de **l'implémentation « coté serveur » du noyau,** peu de calculs sont effectués par les clients. Ceci se révélera intéressant lors de l'utilisation de Smartphones ou de tablettes dont l'autonomie de la batterie se révèle être une faiblesse.

Finalement, notre application est utilisable sur **plusieurs plateformes, indépendamment** de **l'OS** et du **navigateur**, pour peu que le JavaScript soit activé sur le navigateur, vu que deux de nos modules utilise Google maps.

4.3. Les inconvénients et faiblesses

Contrairement aux solutions proposées (voir section 2.3), notre solution ne travaille qu'avec des web services. Il n'est donc pas possible de traduire ou transformer n'importe quelle page d'un site web classique. Une solution est que ce site *classique* utilise un web service, alors la création d'un module adéquat permettra l'adaptation.
Ceci nous amène à soulever un autre inconvénient : notre solution ne propose pas de module générique s'adaptant à n'importe quel web service. Il est donc obligatoire de créer un module sémantiquement lié à chaque web service.

Dans notre objectif de minimisation des coûts au niveau des transferts de données, nous perdons beaucoup dans le transfert du fichier DCO entre le client et le proxy. La gestion de l'envoi de ce fichier n'étant pas encore abordée par le W3C, nous osons espérer qu'une solution pourra être dégagée concernant la taille et le transfert de celui-ci.

Enfin, contrairement au mode WSDL, l'utilisation du mode REST nécessite, pour le développeur du module, une documentation manuelle sur le web service avant l'implémentation du module correspondant.

4.4. Les critères d'efficacité

Dans cette section, nous allons évaluer notre solution selon quatre critères d'efficacité.

4.4.1. Informationnelle

Nous avons choisi de délivrer les informations de manière claire et simple pour l'utilisateur afin qu'il ne perde pas de temps à les rechercher dans la page.

Pour le module « Weather », nous avons ainsi privilégié l'affichage des températures minimales et maximales sous la forme d'un thermomètre vertical, la direction et la vitesse du vent sous la forme d'une rose des vents pour les affichages de types graphiques (tablette et PC). Dans le même ordre d'idée, l'affichage des graphiques d'évolution de températures, de précipitations et de couverture nuageuse permet à l'utilisateur d'avoir un aperçu immédiat sans devoir réaliser une comparaison entre toutes les valeurs.

Pour le module « WalkAware », l'affichage Google maps permet à l'utilisateur de situer les différents points d'intérêts plus facilement que si nous lui avions fourni une liste.

4.4.2. Organisationnelle

Le choix de plusieurs paramètres permet à différents profils d'utiliser notre solution. Ainsi, le débutant pourra se laisser guider car la marche à suivre est logique et aussi parce que la configuration est déjà pré-remplie selon les règles d'adaptation, tandis que l'utilisateur confirmé pourra affiner sa requête en fonction de ses propres besoins.

4.4.3. Économique

Ce critère peut être développé selon les différents points de vue ci-dessous :

- **le web service** a un coût de fonctionnement plus faible dû à l'utilisation de REST ainsi qu'à la requête optimisée par notre solution ;
- **le Proxy** a coût de fonctionnement plus important dû au travail qu'il doit effectuer ;
- **le client** a un coût de fonctionnement plus faible et réalise également des économies, s'il utilise le réseau EDGE/3G, sur son forfait de données grâce aux requêtes optimisées.

4.4.4. De réalisation

Grâce au fichier XML résumant le fichier DCO, la possibilité d'inclure facilement un nouveau module ou de modifier le noyau, tant que les spécifications sont respectées (le noyau doit fournir un fichier XML formaté et le module doit contenir un dossier cache où le noyau va le déposer). De plus, l'utilisation du pattern MVC permet une maintenance et une mise à jour facilitée du noyau. L'indépendance par rapport à l'OS et le navigateur facilite également la création d'un nouveau module.

4.5. Tests de performance

Pour ces tests, nous allons comparer les performances du module « Weather », qui va nous servir de standard de test. C'est en effet le seul qui nous permettra une comparaison entre WSDL et REST, puisqu'il est le seul module à être doté de ces deux modes. Mesurer le temps d'exécution des requêtes ainsi que la taille totale des paquets échangés entre le module et le web service nous semble les tests les plus pertinents à réaliser.

Pour permettre une précision des relevés de nos tests et dans l'optique de ne pas surcharger le web service, nous avons définis le protocole de test suivant :

- chaque requête, qu'elle soit en mode WSDL ou en mode REST, est réalisée 1 seconde avant la suivante ;
- après 20 requêtes, une moyenne sera calculée et 15 secondes sépareront le début du calcul de chaque moyenne ;
- 10 moyennes seront utilisées pour calculer la moyenne finale ;
- la suspension de l'exécution se fera à l'aide de la fonction PHP *usleep()* (qui arrête l'exécution durant quelques microsecondes) ;
- lorsque nous montrerons une évolution, seul le mode REST sera utilisé. Nous supposerons que le mode WSDL évolue de la même manière, toutes proportions gardées ;
- le temps d'exécution sera calculé en utilisant la fonction PHP *microtime()* (qui retourne le timestamp Unix en microsecondes) avant et après l'exécution de la requête. Une soustraction des deux valeurs ainsi obtenues, nous donnera la durée de l'exécution ;
- le logiciel « Wireshark » nous permettra de calculer la taille de chaque paquet échangé. Une addition de ces différentes tailles nous donnera ainsi la quantité totale de paquets échangés.

4.5.1. Le temps d'exécution

Dans les chapitres précédents, il a été discuté du fait qu'un module en mode WSDL prenait plus de temps pour la réalisation de sa requête que si elle était réalisée en mode REST. La Figure 53 indique, pour chaque mode, la moyenne de 200 requêtes demandant 2 jours de prévisions avec 9 paramètres sélectionnés, selon le protocole de test définit ci-dessus.

Figure 53 : Comparaison du temps d'exécution

Nous avons donc confirmé, grâce à ces tests, l'intuition que nous avions concernant le fait que WSDL prenait plus de temps, à cause de la demande du fichier WSDL au web service, précédant la requête proprement dite.

Nous avons également voulu connaitre le temps d'exécution d'une requête en fonction de la quantité de données demandées. La Figure 54 montre l'évolution de la moyenne de 50 requêtes REST, en fonction du nombre de paramètres sélectionnés pour 2 et 6 jours de prévisions.

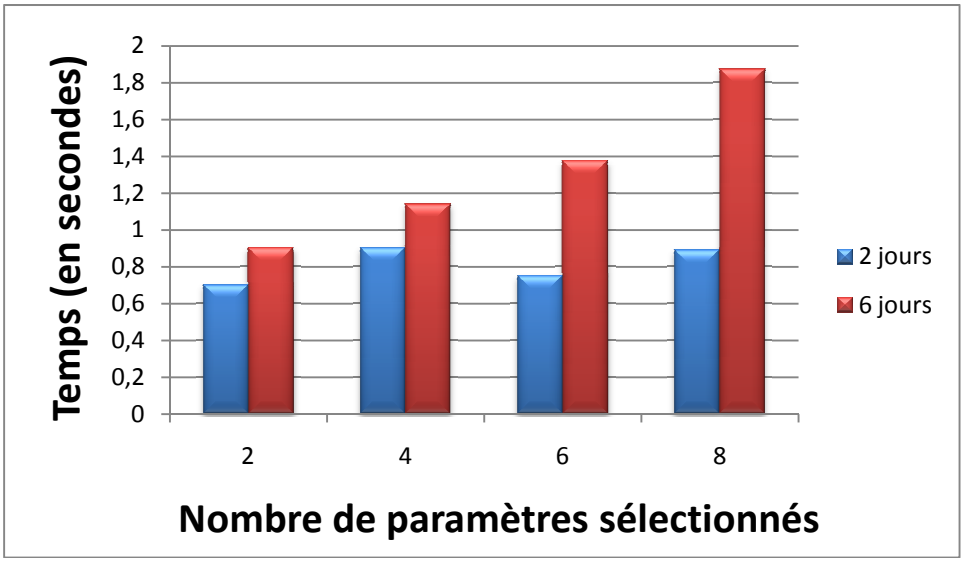

Figure 54 : Évolution du temps d'exécution

Les tests nous montrent qu'il existe bien une relation entre le temps d'exécution et la quantité de données demandées au web service.
Remarquons que le temps pris par la requête de 4 paramètres pour 2 jours de prévision est supérieur aux autres requêtes, pour 2 jours également. Cela illustre le fait que lorsqu'il y a peu de données échangées, des problèmes de transport, comme la perte d'un ou plusieurs paquets, peut influencer le temps d'exécution.

4.5.2. La taille des données échangées :

Ces tests ont pour objectif de démontrer la supposition que nous avons faite dans les sections précédentes : le mode WSDL a un coût plus important que le mode REST.

La Figure 55 indique, pour chaque mode, la moyenne de 200 requêtes demandant 2 jours de prévisions avec 9 paramètres, réalisée également selon le protocole de test que nous avons définit.

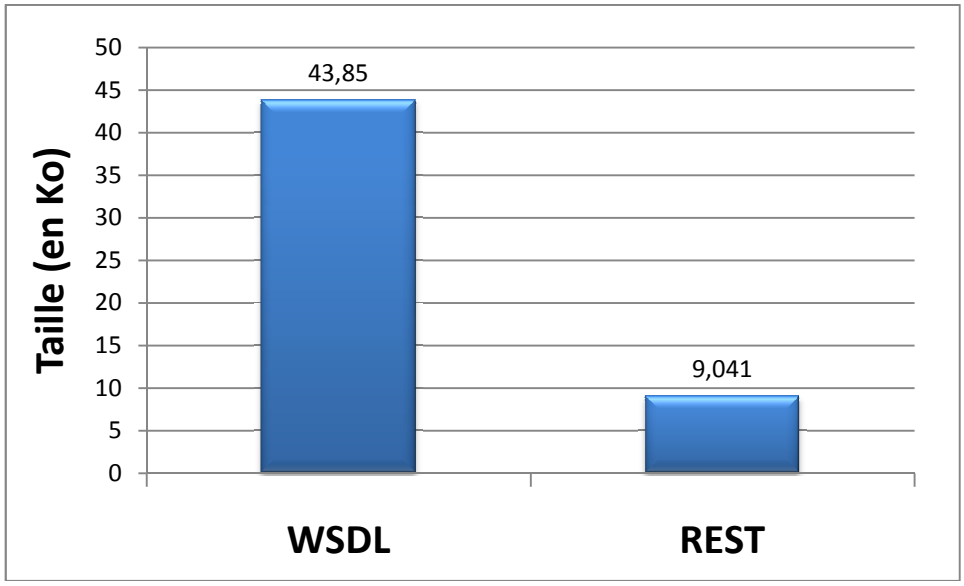

Figure 55 : Comparaison de la taille des données échangées

Grâce à ces tests, nous avons confirmé notre supposition : WSDL a un coût plus important, à cause du transfert du fichier WSDL.

Logiquement, nous nous doutons que la taille totale des données va évoluer selon la quantité demandée. L'intérêt ici est de savoir si cette évolution est constante ou non. La Figure 56 ci-dessous montre l'évolution de la moyenne de 50 requêtes REST, en fonction du nombre de paramètres sélectionnés pour 6 jours de prévisions.

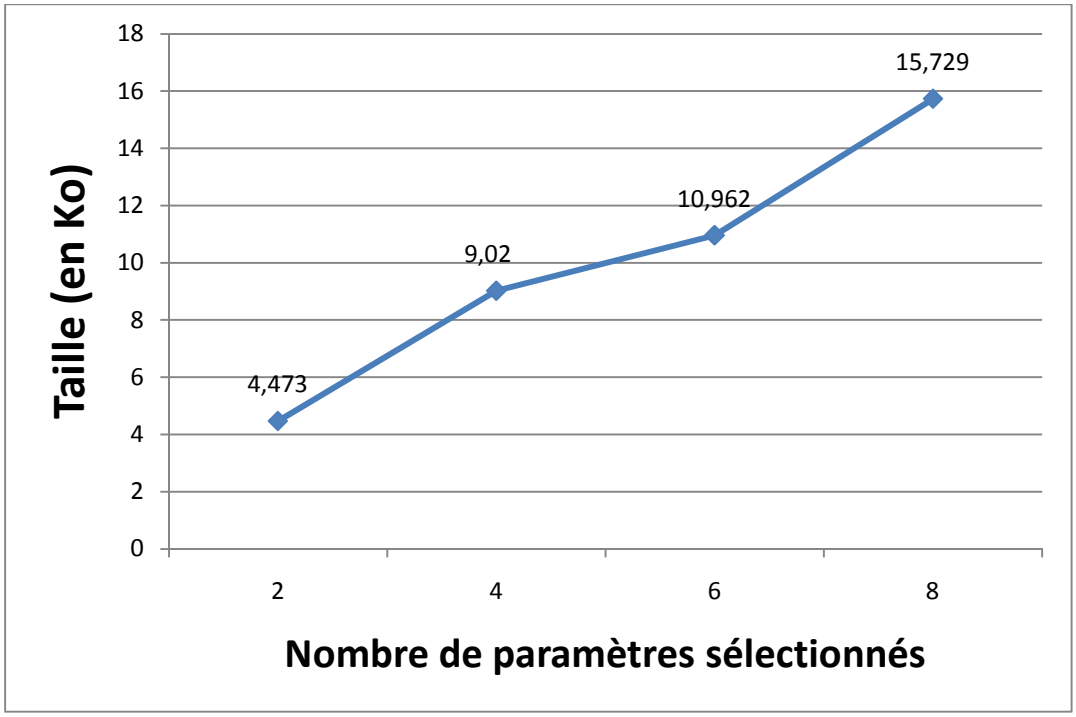

Figure 56 : Évolution de la taille des données

Nous avons pu constater, grâce à ces tests, que l'évolution dépend non seulement du nombre de données demandées, mais également du type de paramètres demandés. Certains paramètres alourdissent en effet plus d'autre, le fichier XML retourné par le web service, puisque plus d'informations sont renvoyées.

4.6. Conclusion

Nous venons d'exposer le fonctionnement pratique de notre solution via la mise en application de trois études de cas. Celles-ci nous ont permis d'évaluer, pour plusieurs types de clients, la création et l'adaptation du front-end propre aux deux web services sélectionnés, ainsi qu'à leur combinaison.

L'évaluation des tests de performance de notre solution a pu démontrer que le choix de REST était le plus optimal au point de vue temps et bande passante.

Chapitre 5 : Conclusion finale

Vous pourrez trouver dans ce chapitre une synthèse de l'ensemble des concepts et des conclusions en découlant, que nous avons pu développer dans les pages que vous venez de lire. Il vous sera ainsi possible de cerner l'entièreté du contenu général de notre mémoire.

5.1. Résumé des résultats obtenus

En faisant l'état de l'art dans le chapitre 2, nous avons pu définir ce qu'est un web service (section 2.1) utilisant deux modes de fonctionnement possibles, que sont WSDL et REST. Avec le standard « Delivery Context Ontology » (section 2.2) nous avons abordé le concept de contexte d'utilisation du client, mais aussi le manque de définition formelle du W3C au sujet de l'envoi de ce fichier DCO, ce qui nous a contraints à simuler l'envoi de celui-ci dans notre prototype. L'étude de quelques solutions existantes dans le monde de l'adaptation du web vers le mobile (section 2.3) nous a permis de relever certains concepts intéressants, qui ont alimenté notre réflexion au sujet de nos propres choix d'implémentation. Ces différents choix ont été décrits dans le chapitre 3, d'abord par le schéma général du fonctionnement de notre solution (section 3.1) qui a repris les différentes étapes théoriques, puis en détaillant les différentes parties qui la composent. Le cœur du Proxy (section 3.1.2) s'est révélé être composé d'un noyau, sur lequel peuvent être greffés de nouveaux modules, ceux-ci étant assortis sémantiquement à un ou plusieurs web services particuliers. Les règles d'adaptation nécessaires au bon déroulement du processus d'adaptation ont également été définies (section 3.4). L'adaptation réalisée à l'aide de ces règles a pu être visible dans le chapitre 4, où trois études de cas pratiques (section 4.1) ont pu révéler les avantages et inconvénients dû à l'utilisation de notre solution (sections 4.2 et 4.3). Elles nous ont également permis d'évaluer, pour plusieurs types de clients, la création et l'adaptation du front-end propre aux deux web services sélectionnés, ainsi qu'à leur combinaison, via les trois modules implémentés sur le Proxy. L'évaluation des tests de performance de notre solution a pu démontrer que le choix de REST était le plus optimal au point de vue temps et bande passante (section 4.5).

C'est de par sa particularité à intégrer le contexte d'utilisation client via son fichier DCO, de permettre l'ajout de nouveaux modules assortis sémantiquement à un ou plusieurs web services particuliers, d'alimenter la création d'une interface graphique par l'ajout de règles d'adaptation, que vous aurez remarqué l'aspect « inédit » de la solution que nous proposons ici pour aborder l'adaptation du front-end d'un web service. L'intérêt d'une telle solution est d'avoir proposé un prototype capable d'intégrer rapidement et efficacement le « Delivery Context Ontology ». Dés que l'envoi de celui-ci sera standardisé, cela devrait lui permettre d'atteindre une certaine notoriété et, par conséquent, de susciter l'intérêt des constructeurs des différents matériels. L'impact de notre solution ne pourra être réellement mesuré qu'à partir de ce moment là.

5.2. Analyse critique

L'évaluation des tests de performance de notre solution a pu démontrer que le choix du mode REST correspondait mieux que le mode WSDL à notre vision d'optimisation du temps et de la bande passante. Nous économisons bien du temps parce qu'il ne faut plus demander le fichier WSDL au web service avant de réaliser la requête, qui peut être formulée directement, et de la bande passante, vu que nous n'avons plus le coût de transfert du fichier WSDL, dont la taille peut varier selon les fonctions proposées par le web service. Ce choix porté vers REST vient également du fait de pouvoir directement choisir les ressources qui nous intéressent, contrairement à WSDL, où nous restons dépendants de la manière dont les fonctions prédéfinies accèdent aux ressources.

Contrairement aux solutions que nous avons relevées, notre solution ne travaille qu'avec des web services. Il n'est donc pas possible de traduire ou transformer n'importe quelle page d'un site web classique. Une solution est que ce site classique utilise derrière un web service, alors la création d'un module adéquat permettra l'adaptation.

5.2.1. Avantages et Inconvénients

Les **avantages** de notre solution sont multiples :

- une **optimisation « pré-requête »** de par la prise en charge du contexte d'utilisation dès la connexion du client, pour optimiser la requête créée et envoyée au web service ;
- un **fichier XML simplifié** qui permet de ne pas transférer l'entièreté du fichier DCO aux modules. Notre vision du contexte simplifié y côtoie une section « Hardware » contenant certaines informations provenant du fichier DCO ;
- le **concept du « CSS Generator »** qui produit un fichier CSS le plus concis possible ;
- **l'implémentation « coté serveur »** qui permet de minimiser les calculs effectués par les clients.

Certains **inconvénients** sont également à relever :

- La gestion de l'envoi de ce fichier n'étant pas encore abordée par le W3C, nous osons espérer qu'une solution pourra être dégagée concernant la taille et le transfert de celui-ci ;
- notre solution ne travaille qu'avec des web services. Il n'est donc pas possible de traduire ou transformer n'importe quelle page d'un site web classique ;
- notre solution ne propose pas de module générique s'adaptant à n'importe quel web service. Il est donc obligatoire de créer un module sémantiquement lié à chaque web service.

5.2.2. Problèmes rencontrés

Malgré le fait que nous n'avons rencontré que peu de problèmes difficiles à surmonter, nous pouvons pointer les plus importants :

- Le problème principal rencontré lors de ce projet fut sans contexte le concept du « Delivery Context Ontology ». De par sa récente standardisation, peu de personnes ne l'ont encore intégré et, de ce fait, nous n'avons pas trouvé beaucoup d'exemple d'utilisation.
- Il nous a fallu un certain temps avant de trouver le web service NDFD, de par les critères de recherche que nous nous sommes fixés :
  - informations météorologiques variées, pour travailler plusieurs paramètres ;
  - fonctionnement en mode REST et WSDL ;
  - documentation publique disponible ;
  - gratuit.
- Un changement inattendu et non indiqué des paramètres par le web service nous a fait perdre du temps et remettre en question notre propre implémentation.

5.3. Travaux futurs

Parmi les différentes fonctionnalités auxquelles nous avons réfléchies durant le développement du prototype, plusieurs d'entres elles n'ont pas été développées ou implémentées, par manque d'expérience au niveau du développement Web ou par manque de temps. Vous trouverez donc ci-dessous une série d'idées, de concepts qu'il serait encore possible d'intégrer pour améliorer la solution actuelle.

5.3.1. Que reste-t-il à faire ?

En supposant la solution actuelle, voici ce que nous aurions prévu de développer à court terme :

(1) implémenter un système de cache performant nous permettant de sauvegarder des informations afin de n'effectuer par la suite que des requêtes concernant de nouvelles informations ;

(2) implémenter une gestion des espaces utilisateurs plus approfondie. Ainsi, nous pourrions sauvegarder les préférences de chaque client, réaliser des statistiques ou encore, avoir un historique des précédentes requêtes déjà formulées ;

(3) avec la fonctionnalité ci-dessus, nous pouvons également envisager la possibilité d'une automatisation totale où le client n'aurait qu'à s'identifier et choisir son module ;

(4) afficher, dans le module « Weather », la couverture nuageuse en mode graphique sous forme d'un nuage plus ou moins transparent selon la prévision ;

(5) travailler la réutilisabilité du code afin de le rendre le plus générique possible ;

(6) améliorer la gestion des messages d'erreurs.

5.3.2. Pour aller plus loin

Il est question ici de fonctionnalités plus avancées nécessitant une adaptation plus importante de la solution actuelle :

(7) la création d'un méta langage permettant de définir des règles d'adaptation, de telle sorte que les développeurs de modules puissent en écrire eux-mêmes dans un format identique pour tous ;

(8) optimiser la solution actuelle par l'ajout de threads s'exécutant en parallèle afin de gagner du temps d'exécution, notamment pour la gestion du DCO ;

(9) permettre aux développeurs de module d'utiliser le langage de programmation qu'ils souhaitent ;

(10) mettre en place un système de feed-back des utilisateurs afin que notre solution s'adapte encore mieux à leurs besoins ;

(11) à la vue de l'évolution matérielle rapide des Smartphones et des tablettes, nous pouvons imaginer de transférer plus d'activité du « coté client » afin de soulager le Proxy ;

(12) transférer l'activité du « coté client » lorsque la plateforme le permet et ainsi réserver les calculs « coté serveur » uniquement pour les appareils qui en ont le plus besoin ;

(13) fournir un panneau d'administration graphique efficace pour les développeurs de module débutants afin qu'ils puissent créer un module-type facilement ;

(14) fournir, pour les développeurs de modules contraints d'utiliser le mode WSDL, un outil qui génèrerait une interface permettant de réaliser des requêtes vers le web service, d'après son fichier WSDL.

Vous avez pu constater que parmi ces fonctionnalités, beaucoup se destinent à faciliter l'utilisation de notre solution. Ceci a pour but de permettre sa prise en main rapide et ainsi la populariser plus rapidement.

5.3.3. Pour élargir la question

Nous énonçons ici les différents champs d'investigation découverts lors de nos recherches et qui nous semblent intéressants à approfondir dans des projets futurs :

(15) création d'un méta-modèle de règles d'adaptation facilitant l'implémentation de l'adaptation grâce à une génération automatique du code en fonction de ces règles [GAN 2007] ;

(16) ajouter d'autres manières d'accéder au contexte d'utilisation, par exemple, JavaScript ;

(17) afin de rendre l'adaptation plus dynamique, l'utilisation d'une architecture orientée événement pourrait être une solution.

D'autres choses pourraient encore être développées, mais si nous devions classifier ces étapes supplémentaires selon leur importance et/ou leur faisabilité, cela ressemblerait à la Figure 57.

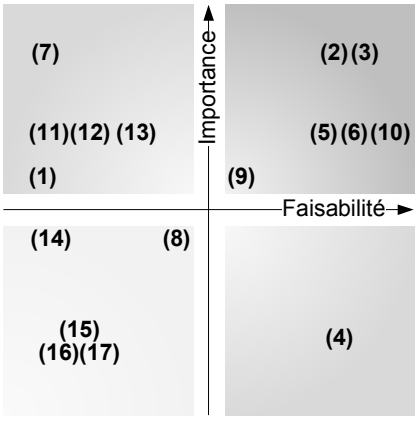

Figure 57 : Importance et faisabilité des étapes futures

Références bibliographiques

[ANT 2011] ANT RESEARCH, *Effective solutions for media marketing intelligence*. Bruxelles. Disponible sur: <http://www.antresearch.com/>(Consulté le 16.05.11).

[BIC 1999] BICKMORE T. & GIRGENSOHN A. & SULLIVAN J. W., Web Page Filtering and Re-Authoring for Mobile Users, received August 18, 1998; revised April 21, 1999.

[BRA 2008] BRAY T., Textuality and Netscape & PAOLI J., Microsoft & SPERBERG-MCQUEEN C. M., W3C & MALER E., Sun Microsystems & YERGEAU F., *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C : Genève. Disponible sur : < http://www.w3.org/TR/xml/> (Consulté le 17.05.11).

[CAN 2009] CANTERA FONSECA J., Telefónica & LEWIS R., Volantis Systems Ltd, *Delivery Context Ontology*. W3C : Genève. Disponible sur : < http://www.w3.org/TR/2009/WD-dcontology-20090616/> (Consulté le 12.05.11).

[CHR 2001] CHRISTENSEN E., Microsoft & CURBERA F., IBM Research & MEREDITH G., Microsoft & WEERAWARANA S., IBM Research, *Web Services Description Language (WSDL) 1.1*. W3C : Genève.

Disponible sur : < http://www.w3.org/TR/wsdl> (Consulté le 11.05.11).

[CLE 2004] CLEMENT L., Systinet & HATELY A., IBM & VON RIEGEN C., SAP AG & ROGERS T., Computer Associates, *UDDI Spec TC*, OASIS. Disponible sur : < http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm> (Consulté le 11.05.11).

[FIE 2000] FIELDING R., *Architectural Styles and the Design of Network-based Software Architectures*, Thèse : University of California, Irvine, 2000, 162 p.

[FIG 2006] FIGER J-C., *REST, un style d'architecture universel. "pluralitas non est ponenda sine necessitate"*. Disponible sur : <http://www.figer.com/publications/REST.htm> (Consulté le 11.05.11).

[GAN 2007] GANNEAU V. & CALVARY G. & DEMUMIEUX R., Métamodèle de Règles d'Adaptation pour la Plasticité des Interfaces Homme-Machine, in Actes de la 19ème Conférence francophone sur l'Interaction Homme-Machine, Paris, 2007, pp. 91-98.

[GIM 2006] GIMSON R., HP & LEWIS R., Volantis Systems Ltd. & SATHISH S., Nokia, *Delivery Context Overview for Device Independence*. W3C : Genève. Disponible sur: <http://www.w3.org/TR/di-dco/> (Consulté le 16.05.11).

[GOM 2011] GOMOBI. Sl. Disponible sur: <http://gomobi.info/> (Consulté le 16.05.11).

[GUD 2007] GUDGIN M., Microsoft & HADLEY M., Sun Microsystems & MENDELSOHN N., IBM & MOREAU J-J., Canon & NIELSEN H., Microsoft & KARMARKAR A., Oracle & LAFON Y., W3C, *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. W3C : Genève. Disponible sur : < http://www.w3.org/TR/soap12-part1/> (Consulté le 11.05.11).

[HIL 2003] HILBERT D. & TREVOR J. & SCHILIT B., Supporting ubiquitous information on very small devices is harder than you think, HCI International 2003, June 22.

[HIL 2011] HILBERT D., *M-Links*. Sl. Disponible sur: <http://www.fxpal.com/?p=mlinks> (Consulté le 18.05.11).

[JER 2009] JEROEN, *UAProf device profiles*. Sl. Disponible sur: <http://www.w3.org/2004/01/ccpp-pressrelease.html.fr> (Consulté le 16.05.11).

[KLY 2004] KLYNE G., Nine by Nine & REYNOLDS F., Nokia Research Center & WOODROW C., Information Architects & OHTO H., W3C/Panasonic & HJELM J., Ericsson & BUTLER M. H., Hewlett-Packard & TRAN L., Sun Microsystems, *Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0*. W3C : Genève. Disponible sur: <http://www.w3.org/TR/CCPP-struct-vocab/> (Consulté le 16.05.11).

[MEE 2010] MEEKER M. & DEVITT S. & WU L., Morgan Stanley, *Internet Trends*. Sl. 2010, 87 p.

[MOB 2011] MOBIREADY. Sl. Disponible sur:<http://ready.mobi/> (Consulté le 16.05.11).

[MOW 2011a] MOWSER, *Mowser mobilizing the web*. Sl. Disponible sur: <http://mowser.com/> (Consulté le 16.05.11).

[MOW 2011b] MOWSER, *Mowser Adaptation Steps*. Sl. Disponible sur: <http://mobiforge.com/developing/page/mowser-adaption-steps> (Consulté le 16.05.11).

[NAT 2011] NATIONAL WEATHER SERVICE, *National Digital Forecast Database*. Silver Spring. Disponible sur: <http://www.weather.gov/ndfd/> (Consulté le 16.05.11).

[OPE 2011] OPERA SOFTWARE. Oslo. Disponible sur:<http://www.opera.com/mobile/features/> (Consulté le 16.05.11).

[PAT 2010] PATERNÒ F. & ZICHITTELLA G., Desktop-to-Mobile Web Adaptation through Parametric Bidimensional Semantic Redesign, in BERNHAUPT R. & FORBRIG P. & GULLIKSEN J. & KRISTÍN LÁRUSDÓTTIR M., Human-Centred Software Engineering, Third International Conference, HCSE 2010 Proceedings. Lecture Notes in Computer Science, 2010, Volume 6409, Springer-Verlag, Berlin, 2010, pp. 79-94.

[PHP 2009] PHP-HTML.net, *Model View Controller(MVC) in PHP*. Sl. Disponible sur : <http://php-html.net/tutorials/model-view-controller-in-php/> (Consulté le 16.05.11).

[RIC 2007] RICHARDSON L. & RUBY S., RESTful Web Services. Première édition. Sebastopol, Californie : O'Reilly Media, Mai 2007, 419 p.

[ROD 2008] RODRIGUEZ A. - IBM, *RESTful Web services: The basics*. Disponible sur : <https://www.ibm.com/developerworks/webservices/library/ws-restful/> (Consulté le 11.05.11).

[SCH 2001] SCHILIT B. & TREVOR J. & HILBERT D. & KOH T., m-Links: An Infrastructure for Very Small Internet Devices, in The 7th Annual International Conference on Mobile Computing and Networking (MOBICOM 2001), Rome, Italy, ACM Press, July 16-21 2001, pp. 122-131.

[SKW 2011] SKWEEZER, *Find what you're looking for faster & easier with Skweezer*. Sl. Disponible sur: <http://company.skweezer.com/> (Consulté le 16.05.11).

[STR 2011] STROUSTRUP B., *Welcome to Bjarne Stroustrup's homepage!*.Sl. Disponible sur:<http://www2.research.att.com/~bs/homepage.html> (Consulté le 16.05.11).

[TEA 2011] TEASHARK, *TeaShark-BETA because mobile web is dead*. Sl. Disponible sur: <http://www.navire.fi/teashark/index.html> (Consulté le 16.05.11).

[TIM 2009] TIMMERER C. & JABORNIG J. & HELLWAGNER H., *Delivery Context Descriptions - A Comparison and Mapping Model*, Department of Information Technology (ITEC) Klagenfurt

University, Klagenfurt, 2009, 18 p.

[VAN 2004] VAN DER VLIST E., *Le triptyque SOAP/WSDL/UDDI*. Sl., Juin 2004, 11 p.

[W3C 2004] W3C, *Le W3C publie la Recommandation CC/PP 1.0*. W3C : Genève. Disponible sur: <http://www.w3.org/2004/01/ccpp-pressrelease.html.fr> (Consulté le 16.05.11).

[W3S 2011] W3SCHOOLS, *Introduction to WSDL*. Sl. Disponible sur: <http://www.w3schools.com/wsdl/wsdl_intro.asp> (Consulté le 11.05.11).

[W3S 2011a] W3SCHOOLS, *Introduction to WSDL*. Sl. Disponible sur: <http://www.w3schools.com/wsdl/wsdl_intro.asp> (Consulté le 11.05.11).

[W3S 2011b] W3SCHOOLS, *SOAP Introduction*. Sl. Disponible sur: <http://www.w3schools.com/soap/soap_intro.asp> (Consulté le 11.05.11).

[WAL 2011] WALKAWARE, *Un outil de création et de gestion de circuits touristiques…* Gembloux. Disponible sur : < http://walkaware.com/>(Consulté le 16.05.11).

[WEB 2010] WEBBER, J. & PARASTATIDIS, S. & ROBINSON, I., REST in Practice: Hypermedia and Systems Architecture. Première édition. Sebastopol, Californie : O'Reilly Media, Septembre 2010, 428 p. ISBN: 978-0-596-80582-1

[WEB 2011] WEBOPEDIA, *Web services*. Sl. Disponible sur: <http://www.webopedia.com/TERM/W/Web_services.html> (Consulté le 11.05.11).

[WIK 2011] WIKIPEDIA, *Representational State Transfer*. Sl. Disponible sur: <http://fr.wikipedia.org/wiki/Representational_State_Transfer> (Consulté le 11.05.11).

Tables des illustrations

Figures

Code

Tableaux

Fichiers

Code source