Serveur d'Applications Web : WebObjects

Nicolas Roard

29 avril 2004

Table des matières

1 Introduction

Les "serveurs d'applications web" sont de plus en plus courants et diverses solutions existent, aussi bien du côté des logiciels libres que propriétaires.
Parmis les solutions les plus connues, on peut citer en particulier ZOPE, un serveur d'application libre en python, et également les possibilitées offertes par Java (J2EE, Struts...).
Il y a cependant une solution de serveur d'application objet qui existe depuis le début des années 90, ayant fait ses preuves sur des sites web importants, très apprécié par sa communauté d'utilisateur, et pourtant presque inconnue : WebObjects.
La raison de cette méconnaissance tient pour beaucoup à un faible marketing, et surtout, à un prix prohibitif jusqu'il y a peu de temps (quelques dizaines de milliers de dollars – le prix actuel est désormais de 700$). Ah ! mais il s'agit d'une solution propriétaire ! pourquoi en parler dans LinuxMag ? pour deux raisons :

– d'une part, WebObjects venant de NeXT (maintenant Apple), son design est élégant et intéressant à découvrir
– d'autre part... il existe aujourd'hui des solutions libres qui implémentent WebObjects !

2 Historique

Le web a commencé par de simples pages HTML statiques (ne changeant pas à l'exécution). De telles pages peuvent sembler suffisantes pour de la présentation de documentation, ou pour un site modifié peu souvent. Mais dès que l'on doit fournir un contenu "volatile" (météo, nouvelles, etc.), on veut pouvoir générer à la volée les pages HTML, par exemple en se servant de contenu stocké en base de données (SGBDR, Systèmes de Gestion de Base de Données Relationnelles), ou de contenu calculé en temps réel.
Les premières tentatives de génération de pages dynamiques utilisaient tout simplement des langages de programmation (Perl, C ...) en créant des programmes appellés par le serveur web et retournant du HTML : des CGI[1] (voir Fig. 1).

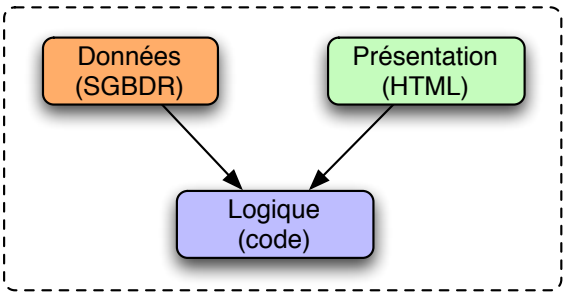

FIG. 1 – Programmes CGI

[1] CGI : Common Gateway Interface, un protocole simple pour appeller un programme et récupérer le résultat

Bien que permettant d'avoir des pages dynamiques, cette solution a plusieurs problèmes :

– premièrement, un coût d'utilisation assez important (à chaque appel d'une page, un processus est lancé sur le serveur pour créer la page !)
– deuxièmement, et plus gênant encore, tout est mélangé : la partie logique (le code), la partie présentation (le HTML) et la partie données (requêtes à la base de donnée)

Rapidement, les créateurs de sites web se sont rendus compte que la plus grande partie d'un site web, même dynamique, était du HTML, pas du code. La création de langages tels que PHP ou ASP, qui inversent le problème par rapport aux CGI (on n'utilise plus un programme qui génère du HTML, mais du HTML qui contient des petits bouts de code et que l'on fait passer dans un interpréteur) a alors permis de démocratiser et de faciliter la création de sites web dynamiques (Fig. 2).

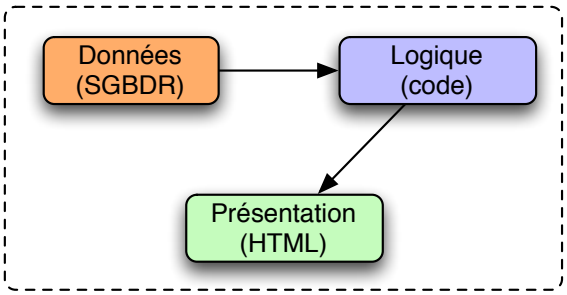

FIG. 2 – Pages Dynamiques (PHP, ASP…)

Bien que plus efficace généralement que des CGI (entre autre, le moteur de script – l'interpréteur – est souvent "intégré" directement au serveur web de façon à éviter de relancer à chaque fois un programme), et d'accès bien plus facile pour des non-programmeurs, tout est toujours mixé dans la page (données, logique, présentation).

D'où l'étape suivante (Fig. 3), qui consiste à séparer la logique (le code) de la représentation (la page HTML) :

Cela a l'avantage de permettre une meilleure réutilisation du code, et surtout d'être indépendant de la présentation – un graphiste pourra alors à loisir changer le design du site web en étant sûr de ne pas "casser" le site par ailleurs.

Avec un peu d'effort il est d'ailleurs possible de réaliser ce genre de séparation en ASP/JSP/PHP.

Cependant, on est toujours dépendant, au niveau des données, de la base choisie (ce qui peut éventuellement être contourné en utilisant une librairie d'abstraction d'accès aux SGBDR), et surtout, à la façon dont les

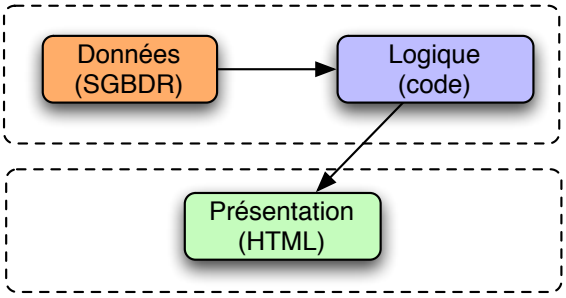

FIG. 3 – Séparation de la logique et de la représentation

données sont organisés dans la base. Impossible de modifier le schéma de la base sans devoir revoir tout le code. WebObjects permet au contraire une séparation claire de ces trois niveaux (Fig. 4), en utilisant d'une part des composants (permettant de séparer clairement la logique et la présentation dans des templates HTML), et d'autre part en utilisant un framework d'accès aux bases de données, EOF (Enterprise Objects Framework).

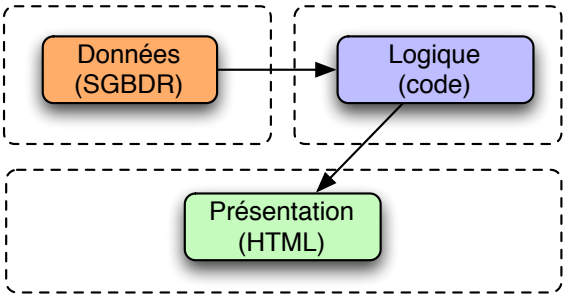

FIG. 4 – Séparation sur trois niveaux (WebObjects

EOF permet d'être indépendant de la base de donnée, mais surtout "mappe" des objets sur le schéma de la base de donnée – du point de vue du programmeur, on n'écrit pas de SQL, on se contente de manipuler des objets de façon tout à fait normale. Le schéma de la base peut alors changer, il suffit de modifier le schéma EOF décrivant la relation entre les objets et les tables : inutile de modifier le code de votre application.

2.1 Implémentation WebObjects

À l'heure actuelle, il existe deux implémentations libres de WebObjects 4.5, en plus de celle, payante, d'Apple :

– GNUstepWeb, projet GNU disponible sur le cvs du projet GNUstep, en LGPL
– NGObjWeb, récemment “libéré” par SKYRIX, disponible également en LGPL

Les deux sont finalement assez proches, bien que NGObjWeb semble un peu plus rôdé (il s’agit du moteur WebObjects utilisé sur le projet OpenGroupware.org). OpenGroupware.org utilise également SOPE, qui propose des extensions pour NGObjWeb similaires dans l’esprit à l’approche de ZOPE.

GNUstepWeb offre quelques avantages, comme la gestion du load-balancing (existant mais non publié sur NGObjWeb).

Il reste à voir si ces deux implémentations vont converger ou non – NGObjWeb/OpenGroupware.org étant en train de passer à GNUstep au lieu de libFoundation par exemple.

Quelques discussions (houleuses) ont eu lieu entre les développeurs (GNUstepWeb a été lancé en partie suite au refus de Skyrix de libérer leur implémentation il y a quelques années) – espérons malgré tout que cela aboutira à moyen terme à une implémentation libre de WebObjects qui tire avantage des deux implémentations actuelles.

Néanmoins, les deux sont tout à fait utilisables actuellement, et fonctionnent avec Apache (version 1 ou 2). GNUstepWeb a par ailleurs récemment intégré le support de la syntaxe de tags utilisé par NGObjWeb, qui lui même a intégré le Key-Value Coding. Il est ainsi facile de passer d’un serveur à un autre.

Une implémentation d’EOF 4.5 existe également : GDL2 (GNUstep Database Layer 2) et est disponible sur le cvs de GNUstep.

Il est à noter qu’aussi bien GNUstepWeb que NGObjWeb utilisent Objective-C comme langage de programmation des composants, alors qu’Apple ne propose plus que Java depuis la version 5 de WebObjects (qui est du coup devenu une plateforme J2EE). L’intérêt pour Apple d’utiliser Java était à priori d’abord une question de marketing (buzzwords et bien sûr le fait qu’il y a plus de développeurs Java qu’Objective-C !), mais en terme de performances/-capacitées, Objective-C reste un bon choix : puissant et simple. GNUstep et Apache compilant sur la majorité des plateformes Unix, et sous Linux/OSX/Windows, la portabilité est également là.

Je n’ai pas traité dans cet article de l’installation de NGObjWeb ou GNUstepWeb – la documentation est disponible (voir les liens en fin d’article) et des listes de diffusions existent. . .

Les sources des exemples de cet article sont disponibles sur `http://www.roard.com/lmf/webobjects`.

2.2 Différences et manques ?

Le principal manque par rapport à la version 4.5 d’Apple (utilisée comme référence par GNUstepWeb et NGObjWeb) reste l’absence d’un équivalent à WebScript, un langage de script qui permettait de créer des composants facilement, sans avoir à passer par de la programmation Objective-C ou Java, et ne nécessitant pas de recompilation de l’application pour prendre en compte les modifications.
Bien sûr, les composants complexes ayant besoin de rapidité avaient la possibilité d’être programmé en Objective-C ou Java, mais les petits composants utilisés par exemple pour le côté “interface” (composants HTML) étaient idéalement écrit en WebScript.
À l’heure actuelle, il existe cependant un excellent framework pour GNUstep, StepTalk, qui permet facilement d’ajouter des fonctions de scriptage dans les applications GNUstep. StepTalk est architecturé pour supporter différents langages de script, et implémente pour le moment SmallTalk et Guile (d’autres langages sont prévus). Aussi bien GNUstepWeb que NGObjWeb envisagent donc d’utiliser StepTalk pour implémenter des composants scriptables. On peut noter également que SOPE permet de réaliser des composants en Python et JavaScript (mais on s’éloigne un peu de WebObjects).
L’autre point important de WebObjects (version NeXT/Apple) est bien sûr la disponibilité d’outils graphiques permettant de créer très simplement des pages web utilisant des tags WebObjects et de les lier aux composants. Le framework EOF utilisé pour l’accès au données dispose également d’outils graphiques très intéressants permettant simplement de créer des connexions aux bases de données, de rétro-modéliser ou synchroniser une base, etc.
Malheureusement, il n’existe pas (encore) de tels outils libres... toutefois, l’utilisation de GNUstepWeb ou NGObjWeb n’est pas spécialement compliqué (grâce entre autre aux makefiles GNUstep, qui font une bonne partie du travail à notre place). Par ailleurs, un éditeur graphique de schémas EOF est en train d’être réalisé.

3 Principes de fonctionnement

Comme la plupart des solutions de web dynamique, WebObjects fonctionne en interceptant les requêtes HTTP.
Un “adaptateur WebObjects” pour Apache se charge ainsi de rediriger les requêtes vers les applications WebObjects du système.
La Fig. 5 page suivante montre ainsi que la page appellée corresponds à un composant. Quand celui-ci est invoqué,

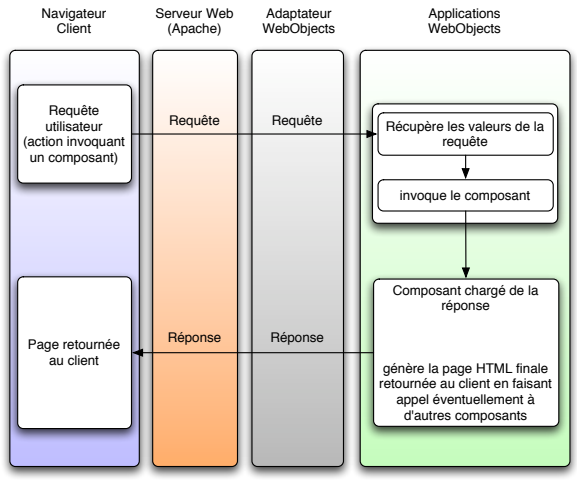

FIG. 5 – La boucle de requête-réponse de WebObjects

il crée les différents composants que lui-même utilise. Pour chacun des composants, la méthode :

```
takeValuesFromRequest:
        (GSWRequest*)aRequest
 inContext:(GSWContext*)aContext
```

est appellée ; chaque composant a ainsi la possibilité d'utiliser les informations contenues dans la requête. Ensuite, la méthode :

```
invokeActionForRequest:
        (GSWRequest*)aRequest
 inContext:(GSWContext*)aContext
```

est appellée pour chacun des composants, jusqu'à ce qu'un d'eux retournent une page complète. Le composant retournant la page complète envoie alors à tous les composants de la page le message :

```
appendToResponse:(GSWResponse*)aResponse
      inContext:(GSWContext*)aContext
```

À la réception de ce message, les composants peuvent ajouter du code HTML à la réponse. Enfin, la page HTML ainsi construite est retournée à l'utilisateur.

On voit donc que l'une des caractéristiques importantes de WebObjects est l'utilisation intensive de composants ; chaque page HTML retournée est en fait un ensemble de composants web. Grâce à ce système, il est alors extrêmement simple de réutiliser des composants (dans l'application bien sûr, mais éventuellement entre applications WebObjects – il suffit de mettre ses composants dans un framework), de gérer la traduction du site, et bien sûr de séparer proprement le fond et la forme du site web.

Chaque application WebObjects est elle même un ensemble de composants et d'objets.

Les composants peuvent être mis en câche automatiquement de façon à accélérer le traitement des requêtes.

La connexion à une base de donnée peut se faire par différentes librairies ; WebObjects utilise normalement Enterprise Objects Framework (EOF), qui permet de "mapper" des objets sur des bases de données classiques et de manipuler ensuite les objets de façon classique. GNUstepWeb propose ainsi GDL2, une implémentation libre d'EOF 4.5. On peut également préférer utiliser une librairie plus classique d'abstraction aux bases de données comme SQLClient, disponible sur le cvs GNUstep. EOF est cependant particulièrement intéressant et puissant…

Un point très intéressant pour le programmeur est donc la possibilité d'utiliser directement n'importe quelle librairie Objective-C ou C (Objective-C étant un sur-ensemble du C ansi) dans son application WebObjects.

Rien de bien compliqué de faire par exemple une application web complètement répartie selon des critères particuliers en utilisant les Distributed Objects (objets répartis) de GNUstep…ou des traitements sur des flux XML en utilisant une libraire telle que skyrix-xml.

Avec le futur gcc 3.5, on devrait également avoir accès directement aux librairies C++ en Objective-C (ce qui est déja possible avec la version actuelle du gcc d'Apple).

3.1 Les composants WebObjects

Les composants doivent donc implémenter les trois méthodes (takeValuesFromRequest :inContext, invokeActionForRequest :inContext, appendToResponse :inContext) décrites précédemment.

Cependant, la classe WOComponent propose une implémentation par défaut de ces trois méthodes, ce qui simplifie grandement la création de composants usuels !

Chaque composant est ainsi divisé en trois parties, suivant le Design Pattern bien connu du "Modèle-Vue-Contrôleur" (MVC), correspondants aux fichiers suivants :

– un objet Objective-C définissant le comportement du composant : le Modèle
– un template HTML définissant l'affichage du composant : la Vue
– un fichiew "WOD" liant la vue et le modèle : le Contrôleur

La Fig. 6 page suivante montre que le template HTML et le contrôleur (fichier WOD) sont rangés dans un répertoire portant le nom du composant et ayant l'extension "WO".

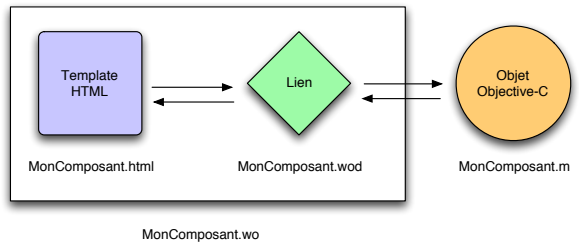

FIG. 6 – Structure d'un composant WebObjects

Le template HTML est en fait un fichier HTML (voire XHTML, donc pensez à fermer vos tags...) qui contient en plus de tags HTML classiques des tags WebObjects. De plus, le template peut ne contenir qu'un "morceau" de HTML, pas forcément un fichier HTML en entier – ce qui permet justement la création modulaire de pages web.
La syntaxe classique d'un tag WebObjects est normalement "<webobject name="nom"></webobject>", mais peut changer suivant le moteur ; NGObjWeb utilise ainsi la syntaxe "<#nom>" et GNUstepWeb accepte les deux précédentes plus "<gsweb name="nom"></gsweb>".
Le fichier WOD permet de lier les attributs des composants WebObjects utilisés à des données membres de l'objet ou de leur affecter une valeur.
En résumé, l'objet servant de modèle n'a donc pas à implémenter les méthodes décrite dans la section précédente, à moins de besoins très particuliers. L'implémentation standard se charge ainsi du traitement de la requête, de la lecture du template HTML et du WOD pour déterminer que renvoyer.

4 Composants HTML

Imaginons que l'on veuille un composant retournant simplement un texte. Il s'agit alors d'un composant simple, qui n'a besoin que d'une partie HTML. On aura alors un seul fichier, contenant le template HTML, "Personne.html" :

```
Prenom : Jean-Pierre<br/>
Nom : Liegeois<br/>
Activite : jeune lecteur du Var
```

Maintenant, on peut aussi vouloir que ce composant soit un minimum paramétré. Pour cela, il nous suffit de déclarer dans le template des tags WebObjects :

```
Prenom : <#prenom/><br/>
Nom : <#nom/><br/>
Activite : <#activite/>
```

Ces tags peuvent avoir n'importe quel nom. Il nous faut maintenant mettre dans le fichier "Personne.wod" à quels composants WebObjects ces tags correspondent. Le fichier "Personne.wod" contiendra ici :

```
prenom : WOString { value = "Jean-Paul" }
nom : WOString { value = "Liégeois" }
activite : WOString {
      value = "jeune lecteur du Var" }
```

Chaque tag est lié à un type de composant suivant la syntaxe :

```
tag : composant { attribut1 = valeur1;
    attribut2 = valeur2; ... }
```

Ici, on utilise uniquement des composants WOString, qui renvoient simplement la chaîne de caractère définie dans leur attribut "value".
Notre composant retournera donc le HTML suivant :

```
Prenom : Jean-Paul<br/>
Nom : Liegeois<br/>
Activite : jeune lecteur du Var
```

Il est bien sûr possible de référencer des composants fait-maison en plus des composants WebObjects ; on pourrait ainsi avoir un composant "Lecteurs" qui soit fait de la façon suivante :

```
<h1>Liste des Lecteurs</h1>
<personne/>
```

pour le template HTML "Lecteurs.html", et

```
personne: Personne {}
```

pour le fichier WOD "Lecteurs.wod". Si on appelle ce nouveau composant, il renverra alors :

```
<h1>Liste des lecteurs</h1>
Prenom : Jean-Paul<br/>
Nom : Liegeois<br/>
Activite : jeune lecteur du Var
```

4.1 Tags Dynamiques

WebObjects propose un ensemble de tags dynamiques permettant de manipuler des éléments HTML classiques (formulaires, boutons, liens, ...), des données, etc.
Un exemple de composant dynamique intéressant est WOContentComponent. Un tag peut en effet indiquer l'utilisation d'un composant à un endroit, mais peut

également encadrer du contenu (balises ouvrantes et fermantes). WOContentComponent symbolise le contenu du composant actuel (Fig. 7).

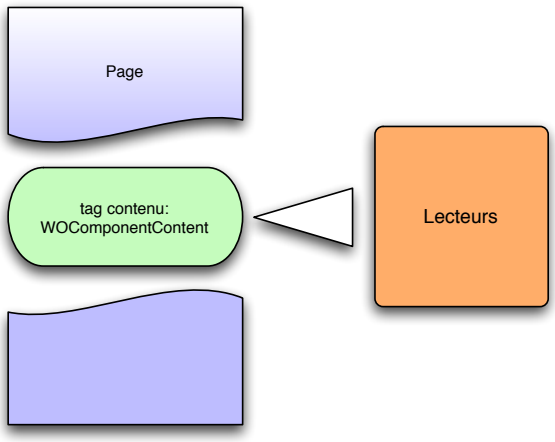

FIG. 7 – Utilisation de WOComponentContent

On peut alors facilement créer un composant “Page” chargé de définir une en-tête et un pied de page ; le template HTML “Page.html” contiendra :

```
<html>
<head>
      <title>Exemple WebObjects</title>
</head>
<body>
      <#contenu/>
</body>
</html>
```

Et on définit dans le fichier WOD le tag “contenu” comme étant un WOComponentContent :

```
contenu : WOComponentContent {}
```

Si l’on veut maintenant que notre composant “Lecteurs” utilise le composant “Page”, il nous faut modifier légèrement le template HTML et le WOD de “Lecteurs” pour englober le texte dans un tag :

```
<#page>
      <h1>Liste des Lecteurs</h1>
      <personne/>
</#page>
```

et définir dans le fichier “Lecteurs.wod” le tag “page” comme un composant “Page” :

```
page : Page {}
personne: Personne {}
```

Si l’on appelle maintenant notre composant “Lecteurs”, on recevra en retour :

```
<html>
<head>
      <title>Exemple WebObjects</title>
</head>
<body>
      <h1>Liste des Lecteurs</h1>
      Prenom : Jean-Paul<br/>
      Nom : Liegeois<br/>
      Activite : jeune lecteur du Var
</body>
</html>
```

5 Composants dynamiques : utilisation des objets

Les exemples précédents restent encore au niveau d’une gestion sophistiquée de templates ; WebObjects permet beaucoup plus que ça !
Par exemple, notre exemple de composant “Personne” paramétré est intéressant, mais n’est paramétré que globalement (et statiquement) ; impossible d’appeller plusieurs fois ce composant avec des valeurs différentes.
Pour cela, nous devons utiliser un objet, qui lui pourra contenir des valeurs dynamiques.
Déclarons donc un objet “Personne.m” chargé de contenir les valeurs utilisées dans notre composant (on se contente de déclarer un objet ayant trois données membres) :

```
@interface Personne : WOComponent
{
      NSString* nom;
      NSString* prenom;
      NSString* activite;
}
@end

@implementation Personne
@end
```

Et modifions notre fichier WOD pour utiliser ces données membres :

```
prenom : WOString { value = prenom }
nom : WOString { value = nom }
activite : WOString { value = activite }
```

Nous pouvons désormais modifier le fichier WOD de notre composant “Lecteurs” pour spécifier les attributs du composant “Personne” :

```
page : Page {}
personne: Personne {
      nom = ”Liégeois”;
      prenom = ”Jean−Pierre”;
      activite = ”jeune␣lecteur␣du␣Var”;
}
```

5.1 Key-Value Coding

Comment WebObjects arrive-t-il à accéder aux données membres alors qu’aucun accesseur n’est défini dans notre objet ? en fait, NSObject – la classe racine de tous nos objets – dispose d’un mécanisme appellé “Key-Value Coding” ou KVC, qui utilise les propriétées d’introspection d’Objective-C pour accéder automatiquement aux données membres. Mieux, rien ne vous empêche de définir vos propres accesseurs, et dans ce cas ce seront eux qui seront appellés !
Les accesseurs doivent juste suivre une notation usuelle pour leurs noms, par exemple “nomVariable” (pour le get) et “setNomVariable” (pour le set).
Gardons cependant à l’esprit que le système de KVC induit un léger surcoût à l’exécution, ce qui peut éventuellement être problématique suivant les cas (boucles énormes ?) – dans ce cas, définissez votre propre accesseur.
On voit ainsi que l’on peut très simplement définir des composants dynamiques réutilisables.

5.2 Présentation des données

Un problème classique dans un site dynamique est de vouloir afficher une liste ; en effet, très souvent cela aboutit à devoir mixer la présentation du site et la logique, in incorporant des morceaux de HTML dans le code.
WebObjects permet élégamment de résoudre ce problème en disposant d’un composant WORepetition. Ce composant prends deux paramètres :
– une liste
– un item
Il va ensuite répéter son contenu autant de fois qu’il y a d’éléments dans la liste fournie, et permettre d’accèder à travers la variable item fournie à l’élément courant de la liste.
Définissons un composant qui retournera une page web affichant une liste de liens clickables. Le fichier template HTML “Liens.html” contiendra le texte suivant :

```
page : Page {}
<#page>
Choisissez un lien :
<br/><br/>
<ul>
<#repetition>
      <li><#lien><#nom/></#lien></li>
</#repetition>
</ul>
</#page>
```

Le fichier WOD correspondant, “Liens.wod” nous permet de définir les attributs et les composants utilisés :

```
page : Page {}
repetition: WORepetition {
      list = listeLiens;
      item = lienCourant;
}

lien : WOHyperlink {
      action = lienCourant;
}

nom : WOString { value = lienCourant }
```

On a ici un composant Page (créé précédemment), une chaîne de caractère (prenant la valeur “lienCourant”), un composant répétition et un lien qui pointant sur la chaîne “lienCourant”.
Le code du composant chargé d’implémenter le modèle devra donc fournir deux données membres, un tableau de chaînes de caractères (listeLiens) et une variable destinée à contenir la chaîne courante (lienCourant).
Notre objet “Liens.m” sera donc :

```
@interface Liens : WOComponent
{
  NSArray* listeLiens;
  NSString* lienCourant;
}
@end

@implementation Liens
− (id) init {
  if (self = [super init])
  {
    listeLiens = [[NSArray alloc]
      initWithObjects:
        @”http :// www.gnustep.org”,
        @”http :// www.gnustepweb.org”,
        @”http :// www.opengroupware.org”,
```

```
    nil];
  }
  return self;
}
@end
```

Notre objet “Liens” définit bien deux données, listeLiens et lienCourant, respectivement un tableau et une chaîne de caractère. Dans l’implémentation de l’objet, on voit que dans le constructeur on initialise notre tableau “listeLiens” avec trois urls.
L’appel à notre composant “Liens” va ainsi retourner la page web suivante :

```
<html>
<head>
      <title>Exemple WebObjects</title>
</head>
<body>
Choisissez un lien :
<br/><br/>
<ul>
  <li>
      <a href="http://www.gnustep.org">
      http://www.gnustep.org
      </a>
  </li>
  <li>
      <a href="http://www.gnustepweb.org">
      http://www.gnustepweb.org
      </a>
  </li>
  <li>
      <a href="www.opengroupware.org">
      http://www.opengroupware.org
      </a>
  </li>
</ul>
</body>
</html>
```

5.3 Sessions et Applications

En plus de l’utilisation des composants et des objets, WebObjects supporte la notion de Session et d’Application. On peut ainsi définir ses propres objets de Session et d’Application – l’objet Session ayant la durée de vie d’une session utilisateur, l’objet Application celle de l’application WebObjects proprement dit. Il suffit pour cela de dériver un objet de WOSession et un autre de WOApplication.

Ces objets sont bien entendus appellables depuis d’autres composants en envoyant un message “session” ou “application” sur le composant :

```
WOSession* maSession = [self session];
WOApplication* monAppli = [self
    application];
```

6 Conclusion

WebObjects est un serveur d’application particulièrement puissant et néanmoins simple à mettre en oeuvre. Nous n’en avons égratigné qu’une très faible partie ! Il faudrait pouvoir parler par exemple d’EOF (GDL2 pour l’implémentation libre), des différents composants dynamiques...
Néanmoins, j’espère que ce tour d’horizon vous aura donné envie d’en savoir un peu plus :-)
Il est à noter que, comme tout développement basé sur le Design Pattern MVC, les premiers pas semblent un peu long par rapport à une solution rapide style PHP, mais cela est plus qu’amplement rattrapé dans la suite du développement.
WebObjects permet d’avoir un développement très propre, orienté objet, et a fait ses preuves... C’est à mon sens une solution originale qui mérite bien plus d’intérêt de la part des développeurs.
Nicolas Roard mail : nicolas@roard.com

7 Liens

`http://www.roard.com/lmf/webobjects`
(code source des exemples de l’article)

`http://developer.apple.com/documentation/LegacyTechnologies/WebObjects/WebObjects_4.5/webobjects.html`
(les docs apple sur WebObjects 4.5)

`http://www.gnustepweb.org`
`http://www.opengroupware.org/en/devs/sope/skyrix_sope/index.html`
(les sites de gnustepweb et ngobjweb)
`http://rentzsch.com/webobjects/introTo5`
`http://rentzsch.com/webobjects/wo5in15`
(un article intéressant sur WebObject ainsi qu’un film Quicktime montrant la création d’une application en 15 minutes avec WebObjects 5)

`http://www.gnustep.org`
(le site du projet GNUstep)
`http://www.opengroupware.org`
(le site du projet OpenGroupware.org)