UNE MARCHE DU CÔTÉ DE LA MER :
SEASIDE

Ce tutoriel est une introduction exploratoire au cadre de développement web (web framework) **SEASIDE 2.0.**

Pour débuter vous aurez besoin de:

- une installation fonctionnelle de Squeak Smalltalk version 3.6 qu'on peut trouver à ftp://st.cs.uiuc.edu/Smalltalk/Squeak/
- le serveur web KomHttpServer6.2 sur Squeak Map
- la version courrante de SEASIDE 2.0 (59)
- Une familiarité de base avec Smalltalk et avec HTML..

L'adresse suivante explique comment installer et démarrer Seaside (en anglais).

http://blogs.inextenso.com/seaside/blog/learning

Une fois Squeak installé vous devrez:

a) charger dans l'ordre KomHttpServer, puis SEASIDE dans SQUEAK, voici comment.

- Dans Squeak, menu (World/open/Package loader) Squeak vous offrira de charger des éléments pertinents pour le Package Loader dites oui à tout.

- Lorsque le Package loader s'affiche à l'écran, sélectionner dans la liste des applications (panneau gauche) "KomHttpServer 6.2" et la commande "install" du menu contextuel.

- Puis sélectionner Seaside 2 (59) dans la liste des applications et "install" du menu contextuel.

b) démarrer le service Seaside. { *à modifier* Seaside fournit la classe WAKom pour s'interfacer avec le serveur web Comanche.} Pour démarrer une instance de Comanche configurée avec Seaside, évaluez le code suivant dans un Workspace:

```
{WAKom startOn: 9090}
```

Seaside devrait maintenant être à l'écoute des arrivées de requêtes sur le port 9090. À partir de maintenant, chaque fois que vous sauvegardez et redémarrez cette image, le service Seaside entre en fonction automatiquement. Sauvegardez maintenant votre image.

Concepts de base: les sessions et les composants

Pour vérifier que Seaside fonctionne, pointez votre navigateur (Internet Explorer) sur l'adresse suivante: http://localhost:9090/seaside/counter. Ceci devrait exécuter la plus minimale des applications Seaside : un simple compteur avec des liens permettant d'incrémenter ou de décrémenter ce compteur. Pour le moment assurez-vous seulement que le compteur fonctionne en cliquant sur les liens "++" et "- -".

En jouant avec le compteur, portez attention à l'URL dans le champ adresse de votre fureteur. Cet URL est constitué de deux parties, chacune disant quelque chose à propos du fonctionnement de Seaside. À la fin il y a une longue série de lettres et de chiffres au hasard ressemblant à ceci

;f1727c45-b4ba-4af0-99f9-d25ae484bf75

Cette série ne change jamais lorsque vous utilisez l'application. C'est une clé unique qui identifie la session courrante de l'utilisateur.

Contrairement à la pluspart des frameworks disponibles pour le développement web, où une session n'est qu'un objet servant à conserver des états, et où tout est centré sur les cycles requête/réponse individuels, Seaside traite une session comme un thread ou un processus: il débute l'exécution d'une session à un point d'entrée bien défini, et l'application procède de façon linéaire à partir de là, faisant des pauses pour afficher des pages web pour l'utilisateur ou attendant pour un 'input'.

Notez un lien nommé "New Session" sur une barre qui s'étend le long du bas de la page. C'est la barre d'outil de Seaside, qui apparaît durant le développement. Le lien "New Session" permet de discarter le processus en cours et d'en débuter un nouveau. Si vous activez ce lien maintenant, la seule chose que vous verrez est que le compteur se remet à 0, et que le **id** de session change.

Devant le point virgule du **id** de session, il y a un petit entier qui s'incrémente constamment au fur et à mesure que l'application progresse. Ce segment de l'URL ne réfère pas à une action particulière, ni n'encode l'état de la session d'une quelconque façon; toutes les

informations sur l'état d'une session sont conservées du côté serveur, et une session Seaside progresse de façon linéaire, non par des sauts d'une action à l'autre. Ce petit entier ne fait que compter le nombre de requêtes dans la session courrante, et permet à Seaside de traquer les progrès de l'utilisateur dans l'application.

De même chaque lien et chaque champ (form field) d'une page a un numéro unique, un **id** séquenciel : ces entiers ne prennent un sens que lorsqu'ils retournent au serveur. Le désavantage de cette façon de faire est de rendre l'utilisation de signets inutilisables parce qu'inintelligibles, par contre çà nous donne une immense flexibilité.

Au lieu de pages ou d'actions possédant des noms, une application Seaside est essentiellement composée de "composants" interagissant entre eux. Ceux-ci sont des objets qui modèlent l'état et la logique d'une portion de l'interface utilisateur.

Quand une session débute, une seule instance de la classe composant est créée, et ce composant entre dans une boucle-réponse: il s'affiche lui-même à l'usager, et attend un input. L' input (un lien activé ou une soumission de formulaire) va déclencher une méthode correspondante du composant, et après exécution de la méthode, le composant s'affichera lui-même encore une fois. Ces méthodes déclenchées peuvent simplement mettre à jour l'état de l'application ou du composant courrant, ou elles peuvent créer un nouveau composant et lui transférer le contrôle, ce nouveau composant entrera alors dans sa propre boucle-réponse : [Affichage - Mise à jour - Affichage].

La classe du composant qui modèle l'application "Counter" se nomme WACounter. Nous allons maintenant l'examiner de plus près.

État, Action, Affichage : l'essentiel des composants

Chaque composant a trois responsabilités :

- entretenir l'état de l'Interface Utilisateur,
- réagir aux 'inputs' de l'utilisateur,
- s'afficher lui-même en HTML.

Voyons comment WACounter se débrouille avec ces responsabilités.

ETAT

Même si presque tout ce qui constitue l'état d'une application est stocké dans des objets-d'affaire

(business objects) ou une base de données, l'interface utilisateur a souvent des états (attributs) qui lui sont propres. Ceci peut inclure, par exemple, la valeur courante d'un champ de formulaire, ou quel enregistrement d'une base de données était affiché, ou encore quels nodes d'une vue en arbre était déployée. Tout cela est stoké dans des variables d'instance des composants qui constituent l'Interface Utilisateur.

Dans le cas de WACounter, le seul état dont on doit se préoccuper est le compte lui-même. Lorsqu'une instance de WACounter est créée au début de la session, sa variable d'instance 'count' est initialisée à zéro (0). En cliquant sur les liens "++" et "--" on ne fait que changer la valeur de cette variable d'instance. Une bonne façon de vérifier cela est de cliquer sur le lien "Inspect" dans la barre d'outil de Seaside, un inspecteur adapté au web s'affichera sur le composant courrant. Vous verrez donc un WACounter avec un nombre de variables d'instance héritées de sa superclass WAComponent, mais aussi 'count'. La valeur courrante de 'count' sera toujours la même que celle affichée sur la page d'où vous avez cliqué sur le lien "Inspect".

Si vous désirez poursuivre l'exploration, le lien sur le nom d'une variable d'instance vous mènera vers un nouvel inspecteur pour cet objet ; vous pouvez utiliser le bouton 'précédent' du navigateur web pour retourner en arrière dans l'arbre. Lorsque vous avec terminé, "OK" vous ramène à l'application 'Counter'.

ACTION

Deux actions sont possibles dans l'application 'Counter', et WACounter a une méthode appropriée pour chacune d'elles. En cliquant le lien "++" un appel de la méthode #increment est effectué. Voici la méthode :

```
increment
   count := count + 1
```

Ceci réalise exactement ce que vous attendez : la valeur de 'count' s'incrémente de un. Lorsque la méthode a terminé son exécution la boucle-réponse continue et le composant se réaffiche lui-même à nouveau avec la valeur nouvelle.

Pour le plaisir tentons de modifier la méthode. Sur la barre d'outil il y a un lien "Browse" qui affiche un System Browser adapté au web et très fonctionnel, focussé sur la classe du composant courrant. Trouvez la méthode #increment et modifiez là pour que l'incrément soit de deux à la fois au lieu d'un. N'oubliez pas 'Accept'. Le lien "OK" vous ramènera à l'application.

AFFICHAGE

Pour afficher un composant, Seaside envoie le message #renderContentOn: , avec une instance de WAHtmlRenderer comme argument. Ceci est un pattern familier pour tous ceux qui on réalisés un applet en Java, ou une nouvelle sousclasse de Morph en Squeak: on donne un canvas à l'objet qui l'utilise pour s'afficher lui-même. Dans le cas présent le canvas sait comment rendre des éléments HTML. Le bidule qui effectue le travail de rendu fonctionne comme un stream (un flux de données) : chaque message qui lui est envoyé s'ajoute en tant que texte ou éléments au document dont il faut faire le rendu.

Voici la méthode #renderContentOn: :

```
renderContentOn: html
   html heading: count.
   html anchorWithAction: [ self increment] text: '++'.
   html space.
   html anchorWithAction: [self decrement] test: '- -'.
```

Trois messages sont envoyés au bidule qui effectue le travail de rendu (le rendereur) chacun produisant un élément HTML différent.

Le premier #heading:, produit une simple section d'entête : si la valeur courrante de 'count' était 42, il ajoutera <h1>42</h1> au document.

Le second #space quant à lui ajoute simplement un caractère d'espace non-sécable.

Le dernier #anchor WithAction:text: est plus intéressant. Il est appelé, évidemment, **pour produire les liens** "++" et "- -" apparaissant sous le compte. L'argument text: spécifie clairement la chaîne de caractères formant le lien. Mais ces liens pointent sur quoi? Que font-ils?

La réponse courte: ne vous en souciez pas. En Seaside, les liens n'ont pas de destination, ce sont des "appels avec retour" (callbacks) : chaque fois que vous générez un lien ou un bouton, il est associé avec un block. Lorsque ce lien ou ce bouton est activé, le block est évalué. Dans le cas présent, en cliquant sur "++" un appel à 'self increment' est effectué.

Modifions légèrement cette méthode : au lieu de liens pour incrémenter et décrémenter nous utiliseront un bouton 'submit'. Trouvez WACounter>>renderContentOn: dans un browser Squeak et changez le code pour celui-ci :

```
renderOn: html
    html form:  [
       html heading: count.
       html submitButtonWithAction: [self increment]
text: '++'.
       html space.
       html submitButtonWithAction: [self decrement]
text: '- -'
     ].
```

La chose principale qu'on a fait est de remplacer #anchorWithAction:text: par #submitButtonWithAction:text:. Les deux méthodes sont pratiquement identiques, il est donc très facile de passer d'un lien à un bouton selon les besoins de l'interface. *Toutefois un bouton 'submit' ne fonctionnera que s'il est à l'intérieur d'un formulaire, c'est à quoi sert la méthode #form:.* Cette dernière est très simple elle ne prend qu'un simple block comme argument. Cette fois il n'y a pas de "d'appel avec retour"; on l'utilise plutôt de façon structurelle pour emballer les contenus du formulaire. Quand on utilise #form: on s'assure que tout tient à l'intérieur d'une paire de 'form tag': [...]. La même convention est utilisée pour la méthode #table:, #tableRow:, et #tableData:.

La boucle-réponse de tout composant:

Maintenant que vous avez été introduit aux responsabilités d'un composant, il est utile de comprendre comment ceux-ci sont inter-reliés. Chaque composant exécute une boucle-réponse...en terme de méthodes çà ressemble à ceci:

- ◇ Une instance de WACounter est créée; 'count' est initialisé à 0.
- ◇ Seaside appel #renderContentOn: sur le compteur. Ceci produit un document HTML affichant la valeur courante de 'count', ainsi que deux liens avec les appels qui leur sont associés. Ce HTML est envoyé à l'utilisateur.
- ◇ L'utilisateur clique sur un des liens, invoquant l'appel qui lui est associé, lequel tour à tour invoque #increment ou #decrement. Ce qui modifie la valeur de 'count'. Après quoi la méthode retourne.
- ◇ Le compteur est rendu à nouveau : une nouvelle page HTML s'affiche avec le nouvel état du compteur.

Regardons de plus près le transfert de contrôle :

Pour transférer le contrôle à un autre composant, WAComponent fournit la méthode #call:. Cette méthode prend un composant comme paramètre, et amorce immédiatement la boucle-réponse de ce composant en l'affichant à l'utilisateur. Pour tester cela, on peut utiliser la classe de composant WADialog, où nous modifierons la méthode #decrement pour qu'elle imprime un message si l'utilisateur essaie d'aller sous zéro:

```
decrement
  count = 0
     ifFalse: [count := count - 1]
     ifTrue: [self call: (WADialog new message: 'Évitons
les chiffres négatifs.')]
```

Si 'count' est zéro, ceci crée une nouvelle instance de WADialog, lui donne un message d'information à afficher, et l'appelle. Maintenant débutons une nouvelle session et essayons le lien "- -". Le compteur devrait disparaître, et on devrait voir le dialogue à sa place, et le message en gros titre.

Afficher un simple dialogue est tellement commun que WAComponent fournit une méthode #inform: qui mimique la méthode par défaut #inform: fournit par la classe Object. Essayer de changer #decrement pour qu'il l'utilise, comme ici :

```
decrement
  count = 0
     ifFalse: [count := count - 1]
     ifTrue: [self inform: 'Éviter les nombres négatifs.']
```

Si vous l'essayez, vous noterez une légère différence : cette fois, le dialogue a un bouton nommé "OK". Qu'arrive-t'il lorsqu'on le presse ? Bien, le dialogue s'en va et le compteur est afficher une fois de plus. Derrière la scène, l'instance WADialog invoque une méthode compagne de #call: nommée #answer, laquelle force le retour au composant qui avait appelé une méthode de WADialog. C'est ce que nous avons désigné comme "l'appel avec retour". En effet appeler un autre composant est un simple appel à une sous-routine : si vous voulez, vous pouvez penser à #call: comme si elle poussait un nouveau composant sur la pile et à #answer comme retirant le composant du dessus de la pile pour retourner au composant précédent.

Dans les faits cependant il n'y a aucune pile d'utilisée

et ce que fait #answer est un peu plus compliqué : il force le retour au message original et le programme continue de ce point. Puisque l'appel du dialogue était la dernière instruction de #decrement, la méthode retourne et la boucle-réponse de compteur redevient active.

Comme ceci est complètement non-intuitif, çà requiert plus d'explication. Examinons de près la séquence exacte de ce qui se produit :

1. WACounter>>decrement fait un appel à WAComponent>>inform:

2. WAComponent>>inform: crée une nouvelle instance de WADialog, et le passe en argument à WAComponent>>call:. Appelons ce point dans la séquence "le point d'appel".

3. WADialog débute sa boucle-réponse, et s'affiche lui-même dans le navigateur de l'utilisateur.

4. L'utilisateur clique le bouton "OK". Ce qui amène WADialog à invoquer WAComponent>>answer.

5. Maintenant la partie étrange. WAComponent>>answer ne retourne jamais puisqu'en fait, il exécute un saut, un retour arrière au "point d'appel" immédiatement après l'envoi de #call:.

6. WAComponent>>call: retourne donc à WAComponent>>inform:.

7. WAComponent>>inform: retourne à son tour à WACounter>>decrement.

8. WACounter>>decrement retourne à la boucle-réponse de WACounter, et il est réaffiché.

La chose vraiment intéressante à propos de #call: est ceci : si un composant appelé fournit un argument à #answer:, cet argument sera retourné à partir de #call:. En d'autres mots, appeler un composant peut fournir un résultat. Ce qui est beaucoup plus puissant que simplement pousser et retirer un composant d'une pile. Par exemple, çà rend facile l'implémentation de #confirm:, lequel affiche une question et retourne "true" ou "false" dépendant de ce que l'utilisateur à cliqué. Modifiez #decrement comme suit:

```
decrement
  count = 0
    ifFalse: [count := count - 1]
    ifTrue:
        [(self confirm: 'Vous acceptez les nombres
```

```
négatifs?')
                ifTrue: [self inform: 'Ok, allons-y!'.
                count := -100]].
```

Si vous jouez avec l'application maintenant, vous réaliserez qu'à l'intérieur de cette seule méthode nous avons trois vues, trois pages : le dialogue de confirmation, le message "Ok, allons-y!", et finalement re retour au compteur lui-même (pour la curiosité utilisez le bouton `précédent' durant cette séquence et voyez ce qui arrive). C'est une structure typique des applications Seaside : plutôt qu'une série de pages étroitement couplées, chacune sachant quelle page vient avant et laquelle vient après, chaque page collectera et retournera une seule pièce d'information de l'utilisateur, avec la logique les chaînant ensemble et les maintenant à une seule place. Le résultat : des pièces de code étonnamment réutilisables.

Vous ne le réalisez peut être pas mais ce n'est pas la première fois que vous avez vu #call et #answer en action. Chaque fois que vous cliquez "Inspect" ou "Browse", vous faites simplement un appel à une nouvelle instance du composant WAInspector ou WABrowser; en cliquant "OK" on invoque #answer pour retourner au composant original.

Mais qui fait les appels ? et où ces liens sont-ils générés ? La réponse est, ce que vous avez vu jusqu'à présent est une instance de WACounter imbriquée à l'intérieur d'un autre composant, une instance de WAToolFrame, qui fait le rendu de la barre d'outil de Seaside. Cette façon d'imbriquer des composant est très commune en Seaside, et les pages sont souvent faites de plusieurs composants individuels. Les détails de l'imbrication des composant dépassent la portée de ce tutoriel, mais pour un exemple simple vous pouvez jetter un coup d'oeil à WAMultiCounter (http://localhlst:9090/seaside/multi) qui contient plusieurs WACounter indépendants.

Aller de l'avant avec sa propre application

J'espère que maintenant vous avez une idée de ce à quoi ressemble une application Seaside. Pour commencer à explorer avec vos propres applications vous voudrez jetter un coup d'oeil à l'application configuration à http://localhost:9090/seaside/config. Celle-ci permet d'ajouter facilement une nouvelle application, de lui donner un nom, et de lui associer une classe de composant de votre choix.

Merci

(Traduit par Raymond Asselin)