SeaSide: Des Composants pour le Web

Prof. Stéphane Ducasse www.iam.unibe.ch/~ducasse/

Le mois dernier nous vous avons montré la puissance de SeaSide pour construire des applications web. Nous avons montré comment le modèle à base de continuations de SeaSide permet d'implémenter des applications webs comme n'importe quelle autre application, c-à-d sans avoir à modéliser le flôt de contrôle de l'application de manière dédiée au modèle d'interaction du web. Ce pouvoir d'expression nouveau nous a permis de très simplement transformer un jeu en une application web. Ce mois-ci nous présentons le modèle de composants offert par SeaSide et aborder plus précisément l'architecture interne de SeaSide.

Contrairement aux modèles servlets qui nécessitent un handler pour chaque page ou requête, Seaside représente *l'entière* session comme un morceau de code avec un *flôt de contrôle linéaire* et naturel : les pages sont des composants qui s'appelent les uns les autres par simple appel de méthodes, les composants sont passés comme des références, plus besoin de les tronçonner en Urls ou champs cachés et tout ceci avec la possibilité de retour en arrière et la concurrence des choix inhérents aux browseurs webs. Une application SeaSide se compose de composants sachant s'afficher sur l'écran et se passant le contrôle de l'application.

Obtenir et démarrer SeaSide

Obtenez Squeak 3.4 (www.squeak.org), le serveur web Comanche 5.0 http://fce.cc.gatech.edu/~bolot/kom/kom-5-0.02May1824.cs.gz et la dernière version de SeaSide (2.21) disponsible à http://beta4.com/seaside2/. Chargez Comanche et SeaSide dans Squeak (open File List...). Pour démarrez Comanche configuré pour SeaSide, exécutez l'expression `WAKom startOn: 9090`. SeaSide doit maintenant écouter des requêtes sur le port 9090. Notez que si vous sauvez votre image, le service sera automatiquement lancer. Pour vérifier que SeaSide fonctionne bien ouvrez la http://localhost:9090/seaside/counter à votre browseur. Vous devez arriver sur un très simple exemple: un compteur comme montré par la figure 1. Cliquez sur les "++" et "--" (ce dernier doit produire une erreur comme nous l'expliquons un peu plus loin).

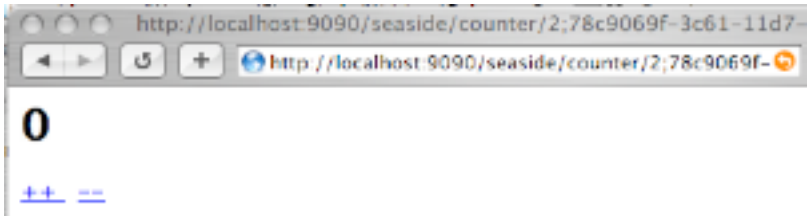

Figure 1: SeaSide fonctionne et montre un compteur

Concepts de Bases: Sessions et Composants

Regardez l'url qui apparait dans le browseur, il est composé d'un nombre et d'une

séquence de lettres et chiffres séparées par un point-virgule comme par example 2;78c9069f-3c61-11d7-a1d2-000393b2aa24. La seconde partie qui n'a pas changée durant votre utilisation est un identifiant unique qui représente la session en cours. Une session est un concept central en SeaSide. Contrairement à la plupart des frameworks web qui traitent les sessions principalement comme des entités pour conserver l'état de l'application et s'occupe de gérer individuellement les cycles de requêtes et réponses, Seaside traite une session à la manière d'un processus (thread): une session débute à un point donné et l'application est exécutée linéairement depuis ce point, affichant les pages web et attendant les données de l'utilisateur. Le lien "New Session" en bas de la page permet de créer une nouvelle session, *i.e.,* crée un nouveau flôt et initialise l'application.

Le nombre avant le point-virgule augmente régulièrement alors que l'application s'exécute. Ce nombre représente le nombre de requêtes par session et permet à SeaSide de connaitre la progression de l'utilisateur. Notez que tout l'état de l'application est gardée sur le serveur. De manière similaire tout lien ou champ de formulaire a un unique identifiant que le serveur sait interpréter. Cette pratique a le désavantage de rendre les favoris illisibles mais permet une très grande flexibilité.

Une application en SeaSide n'est pas composée de pages ou d'actions mais de *composants*, *i.e.*, des objets qui modèlent l'état et la logique d'une partie de l'interface utilisateur (la page web). Ces composants peuvent être imbriqués les uns dans les autres et réutilisés entre applications. SeaSide propose un ensemble de composants prédéfinis. Quand une session démarre, une instance d'un composant est créée et celui-ci entre dans une *boucle de réponse* : il s'affiche et attend des entrées. Les entrées (sélection d'un lien et soumission d'un formulaire) déclenchent la méthode correspondante du composant. Quand cette méthode se termine, le composant s'affiche de nouveau. Les méthodes peuvent simplement changer l'état de l'application, du composant courant ou elles peuvent créer d'autres composants ou encore transférer le contrôle de l'application à un autre composant qui entre alors dans sa propre *boucle de réponse*.

Caractéristiques d'un component: état, action et affichage

Un composant a trois responsibilités: le maintien l'état interne de l'interface (lorsque l'on se place dans une architecture découplant les objets domaines de l'application de leur représentations), la réaction aux actions de l'utilisateur et l'affichage. Regardons comment la classe `WACounter` fonctionne.

Etat. Alors que la plupart de l'état d'une application est généralement stocké dans les objets du domaine ou une base de données, l'interface utilisateur possède aussi son propre état comme la valeur courante d'un champ. De telles informations sont stockées dans les variables d'instances du composant.

Pour notre compteur, le seul état à considérer est le compteur lui-même comme vous pouvez le voir en lisant la définition de la classe `WACounter`, soit dans Squeak ou directement en cliquant sur le lien "Browse" (figure 3). Quand une instance est créée au début de la session, elle initialise la variable `count` à 0 (méthode `initialize`). Les liens "++" et "**--**" changent simplement la valeur de cette variable. Vous pouvez voir l'état du composant en cliquant sur le lien "Inspect"

qui va ouvrir un inspecteur adapté au browseur web comme le montre la Figure 2. En plus de la variable `count` dont la valeur est utilisée dans le titre, un compteur hérite certaines variables de la class `WAComponent` superclasse de `WACounter`. Cliquez sur le lien "OK**"** pour revenir à l'application.

Figure 2: Inspecteur sur un compteur.

Action. Les actions sont représentées par les méthodes du composant. Dans le cas du compteur deux actions sont possibles pour changer la valeur du compteur. Le composant possède deux méthodes correspondantes (`increment` and `decrement`). Cliquez sur le lien "++" invoque la méthode `increment` dont le code est le suivant et qui ne fait rien de magique. Quand cette méthode se termine, la boucle de réponse se poursuit, le composant s'affiche en attendant la prochaine interaction.

```
WACounter>>increment
  count := count + 1

WACounter>>decrement
      self error: 'foo'.
      count := count - 1
```

La version 2.21 de SeaSide montre comment les erreurs sont controlées et affiche la pile d'exécution dans un browseur en levant une erreur (méthode `error:` dans la méthode `decrement`). Afin de pouvoir interagir avec le compteur redéfinissez cette méthode comme suit.

```
WACounter>>decrement
      count := count - 1
```

Changez le code de cette méthode afin de pouvoir décrémenter le compteur en utilisant le lien "Browse**"**. Pour compiler le code cliquez sur le bouton "Accept**"** en bas de l'éditeur. Cliquez sur "Ok**"** pour revenir dans l'application.

Affichage. Pour afficher un composant, la méthode `renderContentOn:` est invoquée avec comme paramétre une instance de `WAHtmlRenderer`. Chaque message ajoute du texte ou des éléments de documents comme le montre la méthode `renderContentOn:` du compteur.

```
WACounter>>renderContentOn: html
    html heading: count.
```

```
    html anchorWithAction: [self increment] text: '++'.
    html space.
    html anchorWithAction: [self decrement] text: '--'.
```

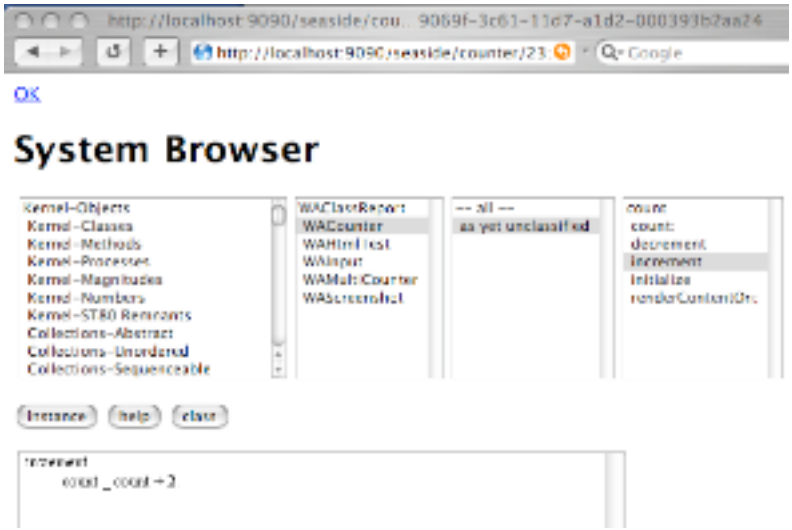

Figure 3. Pour coder ou lire le code directement dans un navigateur Web.

Trois differents messages sont utilisés chacun produisant des éléments HTML différents. `heading:` produit une section heading, `space` ajoute un caractère. `anchorWithAction:text:` produit les liens “++” et “--”. En SeaSide, les liens n’ont pas de destinations, ils ont des callbacks: à chaque fois que l’on crée un lien ou bouton on y associe un block. Lorsque ce lien ou bouton est activé le block est évalué. Dans notre cas les méthodes `increment` ou `decrement` sont invoquées.

La Boucle de Réponse

Regardons maintenant comment les composants sont connectés. Chaque composant effectue une *boucle de réponse*: s'affiche, attend des données, les traite et se réaffiche. Dans notre cas, une instance de `WACounter` est créée, la variable `count` est initialisée à zéro. SeaSide appelle la méthode `renderContentOn:` du compteur qui produit le document HTML montré dans la figure 1 avec tous les liens, document qui est finalement affiché dans le browseur. Lorsque l'utilisateur clique sur un des liens le callback associé est exécuté et appelle les méthodes increment ou decrement qui change l'état de l'objet. Une fois la méthode exécutée le compteur est de nouveau affiché avec son nouvel état.

Interaction entre Composants

Pour transférer le flot de controle à un autre composant, la class `WAComponent` offre une méthode spéciale nommée `call:`. Cette méthode passe en argument le composant qui reçoit le controle et commence immédiatement la boucle de réponse du composant, qui s'affiche alors à l'utilisateur. Mais essayons plutot cela, modifiez la méthode `decrement` afin qu'un message soit affiché lorsque l'utilisateur essaye d'obtenir une valeur negative du compteur.

```
WACounter>>decrement
  count = 0
     ifFalse: [count := count - 1]
```

```
    ifTrue: [self call:
                (WADialog new
                    message: 'Pas de negatif!')]
```

Lorsque la variable d'instance `count` est nulle, une instance de `WADialog` est créée et passée comme argument à un appel à la méthode `call:`. Vous devez obtenir un message. En fait afficher un dialogue est si courant que SeaSide offre la méthode `inform:` que l'on utilise comme suit.

```
WACounter>>decrement
  count = 0
     ifFalse: [count := count - 1]
     ifTrue: [self inform: 'Pas de negatif!']
```

La méthode `inform:` est légéremment différente. Cette fois le dialogue propose un bouton "OK" qui vous ramène au compteur quand vous le pressez. La méthode `inform:` est définie comme suit:

```
WAComponent>>inform: aString
      self call: (WADialog new
                  message: aString;
                  on: 'OK' answer: [true])
```

Comment tout cela fonctionne-t-il? L'instance de `WADialog` utilise la méthode `answer` qui rend le flot de controle au composant ayant invoquée la méthode `call:` originalle. La méthode `answer` force la terminaison de l'invocation originale de `call:`. Le programme continue alors à partir de ce point. Dans notre exemple, l'appel au dialogue est la dernière instruction de la méthode `decrement`, donc cette méthode se termine et la boucle de réponse du compteur se poursuit.

Regardons exactement ce qu'il se passe.
(1) `WACounter>>decrement` appelle `WAComponent>>inform:`.
(2) `WAComponent>>inform:` crée une nouvelle instance de WAdialog et la passe en argument de `WAComponent>>call:`. Appelons ce point, le point d'appel. (3) Le WADialog commence sa boucle de réponse et s'affiche dans le browseur. (4) Quand l'utilisateur presse le bouton "OK", le dialogue invoque la méthode `WAComponent>>answer`. En fait, la méthode `WAComponent>>answer` ne termine jamais car il saute directement au point d'appel juste après l'appel de la méthode call:. (5) `WAComponent>>call:` retourne à `WAComponent>>inform:`. (6) `WAComponent>>inform:` retourne à `WACounter>>decrement`. (7) `WACounter>>decrement` retourne à la boucle de réponse du compteur qui est alors affiché.

La méthode `call:` est en plus très puissante car si un composant appelé spécifie un argument lors de l'appel de la méthode `answer:`, cet argument sera retourné par la méthode `call:`. Ainsi invoqué un composant peut rendre un résultat, ce qui est bien plus que simplement mettre et enlever des composants d'une pile. Il devient alors très simple d'implanter la méthode `confirm:` qui affiche un texte et rend vrai ou faux suivant le choix de l'utilisateur. Modifiez la méthode `decrement` as comme suit:

```
WACounter>>decrement
  count = 0
    ifFalse: [count := count - 1]
    ifTrue:
        [(self confirm: 'Allons dans les negatifs?')
               ifTrue: [self inform: 'Ok, allons-y'.
                       count := -100]].
```

Maintenant si vous cliquez sur le compteur vous allez obtenir trois différentes pages: le dialogue, le message 'Ok, allons-y' et finallement le compteur lui-meme. Notez également que le bouton retour du browseur est bien géré.

Un dernier point

Cet exemple montre la structure typique des applications développées avec SeaSide: au lieu d'avoir une serie de pages chacune connaissant exactement les pages qui la précédent et la suivent, avec SeaSide chaque page contient de l'information présentée à l'utilisateur et ne doit pas se soucier du contexte dans lequelle elle est utilisée, ce qui a l'extrememment interresante propriété de permettre de construire des pages reutilisables.

Vous ne l'avez peut etre pas réalisé mais ce n'est pas la première fois que vous avez rencontré le couple `call:/answer`. A chaque fois que vous avez cliqué sur "Inspect" ou "Browse" vous avez simplement invoqué des instances des composants `WAInspector` or `WABrowser` components, cliquant 'OK' utilise la méthode `answer` pour retourner à l'application principale. En fait l'application principale est une instance de WAToolFrame qui affiche la barre d'outils dans laquelle le compteur est inséré. SeaSide permet en effet d'imbriquer des composnants les uns dans les autres comme le montre l'exemple http://localhost:9090/seaside/multi mais ceci est une autre histoire. Notez que SeaSide permet de gérer de manière plus controlée que ce que nous vous avons montré ici la sémantique du bouton back. Ceci permet de s'assurer que les objets restent dans des états cohérents par rapport à la page dans laquelle ils ont été modifiés et évite donc d'avoir des pages web désynchronisées quand on utilise les boutons de navigations comme cela est encore le cas sur un grand nombre de sites. Mais nous n'avons pas la place de vous le montrer dans cet article.

Références

Squeak: http://www.squeak.org/
SeaSide: http://beta4.com/seaside2/
ESUG: http://www.esug.org/
Wiki Francophone: http://www.iutc3.unicaen.fr:8000/fsug/
Livres Gratuits : http://www.iam.unibe.ch/~ducasse/WebPages/FreeBooks.html
ESUG CD : http://www.ira.uka.de/~marcus/EsugcD.html
Squeak CD : http://www.ira.uka.de/~marcus/SqueakCD.html