SeaSide: Des applications web complexes trés simplement...

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

SeaSide est un puissant framework pour developper des applications web *complexes*. La fonctionalité la plus remarquable et unique de SeaSide réside dans son approche de la gestion de sessions. 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 *flot de contrôle linéaire* et naturel : les pages s'appelent les unes les autres comme des invocations de méthodes, les formulaires sont de simples objets, les objets 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 inhérente aux browseurs webs et du bouton de retour en arrière.

SeaSide offre aussi un modèle d'événements basés sur des call-backs, la gestion des pages devant s'autodétruire, une approche simple de la génération d'HTML, un ensemble de components reutilisable, une gestion des templates pour designers, et un ensemble d'outils de développements.

Dans cet article nous allons faire nos premiers pas en SeaSide en montrant comment un simple jeux peut etre porter en SeaSide, dans l'article suivant nous abordons l'architecture de SeaSide et son modèle de composants ainsi que la possibilités de backtrack, des sujets points plus avancés de SeaSide.

Obtenir et démarrer SeaSide

Vous devez avoir une version récente de Squeak (3.2 ou 3.4) www.squeak.org, le serveur web Comanche 5.0 pour Squeak 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/downloads/latest.html. 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 "--". Si SeaSide vous demande un nom de passe donnez : seaside/admin.

Un simple jeu de devinettes

Nous allons implémenter un sujet de devinettes qui fonctionne de la manière suivante : (1) le système demande à l'utilisateur de penser à un fruit, (2) l'ordinateur pose une série de questions pour identifier le fruit. Quand l'ordinateur a cerné le fruit en question il propose une solution. (3) Si c'est le bon fruit, le jeu est fini. Si ce n'était pas le bon fruit, il demande à l'utilisateur le nom du fruit ainsi

qu'un indice qui lui permettra d'identifier le fruit. Ces informations sont ajoutées dans une base de données pour utilisation future.

L'idée n'est pas de faire un cours sur les jeux de devinettes, donc voila lalogique interne du jeux. Les indices et les fruits sont représentés comme les noeuds d'un arbre binaires étiquetté. Chaque indice est à la tête d'un arbre dont les branches représente les réponses positives ou négatives de l'utilisateur. En partant de la racine de l'arbre, l'ordinateur va donc traverser l'arbre, poser les questions basées sur les indices et naviguer dans l'abre en suivant les réponses de l'utilisateur. Lorsqu'il arrive dans une voie terminale, il a soit trouvé le bon fruit soit il doit insérer un nouveau noeud contenant un fruit et indice.

Comme Squeak ne propose pas d'arbre binaires dans sa distribution, nous allons en définir un. Il a besoin d'une etiquette et des deux branches. Comme nous ne le définissons que pour ce jeux appelons les key, yes et no. Définissez la classe FruitTree comme suit

```
Object subclass: #FruitTree
	instanceVariableNames: 'key yes no '
	classVariableNames: ''
	poolDictionaries: ''
	category: 'Seaside Game'
```

Définissez les accesseurs key: aString, key, yes: aTree, yes, no et no: aTree, sur le modèle

```
FruitTree>>key: aString
  key := aString

FruitTree>>key
  ^ key
```

Quand on insère un nouvel indice et un fruit, nous voulons insèrer l'indice en premier avec la branche yes de l'indice pointant sur le nouveau fruit. Pour nous faciliter la tache, nous définissons donc la méthode de classe newFruit:withClue: définie comme suit:

```
FruitTree class>>newFruit: aFruitString withClue: aClueString
  ^ self new
      key: aClue;
      yes: (self new key: aFruitString)
```

Pour commencer le jeu nous avons besoin d'une racine que nous définissons ici comme une variable globale pour plus de simplicité. Evaluez dans un workspace le code suivant. Bien sûr une aubergine n'est pas un fruit mais c'est le symbole de SeaSide. Le terme SeaSide (Squeak Enterprise Aubergines Server) vient d'une caricature des Beans.

```
FruitRoot := FruitTree newFruit: 'an eggplant' withClue: 'purple'.
```

Maintenant nous devons écrire la logique du jeu. Nous l'avons défini d'un seul bloc de façon à ce que l'on puisse l'exécuter dans un workspace. Ouvrez un

workspace et tapez le code suivant:

```
|node response next|
Object inform: 'Think of a fruit.'.

node := FruitRoot.
[response := Object confirm: 'Is it ', node key, '?'.
next := response ifTrue: [node yes] ifFalse: [node no].
next notNil]
    whileTrue: [node := next].

response
    ifTrue: [Object inform: 'Got it!']
    ifFalse:
        [|fruit clue|
         fruit := FillInTheBlankMorph request: 'Give the name of your fruit'.
         clue := FillInTheBlankMorph request: 'What makes it different from other fruits?'.
         node no: (FruitTree newFruit: fruit withClue: clue)].
```

Voila une implémentation assez simple de la logique du jeu telle que nous l'avons décrite plus avant: on traverse l'arbre jusqu'à ce que l'on arrive à une branche terminale. A cepoint, si la réponse est vraie on a trouvé le fruit sinon on crée un nouveau noeud dans l'arbre. Exécutez cette expression plusieurs fois de suite pour voir l'arbre grandir.Vous pouvez ensuite essayez FruitRoot explore pour le visualiser.

Pour le web

Ce jeu somme toute trivial, 15 lignes de code dans un workspace, est loin d'etre trivial à implanter pour en obtenir une application web avec des moyens traditionels. Si vous avez déjà développé des applications web, prenez un moment pour imaginer comment vous vous prendriez. En premier lieu vous devriez transformer l'algorithme de sa forme actuelle iterative en une forme récursive liant chaque réponse aux questions yes/no à la suivante. Vous devriez pensez aussi aux appels à la widget FillInTheBlankMorph à la fin: pourriez utiliser la même page pour les deux appels, comme savoir ou rentrer l'information? Peut-être que vous devriez paramétrer la page avec la suivante? Ect...Bref un enfer pour un jeu aussi ridicule donc imaginez pour une véritable application.

En SeaSide toute ces questions pourtant importante dans d'autres systèmes n'ont aucune importance. Sans rentrer dans les détails que nous aborderons dans l'article suivant, nous allons transformer notre jeu. Tout d'abord, nous créons une nouvelle sorte de composant web que nous appelons FruitGuessing comme suit:

```
WAComponent subclass: #FruitGuessing
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'Seaside Game'
```

Ensuite nous définissons comment le composant doit s'afficher dans le

navigateur web en définissant la méthode renderContentOn:

```
FruitGuessing>>renderContentOn: html
        html form: [html submitButtonWithAction: [ self play ] text: 'Play' ]
```

Ici cette méthode crée simplement un bouton ayant comme étiquette ‘Play’ (Figure 1) et comme action d’invoquer la méthode play que nous allons définir plus loin.

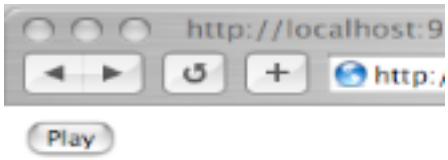


Figure 1: L’interface ultra minimaliste de notre jeu.

Nous allons enregistrer notre application (notez que nous pourrions le faire en exécutant l’expression FruitGuessing registerAsApplication: 'fruit') en utilisant la page de configuration de SeaSide. Dans le navigateur allez sur la page http://localhost :9090/seaside/config (le login est seaside/admin) vous devez alors arrivez sur une page contenant la liste des applications demoes. Entrez le nom de votre application par exemple ‘Fruit’ dans le champ et cliquez sur le bouton **add** et sélectionnez la classe que vous venez de créer comme montré dans la Figure 2.

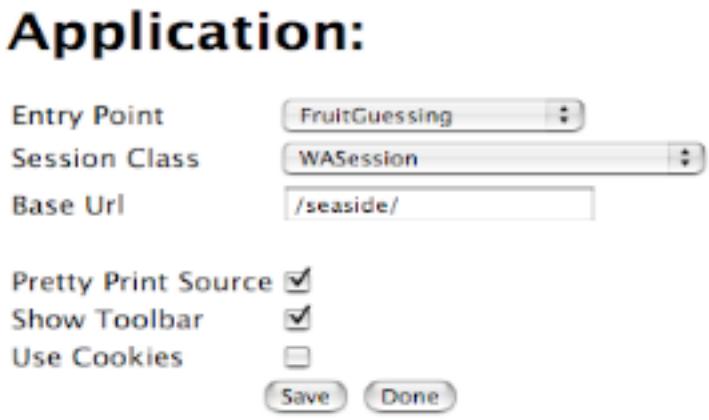


Figure 2 : configuration de l’application

Une fois ceci fait vous devez pouvoir lancer votre application soit depuis le panneau de configuration en cliquant sur l’élément de la liste, soit en allant sur la page http://localhost:9090/seaside/fruit. Vous devez obtenir la Figure 1. Si vous pressez sur le bouton vous allez invoquer une méthode non définie donc vous allez obtenir la trace de l’erreur. Il nous faut donc convertir notre script en une méthode. En SeaSide rien de plus simple. Copiez le code du workspace dans l’éditeur, changer Object et FillInTheBlankMorph par self comme suit :

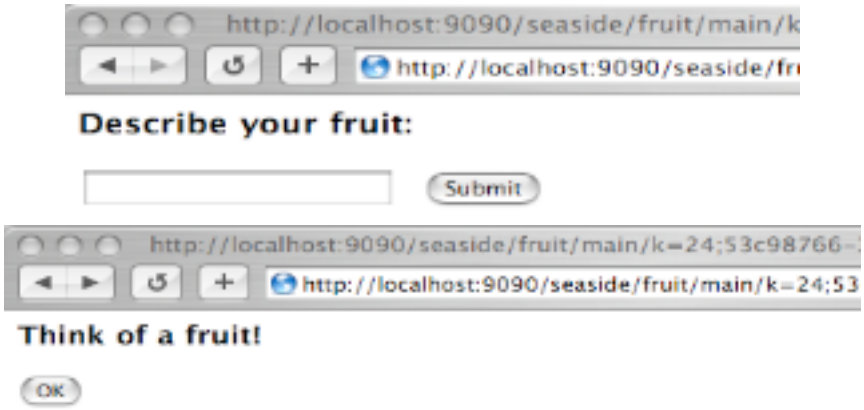
```
FruitGuessing>>play
	| node response next |

	"Initialize the game"
	node := FruitRoot.
	self inform: 'Think of a fruit!!'.

	"Walking down the tree"
	[ response := self confirm: 'Is it ', node key, '?'.
	  next := response
		ifTrue: [ node yes ]
		ifFalse: [ node no ].
	  next isNil not ] whileTrue: [ node := next ].

	"Update the tree"
	response
		ifTrue: [ self inform: 'Got it!!' ]
		ifFalse: [ | fruit clue |
			fruit := self request: 'Describe your fruit:'.
			clue := self request: 'What makes it different form other fruit?'.
			node no: (FruitNode newFruit: fruit withClue: clue) ]
```

Et voila votre application fonctionne. Cliquez sur le bouton play et jouez. Comme vous le voyez vous m'avez pas eu à transformer votre application ni à changer de paradigme.



Objets et Continuations

Une application en SeaSide est un ensemble d'objets se passant lecontrôle de l'application et ayant la responsibilité de gérer leur affichage et leur entrées. Comme nous le verons dans le prochain article cela n'est pas trivial en présence du bouton de retour des navigateurs. Contrairement à la plupart des autres frameworks pour le développement d'application webs, SeaSide intègre le cycle requete/reponse de applications web avec le flot de controle illimité offert par Smalltalk. Ainsi les boucles, les conditions, les appels et retours de méthodes peuvent tous créer de multiple pages web indépendentes. A titre d'exemple notez que l'outil de configuration ainsi que tous les dialogues sont des composants

SeaSide. Essayez le browser de code en cliquant sur le Lien **"Browse"**. Il est aussi développé en SeaSide de la manière la plus simple qu'il soit.
L'idée est que SeaSide utilise la notion de continuation des langages fonctionnelles ce qui lui permet de conserver des points dans l'exécution d'un programme pour potentiellement y revenir. Bien sûr, SeaSide offre des abstractions sur ce modèle et le programmeur n'a pas à connaitre la logique interne de SeaSide, il doit juste utiliser les abstractions offertes. Dans l'article de moi prochain, nous détaillerons le modèle de SeaSide ainsi que les façons dont le flot de contrôle est géré. Nous présenterons les messages spécifiques qui offrent à SeaSide ce comfort d'utilisation et ce pouvoir d'expression aussi puissant et naturel qu'inattendu.

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