Ruby on Rails

Framework "Agile" de développement Web

Juin 2006

(corrigée le 03 juillet 2006)

Jérôme MORDELET

Introduction

Après l'explosion de la bulle internet en 2001, on assiste à un renouveau des applications web. Le terme "Web 2.0" fait son apparition durant l'été 2004. Révolution pour certain, ballon de baudruche pour d'autres, force est de constater que les applications web ont fortement évolué au cours de ces deux dernières années :

- évolution de la place de l'utilisateur avec les blogs, les wiki, le podcasting… (Joël de Rosnay utilise le néologisme de "pronétariat"[1]) ;
- évolution de l'ergonomie et de l'accessibilité des pages (drag n' drop, zones dépliantes…) ;
- évolution du sens (respect des règles sémantiques dans la conception, recherches plus pertinentes des moteurs…).

La prospérité de Google et le succès de ses applications[2] sont étroitement liés à l'intégration de ces évolutions. On peut citer Gmail (www.gmail.com), Google Calendar (www.google.com/calendar) ou encore Google finance (finance.google.com) qui proposent des interfaces riches en fonctionnalités, pourvues d'une forte interactivité, illustrant bien les services étiquetés Web 2.0.

"Les sites web ne sont plus seulement des bases informatives mais deviennent des fournisseurs de services".[3] Pour proposer des services internet pertinents, les entreprises doivent faire preuve de réactivité tout en proposant des applications performantes, dotées d'interfaces simples et fortement orientées utilisateurs. Une application basée sur une idée excellente ne sera jamais adoptée si elle est présentée plusieurs mois après celle de la concurrence ou si son interface est trop complexe pour l'utilisateur final. Les développeurs doivent donc s'adapter, utiliser des méthodes et des

[1] Joël de Rosnay, "La révolte du pronétariat" (Fayard, 2006)
[2] Jean-Louis Gassé "Comment Google renverse Microsoft" (http://gassee.blog.20minutes.fr/archive/2006/03/15/ajax-le-talon-d-achille-de-microsoft.html)
[3] Dominique Vilain "Les briques du web 2.0 par la pratique" (http://blog.domy.be/?2006/05/17/19)

outils leur apportant souplesse et performance. Moins de lourdeur, pour une réponse plus rapide.

Fort de ce constat, le Framework de développement web "Ruby on Rails" présente de nombreux atouts qui lui ont permis de susciter un enthousiasme réel dans le monde du développement Web, aujourd'hui dominé par JAVA, PHP ou .NET.

Orienté RAD (Rapid Application Developpement), Ruby on Rails reprend de nombreux principes communs aux méthodes "Agiles". Il propose aux développeurs d'améliorer considérablement leur productivité grâce à :

- un code plus concis, plus évolutif et produit plus rapidement ;
- une configuration minimale en favorisant certaines conventions ;
- des technologies déjà intégrées comme "Ajax" permettant d'offrir aux utilisateurs finaux une interface "riche" et plus ergonomique.

En résumé : **Simplification → Réactivité → Productivité**

Dans la première partie de ce document, après un bref rappel sur la notion de framework, nous verrons comment est né Ruby on Rails et les principes essentiels sur lesquels il repose.

Dans la deuxième partie, nous détaillerons les différents éléments qui le constituent.

Enfin, la dernière partie sera consacrée à la crédibilité de Ruby on Rails face aux autres solutions du marché.

Partie 1
Origine, influences et philosophie

"Rails is optimized for programmer happiness..."
David Heinemeier Hansson

1. Pourquoi un framework ?

Le framework, littéralement "cadre de travail", est une couche logicielle, constituée d'un ensemble de composants, servant de base au développement d'applications.

Dans le cadre du développement d'applications web, il apparaît que de nombreuses fonctionnalités sont communes aux développements successifs. Paradoxalement, ces fonctionnalités peuvent être re-développées à chaque fois si elles ne sont pas conçues dès le départ dans une optique de réutilisation.

Dans l'objectif de limiter ces répétitions, un framework offre un socle commun et une structure identique aux applications. Il assure ainsi toutes les tâches communes et facilite le développement en allégeant le nombre de fonctionnalités à implémenter.

Très structurant, un framework fournit aussi des contraintes de développement qui permettent l'écriture d'un code plus évolutif et plus simple à maintenir.

En règle générale, le framework apporte donc :

- un gain de productivité ;
- un cadre favorisant le développement de composants réutilisables ;
- une simplification de la maintenance ;
- une diminution des risques.

Il n'en demeure pas moins que le choix du framework est décisif car il impose un temps d'apprentissage et un travail de prise en main pouvant être importants. Il est aussi essentiel de garder à l'esprit que les applications ainsi développées sont étroitement liées au framework et que l'utilisation d'un autre produit demandera une réécriture partielle de celles-ci.

2. 37signals

"37signals is a small team that creates simple, focused software."

37signals

"37signals products are beautifully simple, elegant and intuitive tools that make an Outlook screen look like the software equivalent of a torture chamber."

Jeremy Wagstaff, the Wall Street Journal.

37signals est une société, composée de sept personnes (réparties dans sept fuseaux horaires), qui depuis deux ans est à l'origine d'applications web innovantes et d'ouvrages faisant références.

Les applications proposées par 37signals visent à faciliter le travail collaboratif et le partage d'informations via internet. D'une efficacité redoutable de par leur simplicité, elles sont des exemples en terme d'ergonomie des interfaces utilisateurs.

On peut citer :

- Basecamp : gestionnaire de projet.
- Campfire : messagerie instantanée d'entreprise.
- Backpack : gestionnaire d'informations.
- Writeboard : éditeur de texte collaboratif.
- Ta-da List : gestionnaire de tâches.

C'est lors du développement de Basecamp, débuté mi-2003, que **David Heinemeier Hansson** crée Ruby on Rails. Il extrait Ruby on Rails de Basecamp et propose la première version du framework en juillet 2004. Pour ce travail, il gagnera en Août 2005 le prix du "Best Hacker of the Year" délivré à l'OSCON, l'O'Reilly Open Source Conference, par Google et O'Reilly.

Ce framework sera la base des applications suivantes. En plus de cet outil, l'équipe de 37signals met en avant une méthode de travail pouvant paraître peu conventionnelle, mais à priori efficace. L'ouvrage "Getting Real" en explique les grands principes, comme :

- "Build Less" (*construire moins*) ;
- "Fix Time and Budget, Flex Scope" (*fixer le temps et le budget, ajuster les fonctionnalités)* ;
- "Small Team" (*petite équipe*) ;
- "Scale Later" (*agrandir plus tard*) ;
- …

Cette approche n'est pas forcément nouvelle. On peut facilement observer de nombreux points communs avec des méthodes qui ont déjà fait leurs preuves : les méthodes "Agiles".

3. Méthodes Agiles

Dans les années 80, James Martin élabore la méthode RAD. Il s'ancre sur le double constat suivant :

- Premièrement, le manque de communication entre développeurs et utilisateurs conduit souvent à la réalisation d'applications mal adaptées.

- Deuxièmement, les méthodes classiques sont inadaptées aux rapides évolutions des technologies et la durée des projets, ainsi gérés, est beaucoup trop longue.

Le principal objectif de cette méthode est l'amélioration de la qualité des développements tout en diminuant les délais et en facilitant la maîtrise des coûts.

Cette méthode est aujourd'hui considérée comme la racine commune des méthodes "Agiles".

En février 2001, les promoteurs des principales méthodes issues de RAD se sont regroupés pour former l'"Agile Alliance". Mutualisant les différentes méthodes, ils ont élaboré le "Manifeste pour le développement agile d'applications" ou "**Agile Manifesto**". D'ailleurs, un des auteurs du "Agile Manifesto", Dave thomas est coauteur du premier livre sur Ruby on Rails : "Agile web development with Rails".

Les quatre valeurs fondamentales citées dans ce manifeste sont :

- **"Personnes et interactions plutôt que processus et outils"**

  L'équipe : Dans l'optique Agile, elle est bien plus importante que les moyens matériels ou les procédures. Il est préférable d'avoir une équipe soudée et qui communique, même si elle est composée de développeurs moyens, plutôt qu'une équipe composée d'individualistes brillants. La communication est donc une notion fondamentale.

- **"Logiciel fonctionnel plutôt que documentation complète"**

  L'application : Il est vital que l'application fonctionne. Le reste, et notamment la documentation technique, est secondaire. Même si une documentation succincte et précise est utile comme moyen de communication. La documentation représente une charge de travail importante et peut être néfaste si elle n'est pas à jour. Il est donc préférable de commenter abondamment le code lui-même, et surtout de transférer les compétences au sein de l'équipe (on en revient à l'importance de la communication).

- **"Collaboration avec le client plutôt que négociation de contrat"**

  La collaboration : Le client doit être impliqué dans le développement. On ne peut se contenter de négocier un contrat au début du projet, puis de négliger les demandes du client. Le client doit collaborer avec l'équipe et fournir un feed-back continu sur l'adaptation du logiciel à ses attentes.

- **"Réagir au changement plutôt que suivre un plan"**

  L'acceptation du changement : La planification initiale et la structure du logiciel doivent être flexibles afin de permettre l'évolution de la demande du client tout au long du projet. Les premières versions du logiciel provoquent souvent des demandes d'évolution.

Les méthodes Agiles les plus connues sont : RAD, XP, DSDM, Scrum…

Reflet de la méthode de travail utilisée par l'équipe de 37signals, Ruby on Rails a été pensé dès le début pour répondre à l'ensemble de ces principes. Comme nous allons le voir maintenant, le choix du langage de programmation découle lui aussi de cette réflexion.

4. Ruby

"I wanted a language more powerful than Perl and more object-oriented than Python."

"Ruby is designed to make programming not only easy but also fun."
Yukihiro Matsumoto

Ruby est un langage de script orienté objet, créé au Japon, par Yukihiro "Matz" Matsumoto en 1993. La première version a été publiée en 1995.

Entièrement orienté objet, Ruby peut être comparé à Smalltalk, mais aussi à Python pour sa simplicité ou à Perl pour sa flexibilité (le nom de Ruby, une autre pierre précieuse, y fait d'ailleurs référence).

Ce langage offre :

- une syntaxe très simple ;
- des variables non typées, dont la déclaration est facultative ;
- du tout objet ;
- une gestion interne de la mémoire (comparable au ramasse miettes de Java);
- un système de gestion d'exceptions ;
- la possibilité de remplacement des méthodes et la modification des classes pendant l'exécution du programme ;
- l'utilisation des expressions régulières (gérable en japonais).

4.1. Cohérence et simplicité

Lors de la création du langage, Yukihiro Matsumoto a mis en œuvre le principe **PoLS** (principle of least surprise) pour faire de Ruby un langage "rapide et simple".

Ainsi dès les premières lignes de code, on est surpris par la cohérence du langage qui rend la syntaxe tellement intuitive qu'elle permet quasiment de coder dans un langage parlé.

Autre facteur de cohérence, Ruby est nativement objet. Il n'a donc pas eu à souffrir des modifications plus ou moins réussies que d'autres langages de script (Python, Perl ou Php) ont subi pour le devenir.

4.2. Extensibles

Ruby est multi-plateforme et open source ce qui contribue en grande partie à son extension. Sa communauté de développeurs est très active, elle centralise et met à disposition sur le site internet Rubyforge[1] l'ensemble des projets open source.
On y trouve de nombreuses librairies, la plupart compatibles avec le système de gestion de paquetage "Rubygems" qui facilite leur installation.

[1] http://rubyforge.org/

Parmi ces librairies, on peut citer :

- Rails : LE framework Ruby on Rails.
- RedCloth : gestion de mise en forme du texte façon Wiki.
- Syntax : coloration syntaxique.
- Feedtools : gestion des flux rss et atom.
- PDF::Writer : génération de PDF.
- Capistrano (anciennement Switchtower) : outil de déploiement automatique.
- Flickr : permet d'utiliser l'API de Flickr (site de partage de photo).

4.3. Exemples rapides de code

```
5.times { print "Hello World !" }
```

Cinq fois affiche "Hello World !"

```
exit unless "Hello World".include? "ello"
```

Arrête à moins que "Hello World" ne contienne "ello"

Partie 2
Le Framework Rails

“Rails is the most well thought-out web development framework I’ve ever used. And that’s in a decade of doing web applications for a living. I’ve built my own frameworks, helped develop the Servlet API, and have created more than a few web servers from scratch. Nobody has done it like this before.”
James Duncan Davidson, Creator of Tomcat and Ant

“Rails is the killer app for Ruby.”
Yukihiro Matsumoto, Creator of Ruby

Malgré ses nombreuses qualités, il manquait au langage Ruby un Framework puissant, exploitant au mieux ses nombreuses "facettes". Ruby on Rails s'avère être le candidat idéal.

1. Framework "Agile"

Comme nous l'avons vu plus haut, les méthodes Agiles s'appuient sur quatre valeurs fondamentales. Nous allons voir maintenant comment Ruby on Rails permet de les concrétiser.

1.1. Conventions plutôt que configurations

Après installation, les frameworks doivent être configurés pour pouvoir être utilisés. Les fichiers de configuration, généralement codés en XML, permettent d'indiquer les règles à respecter pour communiquer avec l'environnement d'installation.

Ruby on Rails suit une autre logique, privilégiant l'utilisation de conventions à l'utilisation de nombreux fichiers de configuration. Des règles d'usages, plutôt intuitives pour les développeurs, sont adoptées, évitant la fastidieuse édition de fichiers XML et permettant donc de gagner un temps précieux.

L'environnement de développement est disponible très rapidement, le gain en **réactivité** est donc important.

> ***exemple***
>
> *Voir l'ORM dans la partie Active Record de ce document.*

1.2. Serveur Web intégré

Dans la phase de développement, Ruby on Rails permet de travailler avec le serveur web Ruby : "WEBrick".

Par une simple ligne de commande, le développeur exécute sur son ordinateur un serveur web. Il est **tout de suite opérationnel** et peut se consacrer entièrement au développement de l'application.

1.3. Un seul langage

Que ce soit le script de création de la structure de la base de données, le développement de la couche métier, l'écriture des templates de la Vue ou l'interactivité des pages (généralement en Javascript), tout peut être développé avec un seul et même langage : Ruby.

Ainsi combiné à Rails, Ruby est parfois qualifié de "langage de domaine[1]" permettant au développeur d'aller à l'essentiel en s'abstrayant des choses complexes.

[1] http://en.wikipedia.org/wiki/Domain-specific_language

Les concepts sont exprimés avec précision et concision grâce à un langage unique. Par conséquent, **le code est plus clair et plus facile à modifier.**

1.4. Générateurs

Rails est fourni avec de nombreux générateurs de code qui permettent d'élaborer rapidement une première version de l'application. Ce prototype peut être présenté très tôt et ainsi concrétiser les idées du développeur auprès des utilisateurs.
L'interaction avec le client se fait très tôt dans le cycle de développement.

> ***exemple***
>
> *Voir le générateur de scaffold dans la partie Action Controller.*

1.5. Scaffolding

Incontournables dans une application interagissant avec une base de données, les fonctions CRUD (Create, Read, Update et Delete) sont les phases élémentaires du cycle de vie des données. Rails mets à disposition du développeur un générateur appelé "scaffold" (échafaudage) offrant la possibilité de générer les Vues et Contrôleurs associés à ces fonctions, soit de façon dynamique (par l'appel d'une simple méthode), soit en statique (en produisant du code que le développeur pourra ensuite personnaliser à sa guise).

En une ligne de commande, les interactions élémentaires avec la base de données sont fonctionnelles et **les utilisateurs observent rapidement les évolutions** apportées au produit.

> ***exemple***
>
> *Voir le scaffold dans la partie Action Controller.*

1.6. DRY Don't Repeat Yourself

Un code plus concis est un code où les modifications se feront plus rapidement donc mieux acceptées par le développeur.

Rails implémente plusieurs fonctionnalités permettant au développeur de ne pas se répéter. Ainsi les éventuelles modifications ne se feront qu'à un endroit. L'appréhension de la "confrontation" avec les utilisateurs disparaît et les **rencontres avec les utilisateurs** deviennent des séances d'exploration des possibilités.

> ***exemple***
>
> *Voir Helpers, Layout, Pages partielles dans l'Action View.*

1.7. Test

La proportion de temps que doit représenter la phase de test dans le cycle de vie d'une application est toujours difficile à estimer. La tentation est parfois forte d'abréger cette étape pour respecter des délais contractuels.

RoR permet de gagner du temps, mais il ne fait pas abstraction de cette phase indispensable au bon développement d'un logiciel.

Il intègre des procédures de test permettant d'assurer la livraison d'un **logiciel fonctionnel** tout en acceptant les demandes d'évolution.

1.8. Rdoc

Les méthodes Agiles préconisent un "logiciel fonctionnel plutôt qu'une documentation complète". Cette recommandation n'implique pas la suppression de la documentation.

Grâce à la librairie Rdoc fournie avec Ruby, le développeur peut facilement créer une **documentation de l'application au format HTML**.

Pour être concis, Rdoc parcourt votre projet et référence l'ensemble des fichiers, les classes et les méthodes associées, et les descriptions ajoutées en commentaires par le développeur.

La documentation de Ruby on Rails est bien évidemment réalisée à l'aide de cette librairie, elle est consultable à l'adresse suivante : http://api.rubyonrails.com/.

2. Framework MVC

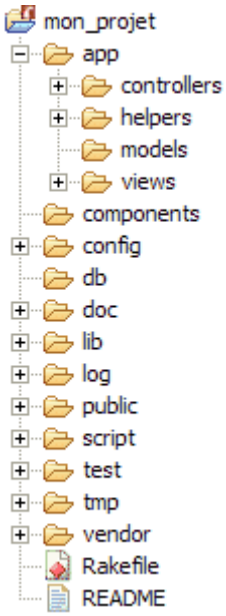
Le design pattern (motif de conception) MVC est proposé en 1979 par Trygve Reenskaug. Ce modèle distingue trois composantes dans une application :

- Le "**Modèle**" qui implémente la logique métier ;
- La "**Vue**" qui implémente l'interface utilisateur ;
- Le "**Contrôleur**" qui "orchestre" les interactions entre le Modèle et la Vue.

La séparation claire entre la logique métier et la présentation permet, par exemple, de pouvoir proposer plusieurs Vues pour un même traitement métier (distinguer un accès depuis un ordinateur d'un accès depuis un téléphone mobile). Elle rend surtout l'application plus simple à maintenir et à faire évoluer.

Ce modèle a depuis longtemps fait ses preuves et s'est largement répandu. De nombreux frameworks l'utilisent déjà de façon plus ou moins complète.

Ruby on Rails supporte le modèle dans son intégralité. Il impose au développeur l'utilisation d'une arborescence stricte qui répartit les fichiers contenus dans le dossier application en trois sous-dossiers : "Model", "View" et "Controller".



Le Modèle est géré par le composant "Active Record". La Vue et le Contrôleur sont gérés respectivement par l'**Action View** et l'**Action Controller**, regroupés dans l'**Action Pack**.

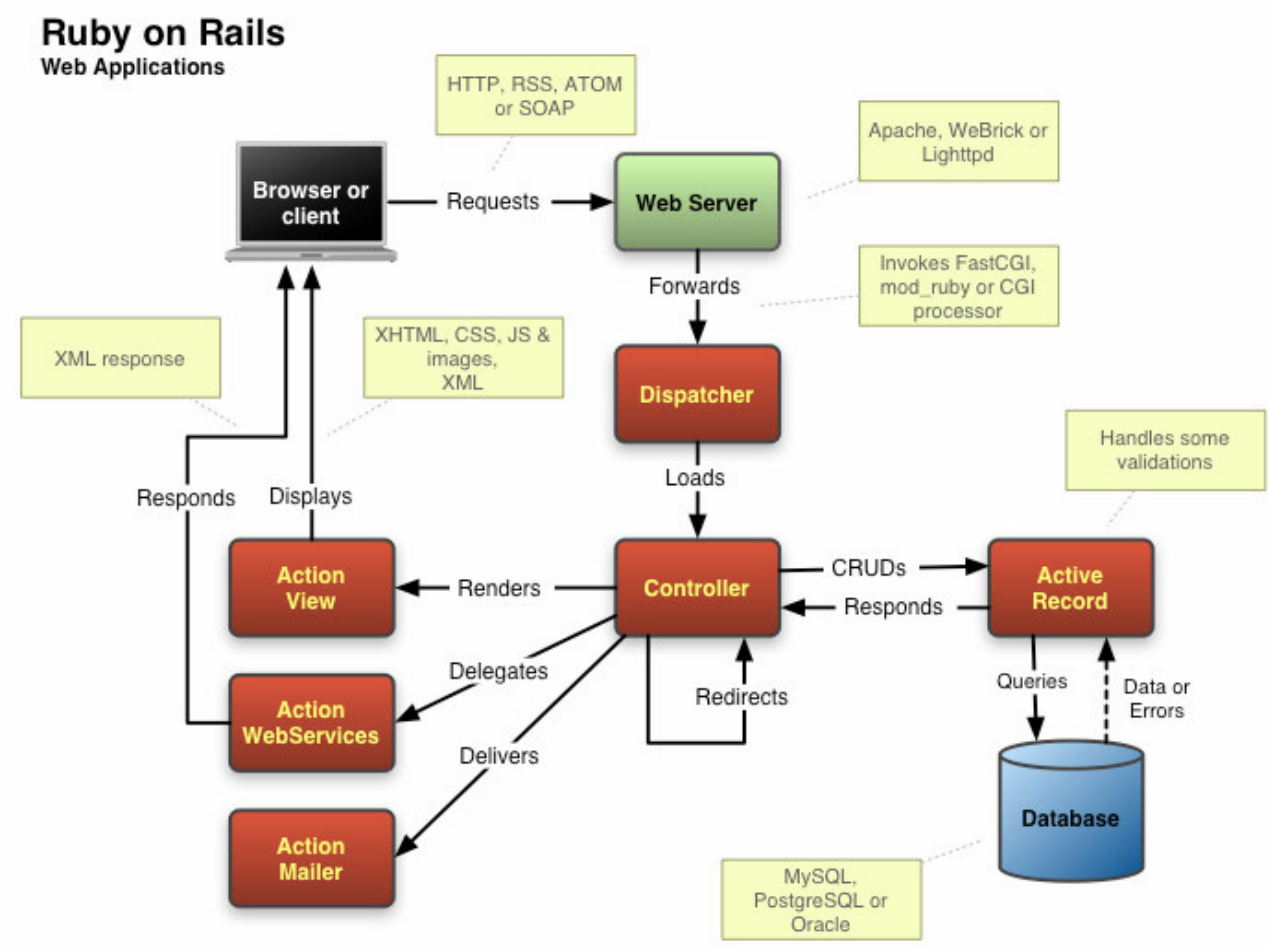


Image : http://rubyonrails.org/

3. Active Record

Comme nous venons de le voir, le "M" de MVC est le Modèle. Mais il peut aussi correspondre à "Métier". Il constitue en effet le cœur de l'application en englobant toute la logique métier. C'est pour cette raison, le point névralgique du développement.

Une des fonctions majeures du Modèle est l'interfaçage avec la base de données et plus précisément l'**object-relationnal mapping**.

3.1. Mappage objet-relationnel

L'ORM ("object-relationnal mapping" ou mappage objet-relationnel) permet d'établir automatiquement un lien entre la logique objet de l'application et le modèle relationnel de la base de données. Ce mappage est une fonctionnalité courante proposée par de nombreux frameworks. Elle nécessite une configuration importante, précisant par exemple quelle table sera transformée en quel objet, quel champ deviendra quel attribut…

Appliquant le principe édicté plus haut, de **"convention plutôt que configuration",** Ruby on Rails gère automatiquement ce mappage sans aucun fichier de configuration grâce à des conventions de nommage.

Les quelques règles à respecter sont par exemple : noms des tables au pluriel, "id" en clé primaire, "nomtable_id" en clé étrangère, déclaration des cardinalités dans le Modèle…

Bien que vivement conseillées, ces règles ne sont en aucun cas une obligation. Si le développeur le souhaite ou si l'environnement de travail l'impose (par l'utilisation d'une base de données existante ou le respect strict de la troisième forme normale de Boyce, Code et Kent[1]…), il est tout à fait possible de "configurer" Ruby on Rails pour travailler autrement.

Les données ainsi mappées sont directement utilisables dans le Modèle.

exemple

Pour un client donné, issu de la table "clients", la valeur du champ "nom" sera accessible sous la forme "Client.nom".

```
mon_client = Client.new
# on crée un nouveau client
mon_client.nom = "DURAND"
# son nom est DURAND
mon_client.prenom = "Jean"
# son prénom est Jean
mon_client.save
# le client est ajouté à la base de données.
```

[1] Les formes normales sont différents stades de qualité permettant d'éviter des anomalies dans les bases de données. La troisième forme normale de Boyce, Codd et Kent (3FNBCK) est l'une d'entre elles.

Dans cet exemple, la base de données est modifiée sans jamais avoir utilisé le langage SQL. L'ORM de l'Active Record permet de ne pas utiliser la syntaxe SQL pour les accès les plus fréquents à la base de données. Ainsi Ruby on Rails simplifie considérablement la manipulation des données. La syntaxe SQL sera réservée aux requêtes plus complexes.

exemple

```
Client.find(:all, :conditions => { :nom => 'DURAND' }, :limit => 10)
```

3.2. Associations et cardinalités

Le modèle relationnel est utilisé dans la conception des schémas de bases de données. Sa force est basée sur les relations qu'il permet d'établir entre les tables et les données qu'elles contiennent. L'affichage des données est donc souvent le fruit de jointures, imposant, lors de l'écriture de la requête SQL, de préciser quelles tables seront sollicitées et quels champs feront charnières. Nous parlerons alors de "clés étrangères".

Une fois de plus, RoR simplifie l'usage de ces interactions. Pour cela, il suffit d'indiquer dans les classes du Modèle comment sont associées les tables.

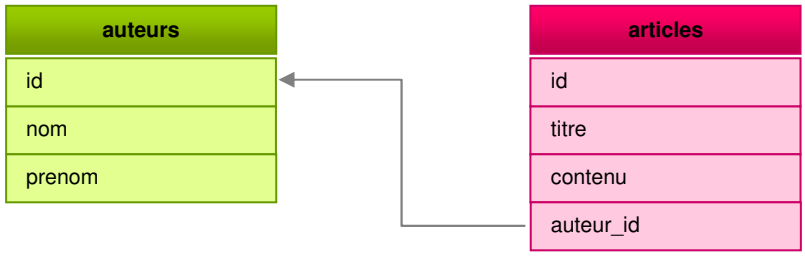
Quatre associations de bases sont disponibles :

- has_one (a un)
- has_many (a plusieurs)
- belongs_to (appartient à)
- has_and_belongs_to_many (a et appartient à plusieurs)

exemples

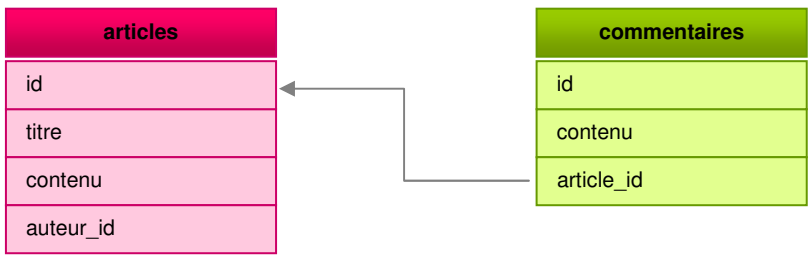
Belongs_to

```
Article belongs_to :auteur
```



Has_many

```
Article has_many :commentaires
```



Has_and_belongs_to_many

```
Article has_and_belongs_to_many :categories
```

Il est inutile de préciser quels champs serviront de clés primaires ou de clés étrangères puisque par convention, ce sont les champs "id" et "autretable_id" des différentes tables.

Exemple de déclaration :

```
Class Article < ActiveRecord::Base
      belongs_to :auteur
      has_many :commentaires
end
# Un article appartient à un auteur (au singulier)
# et possède plusieurs commentaires (au pluriel).
# Trois tables sont associées dans la base de données :
# la table articles, la table auteurs et la table commentaires.
```

Le nom de l'auteur de l'article sera accessible par "`article.auteur.nom`".
Un article pouvant recevoir plusieurs commentaires, les contenus respectifs de ces commentaires seront consultés par l'intermédiaire d'une boucle itérative ("for" par exemple) :

```
for le_commentaire in article.commentaires
      puts le_commentaire.contenu
end
```

3.3. Validation des données

Pour assurer la cohérence des données, Active Record dispose de plusieurs "validations" qui peuvent être ajoutées au Modèle. Ces méthodes permettent entre autre, de valider le contenu d'un objet avant de le sauvegarder dans la base de données.

- validates_presence_of ;
- validates_lenght_of ;
- validates_comfirmation_of ;
- validates_format_of…

exemple

```
Class Client < ActiveRecord::Base
      validates_presence_of :nom
end
# un client sera ajouté dans la base que si son nom est renvoyé par le formulaire de
# saisie.
```

3.4. Procédures de rappel : Callbacks

Active Record permet au programmeur d'intervenir tout au long du cycle de vie des objets d'un Modèle. Des actions peuvent être déclenchées à n'importe quel moment critique de ce cycle. Active Record définit seize procédures de rappel : sept paires avant/après et deux procédures "après".

1. before_validation / after_validation
2. before_validation_on_create / after_validation_on_create
3. before_validation_on_update / after_validation_on_update
4. before_save / after_save
5. before_create / after_create
6. before_update / after_update
7. before_destroy / after_destroy

1. after_find
2. after_initialize

Il suffit au développeur d'associer son code à la procédure de rappel ad hoc pour qu'il soit activé le moment venu.

3.5. Transactions

Les transactions garantissent le "tout ou rien". Elles permettent d'effectuer un ensemble d'actions et de garantir le bon déroulement de l'ensemble. Dans le cas contraire, si l'une des actions ne s'est pas exécutée convenablement, aucune des modifications n'est prise en compte. Le système est laissé dans l'état dans lequel il se trouvait avant le déclenchement de la transaction.

Active Record met à disposition la méthode "`transaction()`" pour exécuter un bloc de code monolithique.

Deux points d'attention :

1. Dans le cas de l'utilisation d'une base de donnée MySQL, il faut être vigilant et utiliser un moteur de table adapté aux transactions. En effet, tous les moteurs proposés par MySQL ne sont pas transactionnels (MyISAM par exemple). Il est donc préférable d'utiliser le moteur InnoDB si cette fonctionnalité est nécessaire.

2. Rails ne supporte pas les validations à deux phases distribuées (two-phase commit[1]). En d'autres termes, il ne sait pas gérer la synchronisation des transactions entre plusieurs bases de données.

[1] Technique assurant qu'une transaction a fait avec succès les mises à jour dans un environnement de bases de données réparties. Tous les systèmes de gestion de base de données (SGBD) concernés confirment dans la première phase que la transaction a été reçue et est récupérable. Dans la deuxième phase, chaque SGBD indique de valider la transaction.

4. Action Controller

"Chef d'orchestre" de l'application, le "Contrôleur" reçoit la requête du client, la fait traiter par le "Modèle" et renvoie le résultat au travers de la "Vue" la plus adaptée.

Pour que le processus soit amorcé, il faut donc que le Contrôleur reçoive la demande du client. Cette première action n'est autre que la saisie de l'URL.

4.1. URL

Vecteur de la demande, l'URL permet aux clients de s'adresser aux différents Contrôleurs de l'application.

Les URLs traitées par les applications basées sur Rails ont la forme suivante :

"www.mon_application.com/controller/action/id"

En d'autres termes :

1. on s'adresse à l'application, exemple : www.mon_application.com
2. puis au Contrôleur, exemple : /clients
3. puis on précise l'action à effectuer, exemple : /visualiser
4. enfin on indique la valeur de la/des variables, exemple : 5

Ainsi, l'adresse "www.gestion_client.com/clients/visualiser/5" renvoie une page HTML (la Vue), contenant les caractéristiques du client ayant l'identifiant n°5 dans la base de données.

L'application dispose donc d'URLs propres, conformes aux règles d'accessibilité du Web, améliorant de plus le référencement des pages internes auprès des moteurs de recherches.

4.2. Filtres

Les filtres sont déclarés au sein des Contrôleurs. A la manière des callbacks pour la partie Modèle, ils permettent de déclencher l'exécution de code à des moments précis liés aux actions. Trois types de filtres sont mis à disposition :

- les pré-filtres ;
- les post-filtres ;
- les péri-filtres.

Comme leur nom l'indique, les pré-filtres agissent avant le démarrage de l'action, les post-filtres après et les péri-filtres autour, c'est-à-dire avant et après.
Cas le plus évident de l'utilisation d'un pré-filtre, l'authentification peut être déclenchée avant chaque appel d'une action protégée. Le post-filtre peut-être quand à lui utilisé pour incrémenter des statistiques d'utilisation, par exemple. Les péri-filtres encadrent l'action par une méthode before() et after() pouvant, par exemple, démarrer et arrêter un chronomètre permettant de mesurer le temps d'exécution de la requête.

4.3. Vérification

La vérification permet de s'assurer de la présence de certains pré-requis avant l'exécution d'une action. "`verify()`" offre pour ce cas précis une syntaxe plus simple que l'utilisation d'un pré-filtre.

exemple

Dans une application de e-commerce, la vérification peut être utilisée pour valider la présence d'un identifiant de session avant d'afficher le contenu d'un panier d'achat. Dans le cas contraire, un message d'erreur peut être déposé dans le "Flash".

4.4. Communication entre actions : Flash

Le mot flash, au sein d'une application Ruby on Rails, n'a aucun rapport avec le format d'animation proposé par Macromedia (devenu Adobe). Plus proche de son sens premier, un flash est un message d'alerte. C'est en fait un tableau associatif, stocké dans la session de l'utilisateur, où l'action en cours peut ajouter des données. Si le flash contient des informations, elles seront disponibles pour l'action suivante, puis effacées (par défaut).

L'usage le plus courant de cette fonctionnalité est le passage de messages d'erreurs.

4.5. Sessions

Le protocole HTTP est un protocole asynchrone non connecté, cela sous entend que d'une requête à l'autre, le client n'est pas automatiquement reconnu par le serveur, donc non reconnu par l'application qu'il héberge.

Pour combler cette lacune, les applications web utilisent les sessions.

Dans Rails, la session se comporte comme un tableau associatif. Stockée sur le serveur, la session permet d'enregistrer des données qui resteront accessibles entre deux requêtes. La session est identifiée par une clé enregistrée sur le poste client dans un cookie.

exemple

```
session[:montant_panier]  = 100
```

5. Action View

Une fois traitée, l'information doit être retournée à l'utilisateur. Le rendu de l'information est géré par l'Action View, qui la plupart du temps encapsulera les données dans du code HTML qui sera interprété par le navigateur de l'utilisateur.

Ruby on Rails intègre un moteur de "template" facilitant la séparation du fond et de la forme.

5.1. Moteur de Template

Par défaut, Rails propose deux formats de sortie : RXML et RHTML.

Les fichiers RXML s'appuient sur la bibliothèque "Builder" pour renvoyer des réponses en XML.

Pour la génération des pages HTML, c'est le format RHTML, mélangeant code HTML et code Ruby, qui est utilisé. Rails s'appuie sur un outil Ruby appelé Erb (Embedded Ruby) pour l'interprétation. Toute les données statiques sont écrites en HTML et le code Ruby inséré dans des balises "<% %>" s'occupe des données dynamiques.

Cette approche sera familière aux développeurs utilisant déjà PHP, ASP ou JSP.

5.2. Helpers

Les "Helpers" sont des assistants dont la principale tâche est de limiter un maximum le code Ruby à écrire dans les formats des Vues (conformément au principe DRY). Ainsi le format sera essentiellement composé de HTML (ou de XML). Rails est livré avec un grand nombres d'assistants facilitant :

le formatage des données :

```
<%= number_with_delimiter(12345678) %>
⇨ 12,345,678
```

les liens entres les pages :

```
<%= link_to "Afficher le dossier du client", { :action => visualiser, :id
=> @client } %>
⇨ <a href="/gestion_client/visualiser/5"> Afficher le dossier du client
</a>
```

la pagination :

le paginateur est déclaré avant dans le controller :

```
@client_pages, @clients = paginate(:clients, order_by => 'nom')
```

puis dans la Vue :

```
<%= pagination_links(@client_pages) %>
⇨ <a href="/gestion_client/list?page=1">Previous page</a><a href="/
gestion_client/list?page=3">Next page</a>
```

la gestion des erreurs :

```
<%= error_message_for( :client ) %>
```

la création de formulaire :

si la variable "@clients" contient une sélection de noms :

```
select(:client, :nom, @clients)
⇨ <select id="client_nom" name="client[nom]">
<option value="DUPONT">DUPONT</option>
<option value="ROGER">ROGER</option>
<option value="ALBERT">ALBERT</option>
<option value="DUMON">DUMON</option>
<option value="MIRAN">MIRAN</option>
</select>
```

et encore d'autres…

5.3. Scaffolding

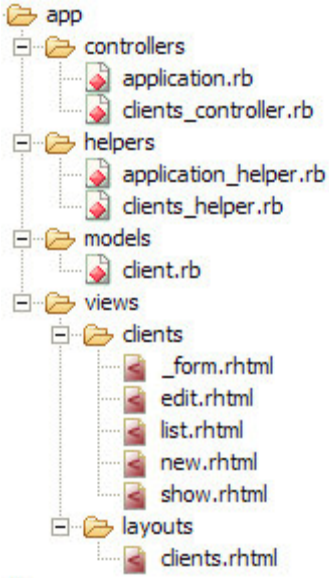
Les Vues associées aux fonctions élémentaires d'interactions avec la base de données peuvent être très rapidement développées. Le générateur de "scaffold" permet de générer, à la volée ou en dur, les Vues nécessaires à la création, l'affichage, la modification et la suppression de données.

Par l'intermédiaire de la simple commande "`ruby script/generate scaffold nom_du_modèle`", le framework génère le Contrôleur et les Vues permettant d'effectuer les actions de création, de lecture, de mise à jour et de suppression, sur les objets liés au Modèle indiqué dans la commande.

exemple

```
c:\> ruby script/generate scaffold client
exists app/controllers/
exists app/helpers/
create app/views/clients
exists test/functional/
dependency model
exists app/models/
exists test/unit/
exists test/fixtures/
identical app/models/client.rb
identical test/unit/client_test.rb
identical test/fixtures/clients.yml
create app/views/clients/_form.rhtml
create app/views/clients/list.rhtml
create app/views/clients/show.rhtml
create app/views/clients/new.rhtml
create app/views/clients/edit.rhtml
create app/controllers/clients_controller.rb
create test/functional/clients_controller_test.rb
create app/helpers/clients_helper.rb
create app/views/layouts/clients.rhtml
create public/stylesheets/scaffold.css
```

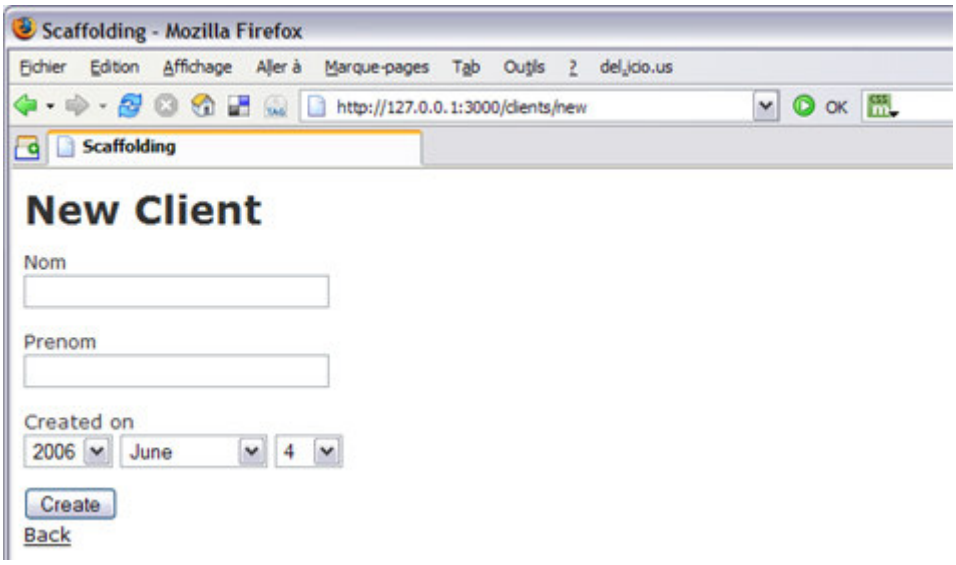
Résultat dans l'arborescence de notre application :



Dans le dossier "controllers" est ajouté un fichier "clients_controller.rb" qui contient six méthodes nous permettant de manipuler les données de la table "clients" :

- **index** : renvoie vers "list" ;
- **list** : récupère le contenu de la table et l'affiche via "list.rhtml" ;
- **show** : récupère le contenu d'un enregistrement et l'affiche via "show.rhtml" ;
- **new** : instancie un nouvel objet et affiche le formulaire de saisie "new.rhtml" ;
- **create** : enregistre les données du formulaire de saisie dans un nouvel enregistrement de la base ;
- **edit** : récupère le contenu d'un enregistrement et l'affiche dans un formulaire de saisie via "edit.rhtml" ;
- **update** : stocke les données du formulaire de saisie en modifiant l'enregistrement dans la base ;
- **destroy** : supprime un enregistrement dans la base.

Pour créer un nouveau client il suffit à l'utilisateur de se rendre à l'adresse : "http://127.0.0.1:3000/clients/new" :



Dans l'exemple que nous venons de voir, le fichier "_form.rhtml", créé par le générateur, ne semble pas appelé par le Contrôleur. Il est en fait utilisé dans l'exécution de deux méthodes : "new" et "edit". Ce fichier est en fait une "page partielle".

5.4. Pages Partielles

Conformément à la règle "DRY" (Don't Repeat Yourself), le développeur peut créer des portions de Vue qui seront intégrées à d'autres lors de l'affichage.

Dans l'exemple précédent le fichier "_form.rhtml" contient un formulaire de saisie. Ce formulaire étant commun aux deux actions de création et de modification, il est extrait des Vues et seule une référence à ce fichier est présente dans "new.rhtml" et "edit.rhtml". Ainsi, si le développeur souhaite modifier le contenu du formulaire, il ne le fera qu'une fois au lieu de deux.

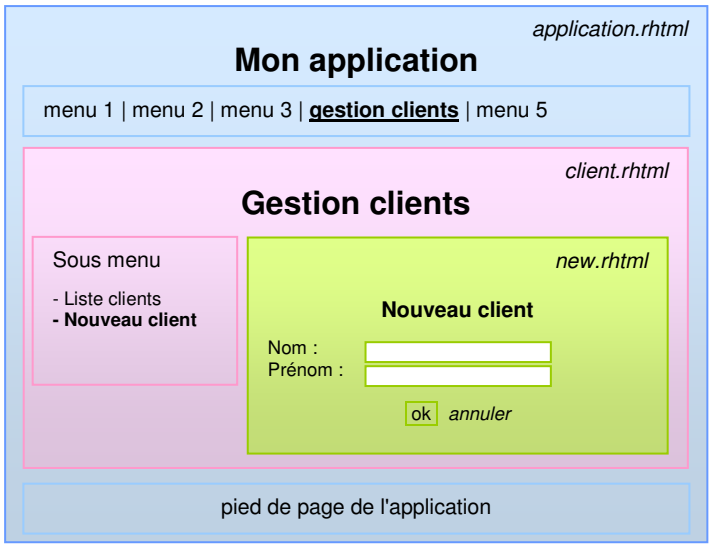
5.5. Mise en page (layouts)

La règle "DRY" est aussi appliquée sur la composition globale des pages affichées dans notre application.
En plus des dossiers contenant les Vues invoquées par les Contrôleurs, le dossier "views" contient un répertoire "layouts". Dans ce répertoire sont stockées différentes "couches" de l'interface de l'application.
Il est d'usage courant dans la création de site internet d'utiliser par exemple, une entête, un pied de page et un menu identiques sur l'ensemble des pages.
Rails propose de centraliser ces différentes mise en pages dans le dossier "layouts" et de les invoquer à la demande plutôt que de répéter du code sur toutes les Vues.

exemple

Le fichier "application.rhtml" pourra contenir l'entête, le menu et le pied de page de notre application.
Le fichier "client.rhtml" pourra lui définir un fond et une illustration qui seront affichés dans toutes les Vues liées aux actions sur les données clients.

Il est possible d'indiquer dans chacun des fichiers "Contrôleur", quels seront les "layouts" à utiliser, voire même de le préciser au niveau des méthodes.



5.6. Composants

Troisième représentant majeur de l'application de la règle "DRY", le concept de composant. RoR permet à une action appartenant à un contrôleur d'appeler une autre action appartenant à un Contrôleur différent.

C'est le cas par exemple du panier utilisateur sur un site de commerce qui sera présent sur toutes les pages visitées par le client. Autre cas de figure, dans un blog, la liste des derniers articles pourra être appelée sous la forme d'un composant et sera présente sur toutes les pages du site.

5.7. Ajax et RJS

AJAX ou "Asynchronous JavaScript And XML" est une des pierres angulaires du "Web 2.0". Derrière cet acronyme se cache non pas une nouvelle technologie mais plutôt l'utilisation conjointe d'un ensemble de technologies :

- HTML (ou XHTML) pour la structure sémantique des informations ;
- CSS pour la mise en forme des informations ;
- DOM et JavaScript pour intéragir dynamiquement avec l'information présentée ;
- XML, XSLT et surtout l'objet XMLHttpRequest pour échanger les informations avec le serveur web.

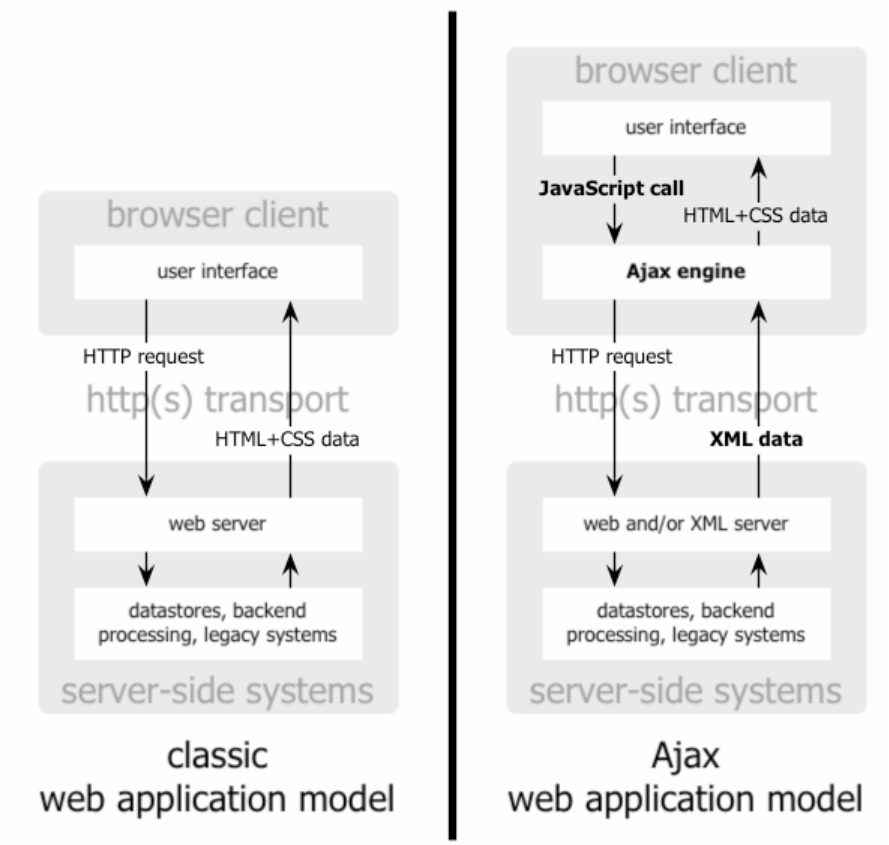


Image : http://www.adaptivepath.com

Les pages internet utilisant ces technologies offrent une meilleure interactivité : les données présentées peuvent être rafraîchies sans rechargement de la page, des effets visuels peuvent animer les pages, les utilisateurs peuvent utiliser le glisser-déposer… La navigation est plus souple et l'information est présentée de manière plus attractive.

Pour faciliter l'utilisation des technologies AJAX, le framework Ruby on Rails met à disposition des développeurs un ensemble de méthodes. Ces méthodes permettent entre autre, d'utiliser les bibliothèques JavaScript telles que Prototype, sans écrire de javascript.

exemple

L'ajout de la ligne "`<%= javascript_include_tag "prototype", "effects"%>`" dans le fichier "app/view/layouts/application.rhtml" permet de disposer sur l'ensemble des pages de l'application des bibliothèques javascript "prototype" et "effect". Ces bibliothèques permettent de modifier le contenu de la page sans rechargement et d'ajouter des effets visuels à l'apparition de ces modifications.

Il suffit ensuite d'appeler ces fonctions dans les pages :

```
<h2><%= link_to_remote client.nom,
      :update => "info_client_#{client.id}",
      :complete => "new Effect.Appear('info_client _#{article.id}')",
      :url => { :action => 'show', :id => client } %></h2>

<div id="info_client _<%= client.id %>"></div>
```

1. La page affiche dans un titre (`<h2>`) un lien portant le nom du client (`client.nom`). Ce lien particulier (`link_to_remote`) va déclencher le changement (`:update`) du contenu de la balise `<div>`.

2. Le nouveau contenu sera fourni par la page "`/clients/show/id_du_client`" (`:url => { :action => 'show', :id => client }`).

3. Cette modification sera agrémentée d'un effet visuel appelé "Appear" (`:complete => "new Effect.Appear('info_client _#{article.id}')"`)

Depuis la version 1.1, Rails va plus loin en intégrant un nouveau format de sortie : le RJS. Le RJS permet de faire du Javascript sans jamais écrire de JavaScript.

En agissant sur l'objet "Page", il est possible avec RJS de manipuler en Ruby, les différents composants de la page HTML présentée à l'utilisateur. Lorsque la Vue est générée, le code Ruby est transformé en JavaScript et inclus dans la page.

Avec RJS, le développeur peut s'il le souhaite, n'utiliser que du Ruby tout le long du développement de l'application.

6. Action Mailer

En plus des trois composants élémentaires du modèle MVC, Ruby on Rails met à disposition du développeur d'autres composants.

Parmi eux, le composant Action Mailer permet à une application d'envoyer et de recevoir très simplement des mails.

Pour l'envoi, Ruby on Rails permet de créer facilement des mails. Comme pour la création de Vues, le script "generate" permet de générer des formats qui seront utilisés pour la mise en forme. Une classe dérivée de ActionMailer est produite dans le dossier "model" qui permettra de créer un objet message. Cet objet sera instancié par le Contrôleur, qui activera ensuite l'envoi du mail.

Pour la réception, la création d'une méthode de classe "`receive()`" permettra à notre application de traiter le contenu des emails reçus. Reste deux problèmes à résoudre : intercepter le message et le rediriger vers cette méthode ?

Pour nous faciliter la tâche, RoR met à disposition le script "runner". La commande "runner" procure un moyen d'intéragir avec notre application sans passer par le navigateur.

Ici s'arrête le développement Web ! En effet, c'est le serveur de messagerie qui reçoit les emails, à charge pour lui de l'identifier parmi les autres, pour ensuite l'adresser à notre application en invoquant le script "runner".

7. Action Web Service

Quand on pense application Web, on pense principalement intéractions avec les utilisateurs. Mais pour devenir un service web à part entière, l'application doit aussi être capable de dialoguer avec d'autres applications, donc disposer d'une API (Interface de Programmation d'Application).

C'est à ce besoin que répond le module "Action Web Service". Il supporte les protocoles SOAP et XML-RPC, permettant à notre application de proposer ces deux interfaces, si nécessaire.

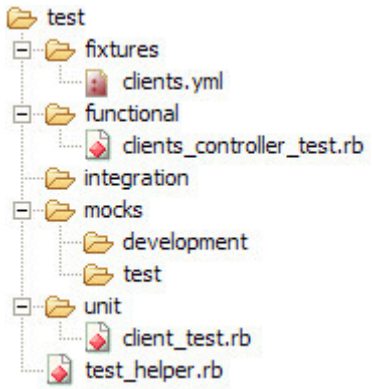
A l'inverse, ce même module nous permettra aussi d'ajouter des services à notre application en offrant quelques classes pour accéder aux API externes en tant que client.

8. Test

La phase de tests est une phase délicate dans le développement d'une application. Bien que les tests ne permettent pas d'affirmer que l'application est exempte de tous bugs, ils contribuent fortement à la qualité de l'application. Une fois mises en place, les procédures de test permettent de détecter rapidement les impacts négatifs que peuvent avoir les mises à jour du code de l'application. Elles facilitent ainsi les réactions aux changements chères aux méthodes Agiles.

Dans RoR, les tests font partie intégrante du cycle de développement. La notion de test apparaît dès la configuration de l'accès à la base de données où sont indiqués trois types d'environnements : l'environnement de développement, de test et de production.

Tout au long du développement, sans que le développeur ait à s'en occuper, Rails complète le répertoire "test". A chaque utilisation du générateur de code, le framework crée en parallèle du Contrôleur ou du Modèle généré, un fichier contenant l'ossature du test correspondant. Ainsi les tests peuvent être développés en même temps que l'application.



Les tests unitaires s'appuient sur le framework de test Ruby "Test::Unit" et portent sur les Modèles. Les tests fonctionnels évaluent les Contrôleurs, et depuis la version 1.1, rails propose aussi les tests d'intégrations qui permettent de valider le fonctionnement de l'ensemble.

9. Sécurité

La sécurité est un point sensible dans tout développement, mais la mise à disposition d'une application sur internet accroît fortement les risques. Rails propose de combler automatiquement certaines failles parmi les plus fréquentes sur les applications web.

En voici deux exemples :

9.1. Injection SQL

Cette attaque consiste à envoyer du code SQL via un des formulaires de saisie du site internet. Ainsi le pirate peut, par exemple, passer outre une page d'identification en saisissant une requête SQL spécifique dans le champ "identifiant utilisateur".

Pour éviter ce genre d'attaques, toutes les fonctions prédéfinies dans Active Record excluent automatiquement tous les caractères spéciaux nécessaires aux requêtes SQL.

9.2. Attaques XSS

Le Cross Site Scripting, abrégé XSS (pour ne pas confondre avec les feuilles de style CSS), consiste à injecter des scripts malveillants dans les données du site internet pour dérober, par exemple, les cookies ou informations de session distribués par ce site. Pour protéger l'application, il est préconisé d'utiliser un assistant fourni par RoR : "`h()`" ou "`html_escape()`". Cet assistant "échappe" le contenu qui sera affiché par les Vues. Ainsi, le code contenu n'est pas interprété.

Si du HTML doit absolument être interprété, la méthode sanitize() est proposée pour rechercher les cas d'attaques connues. Il est évident que cette méthode doit être utilisée avec précaution car de nouvelles attaques sont créées régulièrement.

10. Changement d'échelle

La résistance à la montée en charge et la scalabilité (changement d'échelle) sont des critères déterminant de la qualité d'une application.

Face à la montée en charge, RoR intègre un gestionnaire de cache permettant d'accélérer considérablement le rendu des pages[1].

Si l'application est fortement sollicitée et que cette première étape ne suffit pas, seule la duplication des serveurs permettra d'en assurer la disponibilité. Dans ce type d'architecture plusieurs serveurs sont mis à contribution et se partagent les requêtes des utilisateurs.

Cette répartition implique des modifications. Les utilisateurs ne s'adressant pas forcement au même serveur d'une requête à l'autre, les informations de l'utilisateurs (liées aux cookies) ou le cache doivent être disponibles pour tous les serveurs. Rails accepte naturellement ce type de configurations facilitant le changement d'échelle des applications.

Autre conséquence de la répartition : la duplication des mises à jour.
L'application étant dupliquée sur plusieurs serveurs, ses mises à jour devront l'être aussi ! Pour effectuer cette tâche aisément, Rails utilise un outil nommé Capistrano (ancien SwitchTower). Capistrano automatise les tâches de déploiement, il permet de transférer l'application et ses mises à jour d'une simple commande, depuis le poste du développeur, vers un ou plusieurs serveurs. Les versions successives sont identifiées et les versions posant problèmes peuvent être rapidement désinstallées.

[1] Voir l'article de "Some Numbers at Last" de J. GEHTLAND où une des fonctionnalités de son application passe d'un traitement de 75.59 requêtes/seconde à 1754.39 requêtes/seconde en utilisant le cache. (http://blogs.relevancellc.com/articles/2005/04/04/some-numbers-at-last.)

Partie 3
Une solution crédible ?

"Java meets PHP : Architectural purity of Java, Easy immediacy of PHP."
David Heinemeier Hansson

Ruby on Rails est né en juillet 2004, c'est donc encore un produit jeune. Mais en presque deux ans, il a énormément gagné en maturité, en grande partie grâce à une communauté très active. La version 1.0 est sortie le 14 décembre 2005, suivie de près par la version 1.1 datée du 26 mars 2006.

Son évolution rapide et ses performances font beaucoup parler. Ruby on Rails est donc régulièrement comparé aux solutions existantes : PHP, Java, J2EE, .NET…etc. Langages, frameworks, plateformes applicatives, toutes les comparaisons ne sont pas forcément pertinentes.

Ruby on Rails a certes de nombreux atouts, mais il n'est bien évidemment pas une solution miracle.

1. Comparer ce qui est comparable

1.1. Solutions open source

Les solutions open sources sont idéales pour développer des applications sans investissements financiers importants. Elles ont longtemps été synonymes de projets d'amateurs et de petites applications. Aujourd'hui, ce cliché n'est plus justifié car l'open source se développe fortement et s'installe dans les grandes entreprises[1].

Parmi ces solutions, PHP est le langage de script le plus utilisé pour développer des pages web dynamiques. Simple et rapide à mettre en œuvre, il reste souvent réservé aux petites applications même s'il dispose de frameworks de plus en plus robustes. Pour les applications plus complexes, nécessitant un développement plus structuré, les entreprises préfèrent s'orienter vers des solutions utilisant le langage Java plus apprécié des décideurs. Les applications sont donc basées sur des framework comme Struts, Spring ou Hibernate.

Ruby on Rails peut perturber cet ordre établi. Il permet en effet de développer une application de manière structurée, respectant le Modèle MVC, aussi rapidement et simplement qu'avec le langage PHP, sans la lourdeur et la complexité des frameworks Java.

1.2. Solutions commerciales

Les plateformes J2EE et .NET sont prédominantes dans les grandes entreprises[2]. Ces plateformes sont la base d'applications sensibles, elles permettent une forte intégration au système d'information existant et une gestion accrue des systèmes transactionnels.

Que ce soit WebSphere d'IBM, WebLogic de BEA Systems pour J2EE, ou les produits Microsoft pour .NET, ces plateformes sont distribuées avec de nombreux outils très

[1] Une étude de Forrester d'avril 2006 montre que J2EE et .NET n'ont pas atténué le développement des solutions open source dans les entreprises (http://www.01net.com/article/311832.html).
[2] *Ibid.*

puissants (IDE, interface d'administration…) mais nécessitent de nombreuses heures de formation et sont complexes à maîtriser dans leur globalité.

De telles plateformes sont difficilement comparables avec Ruby on Rails, tant au niveau des fonctionnalités qu'au niveau des coûts engagés ! Comme nous l'avons vu dans la première partie de ce document, Ruby on Rails est un outil volontairement simple, apportant de la souplesse et s'adressant en priorité aux petites équipes.

2. Eléments de décision

Pour synthétiser les nombreuses comparaisons entre ce framework et les autres solutions du marché, voici une liste récapitulative des principaux critères décisifs se rapportant à Ruby on Rails :

- ✚ prêt pour le Web 2.0, il facilite le développement d'interface riche (AJAX) ;
- ✚ courbe d'apprentissage faible ;
- ✚ un seul langage ;
- ✚ code concis ;
- ✚ réactivité accrue ;
- ✚ support des services Web ;
- ✚ modèle MVC ;
- ✚ open source :
  - + indépendance vis-à-vis des éditeurs,
  - + utilise des standards,
  - + pas de coûts logiciels.

- ➖ peu de solutions d'hébergement ;
- ➖ peu de sociétés de services ;
- ➖ manque de références entreprises (produits de 37signals mis à part) ;
- ➖ faible support de l'internationalisation ;
- ➖ prend toute son ampleur sur de nouveaux projets.

Cette liste est encourageante, car les principaux défauts reprochés à Ruby on Rails sont des défauts liés à la jeunesse du produit.

3. Quelques références

De nombreux sites internet ont déjà été développés avec Ruby on Rails, parmi les plus fréquentés on peut citer :

- ➲ **Basecamp** (www.basecamphq.com) : un des sites développés par 37signals. Cette application de gestion de projet compte plus de 250 000 utilisateurs enregistrés.

- ➲ **Odeo** (www.odeo.com) : le célèbre annuaire de podcast met à disposition de ses utilisateurs un million de références.

- **A List Apart** (www.alistapart.net) : ce magazine en ligne sur le webdesign comptabilise 14 millions de pages visitées par mois.

- **Eins.de** (eins.de) : ce site internet allemand dessert 1,2 millions de pages par jour (100Gb de trafic). Son architecture a récemment évolué[1] pour améliorer ses performances.

4. Vocations

Une chose est certaine, l'apparition de Ruby on Rails a provoqué de nombreuses vocations et tous les langages voient apparaître des frameworks copiant les principes de Rails :

.NET
- MonoRail (http://www.castleproject.org/index.php/MonoRail)

Java
- Wicket (http://wicket.sourceforge.net/)
- Sails (http://www.opensails.org/)
- Trails (https://trails.dev.java.net/)

Javascript
- TrimJunction (http://trimpath.com/project/wiki/TrimJunction)

Perl
- Catalyst (http://www.catalystframework.org/)

PHP
- Cake (http://cakephp.org/)
- Symfony (http://www.symfony-project.com/)
- Seagull (http://seagull.phpkitchen.com/)
- PHP on Trax (http://phpontrax.com/)

Python
- Django (http://www.djangoproject.com/)
- Fanery (http://fanery.python-hosting.com/)
- Subway (http://www.gosubway.org/)
- TurboGears (http://www.turbogears.org/)

Smalltalk
- Seaside (http://seaside.st/)

[1] "The adventures of scaling" : http://poocs.net/articles/2006/03/13/the-adventures-of-scaling-stage-1

Conclusion

Le monde du Web rencontre actuellement de nombreuses mutations. Dans ce nouveau Web aussi appelé Web 2.0, l'utilisateur ne doit plus subir les contraintes techniques. Respect des normes, ergonomie et accessibilité sont les maîtres mots. L'utilisateur est considéré différemment et son rôle évolue. Il n'est plus un simple consommateur, il devient auteur de l'information.

Preuve de cette volonté de proximité avec l'utilisateur, la majorité des applications récemment apparues sur le Web, sont proposées dès leur version "bêta" et confrontées à la validation des utilisateurs. Cette procédure, bien que courante dans le cadre du développement de logiciels, n'était pas encore utilisée sur internet. Elle est maintenant, quasiment devenue un critère d'obtention du label "Web 2.0".

Les entreprises voulant exister dans le domaine des services web doivent intégrer ces contraintes et considérer les demandes des utilisateurs. Elles seront donc tenues de faire preuve d'innovation et de réactivité pour se démarquer.

En terme d'innovation, l'utilisation de technologies comme AJAX, compense la réduction qualitative subie par les interfaces utilisateurs, lors du passage du client lourd au client web. Ces évolutions permettent aujourd'hui, de développer des clients "riches", proposant à l'utilisateur des interfaces plus adaptées et plus interactives.

Pour gagner en réactivité, les sociétés productrices préfèrent adopter la "stratégie Tetris[1]". Elles privilégient une réaction rapide, risquant d'être fausse, plutôt qu'une analyse poussée et exacte, entraînant un résultat trop tardif. Elles élaborent donc très tôt, des prototypes, qui subiront des modifications adaptées aux propositions des utilisateurs.

[1] http://fr.wikipedia.org/wiki/Effet_Tetris

Face à ces constats, le framework proposé par 37Signals parait tout à fait adapté. Pensé par une petite équipe pour de petites équipes, il concrétise les principes fondamentaux communs aux méthodes "Agiles". De plus, Ruby on Rails :

- permet au développeur de s'abstraire de nombreuses difficultés en exploitant au mieux les performances du langage Ruby sur lequel il s'appuie ;
- favorise le développement d'un code plus concis qui sera plus simple à modifier ou à maintenir ;
- facilite la réaction aux changements et l'adaptation aux demandes du client.
- apporte une souplesse de travail et de nombreuses fonctionnalités contribuant à l'amélioration de la productivité.
- intègre d'origine des librairies Javascript facilitant l'utilisation des technologies AJAX.
- permet d'enrichir les fonctionnalités de l'interface utilisateur grâce à RJS.

L'ensemble de ces capacités se loge dans un cadre de développement strict et éprouvé, sans pour autant ajouter de la lourdeur ou de la rigidité. Ruby on Rails réussit ainsi à faire cohabiter deux concepts jusque là antinomiques : "code structuré" et "rapidité de développement".

Il reste à savoir si malgré tous ses atouts, Ruby on Rails arrivera à pénétrer les entreprises. Majoritairement représenté par J2EE ou par la plateforme .NET, ce type de choix technique est étroitement lié à des décisions politiques, ou à des arguments de communications, face auxquels le langage Ruby n'est pas encore à son avantage.

De manière plus générale, le succès de cette solution passera forcément par l'augmentation des offres intégrant ce choix chez les hébergeurs. Car la moindre des choses pour une application internet est d'être disponible… sur internet !

L'apparition de multiples applications basées sur RoR et la récente décision de Telecom Italia de promouvoir cette solution[1], laisse à penser que l'utilisation de Ruby on Rails va continuer à progresser et que ce framework permettra d'écrire ("facilement") de nombreuses pages de l'histoire d'Internet.

[1] " Développement d'applications Web : Telecom Italia et Nuxos Group font cause commune pour «Ruby on Rails»" (http://entreprises.telecomitalia.fr/NET/document/newsletter_entreprises/ruby_on_rails/index.htm)

Glossaire

AJAX	"Asynchronous JavaScript And XML", acronyme désignant un ensemble de technologies (HTML, CSS, DOM, XML, XSLT, …) utilisées pour le développement de sites Web.
API	"Application Programming Interface", interface de programmation définissant la manière dont un composant informatique peut communiquer avec un autre.
CSS	"Cascading Style Sheets" ou feuilles de style en cascade, langage utilisé pour décrire la mise en forme d'un document HTML ou XML.
Design pattern	"Motif de conception", solution standard répondant à des problèmes de conception logicielle.
DOM	"Document Object Model" recommandation décrivant une interface permettant d'accéder ou de mettre à jour le contenu, la structure ou le style d'un document. DOM est principalement utilisé pour traiter et générer des documents XML.
DSDM	"Dynamic software development method", méthode "Agile" de gestion de projet, originaire de Grande Bretagne, née en 1994.
HTML	"Hypertext Markup Language" langage utilisé pour écrire les pages Web. Son nom est lié à sa capacité à créer des liens "hypertextes" entre les pages.
Javascript	Langage de script principalement utilisé coté client pour la manipulation des objets d'une page web.
RAD	"Rapid Application Development" méthode de gestion de projet parue dans les années 80, à l'origine du mouvement "Agile".
Rails, RoR	"Ruby on Rails".
RJS	Format de template proposé dans Ruby on Rails permettant "d'écrire du Javascript en Ruby"
RPC	"Remote Procedure Call" protocole permettant de faire des appels de procédures sur un ordinateur distant à l'aide d'un serveur d'application.
Scrum	méthode "Agile" de gestion de projet, née en 1996.
SOAP	"Simple Object Access Protocol " protocole RPC basé sur XML permettant le transfert entre objets distants.
Template	Gabarit permettant de séparer le fond de la forme, utilisé dans la conception de sites Web.
Web 2.0	Terme apparu en 2004 dont la définition précise est encore sujet à controverses. Il évoque une évolution du Web original regroupant une implication de l'utilisateur différente, le respect des standards, la mise en place d'interfaces utilisateurs plus riches et des aspects de réseaux sociaux.
XML	"Extensible Markup Language", standard servant à l'élaboration de langage de balisage. L'objectif initial de XML était de faciliter le partage de textes et d'informations structurées, par exemple au travers de l'Internet, en séparant le contenu (les données) du contenant (la présentation des données).
XML-RPC	Protocole RPC basé sur XML permettant le transfert entre objets distants. Ce protocole est plus simple que son successeur SOAP.
XP	"Extreme programming" méthode "Agile" de gestion de projet, née en 2000.

Références

1. Livres

Ruby on Rails
Dave Thomas, David Heinemeier Hansson
Traduction Laurent Julliard, Richard Piacentini
Editions Eyrolles.

Agile Web Development with Rails
Dave Thomas, David Heinemeier Hansson
The Pragmatic Programmers

Rails Recipes
Chad Fowler
The Pragmatic Programmers

Ruby for Rails *Ruby techniques for Rails developers*
David A. Black
Manning Publications

Programming Ruby
Dave Thomas
The Pragmatic Programmers

2. Internet

2.1. Notions générales

"Les serveurs d'application dans les architectures n-tiers" JDN Solutions (11/2003)
http://solutions.journaldunet.com/0311/031107_chronique.shtml
"Les frameworks pour les applications web"
Jean-Michel DOUDOUX (2005)
http://perso.orange.fr/jm.doudoux/java/tutorial/chap038.htm

2.2. Web 2.0

"AJAX, le talon d'achille de microsoft." Blog de Jean-Louis GASSET (03/2003)
http://gassee.blog.20minutes.fr/archive/2006/03/15/ajax-le-talon-d-achille-de-microsoft.html
"Ajax: a new approach to web applications" Adaptive Path (03/2005)
http://www.adaptivepath.com/publications/essays/archives/000385.php
"Les briques du web 2.0 par la pratique" Blog de Dominique Vilain (05/2006)
http://blog.domy.be/?2006/05/17/19
"Web 2.0 : mythes et réalités" Xml.fr (12/2005)
http://xmlfr.org/actualites/decid/051201-0001

"Web 2.0 : à l'origine du terme" Yades (09/2005)
http://www.yades.com/index.php/web-20-a-l-origine-du-terme
"Techniques de développement web : une révolution en cours ?" FredCavazza.net (03/2005)
http://www.fredcavazza.net/index.php?2005/03/30/632

2.3. .Net

".NET: dix arguments pour se décider" ZDNet.fr (01/2003)
http://www.zdnet.fr/entreprise/service-informatique/serveurs-stockage/0,50007198,2127705,00.htm
"Architecture .NET" DotNetGuru (2006)
http://www.dotnetguru.org/
"je débute avec Microsoft Visual Studio .NET" Pixel Technologie (2006)
http://www.pixel-technology.com/rthomas/
".NET : la plate-forme d'entreprise de Microsoft" Microsoft msdn (2006)
http://www.microsoft.com/france/msdn/net/decouvrez/default.mspx

2.4. Java et J2EE

"Hibernate Your Data" O'REILLY, On Java (01/2004)
http://www.onjava.com/pub/a/onjava/2004/01/14/hibernate.html
"Cours et articles de Serge Tahé" Developpez.com (2006)
http://tahe.developpez.com/
"Le framework Struts" Sysdeo (04/2003)
http://www.sysdeo.com/dossiers/le_framework_struts__1
"Livre blanc Framework J2EE" Sysdeo (2004)
http://www.sysdeo.com/sysdeo/content/download/380/4866/file/SYS-FWK-040801-1.2-Sysdeo-J2EE-Framework-Whitepaper.pdf
"Plate-forme J2EE: cinq critères pour faire le bon choix" ZDNET (06/2003)
http://www.zdnet.fr/entreprise/service-informatique/serveurs-stockage/0,50007198,2135287-2,00.htm

2.5. Comparaison

"Y . o . m . b . a . r, Rails from a .NET perspective" (2006)
http://dema.ruby.com.br/
"ASP.NET vs Ruby on Rails" (07/2005)
http://groups.google.com/group/comp.lang.ruby/browse_thread/thread/c20b0c9b400fd528/47655971c922b9e4?#47655971c922b9e4
"Ruby on Rails and J2EE: Is there room for both?" IBM (07/2005)
http://www-128.ibm.com/developerworks/web/library/wa-rubyonrails/
"Questions Ruby on Rails skeptics ask" Updrift (01/2006)
http://www.updrift.com/article/questions-ruby-on-rails-skeptics-ask
"Ruby the Rival" O'REILLY, On Java (11/2005)
http://www.onjava.com/pub/a/onjava/2005/11/16/ruby-the-rival.html?page=1
"Coming to Ruby from Java" Francis Hwang (11/2004)
http://fhwang.net/blog/40.html

"Best Tool For the Job » Alternatives to Ruby on Rails" Blog de Marcus VORWALLER (12/2005)
http://marcusvorwaller.com/blog/archives/2005/12/15/alternatives-to-ruby-on-rails/
"Chris Shiflett: Ruby on Rails Fans" Brain Bulb (02/2006)
http://shiflett.org/archive/190
"Ruby Kool-Aid, drink up" Blog de Nicholas JON (04/2006)
http://www.nicholasjon.com/archives/2005/4/26/ruby_kool_aid_drink_up
"J2EE et .Net n'ont pas éliminé leurs rivaux" 01net (04/2006)
http://www.01net.com/article/311832.html

2.6. Philosophie

"Don't Repeat Yourself: Layouts, Components, and Partials" Blog "The Rails Diary" (09/2005)
http://www.railsdiary.com/diary/dry_layouts_components_partials
"The Agile Web Design Manifesto, An Introduction" Emily Chang (02/2006)
http://www.emilychang.com/go/weblog/comments/the-agile-web-design-manifesto-an-introduction/
"Manifesto for Agile Software Development" (2001)
http://agilemanifesto.org/
"Méthode agile" Wikipédia (2006)
http://fr.wikipedia.org/wiki/M%C3%A9thode_agile
"Usability First: Usability in Website and Software Design" Usability First (2006)
http://www.usabilityfirst.com/index.txl

2.7. Ruby

"Ruby: Programmers' Best Friend" Site officiel de Ruby (2006)
http://www.ruby-lang.org/en/
"Why's (Poignant) Guide to Ruby" (2006)
http://poignantguide.net/ruby/
"Ruby" Wikipédia (2006)
http://fr.wikipedia.org/wiki/Ruby
"Introduction à Ruby" JDN Développeurs (03/2005)
http://developpeur.journaldunet.com/tutoriel/ruby/050318-ruby-intro.shtml
"Interview Laurent Julliard" Librairie Eyrolles (02/2006)
http://www.eyrolles.com/Informatique/Interviews/LJulliard/
"Syntaxe et originalités du langage" RubyFR (2006)
http://rubyfr.org
"Ruby 18e de l'indicateur Tiobe d'avril 2006" JDNet Développeurs (04/2006)
http://developpeur.journaldunet.com/breve/outils/1972/ruby-18e-de-l-indicateur-tiobe-d-avril-2006.shtml
"Ruby" autoblogholog (2006)
http://pagesperso.laposte.net/ibi/index.php?Ruby

2.8. Ruby on Rails

"Ruby on Rails" Site officiel (2006)
http://rubyonrails.org/

"Loud Thinking" Blog de David Heinemeier Hansson (2006)
http://www.loudthinking.com/
"A Quick Introduction to Rails" Jim WEIRICH (2006)
http://onestepback.org/articles/quickrails/takahashi.xul?data=quickrails.data#page1
"StartAtTheBeginning in Ruby on Rails" Wiki Ruby on Rails (07/2005)
http://wiki.rubyonrails.org/rails/pages/StartAtTheBeginning
"HowtoDesignYourRailApplication in Ruby on Rails" Wiki Ruby on Rails (05/2006)
http://wiki.rubyonrails.org/rails/pages/HowtoDesignYourRailApplication
"Integration Testing in Rails 1.1" Blog de Jamis JAMISBUCK (03/2006)
http://jamis.jamisbuck.org/articles/2006/03/09/integration-testing-in-rails-1-1
"Let Java retire from the spotlight of web applications in dignity" Indic Threads Java J2EE Portal (01/2006)
http://www.indicthreads.com/content/view/390/0/1/0/
"Interview de Yann Klis (Novelys)" JDN Développeurs (02/2006)
http://developpeur.journaldunet.com/itws/060202-itw-novelys-klis.shtml
"Chat L. Julliard et R. Piacentini" JDN Développeurs (03/2006)
http://developpeur.journaldunet.com/itws/060324-itw-rubyonrails-julliard-piacentini.shtml
"Rails 1.1: The Forgotten Features" Mike Clark's Weblog (03/2006)
http://clarkware.com/cgi/blosxom/2006/03/28#Rails11
"Petit résumé de l'état de l'art Ruby/Rails" Blog Julien Carnelos (09/2005)
http://julien.carnelos.free.fr/blog/index.php?2005/09/20/11-petit-resume-de-l-etat-de-l-art-ruby-rails-ror-pour-les-intimes
"Rails 1.1: Loaded with 37s extractions" Signal vs. Noise (by 37signals) (03/2006)
http://37signals.com/svn/archives2/rails_11_loaded_with_37s_extractions.php
"Rails 1.1: RJS, Active Record++, respond_to, integration tests, and 500 other things!" Weblog Ruby on Rails (03/2006)
http://weblog.rubyonrails.org/articles/2006/03/28/rails-1-1-rjs-active-record-respond_to-integration-tests-and-500-other-things
"Ruby on Rails, nouveau trublion du développement web" 01net (04/2006)
http://www.01net.com/article/312277.html

2.9. Performances

"Some Numbers at Last" Relevance (04/2005)
http://blogs.relevancellc.com/articles/2005/04/04/some-numbers-at-last
"The adventures of scaling" Poocs.net (03/2006)
http://poocs.net/articles/2006/03/13/the-adventures-of-scaling-stage-1
"Ruby on Rails" Telecom Italia (2006)
http://entreprises.telecomitalia.fr/NET/document/newsletter_entreprises/ruby_on_rails/index.htm

2.10. Déploiement

"Capistrano Deployment on Dreamhost" Ruby on Rails for Newbies (2006?)
http://nubyonrails.com/pages/shovel_dreamhost
"Two Words: rake deploy" Economy Size Geek (11/2005)
http://www.economysizegeek.com/2005/11/08/two-words-rake-deploy/
"Ruby on Rails on Oracle: A Simple Tutorial" Oracle (04/2006)
http://www.oracle.com/technology/pub/articles/haefel-oracle-ruby.html

2.11. Reflexion

"EvaluatingRuby" Martin FOWLER (05/2006)
http://martinfowler.com/bliki/EvaluatingRuby.html
"Why Ruby on Rails won't become mainstream" Otaku, Cedric's weblog (04/2006)
http://beust.com/weblog/archives/000382.html
"Are you sure you want to be mainstream?" Loud Thinking (04/2006)
http://www.loudthinking.com/arc/000584.html

2.12. Outils

"RadRails IDE" (2006)
*http://*www.radrails.org/
"Scite" (2006)
http://www.scintilla.org/SciTE.html
"Komodo" activestate (2006)
http://www.activestate.com/Products/Komodo/?_x=1

2.13. Essayez !

"Try Ruby !" (2006)
http://tryruby.hobix.com/
"Top 12 Ruby on Rails Tutorials" Digital media minute (11/2005)
http://www.digitalmediaminute.com/article/1816/top-ruby-on-rails-tutorials
"Créez votre blog avec Rails !" Librairie Eyrolles (03/2006)
http://www.eyrolles.com/Informatique/Articles/Rails/blog.html?societe=piacentini

Annexes

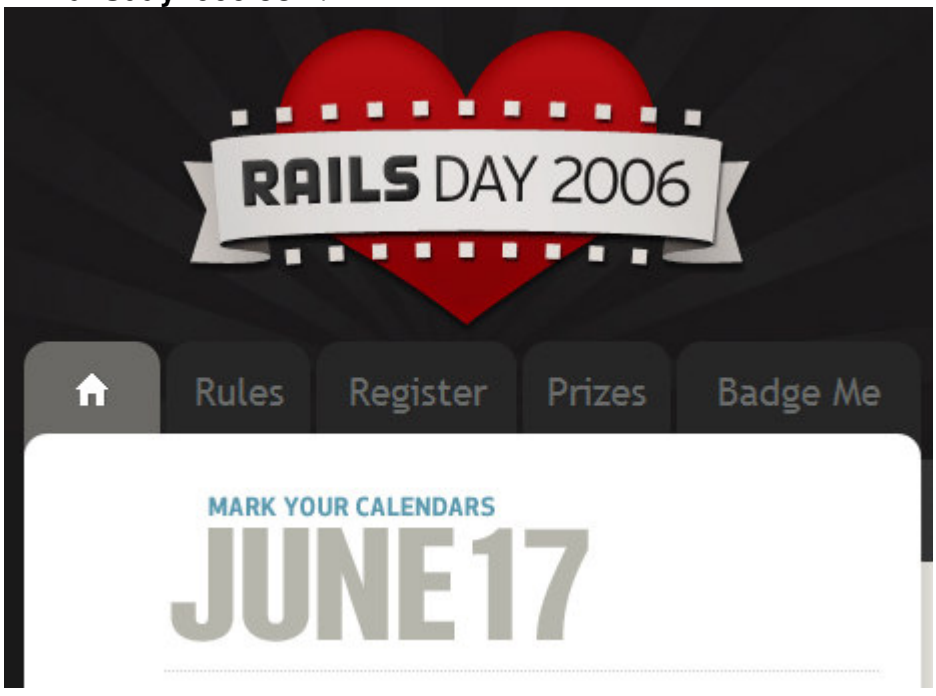
1. Evènements

1.1. Première conférence officielle Rails en France

http://paris.onrails.info/

1.2. Rails Day 2006

http://www.railsday2006.com/



2. Mémento

Methods

Strings
capitalize!
center
chomp!
chop!
concat
count
crypt
delete!
downcase!
dump
each
each_byte
empty?
gsub!
hash
hex
include?
index
intern
length
ljust, rjust
next!
oct
replace
reverse!
rindex
scan
slice!
split
squeeze!
strip!
sub!
sum
swapcase!
tr!
tr_s!
unpack
upcase!
upto

Regex
escape
last_match
new
quote
casefold?
kcode
match
source

Time
asctime
ctime
day
gmt?
gmtime
hour
isdst
localtime
mday
min
mon
month
sec
strftime
tv_sec
tv_usec
usec
utc
utc?
wday
yday
year
zone

DEFAULT DIRECTORY STRUCTURE

- rails_root
  - app
    - apis
    - controllers
      - application.rb
    - helpers
      - application_helper.rb
    - models
    - views
      - layouts
  - components
  - config
    - environments
      - development.rb
      - production.rb
      - test.rb
    - database.yml
    - environment.rb
    - routes.rb
  - db
  - doc
  - lib
  - log
    - development.log
    - production.log
    - server.log
    - test.log
  - public
    - images
    - javascripts
      - controls.js
      - dragdrop.js
      - effects.js
      - prototype.js
    - stylesheets
    - .htaccess
    - 404.html
    - 500.html
    - dispatch.cgi
    - dispatch.fcgi
    - dispatch.rb
    - favicon.ico
    - index.html
  - script
  - test
    - fixtures
    - functional
    - mocks
      - development
      - test
    - unit
    - test_helper.rb
  - vendor

METHODS NOTE

! - Denotes where a tailing ! may be used. A colourless ! denotes that the ! is compulsory.

PRE-DEFINED VARIABLES

$!	Exception information
$&	String of last match
$`	String left of last match
$'	String right of last match
$+	Last group of last match
$N	Nth group of last match
$=	Case insensitive flag
$/	Input record separator
$\	Output record separator
$,	Output field separator
$.	Current line number of last file read
$>	Default output for print
$_	Last input line of string
$0	Name of script
$*	Command line arguments
$stderr	Standard error output
$stdin	Standard input
$stdout	Standard output
$-a	True if -a is set.
$-d	Status of -d switch
$-l	True if -l is set
$-p	True if -p is set
$-v	Verbose Flag

RESERVED WORDS

=begin	elsif	rescue
=end	end	retry
BEGIN	ensure	return
END	false	self
alias	for	super
and	if	then
begin	in	true
break	module	undef
case	next	unless
class	nil	until
def	not	when
defined?	or	while
do	redo	yield
else		

REGULAR EXPRESSIONS SYNTAX

^	Start of string
$	End of string
.	Any single character
(a|b)	a or b
(...)	Group section
[abc]	Item in range (a or b or c)
[^abc]	Not in range (not a or b or c)
a?	Zero or one of a
a*	Zero or more of a
a+	One or more of a
a{3}	Exactly 3 of a
a{3,}	3 or more of a
a{3,6}	Between 3 and 6 of a
!(pattern)	"Not" prefix. Apply rule when URL does not match pattern.

Methods

Arrays
assoc
at
clear
collect!
compact!
concat
delete
delete_at
delete_if
each
each_index
empty?
eql?
fill
first
flatten!
include?
index
indexes
join
last
length
nitems
pack
pop
push
rassoc
reject!
replace
reverse!
reverse_each
rindex
shift
slice!
sort!
uniq!
unshift

Validation
condition_block?
create!
evaluate_condition
validate
validate_on_create
validate_on_update
validates_acceptance_of
validates_associated
validates_confirmation_of
validates_each
validates_exclusion_of
validates_format_of
validates_inclusion_of
validates_length_of
validates_numericality_of
validates_presence_of
validates_size_of
validates_uniqueness_of

Enumerable Mixin
collect
each_with_index
entries
find
find_all
grep
include?
max
min
reject
sort

Available free from ILoveJackDaniels.com

Ruby on Rails Logo used with permission.

image : http://www.ilovejackdaniels.com