Les frameworks de développement web : comment enrichir rapidement et efficacement le système d'information de votre établissement

Benjamin Ninassi
Institut National de Recherche en Informatique et Automatique
655, avenue de l'Europe - Inovallée Montbonnot - 38334 Saint Ismier Cedex France

Simon Panay
Institut National de Recherche en Informatique et Automatique
655, avenue de l'Europe - Inovallée Montbonnot - 38334 Saint Ismier Cedex France

Frédéric Saint-Marcel
Institut National de Recherche en Informatique et Automatique
655, avenue de l'Europe - Inovallée Montbonnot - 38334 Saint Ismier Cedex France

Résumé

Pour de nombreuses raisons (disponibilité, facilité d'administration, nomadisme), les systèmes d'informations des entreprises sont de plus en plus composés de multiples applications web, au détriment d'applications lourdes à installer sur les postes de travail. Cette évolution ouvre de nouvelles perspectives en matière d'enrichissement de système d'information et les web services se multiplient. La possibilité de répondre à des besoins spécifiques par la réalisation d'applications web dédiées venant se greffer au système d'information existant est un enjeu de taille pour les architectes des systèmes d'informations d'aujourd'hui. Ces dernières années, la généralisation des langages orientés objet, les méthodes agiles et l'apparition des frameworks pour le développement des applications web ont fait évoluer les méthodologies de développement.

Cette présentation est un retour d'expériences sur l'utilisation de trois frameworks (Ruby on Rails, Django, Symfony) basés sur trois langages (Ruby, Python, Php) dans des contextes comprenant des maitrises d'ouvrages et des utilisateurs finaux très différents. Les trois applications métiers réalisées sont venues s'interconnecter dans le système d'information existant de l'institut à différents niveau. Nous expliquerons ainsi comment il est aujourd'hui possible de diminuer le coût d'enrichissement du système d'information d'un établissement, en misant sur un investissement initial qui n'est pas à sous-estimer.

Nous aborderons la problématique du choix d'un framework en analysant les similitudes et les principales différences des trois solutions précitées. Nous détaillerons également la complexité de prise en main et de mise en œuvre en soulignant quelques pièges à éviter, ainsi que la description d'outils annexes utiles au développement collaboratif et au déploiement.

Mots clefs

Framework, développement logiciel, application web, Ruby On Rails, Django, Symfony, base de données, web services, Ruby, Php, Python.

1 Introduction

La création et l'utilisation des frameworks web est issue du besoin de développer rapidement des applications, en favorisant la réutilisation de code. Cette philosophie « *Don't Repeat Yourself* » (DRY) [1] doit faciliter la maintenance, le test et les évolutions d'une application. Ainsi un framework web propose un espace de travail modulaire, constitué d'un ensemble de bibliothèques, d'outils et de conventions qui permettent de se concentrer sur la logique métier de l'application. Il fournit aussi une structure pour l'application, généralement basé sur l'utilisation du paradigme Modèle Vue Contrôleur [2]. Cela assure une séparation entre :

- le modèle de données qui est en charge de l'accès et des requêtes à la base de données,
- la « vue », qui est une description de la présentation des données (ex : génération du code HTML, XML, JSON, etc.),

- le contrôleur, qui implémente la logique métier en récupérant les données du modèle pour les mettre à disposition de la vue.

Il existe un grand nombre de frameworks web, écrits dans différents langages qui sont construits autour du même socle MVC. Cette structuration commune facilite l'apprentissage et la compréhension d'un framework. Pour autant ils fournissent chacun une implémentation spécifique qu'il conviendra de prendre en compte pour faire son choix. Après une présentation générale des concepts d'un framework web nous vous présenterons un retour d'expérience sur trois des principaux acteurs des frameworks Web : Ruby on Rails en langage ruby, Django en python et Symfony en Php. Nous détaillerons les motivations de leur utilisation dans la réalisation de trois applications métiers : OSC, un outil de suivi des contrats et des budgets d'équipes de recherches, REMI, un référentiel de comptes informatiques, et Confact, une application permettant l'archivage des actes de conférences des chercheurs.

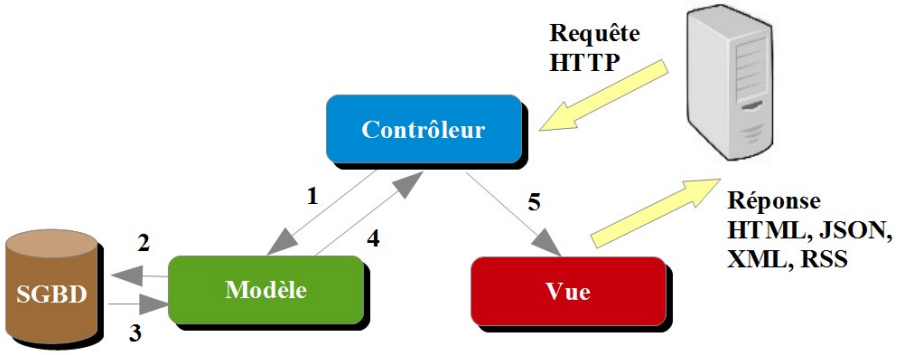

Figure 1: Paradigme MVC

2 Les frameworks Web

2.1 Le modèle de données

Le premier travail pour un développeur consiste à définir son modèle de données pour lancer la création de la base de données de son application Web. Pour cela les frameworks fournissent une abstraction avec la technique d'ORM [3](*Object Relational Mapping*). Elle permet de faire une translation du modèle relationnel d'une base de donnée vers un modèle objet. Ainsi on fait correspondre une table de base de données à une classe objet. On retrouve les informations suivantes dans chaque classe : les types des données, les contraintes d'intégrité (ex : l'unicité du champ) et les relations entre les entités et leur cardinalité.

Par exemple, à partir d'un modèle Symfony2 :

```
class Article{
       /**
        * @var  integer $id
        * @ORM\Column(name=''id'',type=''integer'')
        * @ORM\Id
        * @ORM\GeneratedValue(strategy=''AUTO'')
        */
       private $id ;

       /**
        * @var string $name
        * @ORM\Column(name="name", type="string", length=255)
        */
       private $name ;
```

```
}
```

l'ORM va générer cette table dans une base de donnée SQL de la manière suivante

```
CREATE TABLE articles (
        id int(11) NOT NULL auto_increment,
        name varchar(255),
        PRIMARY KEY  (id)
 );
```

Ensuite l'ORM fournit des opérations de haut niveau pour la manipulation des données afin de s'abstraire de la syntaxe SQL. De plus l'ORM fournit le support pour plusieurs serveurs de bases de données (MySQL, PosgreSQL, SQLite) avec l'utilisation des opérations CRUD pour la persistance des données (*Create Read Update Delete*) [4].Voici une requête Django qui filtre tous les articles contenant la chaîne de caractères « Framework » dans leur nom :

```
articles = Article.objects.filter(name__contains='%Framework')
```

Avec la requête SQL générée par l'ORM

```
SELECT id,name FROM Article WHERE name LIKE '%Framework';
```

2.2 La vue

Les frameworks proposent un système de template Web pour le développement de l'interface utilisateur. Les pages web consistent en une partie statique (HTML) et une partie dynamique avec des variables qui sont interprétées pour afficher les informations extraites de la base de données. Les moteurs de template sont différents d'un framework à l'autre ; d'où un apprentissage nécessaire. Il faut garder à l'esprit le paradigme MVC pour que la vue soit uniquement une logique limitée à la présentation. Dans l'exemple ci-dessous on décrit un template HTML qui affiche une liste d'articles avec le lien hypertexte correspondant dans le framework Ruby On Rails :

```
<% for article in @articles %>
        <%= link_to h(article.name), ligne_path(article) %>
<% end %>
```

2.3 La gestion des URLs et le contrôleur

Le contrôleur fournit la relation entre le modèles et la vue. Il interprète l'URL de la requête du navigateur (afficher un article), interroge le modèle, et transmet le résultat à la vue. On peut voir un exemple de route dans le framework Ruby on Rails avec l'URL de la forme article/<id> qui pointe vers le contrôleur article et l'action show de ce-dernier

```
match "article/:id", :to => "article#show"
```

http://www.example.org/article/3 va appeler la méthode show du contrôleur article

```
def show
        @article = Article.find(params[:id])
        respond_to do |format|
                format.html  # show.html.erb
        end
end
```

2.4 De véritables boîtes à outils

En plus d'un ORM et d'un système de *templating*, les frameworks web intégrent généralement un certain nombre d'outils permettant de faciliter à la fois le développement et les tâches de maintenance. Des communautés propres à chaque framework se sont formées et enrichissent bien souvent la solution initiale par de nombreuses librairies additionnelles (internationallisation, authentification, gestion du cache, des mails, des utilisateurs, interface REST, etc.) et de la documentation exhaustive. Cette dernière permet de réduire la documentation technique d'un projet spécifique à son stricte minimum.

Il faut aussi noter que les frameworks intègrent des mécanisme de sécurisation éprouvés par la communauté (injection SQL, failles Javascript, gestion des sessions, etc...) presque transparents pour le développeur, à partir du moment où celui-ci respecte les bonnes pratiques décrites dans la documentation.

3 Retour d'expérience sur Ruby on Rails

3.1 Présentation du contexte : OSC

OSC (**O**util de **S**uivi de **C**ontrats) [5] est une application web collaborative permettant le suivi des contrats et du budget (réalisé et prévisionnel) de structures de recherche. C'est une application open source distribuée sous la licence LGPL.

L'équipe de développement actuelle est composée de trois développeurs répartis entre l'INRIA et le Laboratoire Informatique de Grenoble. L'évolution technique du produit est le fruit de développements collaboratifs entre ces divers acteurs à temps partiels sur le projet.

L'application est aujourd'hui interconnectée avec les différents systèmes d'information des laboratoires pour :

- authentifier les utilisateurs sur l'annuaire ldap de l'établissement d'appartenance,
- fournir une API d'extraction de données,
- synchroniser toutes les nuits un certain nombre d'informations à partir du système d'information de gestion de l'INRIA (OPSF et GRANT) afin d'éviter les doubles saisies pour les gestionnaires.

Un ingénieur sur contrat à durée déterminée a développé un prototype de manière itérative avec les utilisateurs (chercheurs, gestionnaires, assistantes). Le développeur avait besoin d'un cadre de développement lui permettant d'avancer vite et de pouvoir rapidement montrer un résultat. Son choix s'est porté sur Ruby on Rails, un des frameworks les plus matures et les plus en vogue à l'époque. Une fois le prototype jugé suffisamment fonctionnel, le basculement en production s'est fait dans la foulée. Les développements continuent depuis, sur ce mode itératif.

En quelques chiffres, le dimensionnement de l'application aujourd'hui est le suivant :

- plus de 650 utilisateurs,
- un référentiel de plus de 450 structures,
- près de 5000 contrats ,
- plusieurs centaines de milliers d'entrées budgétaires (dépenses, factures, etc.),
- plusieurs dizaines de milliers de lignes de code réparties entre plusieurs centaines de fichiers,
- des modèles de données complexes avec 66 tables pour 722 champs dans la base de données contenant plusieurs centaines de milliers d'entrées.

3.2 Une application sur les rails

Le framework a effectivement rempli son rôle d'accélérateur pendant la phase de prototypage, en permettant rapidement la réalisation des interfaces fonctionnelles. Le langage ruby est rapide à assimiler et accessible à n'importe qui ayant des bases en programmation : son extrême verbosité facilite énormément la lisibilité du code en minimisant la nécessité des commentaires. Néanmoins, à moins de construire une équipe de développement autour de cette technologie, maintenir une expertise Ruby au sein d'un environnement où le *turn-over* humain est important peut vite devenir problématique.

La philosophie de Rails orientée « convention plutôt que configuration » propose de nombreux comportements par défaut pour la plupart des fonctionnalités. Ces deux facteurs cumulés rendent Ruby on Rails facile à prendre en main, d'autant plus que la documentation et les tutoriaux sont suffisamment abondants sur la toile. Néanmoins, devenir un expert est un travail de longue haleine car les possibilités de paramétrage sont légions.

Il n'y a pas eu de phase de transition entre le développeur initial et les développeurs actuels, et aucune documentation technique n'existait sur le projet. Par contre la structuration du code imposée par le framework a grandement facilité la reprise de la solution, qui s'est opérée en moins d'un mois, en prenant en compte le temps de formation.

De plus, cette structuration permet également de faciliter le développement collaboratif sur l'application. En effet, il est aisé de distribuer les tâches de développement en utilisant le cloisonnement offert par le framework : un designer peut se concentrer sur une vue pendant qu'un développeur se concentre sur le modèle de données.

OSC utilise la plupart des fonctionnalités intégrées au framework (gestion des formulaires, intégration du javascript, API, etc.). De plus, la communauté étant très active, beaucoup de « packages » sont disponibles (sous la forme de « gem », un système de librairies propre à Ruby) permettant ainsi aux développeurs de se concentrer sur ce qui fait la spécificité métier de leur application, et non sur des problématiques générales (pagination, authentification, etc.).

En complément, Ruby on Rails intègre tout le nécessaire pour mettre en place à moindre coût des tests unitaires et fonctionnels.

Une opération de maintenance commune à l'utilisation d'un framework est le suivi de ses versions, qui nécessitent parfois également des changements de version du langage. Ruby souffre d'ailleurs d'une compatibilité ascendante très limitée entre les versions majeures, ce qui complique les migrations entre les différentes versions du framework. Dans le cas d'OSC cette opération est rendue d'autant plus difficile par le manque de tests (qui n'ont pas été rédigés dès le début du projet, malgré leur simplicité de mise en place) qui permettraient d'automatiser la validation de la non-régression de l'application lors des mises à jour.

C'est pourquoi actuellement la version du framework utilisée est la version 2.1, qui a déjà évolué depuis le prototype mais qui reste loin de la dernière version de RoR (3.1). Cette version demeure néanmoins stable et sans problèmes de sécurité.

Le déploiement des applications Ruby on Rails est communément facilité par l'utilisation d'un outil annexe, Capistrano. Cet outil permet d'automatiser sur plusieurs serveurs un certain nombre de tâches comme le déploiement des sources à partir de système de contrôle de version, les migrations de base de données ou encore l'exécution de scripts. Néanmoins, OSC est actuellement hébergé à l'INRIA dans une équipe qui utilise un service d'administration centralisée permettant de faciliter et d'harmoniser le déploiement des applications. Par conformité c'est donc ce système qui a été utilisé pour gérer le déploiement. La mise en place des diverses tâches de *back-office* (les migrations des données, leur synchronisation périodiques avec le reste du système d'informations, etc.) sont facilités par le module rake fournit nativement avec le framework, qui offre là encore un panel de fonctionnalités facilitant le travail du développeur : liens avec les différents environnements de l'application, outils de *profiling* et de *debuging*, structuration des tâches, etc.

Pour illustrer, voici l'exemple de l'utilisation de rake pour migrer une base de données lors de l'ajout d'un attribut « auteur » à un objet « article ». Le fichier de migration va contenir les lignes suivantes :

```
class AddArticleAuteur < ActiveRecord::Migration
    def self.up
        add_column :auteur, :article, :text
    end
    def self.down
        remove_column :auteur, :article
    end
end
```

Afin d'obtenir le changement sur la base de données, il suffit de lancer la commande suivante et laisser faire rake :

```
rake db:migrate
```

Et enfin, pour revenir en arrière et annuler la dernière migration, il suffit de lancer la commande :

```
rake db:rollback
```

De plus, le module passenger d'Apache rend l'exploitation d'une application Ruby on Rails aussi simple que celle d'une application écrite en Php : il suffit d'installer ce module et de rajouter quelques lignes dans le fichier de configuration pour ne pas avoir besoin d'un autre serveur applicatif comme mongrel pour exécuter les applications web en ruby.

3.3 Retour d'expérience sur Django - Python

3.4 Présentation de Remi :

Remi (**Ré**férentiel des **M**oyens **I**nformatiques) doit devenir le référentiel de la base des comptes informatiques à l'INRIA. L'objectif du projet est de pouvoir remplacer les procédures hétérogènes (et parfois manuelles) de gestion des comptes informatiques propres à chaque service informatique par une solution technique mutualisée, automatisée, et harmonisée. Sa principale fonctionnalité attendue est donc l'alimentation automatique d'annuaires métiers de l'INRIA (LDAP POSIX, Active Directory, etc.). D'autres fonctionnalités sont imaginables avec par exemple l'alimentation des listes de diffusions Sympa, et de la suite collaborative Zimbra.

L'alimentation de ce référentiel doit pouvoir se faire de deux manières : l'interface d'administration web de l'application et le traitement automatique de notifications (fichiers XML et API REST) envoyées par une application des ressources humaines de l'INRIA.

Le dimensionnement de l'application est le suivant :

- plus de 6000 entrées dans le référentiel,
- 800 groupes informatiques,
- 17 annuaires à synchroniser (9 LDAP, 8 Active Directory) sur 8 sites géographiques différents.

3.5 Un framework qui swing

Python est un langage largement connu des administrateurs systèmes et réseaux qui ont été amenés à contribuer au projet. Afin de faciliter la reprise de nombreux scripts existants effectuant des traitements métiers sur les différents annuaires, nous avons choisi de conserver son utilisation. C'est un langage interprété, orienté objet, doté d'un typage dynamique fort, similaire à ruby en terme de fonctionnalités.

Pour réaliser cette application, nous avons donc opté pour le framework de développement web le plus connu en langage python : Django. Ce framework a l'avantage de proposer par défaut une interface évoluée d'administration générée par introspection du modèle de données. Pour cela, il suffit d'indiquer dans un fichier quels modèles on souhaite faire apparaître dans l'interface, et celle-ci est générée dynamiquement. Il est toutefois possible de surcharger certaines méthodes pour adapter le comportement de l'interface (si l'on souhaite rendre un champ non éditable par exemple). Pour cette application de gestion de données, nous avons donc pu nous concentrer sur le modèle et les traitements métiers.

Il est aussi possible avec Django d'utiliser tout une panoplie de modules et librairies réutilisables (tels que l'authentification avec un annuaire LDAP) réalisés par la communauté.

3.6 Organisation du développement

Nous avons décidé d'utiliser un cycle de développement itératif afin de proposer le plus tôt possible et à un intervalle régulier un prototype, pouvant ainsi être testé par les usagers de l'application. Il est ainsi possible d'identifier et de régler le plus tôt possible dans la phase de développement des problèmes fonctionnels. Voici une liste des outils utilisés pour aider à mettre en place cette méthodologie de gestion de projet.

3.6.1 Migrations de la base de données

Un des premiers problèmes auxquels on est confronté avec un cycle de développement itératif est l'évolution régulière du schéma de la base de données. Il existe une application tierce permettant d'appliquer les migrations de schéma et de données : South.

Lorsque l'on fait évoluer notre modèle de données (ajouter un champ, une table, modifier un type) il suffit de modifier sa description et d'exécuter les commandes suivantes permettant de générer un fichier décrivant la modification du schéma :

```
python manage.py schemamigration <application>
python manage.py datamigration <application>
```

Pour appliquer cette migration sur la base de donnée, il suffit ensuite d'entrer cette commande :

```
python manage.py migrate <application>
```

Cette couche d'abstraction gère un historique nous permettant de facilement passer d'une version à une autre.

3.6.2 Test unitaires et plate-forme d'intégration continue

Nous avons aussi souhaité que l'application possède une couverture de tests unitaires complète. Ces tests unitaires permettent au développeur de vérifier qu'à chaque modification de code source, le résultat des modifications ne produit pas de régressions dans l'application en cours de développement. Pour visualiser le résultat de ces tests, nous utilisons une plate-forme d'intégration continue (**Jenkins [6]**) bien connue des développeurs Java.

Cette plate-forme exécute l'ensemble des tests unitaires en scrutant les changements sur notre gestionnaire de version (Mercurial) et affiche les résultats à travers une interface web. En plus des tests unitaires, l'outil (Django-jenkins) fournit un rapport de couverture de code qui nous permet de repérer quelles sont les parties du projet qui ne sont pas testées.

3.6.3 Déploiements

Afin de faciliter toutes les tâches de déploiement nous avons utilisé librairie python incluant des outils en ligne de commande pour rationaliser les tâches d'administration à travers SSH : Fabric.

Pour l'utiliser, il suffit d'une description des serveurs cibles et d'une analyse des tâches redondantes effectuées lors d'un déploiement. Nous les avons réparties sous différentes fonctions comme par exemple : déploiement des sources sur un serveur, création de dossiers et liens symboliques, mise à jour de paquets de la distribution, installation, configuration et redémarrage des services (ex : apache).

Ainsi, pour déployer une nouvelle version de notre application, il suffit maintenant d'une ligne de commande à lancer sur le poste de travail du développeur :

```
fab production deploy_and_migrate
```

La mise en place de la chaîne de production (migrations de bases de données, déploiements), nécessite donc d'intégrer des modules externes au framework, contrairement à Ruby on rails où une partie de ces modules font partie du cœur du framework (ex : rake). Mais on constate que ce manque est en train d'être comblé par un travail de la communauté.

4 Retour d'expérience sur Symfony - PHP

4.1 Réalisation d'un outil de gestion des actes de conférences

Ce projet, réalisé en 2010, fait suite à une demande du service Information Scientifique et Technologique (IST), qui souhaitait pouvoir s'appuyer sur un outil informatique pour simplifier la procédure d'archivage des actes de conférences des chercheurs. Le périmètre du projet était relativement restreint : une base de données, un système d'enregistrement de document, une interface d'édition de données, quelques interfaces de visualisation permettant de faire des tris et des sélections dynamiques, et une fonctionnalité d'export des informations.

Le travail de collecte du besoin et de réalisation a été confié à un stagiaire de deuxième année de licence, présent pour une durée de deux mois seulement. Sa principale contrainte était qu'il réalise un outil facilement maintenable par d'autres développeurs après son départ, sachant que la compétence PHP était présente au sein de l'équipe. Le langage PHP est un langage interprété, disposant de fonctionnalité de modèle objet, utilisé pour produire des pages web dynamiques. Nous avons donc décidé d'utiliser le framework PHP Symfony (v1.4), imposant ainsi une structuration à son développement. Le langage PHP étant répandu chez les développeurs d'applications webs, c'est un framework tout à faire adapté à un environnement où le *turn-over* est important.

4.2 Mise en place du framework

Sur des développements de faible envergure comme ce projet, on peut naturellement se demander s'il est rentable d'avoir recours à un framework, surtout s'il faut apprendre à l'utiliser. Néanmoins, en l'espace de quinze jours le stagiaire s'est auto-formé sur le framework à l'aide de la documentation et des tutoriels disponibles en ligne. Le framework incluant par défaut toutes les fonctionnalités nécessaires à l'édition et à la visualisation des données ainsi qu'un ORM performant (Doctrine), le développeur a pu se concentrer sur les tâches spécifiques à l'application : la création du modèle et l'export des données. Le stagiaire n'avait jamais eu d'expérience sur les frameworks auparavant, et a pourtant pu livrer une application opérationnelle avant son départ.

Par la suite, quelques modifications mineures ont du être apportées à la solution afin de satisfaire de nouveaux besoins exprimés par le service IST et le framework a joué son rôle en permettant une reprise en main rapide l'application.

5 Conclusions

Les trois frameworks que nous vous avons présenté bénéficient des avantages classiques de la nouvelle génération des frameworks web. Ils possèdent tous une philosophie commune basée sur l'utilisation de bonnes pratiques et convergent en termes de fonctionnalités. Ils ont tous atteint un degré de maturité suffisante pour assurer la pérénité des applications développées. Leurs communautés sont très actives et offrent une aide et un support efficaces.

Cependant, l'utilisation des frameworks pour le développement d'applications incite à recourir à des cycles de développement itératifs. En terme de maintenance, il est fortement conseillé de suivre l'évolution des différentes versions des frameworks et librairies utilisées (mises au rebut de certaines fonctionnalités, émergence de nouvelles pratiques). Dans le cas contraire, on s'expose au risque de payer cette integration forte par des migrations difficiles pour les différents changements de versions. Dans ce contexte, l'abscence de tests unitaires est un danger supplémentaire.

Enfin les coûts d'apprentissage restent modérés au vu des gains attendus :

- uniformisation de la structuration des développements,
- bénéfice de nombreuses fonctionnalités (outils et librairies) notamment au niveau sécurité,
- réduction importante de la documentation technique nécessaire (remplacée en partie par la documentation officielle du framework),
- développement centré autour de la logique métier,
- facilité pour le développement collaboratif.

Il faut néanmoins garder à l'esprit qu'en plus de l'apprentissage, la première mise en place d'une chaine de développement (integration continue, tests unitaires, outils de déploiement, de migration de base de données, etc.) reste coûteuse, et qu'il vaut mieux mutualiser les efforts en choissant un seul et unique framework pour l'ensemble des développements d'une équipe. Le principal critère de choix reste le langage dans lequel il est écrit et la prise en compte du contexte et des compétences de l'équipe de développement. C'est d'ailleurs pour ces raisons qu'au sein du service SEISM (Service de l'e-Information Scientifique et Multimédia de l'INRIA) nous avons adopté le framework Symfony2 pour les futurs développements d'applications.

6 Bibliographie

[1] Dave Thomas interviewé par Bill Venners, Orthogonality and the DRY Principle, http://www.artima.com/intv/dry.html 2003

[2] Trygve Reenskaug, Models - Views – Controllers, 1979

[3] Scott W. Ambler, Mapping Objects to Relational Databases, 2010

[4] James Martin, Managing the Data-base Environment, 1983

[5] Jean-Marie Vallet, Outils de Suivi des Contrats, http://osc.ligforge.imag.fr/ 2007

[6] John Ferguson Smart, Jenkins: The Definitive Guide, 2011