esup-commons : un *framework* de développement pour ESUP-Portail

ou

Changer de méthodologie, changer d'outils, changer de métier

Pascal Aubry
IFSIC / Université de Rennes 1
Campus de Beaulieu – 35042 RENNES Cedex
Pascal.aubry@univ-rennes1.fr

Résumé

Le projet esup-commons *est né de la volonté de fédérer les développements de la communauté ESUP-Portail. Il est aujourd'hui une recommandation du consortium pour l'écriture des applications, qui peuvent indifféremment être exécutées en* servlet *ou* portlet*.*

Nous décrivons dans cet article la méthodologie et les outils d'esup-commons, discutons de l'opportunité d'adopter ces nouvelles technologies. Une conclusion apparaît clairement : le développement d'applications est devenu un métier à part entière.

Mots clefs

ESUP-Portail, développement, méthodologie, outils, *Spring*, *JSF*

1 Fédérer les développements

Si l'objectif opérationnel du consortium ESUP-Portail [1] était de fournir une solution logicielle pour le déploiement de portails dans les Universités, l'objectif organisationnel, sous-jacent, était de **fédérer les efforts de développement dans la communauté de l'Enseignement Supérieur Français**.

Dès le début du projet (en 2002), la coordination technique du consortium a, après analyse des solutions de développements disponibles, émis des recommandations concernant les formats d'échange, les plateformes, les outils et les méthodes de développement. L'objectif était l'uniformisation des développements de la communauté pour :

– faciliter la prise en main et donc l'amélioration des logiciels développés (contributions) ;

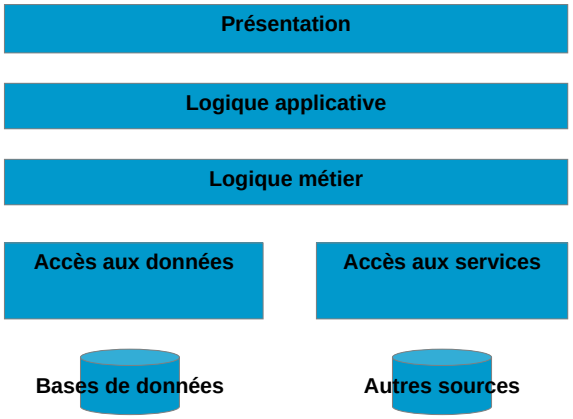
– permettre une mobilité des développeurs entre les projets ;

– homogénéiser le déploiement des applications et réduire le spectre des compétences nécessaires pour les exploitants.

Même si ces recommandations ont eu de réelles retombées, essentiellement en terme d'architecture et de formats d'échange, les logiciels restent relativement disparates, dans leur architecture, la manière dont sont écrits leurs sources, leur configuration et leur procédure de déploiement. L'animation de la communauté s'en est fortement ressentie, par un faible niveau de contribution des établissements de déploiement notamment. L'arrivée de nouveaux standards, en particulier *JSR-168*[1], a été l'occasion de repenser les recommandations, avec une évidence : **l'adoption de normes communes, l'utilisation généralisée d'outils et de méthodes communes ne peut se faire qu'à travers la mise à disposition d'une trousse à outils** au niveau du projet ; c'est la raison d'être de la bibliothèque *esup-commons* [2].

2 L'architecture en couche

La séparation des couches de programmation dans les applications est une condition *sine qua non* de garantie de la modularité. Le modèle suivant est connu depuis les débuts de l'informatique comme étant l'objectif à atteindre :



– La couche **présentation** gère le rendu à l'utilisateur, sous une forme voulue par le client (textuelle pour les applications en ligne de commande, graphique pour les clients dits « lourds », hypertexte pour les clients dits « légers », …) ;

– La couche **logique applicative** est chargée des interactions entre l'utilisateur et l'applicatif, gérant notamment les transitions entre les différents états de l'application ;

– La couche **logique métier** offre une *API* de gestion des objets et procédures métier de l'application ;

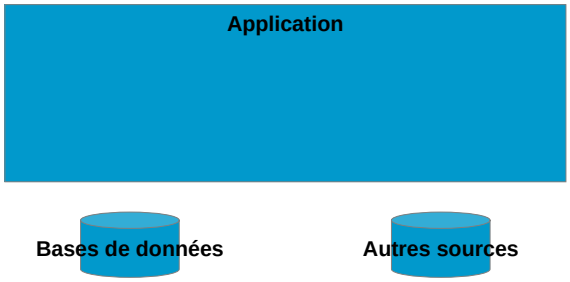
[1] http://developers.sun.com/portalserver/reference/techart/jsr168 : *Introducing Java Portlet Specifications: JSR 168 and JSR 286.*

– Enfin, les couches **accès aux données et aux services** permettent les accès de bas niveau vers les couches physiques.

Dans une application dont la séparation en couches est bien respectée, l'implémentation de chaque couche doit pouvoir être interchangée avec une autre implémentation, dès lors que celle-ci respecte la même *API*.

Cette séparation en couches permet de séparer les tâches au sein d'une équipe de développeurs. Les spécialistes du rendu aux utilisateurs ne sont souvent pas les mêmes que les spécialistes de l'accès aux données ; la programmation en couches permet de limiter les connaissances nécessaires lors d'un développement collaboratif. Ce point sera revu ultérieurement.

Dans la pratique, beaucoup de nos applications sont peu scrupuleuses en matière de séparation des couches :

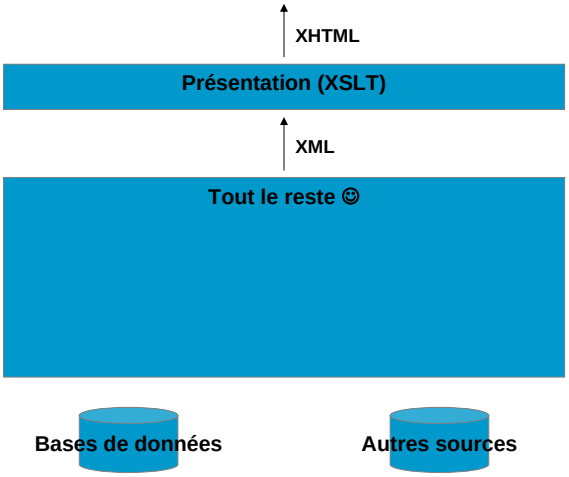


Tout programmeur s'en convaincra aisément en analysant ce petit exemple de programme *PHP*, qui en quelques lignes mélange présentation, logique applicative, logique métier et accès aux données :

```
// logique applicative
if (!empty($_POST["userId"])) {
       // accès aux données
       $c = DB::Connect(

       "mysql://user:passwd@host/db");
       $res = $conn->Query(
           "SELECT * FROM user WHERE id = '"
           + $_POST["userId"] + "'");
       $row = $res->FetchRow(
             DB_FETCHMODE_ASSOC);
       // logique métier
       $displayName = $row["display_name"];
       // présentation
       echo "Hello " . displayName . "!";
}
```

Dès le début du projet ESUP-Portail, la programmation de canaux s'appuyant sur *l'API uPortal*[2] avait forcé la séparation de la couche présentation, basée sur un moteur *XSLT* :



Le reste de l'application restait hélas noyé dans la masse du code et l'adhérence entre la couche présentation et le reste de l'application était encore très importante : dans ce modèle, les objets métiers n'étaient pas remontés jusqu'à la couche présentation et le *XML* produit est souvent dépendant de son utilisation.

Le premier pas vers cette séparation des couches est naturellement l'utilisation d'un langage pour lequel la notion d'interface est native, en l'occurrence *Java*. Notons que l'utilisation de *Java*, si elle permet cette séparation en couches, ne la garantit absolument pas. Une bonne architecture logicielle demande un réel effort de la part des développeurs, d'autant moins contraignant qu'il est bien compris.

3 Les choix techniques

3.1 *Portlet*/*servlet* : même combat

Le mode d'exécution standard des applications en environnement *J2EE* est la *servlet*. Néanmoins, pour une intégration directe dans le portail *uPortal* adopté par le consortium ESUP-Portail comme base d'exécution des applications, celles-ci étaient au début du projet développées comme des canaux *uPortal*, c'est-à-dire des applications s'appuyant sur *l'API uPortal*.

L'évolution actuelle des portails vers l'adoption de *JSR-168* (et *JSR-286* ultérieurement) amène naturellement les développeurs à livrer leurs applications sous forme de *portlets*. Il est cependant très intéressant de **ne pas lier les applications avec leur mode d'exécution** : nous ne développons pas des *servlets* ou des *portlets*, mais plus simplement des applications, qui sont exécutées en tant que *servlet* ou *portlet*, selon l'environnement dans lequel elles sont déployées. D'autres modes d'exécution sont d'ailleurs en passe d'être largement acceptés par notre communauté, tel *WSRP*[3].

Un objectif d'*esup-commons* était donc de dissocier ce qui est de l'application de ce qui est de son mode de déploiement, et de laisser ainsi le développeur se focaliser sur l'application elle-même.

[2] http://www.uportal.org: *Evolving portal implementations for particpating universities and partners.*

[3] http://www.oasis-open.org : *Web Services for Remote Portlets Specification.*

D'autres arguments militent vers cette flexibilité :

– Même si une application doit être finalement livrée sous forme d'une *portlet*, l'environnement de développement d'une *servlet* est beaucoup plus simple que celui nécessaire pour une *portlet* (un portail, dans lequel la *portlet* doit être publiée et accessible aux utilisateurs) ;

– Un exploitant peut, en installant une *servlet*, passer outre les problèmes inhérents au mode d'exécution des *portlets*, de leur publication et leur accessibilité ;

– La distribution des applications sous forme de *quick-start*, paquetages prêts à l'emploi et permettant l'installation rapide et le test d'applications, est très difficile à réaliser avec une *portlet*. Dans ce cas, l'utilisation d'une *servlet*, pré-installée dans un *servlet container* (tel *Tomcat*) rend la chose beaucoup plus simple.

3.2 Le choix du MVC[4] : *JSF*

Un des objectifs fut donc, lors du choix du MVC, de faire en sorte que les applications écrites en s'appuyant sur esup-commons puissent, par simple configuration, être déployées tantôt comme des *servlets*, tantôt comme des *portlets*.

Dans le monde *open-source* berceau du projet ESUP-Portail, deux candidats naturels se dégageaient : *Spring*[5] et *JSF*[6].

La différence essentielle entre les deux candidats tient au fait que **le MVC de *Spring* est dépendant du mode de déploiement** :

```
import org.springframework.web.servlet.*;

public class ServletController
implements Controller {
  public ModelAndView handleRequest(
              HttpServletRequest request,
              HttpServletResponse response)
  throws ServletException, IOException {
       logger.info("returning hello view");
       return new ModelAndView("hello.jsp");
  }
}
```

Le contrôleur pour *servlet* montré ci-dessus peut être très simplement transformé pour être déployé en *portlet* :

```
import org.springframework.web.portlet.*;

public class ServletController
implements Controller {
  public ModelAndView handleRequest(
              PortletRequest request,
              PortletResponse response)
  throws ServletException, IOException {
       logger.info("returning hello view");
       return new ModelAndView("hello.jsp");
  }
}
```

Si le portage est simple, il est néanmoins nécessaire, et il en résulte l'obligation de maintenir deux codes différents pour remplir notre objectif. C'est cette raison qui a amené les développeurs du projet *esup-commons* à choisir ***JSF* pour implémenter le MVC, car celui-ci est indépendant du mode de déploiement**. Les méthodes de transition des contrôleurs renvoient en effet de simples chaînes de caractères :

```
public class JsfController {
  public String callback() {
       logger.info("returning hello view");
       return "hello";
  }
}
```

et les transitions entre les états de l'application sont externalisées dans des règles de navigation au format *XML* :

```
<navigation-rule>
  <from-view-id>anyView.jsp</from-view-id>
    <navigation-case>
      <from-outcome>hello</from-outcome>
      <to-view-id>hello.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

Ce choix crucial offre la possibilité, pour toute application bâtie sur esup-commons, de spécifier le mode d'exécution seulement au moment où l'on déploie l'application, par écriture de propriétés :

```
# portlet deployment:
deploy.type=portlet
deploy.home=C:/uPortal/portlets

# servlet deployment:
deploy.type=servlet
deploy.home=C:/Tomcat/webapps
```

La possibilité de distribuer les applications sous forme de *quick-starts* est native, simplifiant à l'extrême le travail de l'exploitant :

```
# quick-start deployment:
quick-start=true
```

L'implémentation choisie de *JSF* est *MyFaces*[7], implémentation libre de *JSF* du projet *Apache*.

3.3 La gestion des *beans* : Spring

Si *JSF* a été choisi pour le MVC, *Spring* n'en est pas pour autant abandonné : depuis la version 2.0, *Spring* est la référence en matière de gestion des *beans* dans le monde *open-source*.

Il apporte notamment une souplesse extrême dans la configuration des applications, grâce à **l'injection de données et de dépendances**. Cette souplesse est essentielle pour l'adaptation des applications à des configurations locales différentes de celles où l'application a été initialement développée.

La diffusion des applications au sein d'une communauté en est d'autant améliorée qu'il est possible de modifier le comportement d'une application par simple configuration, tout au moins sans modifier le code distribué. Cela prévient la création de branches dissidentes et facilite la

[4] Modèle / Vue / Contrôler

[5] http://www.springframework.com : *Spring, the leading full-stack application framework.*

[6] http://java.sun.com/javaee/javaserverfaces : *JavaServer Faces Technology.*

[7] http://myfaces.apache.org : *MyFaces, the first free open source Java Server Faces implementation.*

coopération (entraide, contributions) entre les exploitants d'une même application.

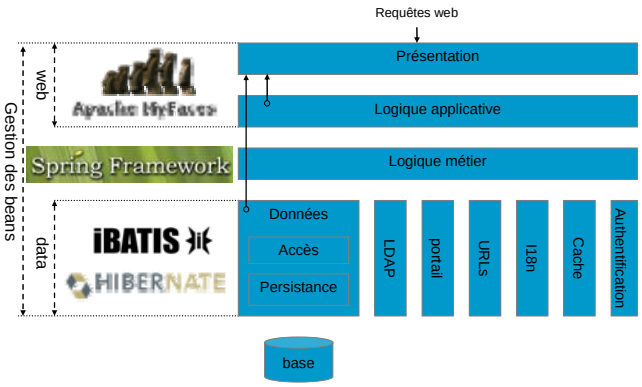
3.4 L'accès aux données

La séparation de la couche d'accès aux données est souvent la plus simple à appréhender pour le développeur novice ; elle est aussi fondamentale pour pouvoir, par simple configuration, changer la manière dont on accède physiquement aux données.

On pourra ainsi tantôt s'appuyer sur une base de données, tantôt sur un stockage dans des fichiers au format *XML*, tantôt utiliser un annuaire *LDAP*, … Quelques applications réalisées à partir d'*esup-commons* n'ont ainsi pas de connexion à une base de données.

Les bases de données étant néanmoins le moyen le plus répandu pour l'accès physique aux données, esup-commons autorise l'utilisation de n'importe quel *backend* ; des implémentations pour *Hibernate*[8] et *Ibatis*[9] sont ainsi proposées. Il a d'ailleurs été montré comment une application peut, par simple configuration, passer d'un *backend Ibatis* à un *backend Hibernate* (et inversement), montrant ainsi la souplesse extrême apportée par l'injection de données et de dépendances de *Spring*.

Les choix techniques de base du projet *esup-commons* sont résumés sur le schéma suivant :



3.4.1 *La préférence* Hibernate

Si l'utilisation de n'importe quel *backend* est possible, les développeurs du projet *esup-commons* recommandent l'utilisation de *Hibernate*, le plus abouti de leur point de vue des bibliothèques libres d'accès aux données. Il peut gérer la structure de la base de données, la création et la mise à jour des tables de la base pouvant être réalisées par de simples commandes (tâches *ant* **`init-data`** et **`upgrade`** dans *esup-commons*).

De ce fait, développeurs et exploitants d'une application peuvent maintenant s'appuyer sur une base de données sans même connaître *SQL* ; l'accent est mis sur les objets, les objets métier en particulier, et leur persistance est entièrement confiée à *Hibernate*.

[8] http://www.hibernate.org : *Hibernate, an object/relational persistence and query service for Java.*

[9] http://ibatis.apache.org : *Ibatis, an XML-based object/relational mapping and persistence framework.*

3.4.2 *Sessions, transactions et exceptions*

Du point de vue du développeur, la gestion des sessions aux bases de données et des exceptions est sans doute **l'apport le plus important *d'esup-commons***. Le développeur est complètement déchargé de cette partie ingrate et délicate, implémentée sous le modèle *one-session-per-request*, la requête *HTTP* étant l'unité de transaction.

Nombreuses sont les applications qui interagissent avec plusieurs bases de données. On verra ainsi des applications qui stockent leurs données dans une base de données propre, tout en accédant à une base de données institutionnelle (de type Harpège ou Apogée par exemple). *esup-commons* prend en charge ce cas de figure, grâce à la notion de magasin de bases de données.

4 Un projet de développeurs pour les développeurs et les exploitants

4.1 *esup-commons* aide les développeurs

Le projet *esup-commons* a été initié par des développeurs d'applications, dans le but d'aider les développeurs d'applications de la communauté ESUP-Portail. En effet, les contextes d'exécutions étant très semblables dans tous les établissements du consortium, les problèmes rencontrés sont très souvent les mêmes.

Prenons en exemple un problème rencontré par tous les développeurs de canaux ou *portlets*, d'applications qui se rendent donc dans un canal. L'envoi d'un courrier électronique, quelque soit son but (information des utilisateurs, remontée d'alerte, …) est une opération dont la longueur n'est pas négligeable. Ainsi, il est en général impossible de demander à un canal d'envoyer plusieurs courriers électroniques lors d'une requête *HTTP* sans rencontrer un timeout au niveau du portail (le temps donné à un canal pour effectuer son rendu est en général limité).

La solution est simple et bien connue, puisqu'il suffit de faire exécuter les envois par des tâches de fonds de manière asynchrone (grâce à des *threads* de basse priorité). Il n'empêche que ce développement simple mérite attention ; cette fonctionnalité est, parmi beaucoup d'autres, proposée en standard dans *esup-commons*.

4.2 *esup-commons* aide les exploitants

Le nombre croissant d'applications dans nos environnements est souvent un casse-tête, notamment dans le contexte des portails où les déploiements se font sur des machines redondantes (des lames). L'installation, et encore plus la mise à jour des applications, nécessitent de la part des exploitants temps, compétence et méthodologie, faisant du **rôle d'exploitant d'applications un métier à part entière**.

esup-commons propose une méthodologie concernant l'installation et la mise à jour des applications qui diminuent le temps passé et les problèmes rencontrés. *esup-commons* fournit ainsi la possibilité, lors d'une mise à jour, de récupérer les fichiers de configuration d'une version antérieure.

Une mise à jour mineure se réduit alors souvent à l'exécution de quelques commandes, et s'effectue en quelques minutes :

```
wget http://serveur/application-x.y.z-t.zip
unzip application-x.y.z-t.zip
cd application-x.y.z
ant recover-config
ant deploy
ant restart
```

Par ailleurs, la diffusion d'applications construites sur le même modèle (hiérarchie de fichiers, format des fichiers de configuration, procédures d'installation et de mise à jour) permet aux exploitants de ne pas être perdus lors de l'installation d'une nouvelle application basée sur esup-commons.

5 *esup-commons* : une méthodologie, des outils

5.1 La méthodologie

esup-commons apporte une méthodologie de développement, qui fait que l'architecture et les procédures de gestion des projets basés sur esup-commons sont les mêmes.

La méthodologie porte notamment sur :

- La hiérarchie des fichiers de l'application ;
- Le format des fichiers de configuration ;
- La programmation par interfaces ;
- Le respect des mêmes conventions de nommage, de codage et de présentation ;
- L'utilisation systématique de *SubVersion* ;
- Les procédures d'installation et de mise à jour.

Il est clair que **l'adoption d'une méthodologie commune facilite énormément la mobilité des développeurs** d'un projet à un autre, pour peu qu'ils soient tous deux basés sur esup-commons. On assiste ainsi dans les établissements qui ont adopté esup-commons comme standard de développement à la naissance du label « développeur *esup-commons* », qui signifie qu'un tel développeur pourra sans peine passer d'un développement *esup-commons* à un autre.

On entend parfois avec plaisir l'expression « **installer une application *esup-commons*** », qui montre l'intérêt de l'uniformisation des procédures de déploiement des applications.

La fédération des développements trouve ici tout son sens.

5.2 Les outils

esup-commons offre de nombreux outils qui permettent le **développement rapide** des applications. Nous en montrons ci après une liste non exhaustive.

5.2.1 La gestion des exceptions

La gestion des exceptions est assurée en lien étroit avec celle des connexions aux bases de données. Elle permet aux développeurs de bénéficier de rapports d'erreur très complets ; ces rapports d'erreur, qui incluent par exemple le contexte de l'application au moment de l'exception, permettent en général de cerner très précisément les dysfonctionnements et de les corriger rapidement.

Il est également possible de remonter les exceptions par courrier électronique, par exemple aux développeurs de l'application (avec un cache pour prévenir les effets de *spam*).

5.2.2 La gestion des versions

La gestion des versions des applications est prise en charge par esup-commons, qui nomme les versions sous une **sémantique unique `x.y.z-t`** (numéros de version majeure, mineure, de correction et de distribution). Un exploitant sait, en fonction du numéro de version d'une distribution proposée par le développeur et du numéro de la version installée sur son système, en quoi va consister la mise à jour :

- Changement de numéro majeur : procédure de mise à jour manuelle, pouvant nécessiter du temps ;
- Changement de numéro mineur : mise à jour semi-automatique pouvant nécessiter un changement des fichiers de configuration ;
- Changement de numéro de correction : mise à jour automatique ;
- Le numéro de distribution change uniquement dans le cas de modifications mineures des documentations ou des fichiers d'exemple.

***esup-commons* gère les incompatibilités de versions entre l'application elle-même et la base de données de l'application**. Ce point est crucial en environnements redondants (clusters) où une mise à jour ne peut pas être faite de façons instantanée et simultanée sur tous les serveurs d'exécution ; l'exécution d'une version d'application incompatible avec la base de données peut en effet conduire à des dysfonctionnements et des incohérences dans les bases de données, elle est empêchée par *esup-commons*.

5.2.3 L'internationalisation

L'internationalisation des applications est native et complète (rendu des pages, messages d'erreurs, …) dans toute application *esup-commons*. Il s'agit là d'un atout évident pour la diffusion des applications au delà des frontières de notre communauté francophone.

L'internationalisation est également un atout pour la distribution des applications à l'intérieur de la communauté francophone, car **elle force le développeur à externaliser toutes les chaînes de caractères**. Par l'utilisation de plusieurs ressources de langues (bundles) par langage, *esup-commons* permet :

- Au développeur de surcharger les messages de esup-commons dans son application ;
- À l'exploitant de surcharger les messages du développeur.

On voit ainsi que l'internationalisation a pour conséquence la **simplification de la personnalisation des applications** par les exploitants.

5.2.4 L'envoi de courriers électroniques

Déjà citée en exemple de ce qu'un développeur trouve dans *esup-commons* sans avoir à faire aucun

développement, l'envoi de courriers électroniques peut se faire de manière synchrone ou asynchrone, au format texte, HTML ou alternatif, avec éventuellement des documents attachés.

5.2.5 *Les commandes en ligne*

Parfois appelées commandes *batch*, elles sont indispensables pour exécuter certaines tâches :

– Trop longues pour être exécutées dans un contexte web, par exemple des manipulations lourdes sur les bases de données ;

– Devant être faites de manière régulière et/ou asynchrone, comme par exemple l'indexation de données, de manière plus générale toutes les tâches périodiques.

Ces commandes doivent, pour bénéficier de la gestion des exceptions et des connexions à la base de données, respecter la méthodologie proposée par *esup-commons*. L'application est alors gérée selon un modèle *one-session-per-command*, avec une granularité de transaction à l'initiative du développeur.

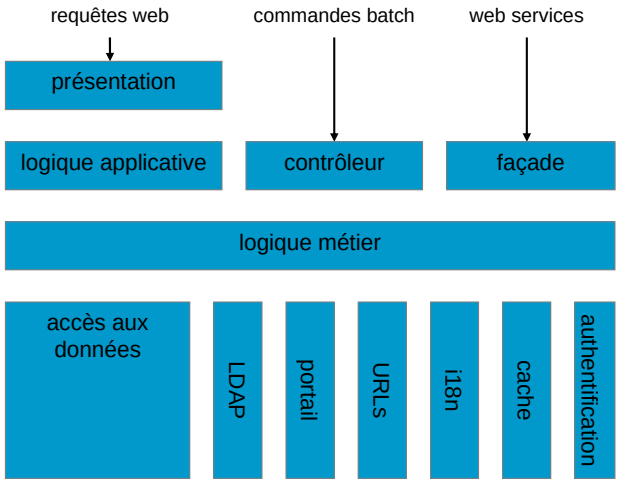
Même s'il a été initialement conçu pour les applications web, ***esup-commons* est aujourd'hui utilisé pour l'écriture d'applications sans interface web**, appelées seulement en ligne de commande.

5.2.6 *Les* web services

Les *web services* sont utilisés dans les applications *esup-commons* dès lors qu'elles doivent interagir avec d'autres applications.

En plus de l'utilisation de la bibliothèque *xFire*[10], qui apporte un niveau d'intégration et une simplification remarquables pour l'exposition des *web services*, l'apport de *esup-commons* dans ce domaine est, comme pour les commandes *batch*, la gestion des exceptions et des connexions aux bases de données.

L'architecture complète des accès à une application *esup-commons* est ainsi donnée par le schéma suivant :



5.2.7 *L'authentification*

Les mécanismes d'authentification sont très nombreux dans nos environnements ouverts. Parmi eux, citons :

– L'utilisation de *CAS* [3], en filtre *J2EE* ;

– L'authentification par le portail, dans le cas d'une exécution en mode *portlet* ;

– Toute authentification propriétaire, basée sur des secrets d'authentification stockés par l'application.

Afin de répondre à toutes les possibilités, **l'authentification est externalisée** (confiée à un service d'authentification, dont on ne pré-suppose pas de l'implémentation), et les principaux mécanismes d'authentification sont proposés par *esup-commons*.

L'ajout de nouveaux mécanismes est d'ores et déjà prévu, notamment l'authentification (et la récupération d'attributs) via *Shibboleth* [4], ainsi que l'appui sur *ACEGI*[11].

En déploiement *portlet*, grâce à l'utilisation des préférences *JSR-168*, les applications *esup-commons* disposent des fonctionnalités de *proxy CAS*, c'est-à-dire qu'elles sont capables de s'authentifier auprès de services tiers sous l'identité de l'utilisateur connecté au portail.

5.2.8 *L'accès au Système d'Information*

Le S.I. des établissements est, en plus des bases de données institutionnelles, composé de l'annuaire *LDAP* et des informations du portail.

esup-commons propose une bibliothèque simple d'accès à l'annuaire *LDAP*, qui permet de récupérer les utilisateurs, les groupes, ainsi que n'importe quelle entité *LDAP* (permettant par exemple de gérer la téléphonie d'un établissement stockée dans un annuaire *LDAP*).

L'accès aux attributs utilisateur et aux groupes du portail est réalisé grâce à l'appel d'un *web service*, proposé par le projet *esup-portal-ws*.[12]

5.2.9 *La bibliothèque de balises* JSF *(*taglib*)*

esup-commons offre une bibliothèque de balises, dont l'intérêt essentiel est d'être configurée dynamiquement par *Spring*.

C'est cette bibliothèque qui permet d'utiliser exactement les mêmes pages *JSF* que l'application s'exécute en *servlet* ou en *portlet* (alors que le rendu à l'utilisateur est différent).

5.2.10 *Les* URLs *directes*

Les *URLs* directes sont ainsi appelées car elles permettent de positionner une application web dans un état donné, en passant éventuellement des paramètres (par exemple, les courriers d'alertes d'un ticket envoyées par l'application *esup-helpdesk* [5] aux utilisateurs donnent des liens hypertextes qui envoient directement sur la page du ticket, sans passer par la page d'accueil ou le tableau de bord).

Cette fonctionnalité, si elle est plutôt évidente pour une *servlet*, est en revanche beaucoup plus difficile à réaliser pour une *portlet*. En effet, il faut tout d'abord que le portail authentifie l'utilisateur, positionne le portail sur l'application, puis transmette les paramètres de la requête à l'application.

---

[10] http://xfire.codehaus.org : *Codehaus xFire: next generation SOAP framework.*

[11] http://www.acegisecurity.org : *ACEGI Security System for Spring.*

[12] http://sourcesup.cru.fr/esup-portal-ws : *Portal Web Service.*

La mise en œuvre de cette partie a été d'autant plus délicate que l'objectif était, pour cette partie également, de conserver le même code que l'application soit déployée comme une *servlet* ou une *portlet*.
esup-commons prend en charge tous les aspects des *URLs* directes :

– Génération des liens hypertextes en fonction des paramètres ;

– Récupération des paramètres et positionnement de l'application dans l'état voulu.

5.2.11 Beaucoup d'autres choses

La liste des fonctionnalités donnée ci-dessus n'est pas exhaustive ; *esup-commons* amène de nombreuses autres possibilités, parmi lesquelles :

– **La gestion du cache**. Très utilisée pour améliorer les performances des applications, par exemple pour limiter les accès réseau, le cache est souvent implémenté de manière disparate, basique et peu fiable. *esup-commons* se base sur *EhCache*[13], ce qui permet de s'appuyer sur une librairie riche, sûre et éprouvée et de centraliser toute la configuration des caches en un seul fichier de configuration.

– **La génération de flux de type *RSS*** bénéficie également de la gestion des exceptions et des connexions aux bases de données.

6 L'effort du consortium ESUP-Portail pour diffuser *esup-commons*

6.1 La genèse

Le projet *esup-commons* est né en avril 2006 à l'Université de Rennes 1 lors du développement d'une nouvelle application sous la forme d'une *portlet* (*esup-diskquota*[14], un gestionnaire de quotas disque basé sur des filtres *LDAP*). Les développeurs de cette application se sont vite heurtés à des problèmes complexes, notamment liés à l'intégration des *portlets* dans le portail.

Il a alors été décidé de mutualiser dans un projet, appelé *esup-commons*, tout ce qui pourrait resservir à d'autres développements semblables.

6.2 Convaincre au sein de son établissement

Quelques mois plus tard, la première version de *esup-commons* était diffusée. Le premier pas de la diffusion fut de faire comprendre puis accepter cette solution au sein du CRI de l'Université de Rennes 1. Après une première formation interne, *esup-commons* a été adopté comme base de tous les nouveaux développements, internes ou externes.

Les défauts de la version d'alors ont été corrigés par de nombreuses contributions, qui montre l'intérêt que les développeurs de l'Université de Rennes 1 ont trouvé dans *esup-commons*.

[13] http://ehcache.sourceforge.net : *EhCache, a simple, fast and thread safe cache for Java.*

[14] http://sourcesup.cru.fr/esup-diskquota : *esup-diskquota, the ESUP-Portail disk quota manager.*

6.3 Convaincre au sein du consortium

Une fois cette première étape franchie, une deuxième formation a été organisée pour les membres de la coordination technique du consortium ESUP-Portail. Il s'agissait là aussi de convaincre de l'opportunité du projet, et d'en faire une recommandation de développement. *esup-commons* a ainsi été présenté aux ESUP-Days [2].

6.4 Convaincre au sein de la communauté

Depuis, le projet *esup-commons* a été présenté à toute la communauté, et quatre autres formations ont été organisées, cette fois ouvertes à notre communauté toute entière. Plus de 80 ingénieurs ont ainsi été formés à *esup-commons*, et d'autres se sont auto-formés en s'appuyant sur les supports mis en ligne.

Les contributions viennent aujourd'hui de toute la communauté pour améliorer et enrichir *esup-commons*.

7 Réfléchissons un peu…

Cette nouvelle manière de développer, cette adoption d'une nouvelle méthodologie et de nouveaux outils constitue une vraie révolution pour la plupart des développeurs. On peut légitimement s'interroger sur la nécessité d'aller vers des technologies de plus en plus complexes et contraignantes.

7.1 Les gains apportés par *esup-commons*

Rappelons tout d'abord les gains engendrés par la méthodologie et les outils présentés ci-avant.

7.1.1 Fiabiliser les applications

L'utilisation d'outils de haut niveau (*JSF*, *Spring*, …) et des bibliothèques dédiées les plus éprouvées dans leur domaine (*Hibernate*, *EhCache*, *LdapTemplate*[15], …) a pour conséquence immédiate de renforcer la fiabilité des applications. Elle permet aussi de bénéficier des évolutions de ces bibliothèques sans développement supplémentaire.

La programmation en couches permet également de spécifier des tests poussés, pas seulement unitaires, qui permettent de garantir la continuité de fonctionnement du code des applications.

7.1.2 Améliorer l'accessibilité

L'accessibilité est devenue un des critères majeurs de jugement des applications web. L'utilisation d'un outil de haut niveau pour la couche présentation (en l'occurrence *JSF*) permet de garantir la qualité du code envoyé aux navigateurs, et d'en améliorer l'accessibilité[16].

7.1.3 Simplifier la maintenance

De l'expérience des développeurs d'applications basées sur esup-commons, la séparation des couches induit la séparation des zones de recherche des erreurs.

[15] http://ldaptemplate.sourceforge.net : *LdapTemplate, Java framework to simplify LDAP operations.*

[16] L'utilisation de *JSF* ne garantit pas l'accessibilité des applications, celle-ci peut nécessiter un effort du développeur pour les interfaces complexes.

De plus, elle permet un travail collaboratif efficace, puisque chaque développeur peut se concentrer sur la partie dont il est spécialiste, par exemple le rendu à l'utilisateur ou bien l'accès physique aux données.

7.1.4 Aider la diffusion des applications

En proposant à l'utilisateur des outils simples de distribution des applications, esup-commons permet d'économiser nombre d'efforts nécessaires à la distribution des applications.

Notons par ailleurs que la distribution est un moyen simple de transfert des applications sous forme de livrables des développeurs aux exploitants, qui ne sont pas toujours les mêmes dans les établissements.

7.1.5 Faciliter l'adaptation

Par ailleurs, l'injection de données et de dépendances de *Spring*, ainsi que l'utilisation systématique d'interfaces permet une très grande facilité d'adaptation des programmes à des configurations locales différentes de celles dans lesquelles les applications ont été initialement écrites.

Cette adaptation se fait d'ailleurs la plupart du temps sans écriture de code Java, ou alors par simple rajout, sans modifier le code principal de l'application. Cela évite la création de branches dissidentes à la branche principale de développement, qui est une des garanties de la cohérence de la communauté qui utilise une application.

7.1.6 Gagner en productivité

Il est difficile de comprendre comment on peut gagner en productivité en ayant à assimiler tant de notions nouvelles. Pour être franc, les développeurs d'esup-commons ont également longtemps douté que l'on puisse, avec un tel niveau d'abstraction, être en fin de compte plus productif. Chacun doit s'en faire une idée propre, il est essentiel de ne pas avoir d'idée préconçue dans ce domaine.

Le nombre d'établissements qui ont aujourd'hui adopté esup-commons comme standard de développement montre que le choix n'est pas celui des seuls développeurs esup-commons. Il doit bien y avoir autre chose…

7.2 Une approche pourtant décriée

Les choix pris par le projet *esup-commons*, et plus généralement ceux du consortium ESUP-Portail sont ci et là vivement critiqués.

Certains critiquent notre approche et vendent de beaux web services en cachant les dessous de l'implémentation ; isoler les couches supérieures est un bon commencement, mais clairement insuffisant lorsque l'on use et abuse de *triggers*[17] pour implémenter les procédures métiers.

D'autres nous appellent les sur-connaissants, en argumentant qu'il faut connaître toutes les couches d'une application *esup-commons* pour développer une application selon ce modèle. Ils ne font que nous conforter dans l'idée que la séparation des couches apporte exactement le contraire : la collaboration entre des développeurs spécialistes chacun dans leur domaine.

Les critiques sont parfois si vives qu'on peut se demander s'il n'est pas gênant de voir les universités mutualiser leur savoir en matière de développement *open-source*. Tant pis !

Nous préférons continuer notre chemin dans la fédération des efforts de notre communauté.

7.3 Et l'existant alors ?

On peut légitimement douter du gain de productivité apporté par l'utilisation de esup-commons, et c'est vrai qu'il n'est visible qu'une fois la première marche franchie (la découverte et la prise en main de la méthodologie et des outils).

Il est clair qu'un établissement vit avec un existant, qu'il faut assumer et entretenir. Les développeurs d'esup-commons ont aussi leur existant à maintenir et sont bien conscients du problème.

Comme on dit parfois, il ne faut pas jeter le bébé avec l'eau du bain. Il faut cependant prendre garde à deux choses :

– En premier lieu, à force de se laver, il arrive que l'on salisse l'eau du bain ; dans ce cas, il faut alors changer l'eau du bain ;

– Enfin, il arrive que le bébé grandisse, tant que le bébé ne soit plus vraiment un bébé, et en tout cas ne tienne plus dans la baignoire ; dans ce cas, il faut changer la baignoire.

7.4 Le métier de développeur a changé !

La seule conclusion sûre et évidente qui s'impose est la suivante : le métier de développeur a changé : on ne programme plus sur un coin de table !

Bibliographie

[1] Pascal Aubry, *The ESUP-Portail project*. JA-SIG Winter 2006 conference, Atlanta, Georgia, United States, December 2006.

[2] Pascal Aubry, *esup-commons : un framework de développement pour le projet ESUP-Portail*, ESUP-Days n°3, Paris, Janvier 2007.

[3] Pascal Aubry, Vincent Mathieu & Julien Marchal, *Open-source Single Sign-On with CAS (Central Authentication Service)*, EUNIS2004, Bled, Slovenia, July 2004, journal Uporabna informatika (Applied Informatics, ISSN 1318-1882, surveyed by INSPEC), ed. Prof. Andrej KovacicSpring.

[4] Pascal Aubry, *Open-source Identity Federation with Shibboleth*, EUNIS'2006, Tartu, Estonia, June 2006.

[5] Pascal Aubry and Alexandre Boisseau , *ESUP-Portail Helpdesk: user support at establishment-level*, JASIG Summer 2007, Denver, Colorado, June 2007.

[17] Procédures stockées implémentant une partie de la logique métier au niveau de la base de données, en totale contradiction avec la programmation en couches, qui entraîne une très forte adhérence entre la logique métier et le *SGBD* utilisé pour la persistance.