Gérer un projet de développement d'application

Méthodes, UML, bonnes pratiques en PHP

Eric Quinton

20/10/2005
Dernière modification : 24/11/2008 - révision 157

Table des matières

I. Les étapes d'un projet – méthodes traditionnelles

A. Maîtrise d'oeuvre, maîtrise d'ouvrage ?

En France, le développement de nouvelles applications implique nécessairement deux acteurs : le demandeur, que l'on désigne maître d'ouvrage, et celui qui réalise l'opération, le maître d'oeuvre.

Cette dichotomie permet de séparer parfaitement les rôles : le maître d'ouvrage définit ce qu'il veut, le maître d'oeuvre le réalise, et le maître d'ouvrage « paye » après avoir vérifié que ce qu'on lui a livré correspond bien à ce qu'il a demandé.

Malheureusement, ce n'est que très rarement que cela se passe aussi bien... Le maître d'ouvrage est un « fonctionnel », c'est à dire la personne compétente dans le domaine à traiter, et pas un informaticien. On assiste donc à la venue d'un troisième acteur, l'assistant à maitre d'ouvrage, qui va être chargé d'une part de traduire la demande initiale en termes compréhensibles par des informaticiens, et d'autre part de vérifier que ce qu'on livre correspond bien à ce qui a été demandé.

Trois acteurs pour un même projet, dont un intermédiaire : ça fait beaucoup ! C'est quasiment indispensable pour les gros projets, impliquant des moyens humains et financiers importants, mais pour les petits projets, ce qui sont réalisés en un mois, pour quelques utilisateurs, qui représentent de petites bases de données, est-ce vraiment utile de déployer une telle approche ?

En fait, sur un projet, on rencontre systématiquement quatre phases essentielles :

- la demande initiale : le demandeur décrit ce qu'il veut, dans les termes qui lui sont propres. Elle est en général insuffisante pour estimer l'ampleur du développement à effectuer.
- La restitution de la demande : l'informaticien va décortiquer la demande, en rencontrant le demandeur, les utilisateurs..., pour circonscrire l'application à réaliser. Le document qui achèvera cette étape constituera le « cahier des charges », c'est à dire ce qui sera demandé à l'application, pour lequel le demandeur est prêt à payer.
- Le développement, étape purement informatique, mais qui nécessite souvent quelques allez-retours avec le demandeur, pour clarifier certains points ou proposer des approches liées aux évolutions techniques ou aux possibilités offertes par les techniques de développement utilisées.
- La mise en production, qui comprend d'une part les tests de fonctionnement, et d'autre part la mise en route : déploiement sur la plate-forme de production, formation des utilisateurs, reprise de l'historique...

Alors, faut-il « jeter » le fonctionnement maître d'ouvrage/maître d'oeuvre ?

Dans la pratique, pour qu'un projet aboutisse, il faut que certains rôles et responsabilités soient parfaitement définis :

- le demandeur est le seul à décider de ce que doit comporter son application. Il doit fournir les ressources financières et humaines nécessaires à son aboutissement. C'est là le rôle du maître d'ouvrage.

- Le réalisateur doit proposer au demandeur la manière de réaliser l'application, les évolutions possibles, et est tenu à l'obligation de résultat. C'est le rôle du maître d'oeuvre.

Les deux doivent travailler ensemble : un maître d'oeuvre qui refuserait de prendre en compte un point particulier apparaissant en cours de développement, en s'en tenant exclusivement au cahier des charges initial, ne jouerait pas son rôle. Si on travaille en externalisation, il faut bien évidemment que les conditions de cette négociation aient été définies dans le contrat (avenants possibles, marge de manoeuvre...). Mais le maître d'oeuvre ne doit jamais imposer son point de vue : ce n'est pas lui qui utilisera l'application, et il ne dispose pas de la compétence « métier » du demandeur.

De même, un maître d'ouvrage qui demanderait au maître d'oeuvre de s'en tenir au cahier des charges, sans être capable d'évoluer en fonction de critères ou d'impossibilités techniques, qui refuserait de trancher sur des points mis à son arbitrage dans des délais raisonnables eu égard à l'ampleur du projet, manquerait à ses responsabilités.

« Il vaut mieux un mauvais accord qu'un bon procès ».

Si on souhaite qu'un projet aboutisse, il faut que chacun assume ses responsabilités et travaille en bonne intelligence. Cela n'exclut pas les conflits, qui sont souvent le résultat d'interprétations liées à des points juridiques sur les droits d'auteur, de licence, de propriété intellectuelle... Si le projet est correctement dimensionné, et si chacun assume son rôle, il n'y a aucune raison qu'il n'aille pas à son terme.

A. Quel cadre pour ce document ?

Cet ouvrage s'attache à donner des pistes pour mettre en oeuvre de petits projets, souvent dimensionnés de l'ordre de un à six mois de développement, développés le plus souvent en interne dans un service ou une entreprise.

B. La demande initiale

Non formalisée, elle tient en général en une ligne ou une demi-page. L'archétype est : *il faut informatiser telle procédure nouvelle, c'est un écran et un état, et c'est pour demain*.

Le premier travail consiste à rédiger la demande pour qu'elle puisse être exploitable, en s'attachant à définir :

- l'objectif visé
- les matériaux (de quoi l'on part)
- le cadre juridique
- les délais souhaités

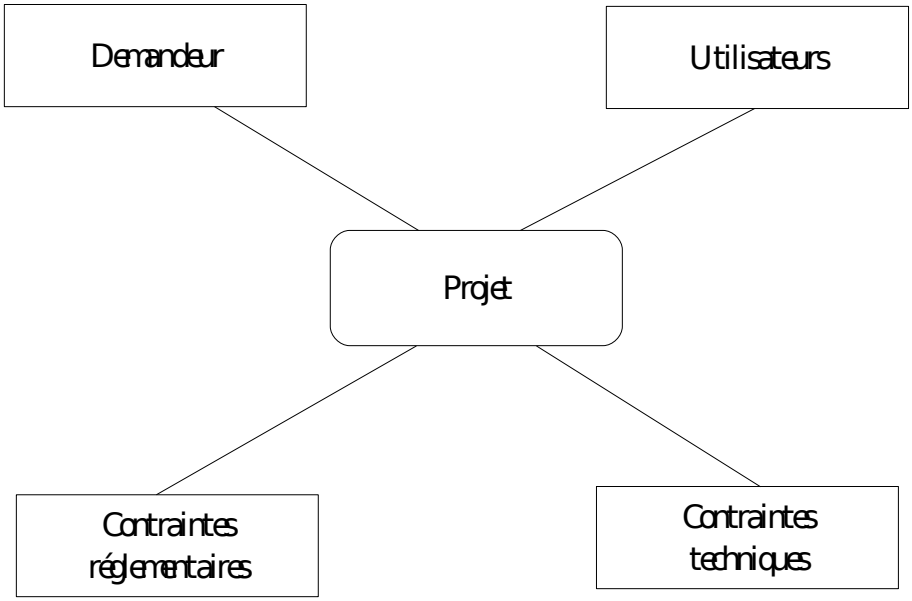
L'informaticien, à ce stade, va jouer un rôle de faciliteur, en aidant à faire exprimer la demande. Bien souvent, il faut « décrypter » le message, en isolant ce qui ressort de la valorisation personnelle du demandeur pour prendre en compte la demande réelle. Deux écueils sont à éviter :

- d'une part, limiter le périmètre du projet en faisant confiance au demandeur, qui va s'auto-censurer. Cas typique : « cette fonction, ce n'est pas la peine, ça représente trop peu de dossiers, ou on n'en aura pas besoin ». En général, il faut, à ce stade, noter la demande qui peut être parfois très structurante pour la suite du projet

- d'autre part, porter un jugement sur la faisabilité (ou plutôt sur la non-faisabilité) d'une fonction ou d'une partie de la demande. C'est le rôle de l'analyse préalable de s'assurer des conditions de réalisation ou de mise en oeuvre, pas de la demande initiale.

En résumé, il faut mettre en forme la demande, qui doit être la plus complète possible, avec, si nécessaire, un premier exercice de définition des priorités.

Il est aussi important, au niveau de la demande initiale, de cadrer le projet dans son contexte.



Le projet ne pourra aboutir dans de bonnes conditions que si l'ensemble des contraintes sont prises en compte :

- le demandeur paye le produit. Il est normal que le logiciel rende les services demandés
- les utilisateurs vont travailler avec le logiciel régulièrement. Il doit s'intégrer dans leurs habitudes de travail, tant en terme d'ergonomie que de processus (si ceux-ci ne sont pas redéfinis)
- les contraintes réglementaires sont trop souvent ignorées ou mal prises en compte. Le demandeur connaît la procédure, mais va passer sous silence « ce qui va de soit ». Il est indispensable de se procurer les textes réglementaires *ad-hoc* et de les intégrer dans sa réflexion
- enfin, un projet ne se crée pas de toute pièce, il s'intègre dans un existant qui peut être très structurant.

Ne pas tenir compte d'une de ces contraintes, c'est s'exposer à de graves difficultés, soit dans la phase de développement (le demandeur se rend compte que le produit ne répond pas à sa demande), soit dans la phase de test et de mise en production (les utilisateurs ne veulent pas utiliser le produit parce qu'il ne correspond pas à l'application qu'ils font de la procédure), soit en production, quand on s'aperçoit que le logiciel ne respecte pas la réglementation.

C. La restitution de la demande

Il s'agit de formaliser ce qu'on a compris de la demande. Cette restitution va permettre :

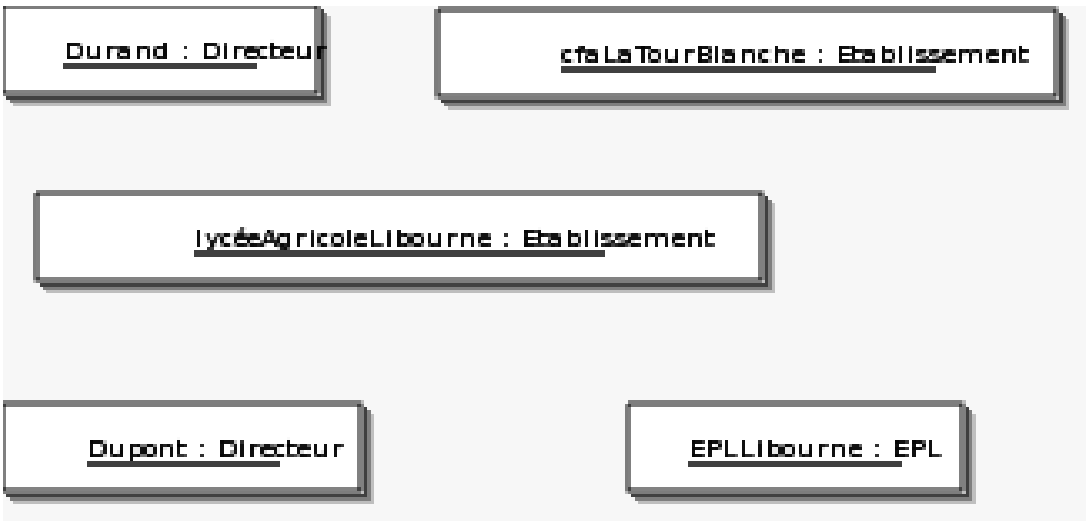
- de vérifier le périmètre du projet

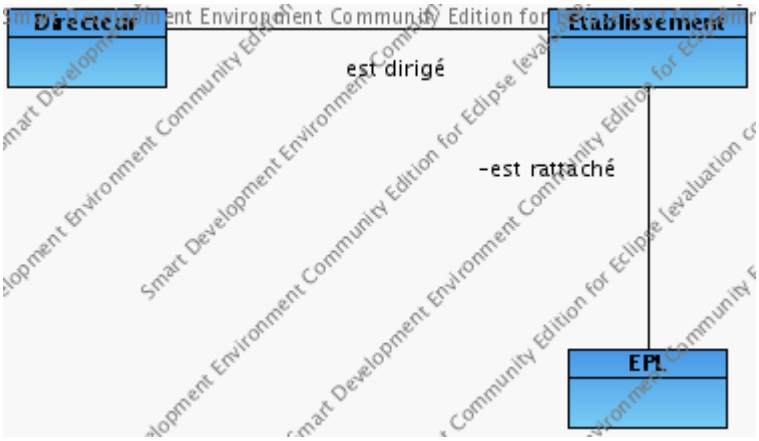
- de préciser certains points techniques. Par exemple, si la demande initiale prévoit l'import de données statistiques, on va approfondir cette demande pour détailler le mode d'import : d'où viennent les données, sous quel format, la fréquence...
- de pouvoir estimer le « volume » du projet, et de définir les différentes phases et modules.

Pour restituer cette demande, et en partant du principe qu'un mauvais dessin vaut mieux qu'un grand discours, on va s'appuyer sur un certain nombre de schémas UML :

- les cas d'utilisation, qui permettent d'identifier les différents modules potentiels. Chaque cas d'utilisation débute par un schéma représentant les acteurs et les grandes fonctions. Mais le plus important est la partie *rédactionnelle* qui suit, et qui va décrire, de façon la plus précise possible, ce qu'on va faire.
- Les diagrammes de fonctionnement (diagramme d'activité, de collaboration), qui vont permettre d'expliciter certains flux ou fonctions complexes
- les diagrammes d'objets et de classes, qui permettent de représenter les objets facilement compréhensibles par des non informaticiens, et de restituer la classification que l'on fait de ces objets.

Voici un exemple, concernant l'organisation des établissements d'enseignement agricole. Un établissement est rattaché à une structure juridique, un EPL. Les établissements sont de plusieurs types : lycées agricoles, centres de formation des apprentis, centres de formations professionnels pour adultes... Chaque établissement est dirigé par un directeur, le directeur de l'établissement support de la structure juridique EPL est le directeur de cet EPL.





Dans cet exemple, le schéma d'objets permet de vérifier qu'on a bien compris comment les établissements sont organisés, et le schéma de classes la conceptualisation correspondante.

Cette restitution de la demande doit faire l'objet d'une validation du demandeur (le maître d'ouvrage). Elle est essentielle : c'est ce document qui va faire foi entre le maître d'ouvrage et l'assistance à maîtrise d'ouvrage.

D. L'analyse initiale

Elle est destinée aux informaticiens. Elle vise à dégrossir les processus, et doit impérativement déboucher sur une maquette.

On va utiliser de façon intensive les diagrammes d'objets et de classes pour décrire les entités manipulées, associés aux diagrammes de collaboration, voire d'activité.

A ce stade, on montre principalement les classes « physiques », celles qui représentent les objets réels, et qui seront pour la plupart implémentées en partie dans la base de données. On peut faire apparaître, si c'est nécessaire, des classes virtuelles, comme les modules de calcul ou de traitement lourd (exports, calculs financiers...).

Les grands ensembles de traitement (les modules) doivent être identifiés. L'ensemble de ces modules et des classes physiques serviront à préparer la maquette.

E. La maquette

De conception simple, elle n'a pas vocation à « fonctionner », même s'il est toujours intéressant de pouvoir naviguer dans les écrans. Elle doit représenter la vision que l'on a de l'application finale, en représentant de façon détaillée les objets et les zones de saisie. Elle peut être incomplète (champs d'une table manquants), à condition de préciser pourquoi on n'a pas pris le temps de tout détailler.

La maquette a plusieurs objectifs. Elle doit être présentée à l'ensemble des personnes impliquées dans le projet, le demandeur et les utilisateurs. Elle va permettre :

- de vérifier qu'on n'a pas oublié une classe d'informations importante
- de vérifier qu'on a bien compris l'enchaînement des étapes et les objectifs recherchés
- comme les utilisateurs voient du « concret », cela les rassure et leur permet de s'assurer que leurs demandes ont bien été prises en compte
- de définir les étapes, en liaison avec le demandeur et les utilisateurs, et de choisir les modules qui seront développés et déployés en premier

Lorsque l'on réalise une maquette, il faut éviter quelques écueils :

- passer trop de temps à fignoler les détails : cela n'apporte rien à la compréhension du système, et le travail réalisé risque d'être abandonné par la suite
- la présentation de la maquette peut faire ressortir des tensions entre les utilisateurs et les demandeurs. Dans ce cas, l'arbitrage doit être rendu par le demandeur, soit immédiatement, soit après expertise ultérieure
- il faut éviter de prendre partie lors de discussions entre les personnes, mais recadrer le cas échéant par rapport à la demande initiale.
- Il ne faut pas annoncer que toute demande est impossible sous prétexte qu'on a fait une maquette qui ne correspond pas ! Prendre alors en compte les besoins exprimés, qui étaient probablement sous-jacents dans les études préalables mais non ou mal exprimés. Si nécessaire, demander un arbitrage ultérieur. Toutefois, il ne faut pas ressortir de la présentation avec un projet complètement remanié. Si c'est le cas, il vaut mieux alors recommencer le projet depuis les phases de demande initiale et d'analyse.

F. L'analyse détaillée – l'approche objet

On rentre vraiment dans le vif du sujet ! Il est fortement souhaitable de privilégier une approche objet. Les avantages :

- facilité de maintenance
- fiabilité du code
- vitesse de programmation

Les inconvénients :

- longue phase de codage avant de pouvoir « voir » quelque chose fonctionner (on code les classes avant l'interface)
- les performances sont moins bonnes en général qu'un développement classique, l'utilisation de classes induisant très souvent plus de lignes de codes et plus de ressources lors de l'exécution. Selon ce qu'on veut faire, il faut alors trouver le bon compromis entre la facilité de maintenance et la vitesse d'exécution. Il faut aussi estimer la charge machine nécessaire : une application destinée à une dizaine d'utilisateurs sera plus tolérante qu'une

application web destinée à supporter plusieurs millions de connexions simultanées !

G. Les tests unitaires

Ils sont à effectuer par le développeur. Ils servent à s'assurer que le code fonctionne de façon correct, de manière indépendante.

On va créer un jeu de tests, portant sur plusieurs items, et prenant en compte l'ensemble des cas de figure qu'on a intégré lors de la programmation. Chaque module doit être testé de façon indépendante, pour ne pas avoir de « pollution » par du code non débogué. L'important est de tester au fur et à mesure, pour s'assurer que ce qu'on vient d'écrire est cohérent. Si on ne fait pas ce travail, le débogage devient alors vite fastidieux, pour retrouver le module qui pose problème. Utiliser des outils susceptibles d'afficher le contenu des variables et de mettre des points d'arrêt apporte un confort certain : la plupart des outils intègrent un débogueur, mais on peut aussi en simuler un en arrêtant l'exécution et en affichant manuellement les variables qui posent problèmes.

Les tests unitaires sont surtout utilisés pour vérifier le fonctionnement de l'interface.

H. Les tests approfondis

Aujourd'hui, les professionnels des tests recommandent d'écrire les tests à réaliser au moment où l'on définit les besoins. Cela va permettre de préciser la demande et d'éviter les biais liés au processus de développement (on modifie inconsciemment ce qu'il faut tester en fonction de ce qu'on a réalisé).

Il est préférable de faire tester par des personnes qui n'ont pas travaillé sur le développement, en y intégrant le demandeur (ou maîtrise d'ouvrage). La phase de test est un processus hélas trop souvent bâclé, qui aboutit souvent à des coûts cachés (bugs découverts sur le tard).

Les tests doivent s'attacher à expertiser la conformité de l'application :

- vis à vis de la demande (l'application répond-elle bien à ce qu'on a demandé ?)
- vis à vis de la réglementation
- vis à vis des utilisateurs (est-elle adaptée au travail et aux habitudes quotidiennes ?)

Il faut ainsi impliquer plusieurs profils d'utilisateurs différents !

Dans la pratique, pour des applications relativement modestes, il faut prévoir une année de mise au point et de correction de bugs à partir de la mise en production. Au delà, les bugs sont nettement plus rares (courbe logarithmique). La correction est en général plus simple, le code plus « propre », en développement objet.

I. La rédaction de la documentation

Avec les logiciels modernes, il est souvent superflu de réaliser des guides d'utilisation très détaillés, surtout si l'application doit être utilisée par un nombre restreint d'utilisateurs qui maîtrisent bien leur procédure. Par contre, il est important de rédiger le guide « Administrateur », qui va décrire la solution technique retenue, l'implantation de la base de données, le paramétrage, et la gestion des droits.

Sans ce document, il est souvent quasiment impossible de mettre en production le logiciel, surtout si celle-ci est décalée dans le temps.

II. Les méthodes Agiles

Face à la complexité de mise en oeuvre de projets et pour limiter les risques inhérents aux démarches traditionnelles, à savoir, effet tunnel, retards de mise en oeuvre, non prise en compte des besoins apparus en cours de développement, inadéquation du logiciel vis à vis des pratiques des utilisateurs..., sont apparues des méthodes basées sur des principes différents.

A. Le manifeste Agile[1]

En 2001, aux États-Unis, dix-sept figures éminentes du développement logiciel se sont réunies pour débattre du thème unificateur de leurs méthodes respectives, dites méthodes agiles. Les plus connus d'entre eux étaient Ward Cunningham l'inventeur du Wiki via WikiWikiWeb, Kent Beck, père de l'extreme programming et cofondateur de JUnit, Ken Schwaber et Jeff Sutherland, fondateurs de Scrum, Jim Highsmith, prônant l'ASD, Alistair Cockburn pour la méthode Crystal clear, Martin Fowler, et Dave Thomas ainsi que Arie van Bennekum pour DSDM (Dynamic System Development Method) la version anglaise du RAD (Développement rapide d'applications). Ces 17 experts venant tous d'horizons différents réussirent à extraire de leur concepts respectifs des critères pour définir une nouvelle façon des développer des logiciels :

De cette réunion devait émerger le Manifeste Agile, considéré comme la définition canonique du développement Agile et de ses principes sous-jacents.

Le Manifeste Agile débute par la déclaration suivante (traduction) :

Nous avons trouvé une voie améliorant le développement logiciel en réalisant ce travail et en aidant les autres à le faire. De ce fait nous avons déduit des valeurs communes.

Le Manifeste Agile est constitué de 4 valeurs et de 12 principes fondateurs.

1. Les 4 Valeurs

Les quatre valeurs fondamentales Agiles sont :

- L'interaction avec les personnes plutôt que les processus et les outils.
- Un produit opérationnel plutôt qu'une documentation pléthorique.
- La collaboration avec le client plutôt que la négociation de contrat.
- La réactivité face au changement plutôt que le suivi d'un plan.

2. Les 12 principes

- Notre première priorité est de satisfaire le client en livrant tôt et régulièrement des logiciels utiles.
- Le changement est accepté, même tardivement dans le développement. Les processus agiles exploitent le changement comme avantage compétitif pour le client.
- Livrer fréquemment une application fonctionnelle, toutes les deux semaines à deux mois, avec une tendance pour la période la plus courte.
- Les gens de l'art et les développeurs doivent collaborer quotidiennement au projet.
- Bâtissez le projet autour de personnes motivées. Donnez leur l'environnement et le soutien dont elles ont besoin, et croyez en leur capacité à faire le travail.

1 Source : http://fr.wikipedia.org/wiki/Manifeste_agile

- La méthode la plus efficace de transmettre l'information est une conversation en face à face.
- Un logiciel fonctionnel est la meilleure unité de mesure de la progression du projet.
- Les processus agiles promeuvent un rythme de développement soutenable. Commanditaires, développeurs et utilisateurs devraient pouvoir maintenir le rythme indéfiniment.
- Une attention continue à l'excellence technique et à la qualité de la conception améliore l'agilité.
- La simplicité - l'art de maximiser la quantité de travail à ne pas faire - est essentielle.
- Les meilleures architectures, spécifications et conceptions sont issues d'équipes qui s'auto-organisent.
- À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son comportement dans ce sens.

B. Les apports des méthodes Agiles

Les méthodes dites « agiles » s'affranchissent des contraintes inhérentes aux méthodes traditionnelles basées sur des cahiers des charges très complets et très lourds, avec rapidement des relations conflictuelles sur les délais de mise en oeuvre ou la complétude fonctionnelle.
Elles sont basées sur quelques principes simples :

- un cahier des charges « minimaliste », de quelques pages maximum
- des arbitrages pour hiérarchiser les cas d'utilisation à développer en priorité
- une présence permanente du client, qui précise ses besoins et valide en temps réel ce qui doit être développé ou non
- des tests unitaires et de recette réalisés à partir de scénarios établis avant le codage, validés ensemble par les développeurs et les fonctionnels
- des livraisons par modules réduits, impliquant des cycles de développement courts (2-3 semaines maximum)
- des itérations courtes, pour faire avancer le produit, et mettre en production au fur et à mesure

Il existe plusieurs méthodes actuellement, dont les plus connues sont[1] SCRUM, Extreme Programming (XP), Rapid Application Developpement (RAD), Dynamic systems development method (DSDM), Adaptive software development (ASD), Feature Driven Development(FDD), Crystal clear, Processus Urbanisant les Méthodes Agiles (PUMA).

1. Un exemple : l'extreme Programming[2]

- Client sur site : un représentant métier accompagne en permanence l'équipe de développement
- jeu du planning, revu au début de chaque itération, en précisant les cas d'utilisation à implémenter
- intégration continue : toute modification apportée est archivée dans le référentiel, le projet est recompilé et les tests sont rejoués. A chaque étape, le projet est alors versé sur une machine de validation
- petites livraisons : pas plus de quelques semaines entre chaque livraison

1 Source : http://fr.wikipedia.org/wiki/Méthode_agile
2 Source : 01 Informatique du 13/03/2008

- rythme soutenable : le développeur doit conserver sa vie personnelle. Pas plus de 7 à 8 heures de travail par jour
- tests de recette, écris conjointement par les développeurs et les fonctionnels à chaque itération. Ils constituent la base de validation de celle-ci (juge de paix)
- tests unitaires : chaque modification de code est soumise à des tests unitaires pour vérifier les anomalies. Ils sont écrits avant le code
- conception simple : on ne développe que ce qui est demandé, sans créer du code générique. Le développeur s'en tient à la tâche de la journée. Ne pas mélanger avec la création d'outils ou de classes génériques conçues hors projet.
- utilisation de métaphores, pour faciliter la compréhension entre les développeurs et les fonctionnels
- refactoring : les équipes remanient le code, et améliorent ses fondement sans changer ni les contrats, ni les services qu'il rend
- appropriation du code : il appartient à toute l'équipe, chacun peut le modifier pourvu que les tests réussissent
- convention de nommage : le pendant du précédent, indispensable pour que le code reste lisible. Normes et méthodes communes pour tous les objets manipulés
- programmation en binôme. Rarement appliqué, mais nécessité de conserver des relations et des échanges permanents entre les développeurs

C. Quelques conseils...

Choisir une méthode est une affaire complexe, et qui nécessite non seulement de la connaître, mais surtout d'en maîtriser les principes. Sans vouloir à tout prix suivre une méthode, on peut simplement suivre des conseils simples :

1. Limiter la phase initiale de description de la demande au strict nécessaire

et dialoguer en permanence avec le demandeur. Si c'est possible, avoir son bureau dans la même pièce ou dans une pièce voisine est un plus certain. Privilégier au maximum l'oral à l'écrit, l'écrit ne venant que dans une seconde étape comme « trace » de ce qui a été dit.

2. Elaborer la maquette avec le demandeur

Il faut obtenir du demandeur du temps pour élaborer ensemble la maquette : le programmeur dessine avec un logiciel quelconque les écrans, sous la directive du demandeur.

Par contre, si le logiciel a vocation à être utilisé par d'autres personnes, il faut prévoir des réunions de validation régulières, pour être sûr que le projet ne soit pas que le fruit d'une personne.

Les maquettes ne s'élaborent pas en une seule fois, mais au début de chaque module.

3. Appuyer le développement sur des méthodes modernes

Les méthodes Agiles nécessitent de coder rapidement. Il faut s'appuyer le plus possible sur des outils qui vont faciliter le codage, et donc privilégier le développement objet. S'appuyer sur un framework, qu'il soit industriel ou

personnel, est quasiment indispensable, pour éviter de devoir coder des fonctions de base (gestion des menus, accès aux bases de données...).

De même, aujourd'hui, on ne devrait développer des logiciels en mode client lourd que dans des cas très précis :

- manipulation de fichiers présents sur le poste de travail
- manipulations complexes à l'intérieur du logiciel : glisser-déposer, clic-droit permanent, multi-fenêtrage obligatoire...
- interactions fortes avec d'autres logiciels (utilisation d'objets OLE par exemple)
- traitements lourds, sollicitant fortement le processeur ou la carte vidéo.

A l'inverse, les applications de gestion traditionnelles « classiques » s'accommodent parfaitement des clients légers, et résolvent de fait les problèmes de mise à jour.

Néanmoins, le choix de l'outil est souvent imposé par le demandeur ou la société qui sollicite le développement ; on ne peut alors intervenir que sur les méthodes.

4. Fournir des modules régulièrement

Sur un petit projet, il faut prévoir une phase d'appropriation des outils puis, ensuite, s'astreindre à fournir des modules tous les quinze jours – trois semaines au maximum.

Fournir un module ne signifie pas fournir une nouvelle version : on peut décider de ne fournir, sur l'ensemble du développement, que 2 ou 3 versions qui représentent un tout cohérent. Néanmoins, le demandeur se verra fournir des modules régulièrement, qu'il devra tester pour vérifier que ce qui est produit correspond à ce dont il a besoin.

Il est également indispensable d'accepter que tout le travail déjà réalisé peut être remis en cause. Ce qui importe, c'est la satisfaction du client, pas la « beauté » du logiciel ou le nombre de lignes produites.

Il ne sert également à rien de travailler comme un « forçat » : au delà d'une journée normale de travail, la qualité s'en ressent et on passe de plus en plus de temps à rechercher des solutions qui seront évidentes le lendemain matin... Un bon week-end permet également de recharger les batteries !

5. Travailler en commun, écrire les tests au préalable

Même si on travaille seul, il faut s'astreindre à se fixer les objectifs pour la journée, de préférence sur un tableau blanc accroché au mur. Si on travaille à plusieurs, ces objectifs sont définis le matin, sur tableau blanc. Bannir les outils informatiques, qui ne permettent pas une bonne visibilité des tâches et, surtout, bloquent les échanges oraux.

Avant de développer un module, écrire au préalable les tests à réaliser, qui permettront de vérifier son fonctionnement. Ne pas écrire les tests après développement : ils auront trop tendance à « suivre » la logique de ce qu'on aura écrit.

Si on travaille à deux sur un projet, se répartir les rôles : l'un s'occupe des tests, l'autre du développement. On peut également envisager d'intervertir les rôles, chacun développant un module différent et se chargeant de tester le module de l'autre.

Enfin, à chaque fois qu'un module est modifié, il doit être retesté entièrement, notamment au niveau des tests unitaires.

6. Amélioration du code ou des outils

Régulièrement, pendant le développement, on peut avoir besoin d'apporter des améliorations soit au code déjà produit, soit au « moteur » utilisé pour coder (*framework*).

Ces améliorations doivent être traitées non sur le moment, mais à des moments choisis ; on se tient à l'objectif de la journée. Si on doit travailler sur le *framework*, on le fait pendant une phase adaptée, dans le cadre des objectifs fixés pour la journée.

7. Documenter et valider le logiciel

Le logiciel ne sera considéré comme valide que si le demandeur décide qu'il répond à son besoin. A contrario, si le logiciel correspond à la demande initiale (ou au cahier des charges), mais pas au besoin réel, il ne sera pas utilisé, et le projet sera un échec.

La documentation ne sert pas à grand chose, hormis :

- les commentaires judicieusement placés dans le code (utiliser les méthodes de description standards, les commentaires pouvant alors être analysés par des moteurs adéquats, comme PHPAnalyzer par exemple).
- La documentation d'implémentation : où et comment installer la base de données et le logiciel, les paramétrages...
- la documentation utilisateur, de préférence en ligne ; si on doit imprimer une documentation papier, ce sera une compilation de la documentation en ligne.

D. Les inconvénients des méthodes agiles

Si elles présentent beaucoup d'avantages, elles ne peuvent être utilisées systématiquement dans tous les cas de figure.

- sur des très gros projets, elles ne peuvent être mises en oeuvre qu'à condition de segmenter les modules en une taille raisonnable ;
- sur le plan réglementaire, il est très difficile de fixer des conditions dans le cadre d'un marché ou d'un contrat. Les objectifs ou les prescriptions pouvant évoluer au cours du développement, il est complexe de mettre en oeuvre des conditions qui permettent d'une part la livraison du logiciel, et d'autre part un bénéfice pour le maître d'oeuvre. L'estimation des coûts est beaucoup plus complexe à faire dans ce cadre. De même, la gestion des délais de livraison, et la qualité des « livrables » est plus difficile à cerner (pas de référence possible à telle fonction décrite dans le cahier des charges et non ou imparfaitement implémentée, par exemple).
- La budgétisation de ces projets est plus complexe : il faut prévoir une partie fixe, et une partie variable, qui ne sera utilisée que si nécessaire.
- Il faut un demandeur « actif », volontaire, et qui dialogue avec les développeurs. La relation doit être une relation de confiance, non de suspicion réciproque.

Néanmoins, dans tous les petits projets, on aura tout intérêt à les mettre en oeuvre, ne serait-ce que parce que l'implication obligatoire du demandeur augmente les chances que le logiciel sera utilisé *in fine*.

III. Les techniques d'analyse

A. La base de données

Elle est créée en fin d'analyse, juste avant le codage, à partir des classes persistantes qui ne contiennent pas de code.

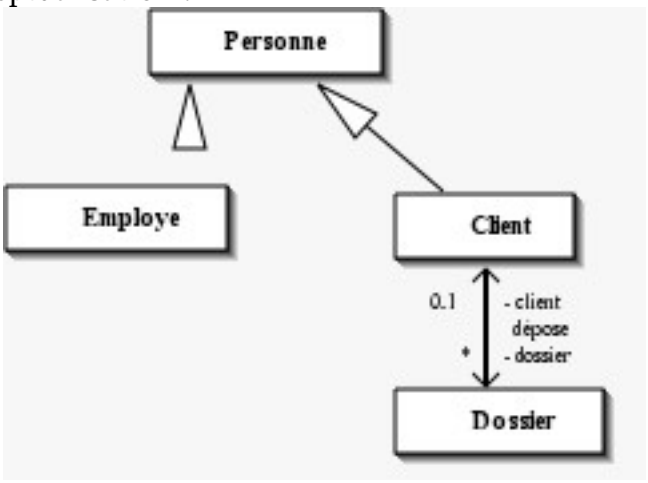
On utilise un schéma relationnel, le langage SQL étant le plus répandu et le plus efficace pour extraire les données dans un cadre de production.

B. L'héritage

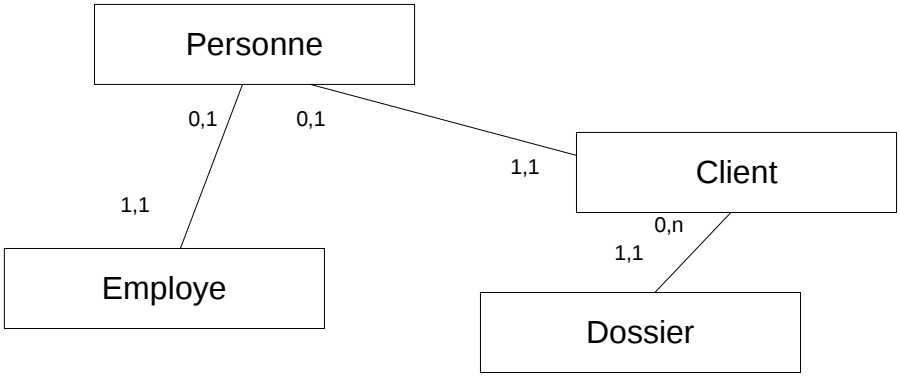
1. Pour les bases de données

Il ne porte jamais sur les données (mais parfois sur les conteneurs des données dans l'application). Les bases de données *objet* n'ont aucun intérêt, sauf pour des usages très particuliers, comme l'information géographique par exemple. Dans ce cas, on incorpore du code à l'objet pour en faciliter la manipulation.

Exemple de conceptualisation :



Le résultat en base de données :



D'une façon générale, l'héritage est sans intérêt dans une base de données, les jointures sont nettement plus efficaces, bien que « moins élégantes ».

2. Les listes

Tous les langages incluent des mécanismes de gestion des listes : en Powerbuilder, les *datastore*, en PHP, les tableaux, en Access : les formulaires...

Une liste est toujours **indépendante** des objets (des tables) de la base de données. On crée systématiquement un objet de base, qui sera soit hérité (on parlera alors de liste non instanciable), soit instancié directement.

Le premier cas permet une gestion plus personnalisée de chaque objet liste, le second est plus rapide à utiliser.

Premier cas : l'instanciation directe.

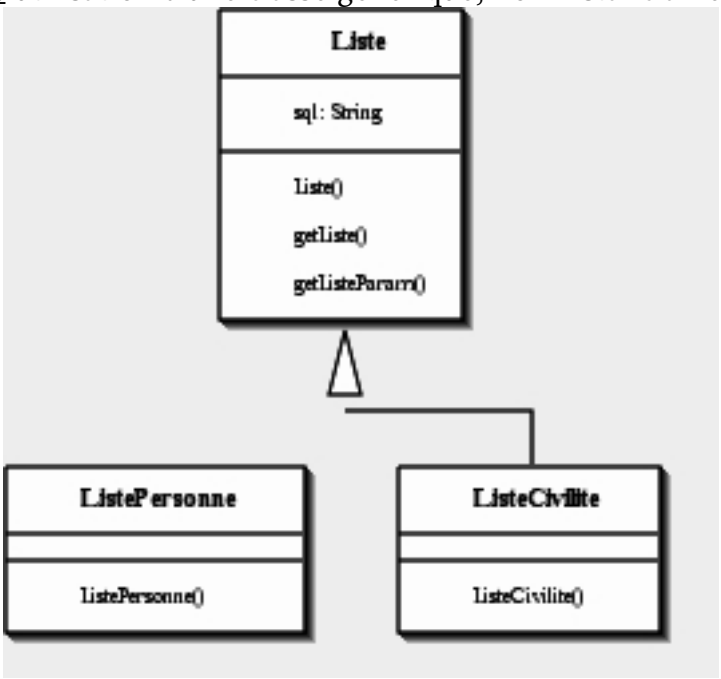
On crée une classe qui va contenir :

- un constructeur, pour assigner le code SQL
- les commandes de lecture (avec ou sans arguments), pour récupérer la liste manipulable dans le langage de développement.

Exemple d'utilisation directe de la classe, en PHP :

```
1 $ls_sql="select NumeroDossier, Libelle from Dossier where NumPersonne = %cle% and
  Annee = %annee%";
2 liste = New Liste($ls_sql);
3 $argument = new array();
4 $argument[0] = clé;
5 $argument[1]=année;
6 $tableau = new array();
7 $tableau = liste.getData ($argument);
```

Deuxième cas : utilisation d'une classe générique, non instanciable.



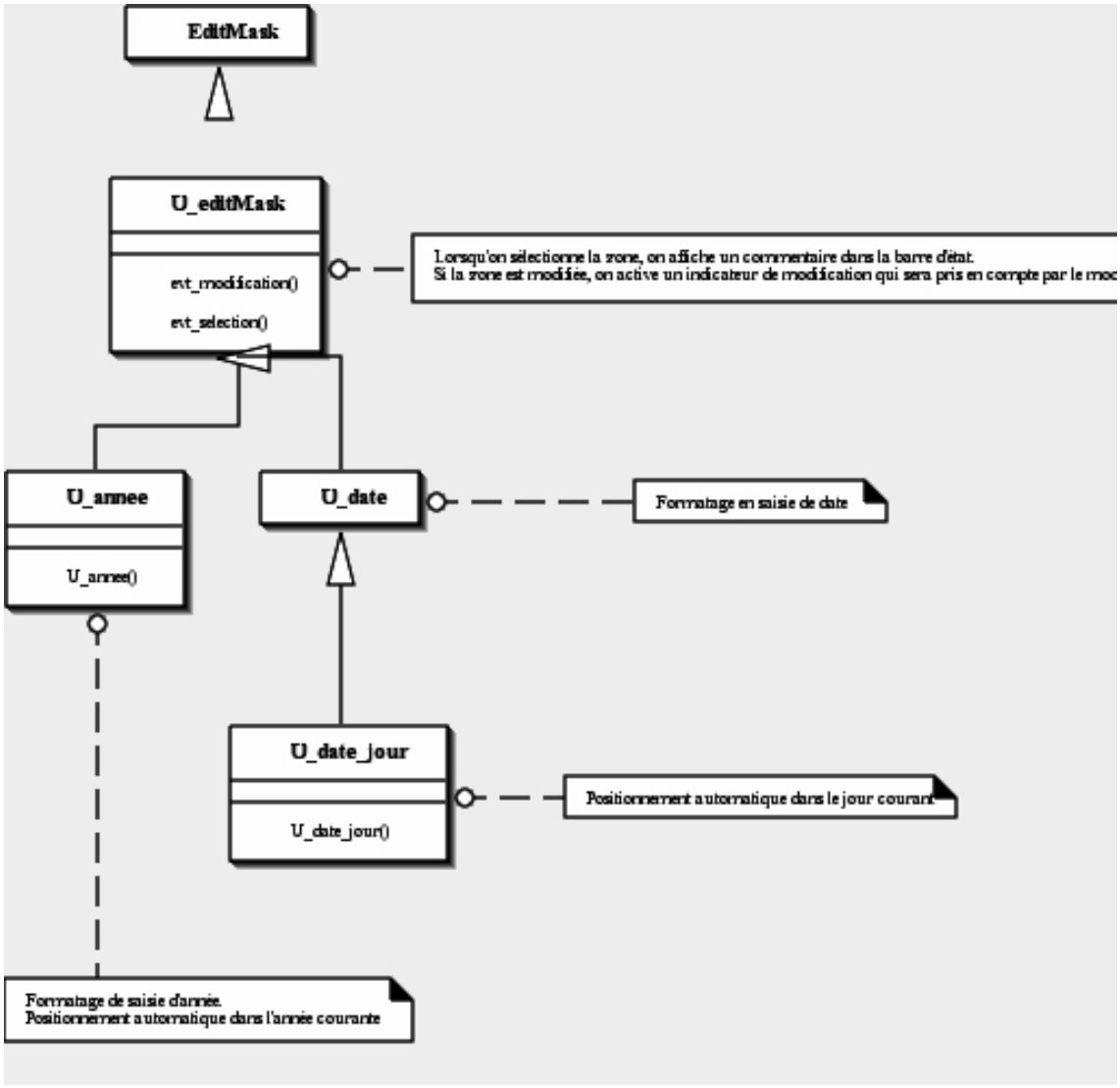
On crée la classe Liste, non instanciable, et on définit les méthodes nécessaires dans les classes filles : getListe(), getListeParam(parametre)...

On hérite ensuite cette classe pour chaque liste dont on a besoin. Dans le script *Constructor*, on modifie les paramètres spécifiques, notamment en indiquant le script SQL qu'il faudra utiliser.

Enfin, dans les modules, il suffit d'instancier les classes spécifiques de chaque liste.

3. Les objets graphiques

Ce sont les objets qui sont par essence les plus souvent sujets à héritage. Ainsi, en PowerBuilder, à partir du type *EditMask* (un objet de saisie de zone de type texte ou numérique), on peut imaginer les héritages suivants :



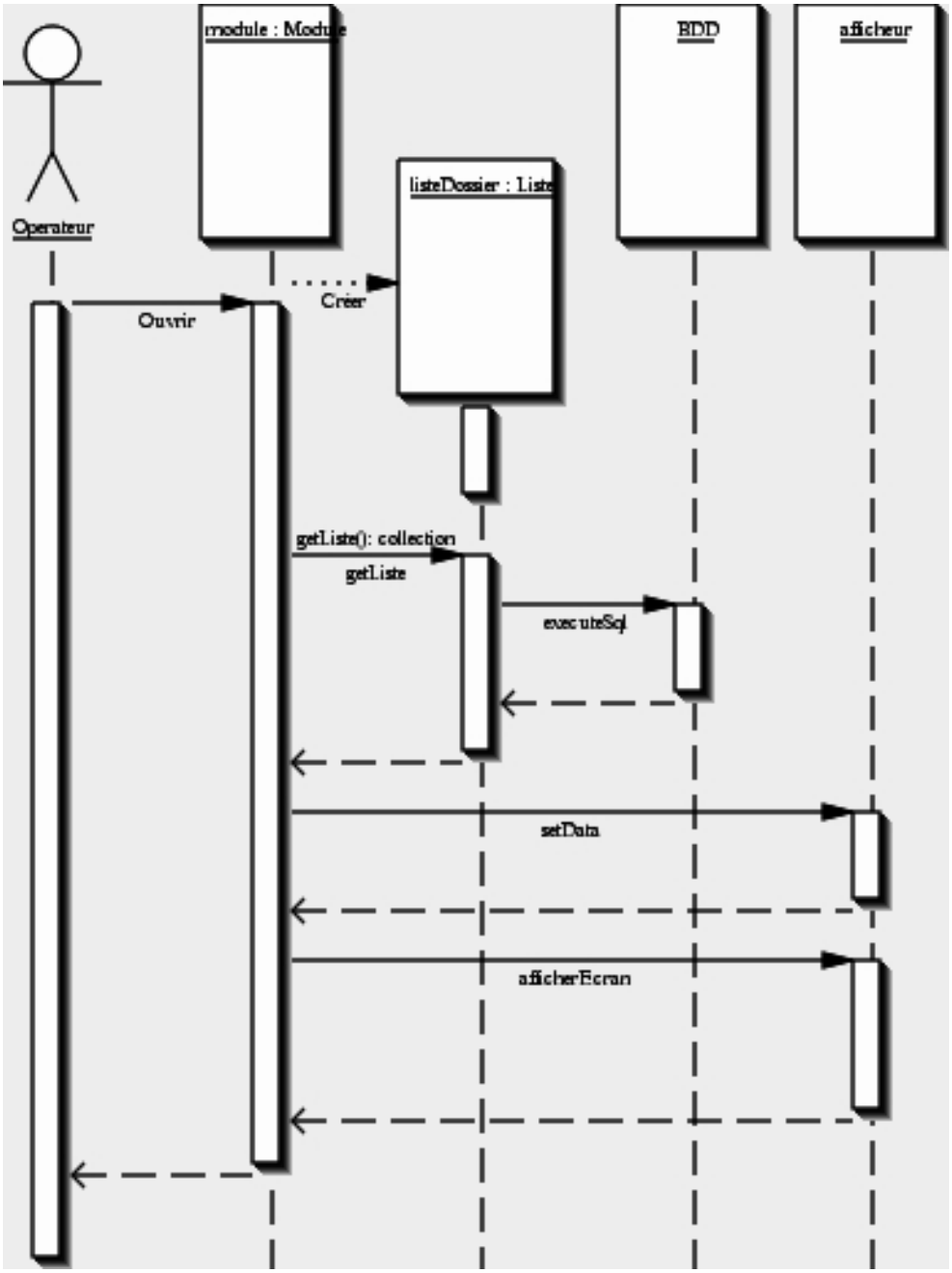
C. Le séquençage des opérations

C'est l'opération la plus complexe à intégrer quand on débute l'analyse objet, tellement elle est différente de l'approche « classique » descendante.

Le principe de base est simple. Un objet n'est accessible que lorsqu'on « l'active », et cette activation n'est possible que par instanciation. Une fois activé, un objet ne fait rien par lui-même : il faut une intervention extérieure pour qu'il réagisse. Cette intervention extérieure est déclenchée par les méthodes. Un objet ne renvoie que ce qu'une méthode est capable de retourner, suite à son activation.

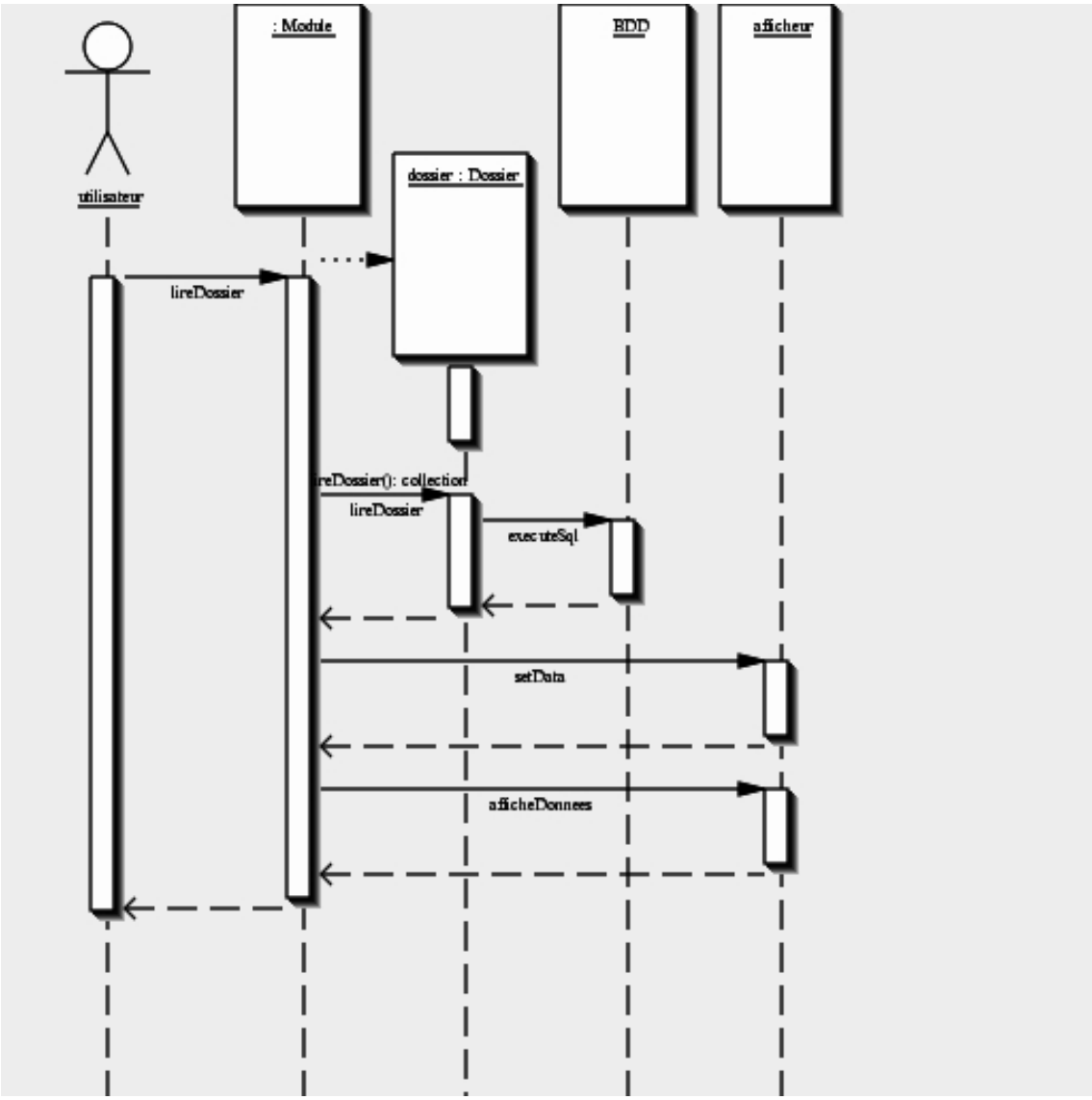
Un objet peut faire appel à d'autres objets, toujours par le principe des instanciations et des appels de méthodes.

Un diagramme est incontournable pour décrire l'ordonnancement des tâches et le séquençage des modules : le diagramme de séquence.



L'utilisateur ouvre le module de saisie. La première partie consiste à afficher la liste des dossiers correspondant. Cet affichage va être réalisé en plusieurs phases :

- création d'une instance de l'objet Liste
- l'objet interroge la base de données, pour récupérer la liste des dossiers
- le module de saisie passe les données au module d'affichage (l'afficheur)
- une fois toutes les données passées, le module demande à l'afficheur d'afficher les données
- l'utilisateur récupère « la main », et peut continuer le travail.



Dans cette seconde étape, l'utilisateur va demander l'affichage d'un dossier particulier. Le scénario est identique : création d'une instance de dossier, lecture des données dans la base de données, attribution des données au module d'affichage, puis affichage des données.

La validation des données saisies fonctionne de la même façon.

De même, on peut utiliser les mêmes mécanismes pour décrire l'interaction entre l'utilisateur et l'écran.

Il est important de noter que le module de saisie est toujours le même, que ce soit pour l'affichage que la validation des données. Mais la validation proprement dite, c'est à dire l'écriture en base de données, est gérée par la classe Dossier, et non par le module : cela facilite la réutilisation et la maintenance, le code n'étant décrit qu'à un seul endroit. Le module de saisie ne sert que d'interface et de séquenceur.

D. L'interface

Une interface est un concept UML qui permet de décrire uniquement les méthodes visibles, sans que la classe sous-jacente ne soit connue.

On va utiliser une interface si on souhaite utiliser, selon le contexte, des classes différentes pour réaliser des opérations ayant la même finalité.

Par exemple, si l'on souhaite identifier les utilisateurs selon plusieurs modalités (login stocké dans une base de données, identification via un annuaire LDAP, via un serveur d'identification unique...), on va créer une interface destinée à présenter la même méthode, quelle que soit le mode d'identification retenu.

Selon le cas, on utilisera des classes différentes, qui fournissent pourtant les mêmes méthodes à l'application.

IV. Les bonnes pratiques de développement en PHP

A. Les outils

« Pas de bon artisan sans bons outils » !

« Il ne sert à rien de réinventer la roue » !

Il existe une multitude d'éditeurs de code, que ce soit sous Windows que sous Linux. Chacun a ses avantages et inconvénients, mais, si aucun outil n'est imposé, il vaut mieux se tourner vers les standards « du marché ».

Aujourd'hui, l'outil de référence pour développer, c'est Eclipse, qui dispose d'un module php très complet (affichage des classes, analyse et auto complétion du code...). Pour les diagrammes UML, on peut utiliser soit des produits payants, intégrés ou non à Eclipse, soit ArgoUML, un « vieux » projet OpenSource qui a été profondément remanié récemment, et qui présente la plupart diagrammes UML 2.

Pour ce qui est de l'éditeur HTML, pour concevoir les pages, on pourra utiliser NVU (remplacé récemment par KompoZer), à condition d'inhiber l'analyse syntaxique ; ces éditeurs syntaxiques posent souvent des problèmes avec l'utilisation de classes de templates de type SMARTY, ils ont tendance à remplacer les caractères spéciaux, comme les accolades, par le code de caractère HTML correspondant, ce qui plante bien évidemment l'exécution du programme par la suite. Très souvent, on finit par utiliser un éditeur graphique pour dessiner la page, que l'on reprend ensuite manuellement sous Eclipse pour rajouter le code adéquat.

B. La séparation du code PHP et du code HTML

PHP permet, intrinsèquement, de développer de deux manières différentes. La première, la plus évidente, consiste à insérer, dans une page HTML, du code PHP quand c'est nécessaire, ou de faire l'inverse, c'est à dire utiliser la commande *echo* pour générer du code HTML depuis une page PHP. C'est la méthode classique des langages procéduraux, où le code s'enchaîne du début à la fin du programme.

L'avantage de cette méthode est la facilité de conception : il suffit d'enchaîner les ordres les uns derrière les autres. L'inconvénient, c'est une lourdeur de code, et des pages complètement illisibles, surtout si on y intègre en plus du JavaScript.

L'autre approche consiste à séparer complètement le code PHP de l'affichage des données. C'est l'approche *objet*. Dans un premier temps, on code le fonctionnement de l'application : les accès aux données, les traitements... dans des classes. Dans un second temps, on décrit l'enchaînement des actions, par l'intermédiaire de modules PHP, des pages qui feront appel aux classes chaque fois que ce sera nécessaire. Ces pages sont écrites en principe de manière procédurale. Enfin, l'affichage des pages HTML sera confié à un module particulier, qui se chargera de gérer l'interface avec l'utilisateur.

Ce module s'appelle un « moteur de templates ». C'est une classe qui a été créée pour gérer uniquement l'interface *homme-machine*. Il en existe plusieurs sur le web. Une des plus connues est la classe SMARTY.

1. La classe SMARTY[1]

L'objectif de cette classe est de séparer complètement le code HTML du code PHP. Pour cela, elle va travailler avec des modèles de pages (les *templates*), dans lesquelles on va rajouter des balises spéciales qui permettront d'afficher les variables qui auront été fournies par l'application.

Ces balises sont encadrées par le signe { }. Dans la pratique, on dessine sa page HTML avec un éditeur quelconque (Kompozer par exemple, issu du projet Mozilla), puis on rajoute les balises SMARTY.

Cet outil est suffisamment puissant pour gérer tous les cas classiques, avec notamment la possibilité de remplir des tableaux(balises <table>, <tr>, <td>...) à partir de tableaux dynamiques PHP. Elle autorise aussi l'intégration de sous-pages dans la même page, ce qui est assez utile pour gérer les affichages de menus communs à toute l'application par exemple.

Voici un exemple de code simple. Il s'agit de l'écran de saisie des utilisateurs :

D'abord, la page php qui va préparer le contenu transmis à Smarty :

```
1  $loginGestion = new LoginGestion($bdd);
2  $list=$loginGestion->lire($id);
3  $smarty->assign("list",$list);
```

On crée d'abord une instance de la classe LoginGestion, qui contient les méthodes permettant de récupérer le contenu de la table.

En ligne 2, on récupère un tableau, dont les attributs contiennent le contenu de l'enregistrement.

En ligne 3, on assigne le tableau $list à la variable Smarty list.

Voici maintenant le template, qui va être traité par Smarty :

```
1  <script language="javascript" SRC="javascript/fonctions.js"></script>
2  <form method="post" action="index.php">
3  <input type="hidden" name="action" value="M">
4  <input type="hidden" name="id" value="{$list.id}">
5                  <input type="hidden" name="module" value="loginmodif">
6                  <input type="hidden" name="password" value="{$list.password}">
7
8  <table class="tablesaisie">
9                  <tr>
10                                 <td>Login :</td>
11                                 <td><input name="login" value="{$list.login}"></td>
12                 </tr>
13                 <tr>
14                                 <td>Nom :</td>
15                                 <td><input name="nom" value="{$list.nom}"></td>
16                 </tr>
17                 <tr>
18                                 <td>Prénom :</td>
19                                 <td><input name="prenom" value="{$list.prenom}"></td>
20                 </tr>
21                 <tr>
22                                 <td>Mél} :</td>
23                                 <td><input name="mail" value="{$list.mail}"></td>
24                 </tr>
25                                 <tr>
26                                 <td>Date de dernière modification :</td>
27                                 <td><input name="datemodif" value="{$list.datemodif}"></td>
28                 </tr>
```

1 http://www.smarty.net/

```
29
30              <tr>
31                              <td>Mot de passe :</td>
32                              <td><input type="password" name="pass1"
    onchange="verifieMdp(this.form.pass1, this.form.pass2)"></td>
33              </tr>
34              <tr>
35                              <td>Répétez le mot de passe:</td>
36                              <td><input type="password" name="pass2"
    onchange="verifieMdp(this.form.pass1, this.form.pass2)"></td>
37              </tr>
38 </table>
39 <div align="center">
40 <input type="submit" name="valid" value="Valider}"/>
41  <input type="submit" name="suppr" value="Supprimer"
    onClick="javascript:setAction(this.form, this.form.action,'S')"/>
42  </div>
43 </form>
```

Toutes les variables Smarty sont entourées des accolades. Comme les variables sont toutes passées à l'intérieur d'un tableau monoligne, on a l'affichage des données qui s'effectue avec l'argument {$tableau.attribut}.

Dans le cas d'un tableau multiligne, on dispose d'outils pour gérer des boucles. Cette page affiche la liste des utilisateurs déclarés dans l'application :

```
1   <a href="index.php?module=loginmodif&id=0">Nouveau login</a>
2   <table>
3               <tr>
4                               <th>Login</td>
5                               <th>Nom - Prénom</td>
6                               <th>addresse mél</td>
7                               <tr>
8                {section name=lst loop=$liste}
9               <tr>
10                              <td><a href="index.php?
    module=loginmodif&id={$liste[lst].id}">{$liste[lst].login}</a></td>
11                              <td>{$liste[lst].nom} {$liste[lst].prenom}</td>
12                              <td>{$liste[lst].mail} </td>
13              </tr>
14              {/section}
15 </table>
```

En ligne 8, nous avons le début de la boucle d'affichage, qui va prendre en compte chacune des occurrences du tableau $liste. La variable lst contient le numéro d'occurrence. Cette boucle se termine en ligne 14.

L'affichage du tableau s'effectue dans les lignes suivantes, $liste[lst].login affiche le contenu de l'attribut login de l'occurrence lst du tableau liste.

A noter : l'utilisation des signes {} interdit l'usage de code JavaScript complexe dans la page HTML. Dans ce cas de figure, on décrit le code JavaScript dans un fichier indépendant (code.js), et on fait ensuite un appel de fonctions.

2. Les feuilles de style

Il est fortement conseillé de travailler avec des feuilles de style CSS, de manière à avoir la même présentation dans l'ensemble de l'application. Cela participe aussi à la vitesse de développement. On peut aussi utiliser d'autres méthodes, mais l'essentiel est d'éviter de coder partout les mises en pages.

L'utilisation de la feuille de style se définit dès le fichier principal, le fameux *config.inc.php*, qui contient tout le code générique à l'ensemble de l'application et

les paramètres spécifiques de chaque implémentation. Voici comment se gère l'instanciation de Smarty, associée à la gestion des feuilles de style :

```
1  $SMARTY_template = $path_inc.'templates';
2  $SMARTY_template_c = $path_inc.'templates_c';
3  $SMARTY_config = $path_inc.'param/configs_smarty';
4  $SMARTY_cache_dir = $path_inc.'plugins/smarty/cache';
5  $SMARTY_cache = FALSE;
6  $APPLI_fds = "CSS/proto.css";
7
8  $smarty = new Smarty;
9  $smarty->template_dir = $SMARTY_template;
10 $smarty->compile_dir = $SMARTY_template_c;
11 $smarty->config_dir = $SMARTY_config;
12 $SMARTY_cache_dir = $SMARTY_cache_dir;
13 $smarty->caching = $SMARTY_cache;
14
15 $smarty->assign("fds",$APPLI_fds);
```

Les lignes 1 à 6 correspondent au paramètres généraux de l'application (en principe, dans un fichier à part, voir le chapitre correspondant).

Les lignes 8 à 13 correspondent à l'instanciation de la classe Smarty.

La ligne 15 permet d'assigner le nom de la feuille de style dans la variable Smarty fds, qui va pouvoir être utilisée dans le template :

Voici le code correspondant dans la page HTML à ces deux lignes :

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
2  <html>
3  <head>
4  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-15">
5  <title>Application</title>
6  <link rel="stylesheet" href="{$fds}" type="text/css">
7  </head>
```

La déclaration de la feuille de style est réalisée en ligne 6.

C. L'organisation en dossiers et sous-dossiers

Une application de taille moyenne est vite confrontée à un nombre impressionnant pages PHP, qui rendent la recherche du code parfois compliquée. Pour simplifier la maintenance (car c'est toujours un objectif prioritaire à ne pas perdre de vue), on a tout intérêt à regrouper les pages par fonction commune, par module.

1. L'organisation des pages en modules

En général, on ne conserve à la racine que la page index.php, la première page de l'application. On peut ensuite créer les dossiers suivants :

- param : va contenir tous les paramètres de l'application
- identification : tout ce qui va toucher à l'identification
- CSS : les feuilles de style
- doc : la documentation
- install : le dossier où on va trouver les informations essentielles pour installer le logiciel, y compris les scripts de création des tables...
- images : les images...

- javascript : les fichiers contenant les fonctions javascript
- locales : les fichiers d'internationalisation
- plugins : un dossier rassemblant l'ensemble des plugins externes utilisés (par exemple, ADOdb)
- templates et templates_c : les fichiers templates utilisés par Smarty

et on pourra ensuite rajouter soit des dossiers par module, soit séparer les fichiers selon leur type. On peut ainsi créer soit des dossiers *gestion, suivi...,* soit des dossiers *classes, affichage, saisie...*

En général, il vaut mieux regrouper tout ce qui traite d'un module au même endroit. C'est d'ailleurs plus conforme avec l'analyse UML, qui isole les fichiers dans des paquetages.

2. Le nommage des pages

Là aussi, les règles, sans être forcément très strictes, doivent être claires pour tous. Comme tous les outils de développement travaillent par classement alphabétique, il faut respecter un classement alphabétique. Une bonne règle est de nommer les pages en respectant ces deux règles :

- on nomme les pages en citant d'abord le module correspondant, puis l'action : LoginAffiche.php, LoginValid.php...
- on garde le même nom pour la page HTML correspondant à la page PHP : LoginAffiche.htm
- les fichiers de classe sont nommés de la même façon, toujours en rangeant les classes par module : commun.class.php, saisiegen.class.php, saisiestat.class.php, consultstat.class.php si des classes spécifiques pour chaque module sont nécessaires.

3. Les pages communes à plusieurs actions

Pour limiter le nombre de pages et éviter de coder plusieurs fois la même chose, il est intéressant de regrouper plusieurs fonctionnalités concernant le même module (consultation – saisie – modification d'une table). En général, on crée une page pour afficher les données, une page pour la saisie des modifications (les formulaires), une page pour la mise en fichier.

L'utilisation judicieuse de variables *action* va permettre de n'utiliser qu'une seule page PHP, page qu'on aura tout intérêt à organiser en créant les fonctions nécessaires. Cette variable action va être transmise en même temps que les variables du formulaire pour déterminer ce qu'on va faire.

Parfois, on utilisera du JavaScript pour coder ces appels, surtout si on préfère utiliser des boutons plutôt que des liens html classiques (<input type="submit">). On utilise cette méthode pour gérer plusieurs boutons, par exemple Valider et Supprimer.

Voici un exemple, concernant la saisie des utilisateurs :

```
1  <form method="post" action="index.php">
2  <input type="hidden" name="action" value="M">
3  <input type="hidden" name="id" value="{$list.id}">
4  ...
5  <div align="center">
6  <input type="submit" name="valid" value="Valider"/>
7   <input type="submit" name="suppr" value="Supprimer"
   onClick="javascript:setAction(this.form, this.form.action,'S')"/>
```

```
8   </div>
9  </form>
```

Le code Javascript correspondant :

```
10 function setAction (nomForm, nomChamp, typeAction) {
11 nomChamp.value=typeAction
12 // Confirmation de la suppression
13    if (typeAction=="S"||typeAction=="suppr") {
14       if (confirm("Confirmez-vous la suppression ?" )!= 1) {
15          nomChamp.value="X"
16       }
17    }
18 nomForm.submit()
19 }
```

En ligne 7, on appelle la fonction javascript setAction, qui va vérifier que l'utilisateur est bien d'accord pour supprimer la fiche (ligne 14). S'il répond ok, la variable action conservera la valeur S, et sinon sera remplacée par la valeur X (ligne 15). Il restera à la page appelée à traiter correctement la variable action pour savoir quoi faire :

```
1   if (isset($_REQUEST["action"])){
2               if ($_REQUEST["action"]=="M") {
3                           //Modification
4                           if (isset($_REQUEST["id"])) {
5                                       $list=array();
6                                       $list=$_REQUEST;
7                                       $ret=$loginGestion->ecrire($list);
8                                       if ($ret>0) {
9                                                   $message= $LANG["message"][5];
10                                                  $_REQUEST["id"]=$ret;
11                                      }else{
12                                                  $message=$LANG["message"][12];
13                                      }
14                          }
15
16              }elseif ($_REQUEST["action"]=="S"&&$_REQUEST["id"]>0) {
17                          //Suppression
18                          if($loginGestion->supprimer($_REQUEST["id"])==1) {
19                                      $message=$LANG["message"][4];
20                                      unset($_REQUEST["id"]);
21                                      $module_coderetour = 1;
22                          }else{
23                                      $message=$LANG["message"][13];
24                          }
25
26              }elseif ($_REQUEST["action"]=="X"){
27                          $module_coderetour = 0;
28              }
29 }
```

En ligne 2, on teste la valeur de la variable $action, qui sera traitée soit pour une mise à jour (lignes 3 à 14), soit pour une suppression (lignes 16 à 24). Si le code retour est X, on ne fait rien.

4. Les appels récursifs

Il n'est pas rare que l'on ait besoin de saisir plusieurs lignes dans un formulaire, par exemple, la liste des produits dans une facture. Plusieurs solutions sont possibles, dont l'utilisation de fenêtres de type Popup. Mais l'approche la plus élégante consiste à utiliser la récursivité : à chaque ajout d'une ligne dans le tableau, on rappelle la page, en lui passant en paramètres l'ensemble des données déjà modifiées.

Voici un exemple, où le code HTML permet de rajouter autant de lignes que nécessaires d'un couple Opérateur/nombre, d'afficher le tableau correspondant, et de supprimer les lignes qui seraient en trop :

```
1       <tr>
2         <td>Op&eacute;rateur :
3         </td>
4         <td>
5         <select name="operateur">
6        <option value="">aucune</option>
7         {section name=lst loop=$listeOperateur}
8    <option value="{$listeOperateur[lst].IDOPERATEUR}" {if $operateur ==
   $listeOperateur[lst].IDOPERATEUR} selected
   {/if}>{$listeOperateur[lst].NOMOPERATEUR}</option>
9    {/section}
10        </select>
11        </td>
12        <td><a href="javascript:recharge('{$phpself}','ajoutoperateur','1')">Ajouter cet
   opérateur</a>
13        </td>
14        <td><a href=# onClick="window.open('{$conf_inc}communs/operateurFormModif.php?
   popup=1','aide','scrollbars=yes,width=600,height=400')">Nouvel opérateur</a>
15        </td>
16      </tr>
17      <tr>
18        <td>Nombre pour ce produit : </td>
19        <td> <input name="nbOperateur" size="5" maxlength="5"></td>
20        <td> </td>
21        <td><br>
22        </td>
23      </tr>
24       {if $operateurajoutes !=""}
25          <tr>
26        <td colspan="2">
27        <table style="text-align: left; width: 100%;" border="0"
28  cellpadding="2" cellspacing="2" class={$tableau}>
29  <tr>
30    <th>Opérateurs du produit</th>
31    <th>Suppression</th>
32   </tr>
33         {section name=lst loop=$operateurajoutes}
34      <tr>
35        <td> <input name="lstidoperateur[]" value="{$operateurajoutes[lst].CLE}"
   type="hidden">
36         <input name="lstnombreoperateur[]" value="{$operateurajoutes[lst].NOMBRE}"
   type="hidden">{$operateurajoutes[lst].NOMBRE}
37                <input name="lstnomoperateur[]" value="{$operateurajoutes[lst].NOM}"
   type="hidden">{$operateurajoutes[lst].NOM}
38        </td>
39        <td>
40            <a href="javascript:recharge('{$phpself}','suppoperateur',
   {$operateurajoutes[lst].CLE})"><img src="../../-.gif" width=15 height=15 border=0>
41           </a>
42        </td>
43      </tr>
44   {/section}
45                                  </table>
46      </td>
47      </tr>
48      {/if}
```

De la ligne 1 à la ligne 13, le programme rajoute un couple Opérateur/nombre, et rappelle la page courante.

La ligne 14 permet de créer un nouvel opérateur dans une fenêtre de type Popup, puis de rafraîchir la page courante.

Enfin, la fin du script permet d'afficher (si nécessaire, cf. ligne 24) la liste des opérateurs déjà saisis, et de les supprimer le cas échéant (ligne 40).

On rappelle systématiquement la même page, qui récupère d'une part une variable action (le code javascript correspondant) et d'autre part toutes les variables du formulaire. On gère alors différemment selon qu'on est en saisie pour la première fois, ou en appel récursif.

Voici le code JavaScript utilisé (et stocké dans un fichier à part, à cause des caractères {} imposés par Smarty) :

```
49 function recharge($adresse,type,val)
50 {
51 document.produit.action = $adresse + "?" + type + "=" + val;
52 document.produit.submit();
53 }
```

C'est par le biais de ce code Javascript qu'on signale qu'on est en mode récursif, et qu'on indique ce que l'on fait. La page PHP va récupérer d'une part toutes les variables du formulaire, d'autre part la variable type, qui lui indiquera ce qu'elle doit faire.

5. Cas particulier de la saisie d'une information complémentaire dans une liste

Dans l'exemple ci-dessus, la ligne 47 permet de gérer la saisie d'un nouvel item dans la table opérateur. Toutefois, après avoir renseigné cet item, il faut forcer le programme à recharger, de manière totalement transparente, la page principale.

On va travailler en deux temps :

- d'une part, on va utiliser la variable *popup* pour savoir si on est ou non en mode popup (saisie indépendante) ;
- d'autre part, on va appeler un page particulière, en validation de l'item créé, pour réafficher la page appelante.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2  <html >
3  <head>
4    <meta content="text/html; charset=ISO-8859-1"
5   http-equiv="content-type">
6    <title>Accueil</title>
7      {$fds2}
8  </head>
9  <body onUnload="window.opener.location.reload()">
10 <h2>{$resultat}</h2>
11 {if $popup!=""}
12
13 <center><a href onClick="window.close()" >Fermer la fenetre</a></center>
14 {/if}
15 </body>
16 </html>
```

C'est la ligne 9 qui va permettre de gérer le réaffichage de la page principale. Il faut donc que celle-ci soit capable de gérer la récursivité (elle se rappelle automatiquement).

Bien évidemment, le mécanisme peut être adapté ! On peut très bien, dans le template SMARTY de saisie de l'item supplémentaire (fenêtre popup), utiliser la variable popup pour afficher ou non le *onLoad* :

```
17 <body {if $popup!=""} onUnload="window.opener.location.reload()"{/if}>
```

6. La variable Path_incl

Lorsque toutes les pages ne sont pas dans le même dossier se pose le problème de l'appel correct aux fichiers répartis ailleurs dans l'arborescence. Ainsi, une page qui serait au niveau 2 (la racine étant au niveau 0) devra accéder quand même au même fichier (par exemple /style/style.css) qu'une page au niveau 1.

Comme les appels ne peuvent être réalisés qu'en relatif, pour un problème de portabilité de l'application sur un autre serveur web que celui utilisé pour le développement, on va créer une variable globale $path_inc, qui contiendra le chemin relatif de la page. Ainsi, une page de niveau 1 aura $path_inc)="../", alors qu'une page de niveau 2 aura $path_inc="../../". On indique ainsi la racine de l'application.

La variable $path_inc sert donc à indiquer le chemin pour revenir à la racine de l'application.

On rajoutera donc, dans la première partie de la page php, la ligne suivante :

```
18 if (!isset($path_inc)) $path_inc="../";
```

Ce test permet de gérer correctement les appels de pages à l'intérieur du programme.

Grâce à cette variable, on gérera facilement l'utilisation de la feuille de style. Voici un exemple du code présent dans le fichier de configuration général de l'application (config.inc.php) :

```
19 $fds = $conf_inc.'templates/fdsDRAF72.css';
20 $fds2='<link rel="stylesheet" href="'.$fds.'" type="text/css">';
```

$fds contient le nom de la feuille de style (on peut très bien imaginer de gérer un profil par utilisateur, qui mettrait à jour cette valeur).

Dans les pages HTML, on n'aura qu'à appeler la variable $fds2 pour que la feuille de style soit directement intégrée :

```
21 <head>
22              {$fds2}
23 </head>
```

Néanmoins, l'utilisation d'une variable de ce type est très dangereux, celle-ci pouvant être modifiée facilement si le paramètre PHP REGISTER_GLOBALS est à ON (ce qui est tout de même de moins en moins fréquent).

Aujourd'hui, cette approche est abandonnée, au profit des méthodes MVC, abordées plus loin dans ce document.

D. La gestion des paramètres

Les paramètres généraux de l'application vont être stockés dans deux fichiers :

- param.default.inc.php, qui va contenir les paramètres par défaut de l'application
- param.inc.php, qui va « écraser » les paramètres par défaut, pour les remplacer par les paramètres propres à l'implémentation.

Ainsi, quand on livrera une nouvelle version de l'application, on ne livrera que le fichier param.default.inc.php, l'autre fichier étant systématiquement conservé.

Ces deux fichiers sont appelés systématiquement l'un après l'autre.

1. Protection du fichier de paramètres

L'accès à ce fichier donne tous les droits sur la base de données... On va donc utiliser trois mécanismes pour sécuriser son accès :

- on va suffixer le fichier avec l'extension .php. Ainsi, le fichier sera systématiquement traité par l'interpréteur php. Un accès direct au fichier ne retournera rien (http://monappli/param.inc.php).
- On va mettre le fichier dans une branche de l'arborescence, qu'on va rendre inaccessible depuis un navigateur. Pour cela, on va créer un dossier appelé param, avec un fichier appelé .htaccess (sous Apache).
  Ce fichier contiendra les lignes suivantes :

```
1 order allow,deny
2 deny from all
```

  ainsi, l'accès à ce dossier est rendu impossible (si Apache le permet)...
- Le dernier point est parfois le plus complexe, selon son architecture. L'idéal est de rendre inacessible le serveur de base de données à l'extérieur. Il faut déjà que le serveur de base de données soit hébergé sur une autre machine (réelle ou virtuelle).
  Si on travaille en réseau étendu (cas sur internet), on peut modifier les paramètres réseau pour inactiver la passerelle (gateway). Si on dispose d'un proxy global, on n'autorise pas l'accès au serveur de base de données.
  Une fois le serveur inaccessible, même en possession du mot de passe de connexion à la base de données, il faudrait arriver à modifier les requêtes pour obtenir des informations sur le système ou falsifier les informations ; on verra plus tard comment se prémunir de « l'injection de code ».

A. L'identification dans l'application

Un des problèmes majeurs dans PHP est de gérer correctement les droits d'accès aux pages. Le fait que PHP fonctionne en mode asynchrone (on peut appeler une page indépendamment de la précédente), il est hors de question de pouvoir demander une identification à chaque page. On est donc obligé de travailler avec les variables de session pour conserver l'authentification de l'utilisateur, et pouvoir vérifier ainsi ses droits.

Deux problèmes sont à régler : l'authentification de l'utilisateur et la conservation de cette authentification, et la gestion des droits d'accès pour chaque page.

1. L'authentification de l'utilisateur

Plusieurs possibilités sont envisageables. La plus simple consiste à stocker le login et le mot de passe (sous forme cryptée, en MD5) dans la base de données. D'autres solutions existent :

- utiliser un annuaire LDAP, qui servira à vérifier le couple *login/mdp*. C'est intéressant s'il existe un annuaire d'entreprise, ou si l'on utilise les fonctions LDAP de *Active Directory*, de Microsoft.
- Utiliser une application globale de gestion des habilitations, qu'on peut alors transformer en services web (avec la classe *NUSOAP*). L'avantage est d'avoir un seul point de gestion des droits des utilisateurs.

Quel que soit le mécanisme choisi, il est indispensable que le module de saisie du mot de passe soit protégé par un cryptage, en général en SSL. De même, le

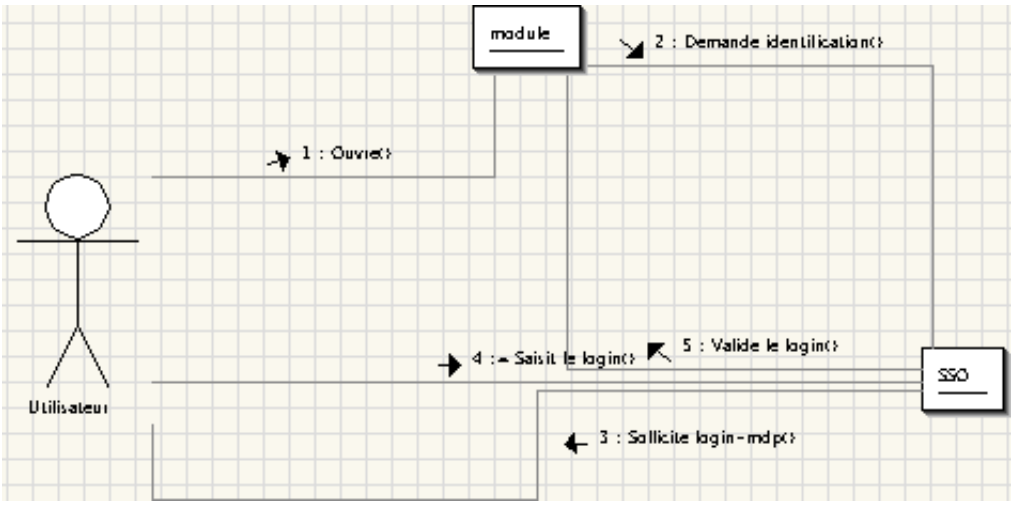
serveur web ne doit pas conserver en mémoire le mot de passe. On ne stocke que le fait que l'utilisateur soit authentifié, et les droits qui lui sont rattachés.

2. Le principe des SSO-CAS

L'objectif du SSO est de regrouper l'identification au même endroit, pour que l'utilisateur n'ait pas à mémoriser 36 mots de passes, et à éviter, le cas échéant, de l'obliger à se réidentifier s'il l'est déjà.

Les SSO (Single Sign On), associés au CAS (Central Authentification Service) permettent d'identifier de façon unique les personnes qui se connectent, quel que soit l'application. On utilise en général la notion de ticket :

- l'application demande au CAS d'identifier un utilisateur
- le CAS demande à l'utilisateur de saisir son login, mot de passe et, s'il est valide, ouvre un jeton ou ticket
- le CAS retourne à l'application demandeuse le login de l'utilisateur (et jamais son mot de passe) : il reste à récupérer les droits de l'utilisateur localement.
- si une nouvelle application, dans la même session, demande l'identification de l'utilisateur au CAS, et si le ticket reste valide, l'utilisateur n'a pas à retaper son mot de passe.



L'intérêt, c'est que l'application appelante ne connaît jamais le mot de passe, et les règles du CAS sont complètement indépendantes du développement en cours. C'est particulièrement pertinent dans le cadre de l'utilisation d'un annuaire d'entreprise, avec mise en oeuvre d'une réelle politique de sécurité.

Les appels aux CAS sont en principe normalisés. Voici un exemple d'utilisation d'un CAS dans un projet PHP. On utilise la bibliothèque PHP-ESUPCAS (ou PHP-CAS)[1] :

```
1  class Identification {
2              var $ident_type = NULL;
3              var $CAS_address ;
4              var $CAS_port;
5              var $CAS_uri;
6              var $password;
7              var $login;
8              var $gacl;
9              var $aco;
```

1 http://www.ja-sig.org/wiki/display/CASC/phpCAS

```
10              var $aro;
11              var $pagelogin;
12
13               * @param $ident_type string
14              function setidenttype($ident_type){
15                              $this->ident_type = $ident_type;
16              }
17               * initialisation si utilisation d'un CAS
18              function init_CAS($cas_address,$CAS_port, $CAS_uri) {
19                              $this->CAS_address = $cas_address;
20                              $this->CAS_port = $CAS_port;
21                              $this->CAS_uri = $CAS_uri;
22                              $this->ident_type = "CAS";
23
24              }
25 * Retourne le login en mode CAS ou BDD
26              function getLogin() {
27                              $ident_type = $this->ident_type;
28                              if (!isset($ident_type)){
29                                              echo "Cette fonction doit être appelée après soit
   init_LDAP, init_CAS, ou init_BDD";
30                                              die;
31                              }
32                              if (!isset($_SESSION["login"])) {
33                                              /*
34                                               * Récupération du login selon le type
35                                               */
36                                              if ($ident_type == "BDD") {
37
38                                              }elseif ($ident_type == "CAS") {
39
40
   phpCAS::client(CAS_VERSION_2_0,$this->CAS_address,$this->CAS_port,$this-
   >CAS_uri);
41                                                                              //
                                if (phpCAS::isAuthenticated()==FALSE) {
42                                                                              phpCAS::forceAuthentication();
43                                                                              //
                                }
44                                                                              $_SESSION["login"] =
   phpCAS::getUser();
45                                              }
46                              }
47
48                              if (isset( $_SESSION["login"])) {
49                                              $this->login = $_SESSION["login"];
50                                              return $_SESSION["login"];
51                              }else{
52                                              return -1;
53                              }
54              }
55               * Déconnexion de l'application
56              function disconnect($adresse_retour) {
57                              if (!isset($this->ident_type)) {
58                                              return 0;
59                              }
60                              if ($this->ident_type == "CAS") {
61                                              phpCAS::client(CAS_VERSION_2_0,$this-
   >CAS_address,$this->CAS_port,$this->CAS_uri);
62                                              phpCAS::logout($adresse_retour);
63                              }
64                              // Détruit toutes les variables de session
65                              $_SESSION = array();
66
```

```
67                              // Si vous voulez détruire complètement la session, effacez
    également
68                              // le cookie de session.
69                              // Note : cela détruira la session et pas seulement les données de
    session !
70                              if (isset($_COOKIE[session_name()])) {
71                                      setcookie(session_name(), '', time()-42000, '/');
72                              }
73
74                              // Finalement, on détruit la session.
75                              session_destroy();
76                              return 1;
77
78              }
79 }
```

L'initialisation de la classe phpCAS se fait en ligne 40. En ligne 44, on récupère le login de l'utilisateur.

3. Gestion de la déconnexion

A partir de la ligne 55, on a un aspect de la déconnexion de l'application, qui va :

- en ligne 61, instancier la classe phpCAS
- en ligne 62, demander au CAS d'effectuer l'opération de déconnexion
- les lignes 65, 70 et 75 font un nettoyage complet de la session, ce qui est toujours souhaitable sur le plan de la sécurité.

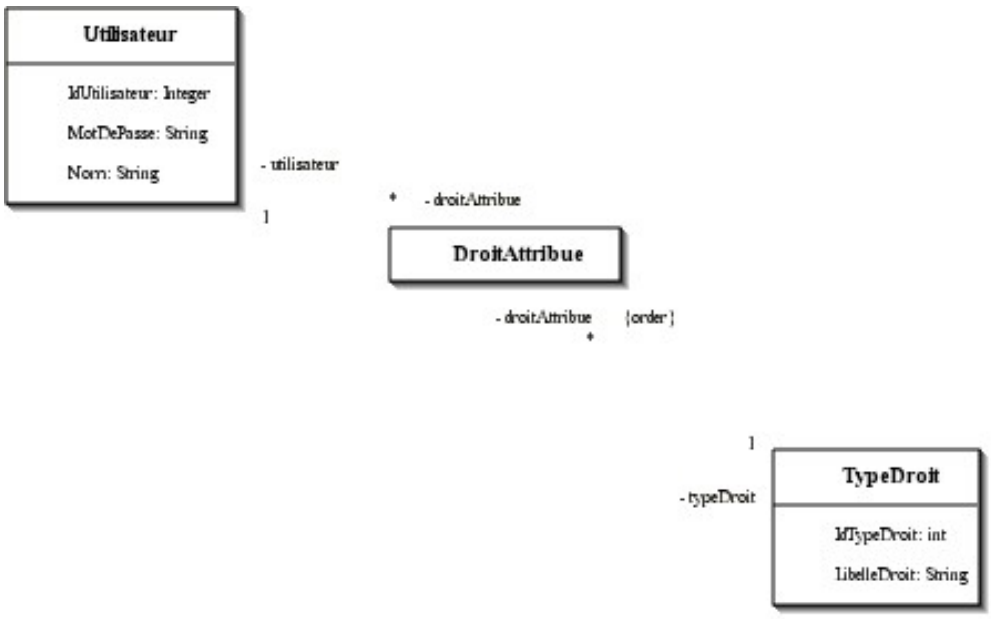
4. La gestion des droits d'accès

En général, on distingue plusieurs types de droits :

- le droit de visualisation des informations, qui peut varier selon le profil de l'utilisateur
- le droit de modification des informations, qui varie lui aussi selon le profil
- le droit d'administration, qui est indépendant des droits d'utilisation de l'application

Une des approches est de gérer les droits de façon pseudo-hiérarchique : on définit des niveaux de droits pour chaque fonctionnalité (visualisation, modification, administration). Ainsi, un utilisateur peut avoir un droit de visualisation de 3, de modification de 1, d'administration de 0. Le droit de visualisation 3 emporte les droits de niveau inférieur. Cela permet d'avoir plusieurs hiérarchies en parallèle.

L'inconvénient de la méthode impose une certaine discipline, pour savoir à quoi correspondent chacun des droits, et pour donner effectivement les droits nécessaires aux utilisateurs.



Avec ce type de schéma, tout utilisateur peut se voir affecter les différents types de droits dont il a besoin. Le modèle peut bien sûr être étendu à d'autres aspects.

Côté PHP, on va créer une classe qui va gérer cette authentification.

Voici un extrait du code de cette classe, pour la partie qui va vérifier si on peut accéder ou non à une page :

```
1  //fonction qui accède à la page si on est authentifié ou amène a la page d'authentification
2     function accederPage($page/*:string*/,$type_droit/*:string*/,$niveau_droit/*string*/) {
3                 $this->URLredir =  $this->host.".$page;
4                 $this->type_droit = $type_droit;
5                 $this->niveau_droit = $niveau_droit;
6                 $controle=$this->effectueControle();
7                 if($controle!==true){
8                                 @header("Location: $controle");
9                                 exit();
10                }
11  }
12
13 //fonction qui controle si l'utilisateur est authentifié
14  function effectueControle()/*:string*/ {
15                @session_start();
16                if(isset($_SESSION['s_droits']))
17                {
18                                $retour= false;
19                                $t_idutil=$_SESSION['s_idutilisateur'];
20                                $t_droit=$_SESSION['s_droits'];
21                                $_SESSION['s_idutilisateur']=$t_idutil;
22                                $_SESSION['s_droits']=$t_droit;
23                                $soapclient = new soapclient($this->pathSW."droitGetNum.php");
24                                $parametres=array('nomdroit' => $this->type_droit);
25                                $numdroit = $soapclient->call("getNumDroit",$parametres);
26                                foreach ($t_droit as $droit) {
27                                                if((strtolower($droit['NOMDROIT'])==strtolower($this-
    >type_droit) && $droit['NIVEAUDROIT']>=$this->niveau_droit)
28                                                                ||
    ($droit['NUMERODROIT']>$numdroit))
29                                                {
30                                                                $retour=true;
31                                                }
```

```
32                              }
33                              if($retour!=true) {
34                                      $mess="Droits insufisants.";
35                                      $retour=$this->URLAccueil."?messErreur=".$mess;
36                              }
37              } else
38              {
39                              $retour=$this->URLIdentification."?redir=".$this->URLredir;
40              }
41                              return $retour;
42  }
```

La fonction *AccederPage* ne sert qu'à gérer l'appel et le retour. Tout se passe dans la fonction *effectueControle*, qui va :

- récupérer un tableau contenant l'ensemble des droits de l'utilisateur pour le droit considéré à partir d'un service web (lignes 24 à 26)
- comparer ces droits au droit nécessaire pour accéder à la page (ligne 27)
- la ligne 28 permet de gérer le cas où un type de droit englobe un autre type de droit. Ainsi, on pourrait vouloir spécifier qu'un administrateur a tous les droits de modification. Dans ce cas, il suffit que l'identifiant de TypeDroit pour Administrateur soit supérieur à l'identifiant de TypeDroit pour Modification. Si on ne souhaite pas cette option, il faut que les identifiants soient de même niveau.
- Les lignes 34 et 35 permettent de spécifier ce qu'il se passe si les droits sont insuffisants
- La ligne 39 gère le cas où l'utilisateur n'est pas encore identifié (variable de session *$s_droits* inexistante. Dans ce cas, il est redirigé vers la page d'authentification.

La page PHP appelée, et sous réserve que l'instance *$identification* ait été créée dans le fichier *common.inc.*php, ne contient plus que la ligne suivante, qui va s'assurer que les droits sont suffisants :

```
43 $identification->accederPage($_SERVER['PHP_SELF'],"Modification","1");
```

Cet exemple permet d'apréhender comment on peut implémenter une authentification et une gestion des droits facilement dans toute application, en gérant différents niveaux de droits et en redirigeant le cas échéant l'utilisateur vers la page d'authentification, si sa session est expirée ou s'il accède à une page directement sans passer par la page d'accueil.

Le code de vérification des droits dans chaque page est dès lors réduit de façon très importante.

5. L'utilisation de PHPGACL[1]

PHPGACL est un projet open source qui vise à fournir une méthode unique d'identification dans les applications. Elle présente l'avantage d'être parfaitement modulaire, et gère tout type de droits. Basé sur une classe objet et une base de données, il gère deux types d'objets :

- les ACO (Access Control Objects), les objets auxquels on donne les droits
- les ARO, (Access Request Objects), les personnes qui souhaitent accéder aux objets.

Basé sur une hiérarchie, tant au niveau des ACO que des ARO, on peut définir de façon très fine les droits, d'autant qu'on peut poser aussi bien des droits positifs (accès autorisés) que des droits négatifs (accès interdit).

1 http://phpgacl.sourceforge.net/

Ainsi, on peut autoriser tel groupe de personnes à accéder à tel module, sauf à telle fonction, qui est réservée à un autre groupe !

Voici un extrait de la doc, avec un exemple basé sur StarWar :

```
Millennium Falcon Passengers
  -Crew       [ALLOW: ALL]
|   -Han
|   -Chewie          [DENY: Engines]
  -Passengers[ALLOW: Lounge]
    -Jedi            [ALLOW: Co kpit]
|     -Obi-wan
|     -Luke          [ALLOW: Guns]
    -R2D2            [ALLOW: Engines]
    -C3PO
```

L'équipage a accès à tout, sauf Chewie qui n'a pas accès aux moteurs.

Les passagers n'ont accès qu'au salon, sauf les Jedi qui peuvent accéder au coockpit, et parmi eux, Luke peut accéder à l'armement. R2D2 peut accéder aux moteurs.

Cet exemple reste simple, mais, dans la pratique, on utilise des sections pour faciliter le regroupement des ACO, ce qui permet de disposer d'un classement complémentaire. Exemple de sections : system, user pour les ARO (les utilisateurs).

Voici un exemple d'implémentation, trouvé dans la doc, et visant à tester si l'utilisateur a le droit de se connecter :

```
44 // include basic ACL api
45 include('phpgacl/gacl.class.php');
46 $gacl=newgacl();
47  $username = $db->quote($_POST['username']);
48   $password = $db->quote(md5($_POST['password']));
49   $sql = 'SELECT name FROM users WHERE name =
50   $sql .= $username .' AND password =' . $ password;
51   $row = $db->GetRow ($sql );
52   if($gacl->acl_check('system','login','user',$row['name' ] ) ) {
53         $_SESSION['username'] = $row['name'] ;
54           return true ;
55    }
56      else
57           return  false ;
```

On teste ici que $row['name'], qui doit être dans l'ARO *user*, dispose bien des droits pour l'ACO *login*, dans la section *system*.

Une troisième dimension a été rajoutée, pour pouvoir gérer de façon plus fine les droits. Si on souhaite, par exemple, définir des droits de lecture, écriture... sur des modules, on va utiliser les AXO. Dans ce cas, on va retrouver l'organisation suivante :

- les ACO correspondent aux actions que l'on veut faire (lire, écrire, supprimer, ajouter...)

- les AXO correspondent aux objets auxquels on veut accéder (les modules : gestion des paramètres, saisie d'un dossier, paiement...)
- les ARO sont toujours le réceptacle des utilisateurs.

On pourra ainsi interroger la base des droits pour savoir si l'utilisateur Jean a le droit de modifier dans le module administration.

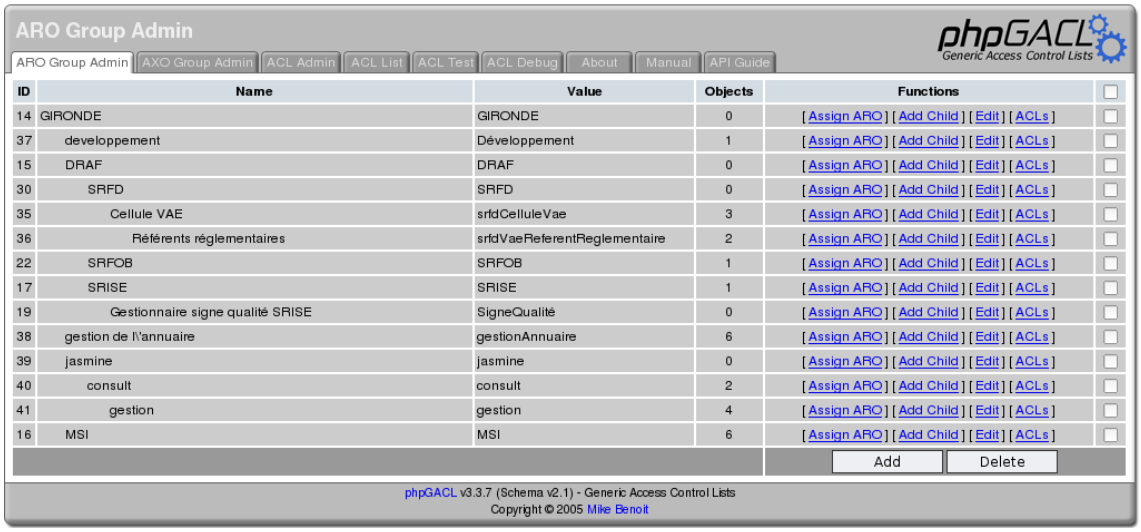
Une interface de gestion des droits est fournie en standard, qui permet la saisie de l'ensemble des informations nécessaires.

PHPGACL est bâti avec les standards modernes de développement : ADODB et SMARTY. Il est possible de paramétrer PHPGACL pour qu'il utilise les bibliothèques déjà présentes dans l'application.

Un exemple d'utilisation simple

On définit, dans l'interface phpgacl, dans l'onglet ACL Admin, dans la partie *Access Request Object* (ARO), une nouvelle section : login. Dans cette section, on rajoute les logins des personnes à qui l'on souhaite donner des droits.

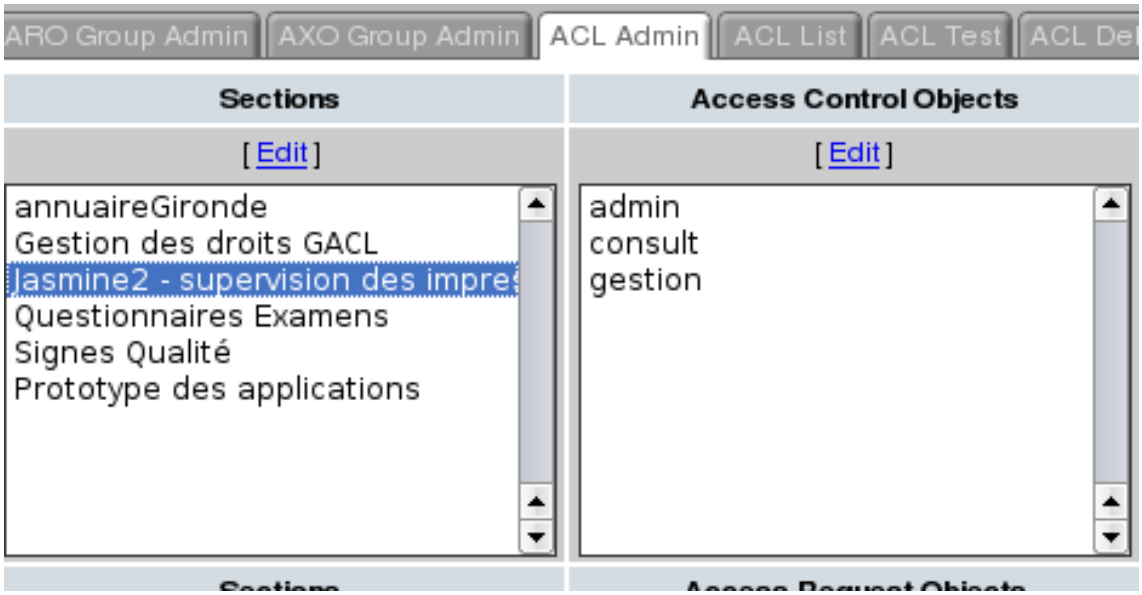
Dans l'onglet ARO Group Admin, on va créer d'une part l'arborescence des groupes des utilisateurs, sachant que les groupes les plus à droite disposeront des droits attribués aux groupes précédents :



Dans cet exemple, les référents réglementaires disposeront des droits attribués aux personnes de la Cellule VAE, qui eux-mêmes disposeront des droits du SRFD...

Les personnes sont affectées aux groupes adéquats, sachant qu'une même personne peut être affectée à plusieurs groupes en même temps (lien *Assign ARO*).

Dans l'onglet ACL Admin, on dans la partie haute (Access Control Object), on va créer les différents droits nécessaires :



Ainsi, pour l'application Jasmine2, trois droits différents ont été définis : admin, consult et gestion. Pour chacun, on définira les groupes qui récupéreront ces droits, dans la zone *Groupe*, en bas à droite de la fenêtre.

Une fois les droits définis dans l'interface, l'utilisation sera aisée dans le programme :

```
58 $resident = $phpgacl->acl_check("jasmine2","admin","login","jean.dupont");
```

$resident ne sera pas nul si le login jean.dupont dispose des droits admin pour l'application jasmine2...

Si on donne les droits *admin* du programme *Jasmine2* au groupe *Référents réglementaires*, tout membre du groupe Référents réglementaires disposera des droits considérés. Cette approche, très souple, permet une gestion toute en finesse des droits dans une application.

Gérer de façon globale les droits

Dans une structure, il est rare que l'on ne développe qu'une application. Lors de l'arrivée ou du départ des personnels, il faut aller dans chacune pour pouvoir attribuer ou supprimer les droits nécessaires.

Pour éviter cette gestion un peu lourde, on a tout intérêt à gérer de façon globale les droits, dans une application indépendante. On créera ainsi une base PHPGACL indépendante, avec l'application de gestion intégrée, qu'on pourra éventuellement adapter si on a besoin d'intégrer un SSO ou une identification sur un annuaire LDAP particulier.

Quoi qu'il en soit, il est indispensable de documenter parfaitement la gestion des droits, tant dans l'application de gestion des droits (PHPGACL) que dans la doc technique d'implémentation et d'administration de l'application cliente.

B. L'accès aux données

1. La classe ADOdb[1]

Les API de connexion aux bases de données ne sont pas standardisées en PHP (ce qui serait, du reste, quasiment impossible compte-tenu de la non standardisation des bases de données). C'est dans ce contexte qu'est né le projet ADODB, qui vise à fournir une classe unique de connexion et d'exécution des ordres SQL vers les bases de données.

ADODB supporte la plupart des bases de données, tant sous Windows que sous Linux, y compris les accès par ODBC.

Voici un exemple de code standard, issu de la documentation :

```
59 <?php include('adodb/adodb.inc.php');
60  $db = ADONewConnection($dbdriver); # eg 'mysql' or 'postgres'
61  $db->debug = true;
62  $db->Connect($server, $user, $password, $database);
63  $rs = $db->Execute('select * from some_small_table');
64  print "<pre>";
65  print_r($rs->GetRows());
66  print "</pre>";
67 ?>
```

2. La classe ObjetBDD[2]

Plusieurs développeurs ont voulu « encapsuler » ADODB, pour faciliter le codage des applications. bien qu'ADODB apporte un plus évident, en apportant une couche d'abstraction, les développeurs doivent encore coder toutes les requêtes manuellement, que ce soit pour les select d'affichage ou les commandes de mises à jour (insert, update, delete). Ces codages sont fastidieux, ils n'apportent rien et surtout, de nombreux outils de développement les encapsulent (PowerBuilder, Access, WinDev...).

L'idée la plus fréquente est de créer une classe, non instanciable (mais héritable), qui permettra d'encapsuler les accès les plus fréquents aux données, pour éviter de perdre du temps en codage fastidieux. Un des outils les plus connus est intégré dans PEAR, mais qui nécessite de décrire, dans un fichier texte, la structure des tables (un utilitaire est livré pour générer automatiquement la structure des tables depuis une base de données).

L'exemple présenté ici est un codage basé sur une interrogation directe de la structure des tables, depuis l'application. Il est en effet possible, en SQL, de récupérer la liste des champs ; une fonction ADODB (FetchField) permet cette opération. Cette fonction permet en outre de récupérer la longueur maximale, et le type du champ, ce qui peut être utilisé pour reformater les dates, par exemple, ou gérer correctement les requêtes d'insertion ou de modification sur les champs de type texte (mise en place des guillemets). Toutefois, la récupération des types de champs n'est pas forcément disponible pour toutes les bases de données ; on est donc obligé d'utiliser une « signalisation » interne, en indiquant les champs qui sont de format date ou numérique.

1 http://adodb.sourceforge.net/
2 http://objetbdd.sourceforge.net/

La classe ObjetBDD a été développée par des vacataires et des stagiaires ; elle est disponible, ainsi que sa documentation d'utilisation, sur sourceforge.net.

Voici un exemple d'utilisation :

Structure de la table

Table personnel :

```
68 id : numeric, cle
69 nom : varchar
70 prenom : varchar
71 dateNaissance : date
72 dateModif : datetime
73 nombreHeureTravaillee : int
```

Classe Personnel

fichier dbclass.phpclass

```
74  Personnel extends ObjetBDD {
75          function Personnel($link) {
76          ObjetBDD::ObjetBDD($link);
77          $this->table = "personnel";
78          $this->cle = "id";
79          $this->id_auto = 1; // Cle automatique gérée par le SGBD
80          $this->types = array (
81          "id"=>1,
82          "dateNaissance"=>2,
83         "dateModif"=>3,
84          "nombreHeureTravaillee"=>1);
85         }
86 }
```

Instanciation de la classe

```
87 include_once('adodb.inc.php');
88 include_once('ObjetBDD.php');
89 include_once('dbclass.php');
90 $BDD_type = "mysql";
91 $BDD_server = "localhost";
92 $BDD_login = "login";
93 $BDD_passwd = "password";
94 $BDD_database = "db";
95 if (!isset($bdd)) {
96          $bdd = ADONewConnection($BDD_type);
97          $etatconn=$bdd->Connect($BDD_server, $BDD_login, $BDD_passwd,
    $BDD_database);
98          if ($etatconn == FALSE) {
99          echo( "Echec de la connexion à la base de données" );
100          die ();
101          }
102}
103$personnel = new Personnel($bdd);
```

Utilisation de la classe

```
104$tab=array();
105$tab=$personnel->lire($id); // Retourne l'enregistrement $id dans $tab :
106/*
107* $tab["id"] = 1, $tab["nom"] = "Dupont", $tab["prenom"] = "Jean",
108 * $tab["dateNaissance"] = "01/01/1970", $tab["dateModif"]="01/09/2008 14:35:26",
109 * $tab["nombreHeureTravaillee"]=180
110*/
111$tab=$personnel->getListe(); // Retourne l'ensemble de la table dans $tab :
112/*
113* $tab[0]["id"] = 1, $tab[0]["nom"] = "Dupont", $tab[0]["prenom"] = "Jean",
114 * $tab[0]["dateNaissance"] = "01/01/1970", $tab[0]["dateModif"]="01/09/2008 14:35:26",
115 * $tab[0]["nombreHeureTravaillee"]=180,
```

```
116* $tab[1]["id"] = 2, $tab[0]["nom"] = "Durant", $tab[0]["prenom"] = "Pierre", (...)
117*/
118$sql = "select id, nom, prenom, dateNaissance from personnel order by nom, prenom";
119$tab = $personnel->getListeParam($sql); // Retourne le tableau correspondant à la
120// requête exécutée, avec conversion de la date
121$ret = $personnel->ecrire($tab); // Ecrit le tableau $tab en base de données
122$ret = $personnel->supprimer($id); //Supprime l'enregistrement $id
```

C. Les contrôles de saisie et la sécurité

Le codage HTML est une vraie « passoire » en terme de sécurité. Si PHP5 permet, dans le cadre de certaines implémentations, de limiter les risques d'injection de code, les applications en PHP4 peuvent laisser passer n'importe quoi.

Le phénomène le plus connu est l'injection de code (mais il existe aussi le débordement de tampon dû à des chaînes trop longues et écrites de façon habile, un classique dans les attaques).

Soit une requête classique d'interrogation, après saisie d'un formulaire de recherche :

select nom from agent where numagent = ?numagent

Il suffit qu'une requête malformée remplace numagent par la chaîne « numagent » (et non plus le numéro de l'agent) pour récupérer l'ensemble des noms des agents présents dans la base ! Une simple vérification sur le type de la donnée, sa longueur... suffit à éviter ce problème.

En général, il est fortement conseillé de :

- ne pas utiliser les appels paramétrés à des pages (include $nompage)
- vérifier les valeurs rentrées, en longueur, et en type
- vérifier que les valeurs qui arrivent sur le site web sont bien du type de celles qu'on attendait (_POST, _GET...)
- dès lors que l'on manipule des informations nécessitant une identification et une non divulgation, il faut passer en mode crypté (https). C'est impératif pour toute demande de mot de passe !

D. L'internationalisation de l'application

1. Pourquoi faire ?

Outre la possibilité de gérer plusieurs langues dans l'application, pour répondre aux besoins évidents de partage ou de travail à plusieurs, il est souvent souhaitable de laisser la possibilité d'adapter certains libellés. C'est souvent le cas pour les informations concernant le login.

2. Les fichiers de langue

En général, les fichiers de langue sont stockés dans un dossier *locales*. Leur gestion va s'appuyer sur deux mécanismes. D'une part, un code permet de savoir quelle langue utilisée. Ce code peut être statique (défini dans un fichier de paramètres) : c'est le cas le plus courant. Il peut également être dynamique, en prenant en compte la langue du navigateur. La troisième possibilité, c'est de stocker dans le profil de l'utilisateur (cookie, base de données...) la langue qu'il souhaite utiliser.

La gestion des libellés est plus classique. On distingue deux approches principales (mais il en existe d'autres), l'une consistant à créer un tableau

contenant tous les libellés, l'autre utilisant un fichier xml. Dans tous les cas de figure, il est fortement conseillé de regrouper les libellés par module, pour faciliter la maintenance. Voici un exemple d'un fichier de langue :

```
123fichier locales/fr_FR.php
124<?php
125$LANG= array();
126$LANG["menu"][0] = "Gestion";
127$LANG["menu"][1] = "Opérations de gestion";
128$LANG["menu"][2] = "Liste des comptes";
129$LANG["menu"][3] = "Liste des logins - identification via la base de données";
130$LANG["menu"][4] = "Administration";
131$LANG["menu"][5] = "Administration de l'application";
132$LANG["menu"][6] = "Déconnexion";
133$LANG["menu"][7] = "Déconnexion de l'application";
134$LANG["menu"][8] = "A propos";
135$LANG["menu"][9] = "A propos de prototypePHP";
136$LANG["menu"][10] = "Aide";
137$LANG["menu"][11] = "Quelques conseils...";
138$LANG["menu"][12] = "Gestion des droits";
139$LANG["menu"][13] = "Gestion des droits d'accès aux modules de l'application";
140$LANG["menu"][20] = "Aide";
141$LANG["menu"][21] = "Utiliser PrototypePHP";
142$LANG["menu"][22] = "Installation";
143$LANG["menu"][23] = "Installer PrototypePHP";
144$LANG["menu"][24] = "Gestion des droits";
145$LANG["menu"][25] = "Gérer les droits dans l'application";
146$LANG["menu"][26] = "Module de lecture-écriture";
147$LANG["menu"][27] = "Créer un module pour lire ou écrire un enregistrement";
148$LANG["menu"][28] = "Paramétrage du module";
149$LANG["menu"][29] = "Configurer le module dans le fichier xml";
150$LANG["menu"][30] = "Classe d'accès aux données";
151$LANG["menu"][31] = "Créer la classe permettant d'accéder à une table";
152$LANG["menu"][32] = "Gestion de la documentation";
153$LANG["menu"][33] = "Gérer les pages de la documentation";
154$LANG["menu"][34] = "Langues";
155$LANG["menu"][35] = "Gestion des différentes langues, internationalisation de l'application";
156$LANG["menu"][36] = "SMARTY";
157$LANG["menu"][37] = "Manipuler les templates SMARTY";
158$LANG["menu"][38] = "Structure";
159$LANG["menu"][39] = "Organisation de PrototypePHP";
160
161
162$LANG["message"][0] = "Bienvenue";
163$LANG["message"][1] = "prototypephp - titre de l'application";
164$LANG["message"][2] = "Module administration";
165$LANG["message"][3] = "Module gestion";
166$LANG["message"][4] = "Suppression effectuée";
167$LANG["message"][5] = "Enregistrement effectué";
168$LANG["message"][6] = "Vidage effectué";
169$LANG["message"][7] = "Vous êtes maintenant déconnecté";
170$LANG["message"][8] = "Vous n'êtes pas connecté";
171$LANG["message"][9] = "Vous n'avez pas le droit d'accéder à cette fonction";
172$LANG["message"][10] = "Identification réussie !";
173$LANG["message"][11] = "Identification refusée";
174$LANG["message"][12] = "Problème lors de la mise en fichier...";
175$LANG["message"][13] = "Problème lors de la suppression";
176$LANG["message"][14] = "Vous ne pouvez pas accéder directement à cette page";
177$LANG["message"][15] = "oui";
178$LANG["message"][16] = "non";
179$LANG["message"][17] = "Attention :";
180$LANG["message"][18] = "Confirmez-vous l'opération ?";
181$LANG["message"][19] = "Valider";
182$LANG["message"][20] = "Supprimer";
```

```
183$LANG["message"][21] = "Rechercher";
184$LANG["message"][22] = "Echec de connexion à la base de données";
185
186$LANG["login"][0] = "Login";
187$LANG["login"][1] = "Mot de passe";
188$LANG["login"][2] = "Veuillez utiliser votre login du domaine pour vous identifier";
189$LANG["login"][5] = "Nouveau login";
190$LANG["login"][6] = "login";
191$LANG["login"][7] = "Nom - prénom";
192$LANG["login"][8] = "Mél";
193$LANG["login"][9] = "Nom";
194$LANG["login"][10] = "Prénom";
195$LANG["login"][11] = "Date";
196$LANG["login"][12] = "Répétez le mot de passe";
197
198?>
```

Il suffit ensuite d'utiliser ce fichier dans l'application, en chargeant d'une part le fichier contenant le tableau des libellés, puis en l'utilisant dans l'application.

```
199$language = "fr_FR";
200include_once('locales/'.$language.".php");
```

3. L'intégration des libellés avec SMARTY

Pour utiliser ce fichier de langue dans Smarty, on va d'abord l'assigner :

```
201$smarty->assign("LANG",$LANG);
```

puis, dans les templates smarty :

```
1  <form method="POST" action="index.php">
2              <table class="tablesaisie">
3              <tr>
4              <input type="hidden" name="module" value={$module}>
5              <td>{$LANG.login.0} :</td><td> <input name="login" maxlength="32"></td>
6              </tr>
7              <tr><td>
8              {$LANG.login.1} :</td><td><input name="password" type="password"
   maxlength="32"></td>
9              </tr>
10             <tr>
11             <td><input type="submit"></td><td> <input type="reset"></td>
12             </tr>
13             </table>
```

En ligne 5, on va retrouver l'affichage du libellé « login », et en ligne _, le libellé « mot de passe », sur cet écran de saisie du login/mot de passe.

4. Les cas particuliers

Cette gestion s'applique bien à des libellés courts, ceux d'un formulaire ou les entêtes de tableaux. Par contre, c'est beaucoup moins adapté à une documentation ou à des pages contenant beaucoup de texte.

On va alors utiliser un autre mécanisme, toujours basé sur notre variable $language.

On veut créer une documentation, qui pourra être disponible dans plusieurs langues.

Dans le dossier *doc*, on crée un dossier par fichier de langue, par exemple fr_FR, ou en_US...

la page index.php, qui va permettre d'accéder à cette documentation, va simplement inclure le chemin correspondant au dossier de langue. Voici un exemple d'intégration dans smarty :

```
1  <?php
2  $handle = @fopen("doc/".$language."/".$t_module["param1"], "r");
3  $doc = "";
4  if ($handle) {
5      while (!feof($handle)) {
6          $buffer = fgets($handle, 4096);
7          $doc .= $buffer;
8      }
9      fclose($handle);
10     $smarty->assign("doc",$doc);
11     $smarty->assign("corps","documentation/index.html");
12 }
13 ?>
```

En ligne 2, $t_module[« param1 »] contient le nom de la page html à afficher. On charge ainsi la page html doc/fr_FR/essai.html (par exemple).

Les lignes 4 à 9 permettent de lire le contenu du fichier. La ligne 10 assigne le contenu de la documentation à la variable $doc de smarty, et la ligne 11 indique à l'application qu'elle doit charger le template documentation/index.html. Ce template va contenir au minimum la ligne suivante :

```
14 {$doc}
```

qui affichera le contenu du fichier html précédemment chargé.

5. Les pièges à éviter...

Il faut penser son application dès le départ : si c'est plus fastidieux lors de la création des écrans de saisie, ce n'en est que plus facile en terme de maintenance.

Il faut également penser au javascript, qui doit, lui aussi, être internationalisé (selon le même principe que pour la gestion de la documentation), faute de quoi on aura des boites de dialogue qui risquent d'être assez peu compréhensibles...

E. La documentation des classes et des fonctions

Le plus souvent, on n'imprime pas la documentation technique de l'application (les classes et leurs méthodes, les paramètres...). Par contre, dès lors que l'on cherche une information, il est utile de pouvoir s'y retrouver rapidement.

Dans cette optique, Sun Microsystems a développé un outil permettant de générer une documentation d'API en format HTML depuis les commentaires présents dans un code source en Java. Javadoc est le standard industriel pour la documentation des classes Java. La plupart des IDEs génèrent automatiquement le javadoc HTML.

1. Les tags Javadoc[1]

Les développeurs utilisent certains styles de commentaire et des tags Javadoc quand ils documentent un code source. Un bloc de commentaire java commençant par `/**` deviendra un bloc de commentaire Javadoc qui sera inclus dans la documentation du code source. Un tag Javadoc commence par un `@`. Quelques tags sont donnés dans le tableau suivant :

Tag	Description
@author	Nom du développeur
@deprecated	Marque la méthode comme dépréciée. Certains IDEs génèrent un avertissement à la compilation si la méthode est appelée.

1 Source : http://fr.wikipedia.org/wiki/Javadoc

@exception	Documente une exception lancée par une méthode — voir aussi @throws.
@param	Définit un paramètre de méthode. Requis pour chaque paramètre.
@return	Documente la valeur de retour. Ce tag ne devrait pas être employé pour des constructeurs ou des méthodes définis avec un type de retour void.
@see	Documente une association à une autre méthode ou classe.
@since	Précise à quelle version de la SDK/JDK une méthode à été ajoutée à la classe.
@throws	Documente une exception lancée par une méthode. Un synonyme pour @exception disponible depuis Javadoc 1.2.
@version	Donne la version d'une classe ou d'une méthode.

Exemple

Un exemple d'utilisation de Javadoc pour documenter une méthode :

```
1  /**
2   * Valide un mouvement de jeu d'Echecs.
3   * @param leDepuisFile   File de la pièce à déplacer
4   * @param leDepuisRangée Rangée de la pièce à déplacer
5   * @param leVersFile     File de la case de destination
6   * @param leVersRangée   Rangée de la case de destination
7   * @return vrai(true) si le mouvement d'échec est valide ou faux(false) si invalide
8   */
9   boolean estUnDéplacementValide(int leDepuisFile, int leDepuisRangée, int leVersFile, int
    leVersRangée)
10  {
11      ...
12  }
```

Un exemple d'utilisation de Javadoc pour documenter une classe :

```
1  /**
2   * Classe de gestion d'étudiant
3   * @author John Doe
4   * @version 2.6
5   */
6   public class Etudiant
7   {
8     ...
9   }
```

2. En PHP...

Le PHP a conservé ces règles de rédaction de la documentation, et utilise les mêmes TAG de base.

Par défaut, Eclipse prend en charge la gestion de la documentation des classes et méthodes ; il suffit de taper /** avant une classe ou une méthode pour que les paramètres de base soient pré-rédigés.

Il existe de nombreux outils qui permettent la génération de la documentation PHP, dont les plus connus sont phpdocumentor[1] et doxygen[2] (dont un plugin eclipse existe[3]).. Ce dernier reprend les tags Javadoc, tandis que phpdocumentor propose plus de tags :

1 http://www.phpdoc.org/
2 http://www.stack.nl/~dimitri/doxygen/
3 Pour installer le plugin : http://www.irisa.fr/bunraku/OpenMASK/EclipseDev/ch02s06.html

3. Les tags de PHPDOCUMENTOR[1]

Tag	Usage	Description
@abstract		Documente une classe abstraite, une variable de classe ou une méthode
@access	public, private or protected	Indique le niveau d''accès d'un élément.
@author	author name <author@email>	Documente l'auteur de l'élément courant.
@copyright	name date	Documente les informations de copyright
@deprecated	version	Indique si une méthode est dépréciée.
@deprec		Identique à @deprecated
@example	/path/to/example	Indique où est stocké le fichier d'exemple
@exception		Documente une exception lancée par une méthode. Voir également @throws.
@global	type $globalvarname	Documente une variable globale, qu'elle soit utilisée dans une fonction ou dans une méthode.
@ignore		Evite de documenter l'élément correspondant
@internal		Informations privées pour les développeurs avancés
@link	URL	
@name	global variable name	Spécifie un alias pour une variable. Par exemple, $GLOBALS['myvariable'] devient $myvariable
@magic		
@package	name of a package	Documente un groupe de classes et fonctions reliées
@param	type [$varname] description	
@return	type description	Ce tag ne doit pas être utilisé pour les constructeurs ou les méthodes définies avec un type void pour la valeur de retour (pas de valeur de retour).
@see		Documente une association vers une autre méthode ou une autre classe.
@since	version	Indique quand une méthode a été rajoutée à la classe.
@static		Documente une classe ou une méthode statique.
@staticvar		Documents a static variable's use in a function or class Documente une variable statique utilisée dans une fonction ou une classe.
@subpackage		
@throws		Documente une exception lancée par une méthode.
@todo		Indique ce qui doit être codé à une date ultérieure.
@var	type	Un type de données pour une variable de classe.
@version		Indique le numéro de version d'une classe ou d'une méthode.

Tout développeur doit impérativement documenter toutes les classes, méthodes et fonctions qu'il rédige, et décrire également, selon le même mécanisme, toute page php générée.

Les balises à utiliser, à minima, sont les suivantes : @author, @param, @return, et @copyright pour les fichiers php.

Cette documentation ne se substitue pas à la documentation d'implémentation cf. *La rédaction de la documentation*, page 7), ou au manuel utilisateur... Elle peut également être complétée par une documentation générale décrivant le fonctionnement global de l'application.

1 http://en.wikipedia.org/wiki/PHPDoc – traduction de l'auteur

V. Le modèle MVC

Le modèle MVC (Modèle, Vue, Contrôleur), est très utilisé dans le monde Java, avec le *framework* Struts. Son principe est simple : on sépare l'interface utilisateur (la vue), du code intrinsèque de l'application (le modèle), le tout coordonné par le contrôleur (le code gérant les enchaînements d'actions).

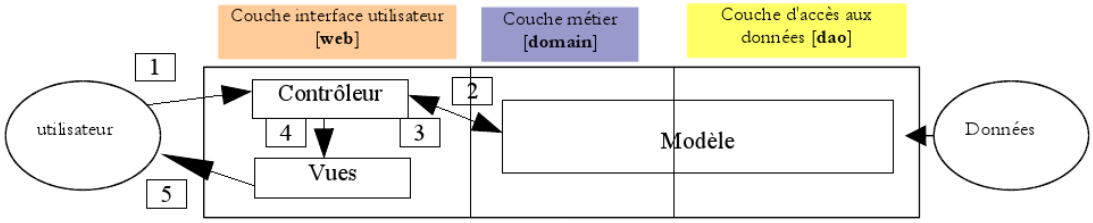


Illustration 1: Le modèle vue-contrôleur. Source : http://tahe.developpez.com/web/php/mvc/?page=introduction

A. Implémenter le modèle MVC dans un projet

L'approche MVC change les habitudes de programmation : fini les appels directs à des pages ! Il faut impérativement passer par le contrôleur, qui va vérifier les droits et réorienter vers le module adapté.

Pour faire simple, on distingue les trois composants :

- la vue, l'interface utilisateur, est générée en Smarty qui, rappelons le, permet de séparer de façon importante le code de l'application de l'affichage ;
- le modèle, ou couche métier, est géré par deux types d'objets (au sens non UML) : les classes, basées par exemple sur ObjetBDD pour tout ce qui concerne l'accès aux bases de données, et les pages de traitement, qui contiennent la logique métier.
- le contrôleur va s'occuper de l'enchaînement des pages, vérifier les droits, gérer le login.

Les deux premiers composants sont déjà connus : la vue, ce sont les *templates* Smarty. Le modèle, se sont les classes et les modules contenant le code de l'application. Par contre, le contrôleur est un nouveau composant, dont le rôle se limite à gérer le fonctionnement général de l'application ; il est générique, et peut être réutilisé pour d'autres applications. Comment l'implémenter ?

B. Le contrôleur

Le contrôleur est constitué d'une page qui va être appelée systématiquement, soit par le biais d'un *include*, soit être le point d'entrée unique de l'application (page *main.php* ou *index.php*, unique lien référencé dans toutes les pages). Cette solution est en général préférable : elle limite l'accès à l'application à un seul point d'entrée.

Cette page va réaliser les opérations suivantes :

- mise en place des paramètres
- identification de l'utilisateur
- vérification, en fonction de l'action demandée, que l'utilisateur dispose des droits nécessaires

- vérification le cas échéant des données entrées (analyse des variables transmises, pour vérifier leur cohérence)
- exécution des modules de traitement
- enfin, génération des pages HTML en retour (appel des pages php qui vont générer les modèles Smarty)

Si on ne manipule qu'une page unique en entrée, cela permet d'éviter le recours à la variable path_inc (tout passe à partir d'une page de base). Par contre, il faut définir une méthode d'enchaînement et d'appel des pages de traitement. Il faut, au minimum, pour chaque action, définir les données suivantes :

- le code du module à appeler (saisie du dossier, p. e.)
- l'action à effectuer dans ce module
- les droits associés à cette action
- la page d'affichage à appeler si l'action se termine correctement (action de sortie)
- le cas échéant, la page à appeler en cas d'échec de l'action.

Ces données peuvent être décrites soit dans un fichier xml (cas de certains *frameworks*, comme *cmvphp)*, soit dans un fichier texte d'un format permettant une lecture rapide par php (fichiers de types paramètres). Il vaut mieux éviter de coder directement les actions dans la page *main.php*, ce qui alourdirait sérieusement le code...

1. Exemple de codage d'une page contrôleur

Le fichier XML

```
1  <navigation>
2          <model action="index.php" droits="gestion" retourko="model"
3                  retourok="model" retournull="model" droitko="model"
   loginrequis="1">
4          </model>
```

Chaque module va faire l'objet d'une déclaration dans le fichier XML, qui va contenir un certain nombre d'attributs :

- action : page php à exécuter
- droits : niveau de droit nécessaire
- retourko : module à appeler si le code de retour vaut -1
- retourok : module à appeler si le code de retour vaut 1
- retournull : module à appeler si le code de retour vaut 0
- loginrequis : est positionné à 1 si le module nécessite une identification préalable, sans pour autant qu'un niveau de droits soit nécessaire.

La classe de manipulation du fichier XML

```
1  <?php
2  /** Fichier cree le 10 juil. 08 par quinton
3   *
4   *UTF-8
5   */
6
7  /**
8   * @class Navigation
9   * @author Eric Quinton
10  *
11  */
```

```
12 class Navigation {
13 		var $doc;
14 		var $module;
15 		var $nommodule = "";
16 		var $t_module = array();
17
18
19 		/**
20 		 * Fonction de creation de la classe
21 		 * lecture du fichier xml contenant la navigation de l'application
22 		 *
23 		 * @param string $nomfichier
24 		 * @return Navigation
25 		 */
26 		function Navigation($nomfichier) {
27 				$this->doc = new DOMDocument();
28 				$this->doc->load($nomfichier);
29 		}
30 		/**
31 		 * Retourne un tableau contenant tous les attributs du module
32 		 *
33 		 * @param string $module
34 		 * @return array
35 		 */
36 		function getModule($module) {
37 				$this->lireModule($module);
38 				return $this->t_module;
39 		}
40 /**
41  * Fonction lisant les attributs du module, et les restituant
42  * dans le tableau t_module
43  * Operation effectuee que si le module n'a pas encore ete charge
44  *
45  * @param string $nommodule
46  */
47 		function lireModule($nommodule) {
48 				if ($this->nommodule<>$nommodule) {
49 						$this->module = $this->doc-
   >getElementsByTagName($nommodule);
50 						$this->t_module = array();
51 						foreach ($this->module as $noeud){
52 								if ($noeud->hasAttributes()) {
53 										foreach ($noeud-
   >attributes as $attname=>$noeud_attribute) {
54 												$this-
   >t_module[$attname] = $noeud->getAttribute($attname);
55 										}
56 								}
57 						}
58 						$this->nommodule = $nommodule;
59 				}
60 		}
61 		/**
62 		 * Fonction retournant la valeur d'un attribut pour le module considere
63 		 *
64 		 * @param string $nommodule
65 		 * @param string $attribut
66 		 * @return string
67 		 */
68
69 		function getAttribut($nommodule, $attribut){
70 				$this->lireModule($nommodule);
71 				return $this->t_module[$attribut];
72 		}
```

```
73 }
74
75 ?>
```

Cette classe va permettre de lire le fichier XML, et de stocker les attributs du module considéré dans un tableau, plus facile à manipuler en PHP. On retiendra deux fonctions :

- getModule("nommodule") : retourne un tableau contenant tous les attributs du module
- getAttribut("nommodule","nomattribut") : retourne la valeur de l'attribut pour le module considéré.

La page contrôleur

```
1.  <?php
2.  /** Fichier cree le 9 mai 07 par quinton
3.   *
4.   *UTF-8
5.   */
6.  include_once("common.inc.php");
7.  /*
8.   * Recuperation du module
9.   */
10. unset($module);
11. if (isset($_REQUEST["module"])) {
12.         $module = $_REQUEST["module"];
13. }
14. /*
15.  * Gestion des modules
16.  */
17. while (isset($module)){
18.         //Recuperation du tableau contenant les attributs du module
19.         $t_module = array();
20.         $t_module=$navigation->getModule($module);
21.
22.         //Verification si le login est requis
23.         if (strlen($t_module["droits"])>1||$t_module["loginrequis"]==1) {
24.
25.                         // Vérification du login
26.                         if (!isset($_SESSION["login"])){
27.                                 /*
28.                                 * Traitement suivant le type d'identification
29.                                 */
30.                                 if ($ident_type == "CAS") {
31.                                         $identification->getLogin();
32.                                 } else {
33.                                         // On verifie si on est en retour de
    validation du login
34.                                         if (isset($_REQUEST["login"])) {
35.                                                 if ($ident_type == "BDD")
    {
36.
    $loginGestion = new LoginGestion($bdd);
37.
    $res=$loginGestion->VerifLogin($_REQUEST['login'],$_REQUEST['password']);
38.                                                                 if
    ($res==TRUE){
39.
            $_SESSION["login"] = $_REQUEST["login"];
40.                                                                 }
41.                                                         }else{
42.                                                                 //
    $ident_type = LDAP
```

```
43.
    $identification->testLoginLdap($_REQUEST["login"],$_REQUEST['password']);
44.                                                                 }
45.                                                         }else{
46.                                                                 // Gestion de la saisie du
    login
47.                                                                 $smarty-
    >assign("corps","ident/login.tpl");
48.                                                                 $smarty-
    >assign("module",$_REQUEST["module"]);
49.                                                                 $message =
    $IDENT_message;
50.                                                         }
51.                                         }
52.                         }
53.         }
54.         $resident = 1;
55.         if ($t_module["loginrequis"]==1&&!isset($_SESSION["login"])) $resident = 0;
56.         //Verification des droits
57.         if (strlen($t_module["droits"])>1) {
58.                         if (!isset($_SESSION["login"])) {
59.                                         $resident = 0;
60.                         }else{
61.                                         $resident = $phpgacl->acl_check($GACL_aco,
    $t_module["droits"],$GACL_aro,$_SESSION["login"]);
62.                         }
63.         }
64.
65.         //fin d'analyse du module
66.         unset($module);
67.         unset ($module_coderetour);
68.
69.         //Execution du module
70.         if ($resident){
71.                         include $t_module["action"];
72.                         //Recuperation du code de retour et affectation du nom du nouveau
    module
73.                         if (isset($module_coderetour)) {
74.                                         switch($module_coderetour){
75.                                                         case -1:
76.
    $module=$t_module["retourko"];
77.                                                                 break;
78.                                                         case 0:
79.
    $module=$t_module["retournull"];
80.                                                                 break;
81.                                                         case 1:
82.
    $module=$t_module["retourok"];
83.                                                                 break;
84.                                         }
85.                         }
86.         }else{
87.                         // On traite le cas échéant le module en cas de droits insuffisants
88.                         if (strlen($t_module["droitko"])>1) {
89.                                         $module = $t_module["droitko"];
90.                         }
91.         }
92. }
93.
94. //if (isset($_SESSION["message"])) $message=$_SESSION["message"];
95.
96. if ($message == "") $message = "bienvenue";
```

```
97. $smarty->assign("message",$message);
98. $smarty->display($SMARTY_principal);
99. ?>
```

Explications...

- ligne 6 : inclusion des paramètres (connexion à la base de données, classes génériques...)
- lignes 10 à 13 : récupération du nom du module, à partir d'une variable passée par le navigateur
- Lignes 19 et 20 : récupération des attributs du module, depuis le fichier XML
- Lignes 23 à 55 : vérification si le login est saisi, et sinon, appel des modules adéquats (accès vers un SSO-CAS, ou saisie via une page de saisie du login). Le login est considéré comme ok si la variable de session $_SESSION["login"] est renseignée. Pour gérer les différents types d'identification possibles (CAS, LDAP..., une classe particulière est utilisée : la classe Identification, décrite en annexe.
- Lignes 56 à 63 : vérification des droits, en utilisant la classe phpgacl.
- Ligne 71 : exécution du module via un include
- ligne 73 à 92 : on récupère le cas échéant le code de sortie, pour enchaîner sur un autre module, puis on boucle pour exécuter le module indiqué
- La fin du code permet de gérer les assignations dans SMARTY, puis d'afficher la page principale.

C. La génération automatique du menu

1. Principe général

La norme CSS2 a apporté de sérieuses avancées en terme de gestion des styles dans les pages web. L'une des plus intéressantes concerne la gestion des menus. Il est maintenant possible de définir rapidement un menu, en gérant des listes avec les balises <ul> et <li> ; ces balises sont utilisées initialement en HTML pour gérer des listes non ordonnées, <ul> encadrant la liste, <li> chaque item.

Voici le menu utilisé sur la page de description du projet PrototypePHP[1] :

```
1  <div class="menu">
2  <ul>
3  <li>
4  <a href="index.html" title="Retour à la page d'accueil">Accueil</a>
5  </li>
6  <li>
7  <a href="Installation.html" title="Installation de PrototypePHP">Installation</a></li>
8  <li><a href="http://sourceforge.net/projects/prototypephp/">Page
9  projet</a>
10 </li>
11 <li>
12 <a href="http://sourceforge.net/project/showfiles.php?group_id=196435">Télécharger
13 le logiciel</a>
14 </li>
15 <li> <a href="phpdocumentor/index.html" title="documentation générée par
   phpdocumentor">Description
16 des classes</a>
17 </li>
18 </ul>
19 </div>
```

1 http://prototypephp.sourceforge.net/

Le menu commence par une balise , qui va permettre de déclarer la classe *menu* de la feuille CSS.

On initialise la liste avec la balise <ul>, puis chaque item du menu est encapsulé dans une balise <li>.

On va rajouter dans notre feuille CSS les styles qui vont permettre de le mettre en forme :

```
1   div.menu {
2     border: #999999 1px solid;
3     background: #ccffcc;
4     font-size: 12px;
5     padding:0.5em;
6     margin-left: 15%;
7     margin-right: 15%;
8   }
9   div.menu h2 {
10    display: none;
11 }
12 div.menu:after {
13    content: "";
14    clear: both;
15    display: block;
16 }
17 div.menu > ul {
18    list-style-type: none;
19    padding: 0;
20    cursor: pointer;
21    margin: 0 0 0 1em;
22 }
23 div.menu > ul > li {
24    margin-top: -2px;
25    float: left;
26 }
27 div.menu > ul span {
28    margin: 0;
29    padding: 0 1em 2px 1em;
30
31 }
32 div.menu > ul li:hover span {
33    background: #FFFFFF;
34      color: #ccffcc;
35
36 }
37 div.menu > ul li ul {
38    border: 1px solid;
39    border-color: #666;
40    margin: -1px 0 0 2px;
41    padding: 1px;
42    width: 10em;
43    height: auto;
44    background: #FFFFFF;
45      color: #ffffff;
46    list-style-type: none;
47    position: absolute;
48    display: none;
49 }
50 div.menu > ul li:hover ul {
51    display: block;
52 }
53 div.menu > ul li ul li {
54    margin: 0;
55    padding-left: 1px;
```

```
56   background: #ccffcc;
57   border-left: solid 1em #CCC;
58     color: #ffffff;
59 }
60 div.menu a:link, div.menu a:visited, div.menu a:active {
61   color: #000000;
62   display: block;
63   margin: 0;
64   padding: 2px 1em 2px 1em;
65 }
66 div.menu a:hover {
67   background: #FFFFFF;
68     color: #666666;
69 }
```

Dans la feuille de style, on décrit chaque niveau de l'arborescence du menu. En ligne 17, on décrit le comportement général du menu de premier niveau (<ul>). Chaque item (balise <li>) est déclaré avec une marge négative de 2 points (ligne 24). En ligne 32, on change la couleur de l'item quand la souris passe par dessus, et ainsi de suite... Le menu intègre la gestion des sous-menus, qui s'afficheront verticalement sous chaque menu de premier niveau. On pourrait également compléter cette feuille avec des menus de troisième niveau, en conservant le même mécanisme.

Pour faire apparaître le menu secondaire vertical, il suffit d'intégrer des balises <ul> et <li> à l'intérieur de chaque balise <li>. Voici un exemple issu d'une application de consultation des impressions sur un serveur CUPS :

```
1  <div class="menu">
2  <ul>
3  <li id="critereliste"><a href="index.php?module=critereliste" title="Consultations par groupe
   ou par imprimante">Consultations globales</a></li>
4  <li id="listeindividuel"><a href="index.php?module=listeindividuel" title="Consultations
   personnelles">Consultations individuelles</a></li>
5  <li id="administration"><a href="index.php?module=administration" title="Opérations
   d'administration">Paramétrage</a>
6  <ul>
7  <li id="ldapinsertform"><a href="index.php?module=ldapinsertform" title="Intégration des
   groupes et des logins depuis l'annuaire LDAP">Import LDAP</a></li>
8  <li id="printerlist"><a href="index.php?module=printerlist" title="Affichage de la liste des
   imprimantes">Saisie des imprimantes</a></li>
9  <li id="queuelistenonaffecte"><a href="index.php?module=queuelistenonaffecte" title="Liste
   des files d'impression non affectées">Files non affectées</a></li>
10 <li id="purgeform"><a href="index.php?module=purgeform" title="Suppression des éditions
   anciennes">Purge</a></li>
11 <li id="gestiondroits"><a href="index.php?module=gestiondroits" title="Gestion des droits
   d'accès aux modules de l'application">Gestion des droits</a></li>
12 </ul>
13 </li>
14 <li id="identdisconnect"><a href="index.php?module=identdisconnect" title="Déconnexion
   de l'application">Déconnexion</a></li>
15 <li id="apropos"><a href="index.php?module=apropos" title="A propos de JASMINE2">A
   propos</a>
16 <ul>
17 <li id="aide"><a href="index.php?module=aide" title="Quelques conseils...">Aide</a></li>
18 </ul>
19 </li>
20 </ul>
21 </div>
```

En ligne 6, on décrit le sous-menu du menu *paramétrage*. De même, on dispose d'un sous-menu pour le menu *à propos*, en ligne 16, qui ne contient qu'un item.

2. L'intégration à notre framework

Pour générer automatiquement notre menu, nous allons compléter notre fichier xml, en rajoutant pour chaque module les informations nécessaires. L'exemple prend en compte un menu à deux niveaux, mais la méthode peut être étendue (presque) l'infini.

Les ajouts dans le fichiers XML

On rajoute les valeurs suivantes :

- menulevel="0" : permet de définir le niveau du menu (0 : niveau principal, 1, menu secondaire)
- menuorder="0" : définit la position de l'item dans le menu. Ce numéro doit être unique dans le niveau de menu considéré, il sert à la fois de clé (pour y rattacher des sous-menus), et d'ordre de tri
- menudroits="gestion" : définit les droits nécessaires pour accéder à ce menu
- menuloginrequis="1" : si aucun droit n'est indiqué, le menu n'est affiché que si l'utilisateur s'est identifié au préalable
- menuparent="0" : pour les sous-menus, indique le numéro de rattachement à l'item du menu précédent

Avec ces informations, on dispose de tout ce qui est nécessaire pour générer dynamiquement notre menu.

L'algorithme d'extraction des informations nécessaires pour gérer le menu

```
1               /**
2                * Fonction lisant l'arborescence sur 2 niveaux du fichier xml
3                * Lecture depuis la racine du fichier xml, des noeuds de niveau 1
4                * et des attributs associes
5                *
6                * @param string $racine
7                * @return array
8                */
9               function lireGlobal() {
10                              $root = $this->dom->getElementsByTagName($this->dom->documentElement->tagName);
11                              $root = $root->item(0);
12                              $noeuds = $root->childNodes;
13                              $g_module = array();
14                              foreach ($noeuds as $noeud){
15                                              // Exclusion du modele
16
17                                              if ($noeud->hasAttributes()&&$noeud->tagName<>"model") {
18                                                              foreach ($noeud->attributes as $attname=>$noeud_attribute) {
19                                                                              $g_module[$noeud->tagName][$attname] = $noeud->getAttribute($attname);
20                                                              }
21                                              }
22                              }
23                              return $g_module;
24              }
25
26 /**
27  * Fonction preparant un tableau multi-niveaux, contenant tous les items
28  * necessaires pour generer un menu a partir du fichier xml.
```

```
29  * Gere 2 niveaux de menu
30  * Le tableau recupere doit ensuite etre trie (via ksort), et les droits
31  * verifies (menuloginrequis et menudroits)
32  *
33  * @return array
34  */
35              function genererMenu() {
36                              $gmenu = array();
37                              foreach ($this->g_module as $key => $value ) {
38                                              if (isset($value["menulevel"])) {
39                                                              // Recuperation des informations sur le
    menu
40                                                              $menu = array();
41                                                              // print_r($value);
42                                                              $menu["menuvalue"] =
    $value["menuvalue"];
43                                                              $menu["module"] = $key;
44                                                              if (isset($value["menudroits"]))
45                                                              $menu["menudroits"] =
    $value["menudroits"];
46                                                              if (isset($value["menuloginrequis"]))
47                                                              $menu["menuloginrequis"] =
    $value["menuloginrequis"];
48                                                              $menu["menutitle"] =
    $value["menutitle"];
49                                                              // Recherche si on est en menu
    principal ou secondaire
50                                                              if ($value["menulevel"]==0) {
51                                                                              foreach ($menu as
    $key1 => $value1) {
52
    $gmenu[$value["menuorder"]][$key1]=$value1;
53                                                                              }
54                                                              } else {
55
    $gmenu[$value["menuparent"]][$value["menuorder"]]=$menu;
56                                                              }
57                                              }
58                              }
59                              return $gmenu;
60              }
```

La première fonction, de la ligne 9 à la ligne 24, va transformer le fichier xml en tableau à deux niveaux. La seconde fonction, à partir de la ligne 35, va récupérer les informations concernant les menus dans le premier tableau généré, et les organiser pour décrire les menus et les sous-menus. Il ne restera alors qu'à générer le menu lui-même.

L'algorithme de génération de la liste des menus

```
1  <?php
2  /**
3   * Preparation automatique du menu a partir du fichier xml
4   */
5  $menuarray = $navigation->genererMenu();
6  //print_r($menuarray);
7  // Tri du tableau selon les cles
8  ksort($menuarray);
9  $menu = "<ul>";
10 foreach ($menuarray as $key => $value){
11              $ok = true;
12              // Verification des droits
13              if ($value["menuloginrequis"]==1 && !isset($_SESSION["login"])) $ok = false;
14              if (strlen($value["menudroits"])>1&& !$phpgacl->acl_check($GACL_aco,
    $value["menudroits"],$GACL_aro,$_SESSION["login"]))
```

```
15              $ok = false;
16
17              // Preparation menu niveau 0
18              if ($ok) {
19                              $menu .= '<li id="'.$value["module"].'"><a href="index.php?
    module='.$value["module"].'" title="'
20                              .$LANG["menu"][$value["menutitle"]].'">'.$LANG["menu"]
    [$value["menuvalue"]].'</a>';
21                              $flag=0;
22                              // Gestion sous-menu
23                              ksort($value);
24                              foreach($value as $key1 => $value1){
25                                              if (is_array($value1)) {
26                                                              $ok1 = true;
27                                                              // Verification des droits
28                                                              if ($value1["menuloginrequis"]==1 && !
    isset($_SESSION["login"])) $ok1 = false;
29                                                              if (strlen($value1["menudroits"])>1&& !
    $phpgacl->acl_check($GACL_aco,$value1["menudroits"],$GACL_aro,$_SESSION["login"]))
30                                                              $ok1 = false;
31                                                              if ($ok1) {
32                                                                              // Preparation du sous-
    menu
33                                                                              if ($flag==0) {
34                                                                                              $menu
    .="<ul>";
35                                                                                              $flag = 1;
36                                                                              }
37                                                                              $menu .= '<li id="'.
    $value1["module"].'"><a href="index.php?module='.$value1["module"].'" title="'
38                                                                              .$LANG["menu"]
    [$value1["menutitle"]].'">'.$LANG["menu"][$value1["menuvalue"]].'</a></li>';
39                                                              }

40                                              }
41                              }
42                              if ($flag == 1) $menu .="</ul>";
43                              $menu .="</li>";
44              }
45 }
46 $menu .="</ul>";
47 ?>
```

Pour chaque niveau de menu, on trie le tableau (lignes 8 et 23), puis on vérifie les droits (lignes 26 à 30 et 37 à 39). A noter que ce script inclut l'internationalisation des libellés (variables $LANG).

Il ne restera plus alors qu'à afficher la variable $menu, qui contient le code HTML d'affichage du menu...

L'intégration dans la page HTML

Dans la page générale de l'application, on insère maintenant les lignes qui vont permettre de générer le menu, puis d'assigner la variable $menu à SMARTY :

```
1  /**
2   * Gestion du menu
3   */
4  include ("menu/menu.inc.php");
5  $smarty->assign("menu",$menu);
```

Et, dans la page SMARTY :

```
1  <div class="menu">{$menu}</div>
```

D. Les avantages et les inconvénients de la méthode

Utiliser un framework apporte beaucoup d'avantages : tous les mécanismes utilisés par le framework ne sont plus à coder. On le voit ici, où le menu est généré automatiquement, les droits bien implémentés partout... On est également obligé de s'astreindre à une certaine rigueur, ce qui n'est pas forcément une mauvaise chose.

Une fois que le framework est maîtrisé, il est très rapide d'y intégrer un nouveau module, il suffit en général de faire un déclaration dans un fichier quelconque pour qu'il soit pris en charge. Si le framework est un projet actif, on peut également récupérer toutes les évolutions assez facilement, soit pour intégrer de nouvelles fonctionnalités, soit qui corrigent les bugs des versions précédentes.

Par contre, il faut prévoir un certain temps d'apprentissage pour le maîtriser, et on est « bloqué » si celui-ci ne répond pas aux besoins. De même, si on doit utiliser des mécanismes particuliers, par exemple pour gérer les droits selon une logique particulière, cela peut devenir très compliqué.

Par contre, un framework, qui intègre par défaut un certain nombre de mécanismes, peut vite devenir un « gouffre » en terme de ressources. Entre décrire un menu « à la main » dans une page HTML, et utiliser un script de génération automatique, le temps de traitement côté serveur ne va pas être le même ! Pour de petites applications, cela peut très bien ne pas poser de problème, mais si les pages concernées sont consultées par plusieurs milliers d'utilisateurs en même temps, le gain de développement peut facilement être perdu par la puissance supplémentaire nécessaire...

En conclusion : les frameworks sont très utiles pour standardiser et sécuriser une application, mais doivent être correctement choisis en fonction de ce que l'on veut faire, et doivent toujours être étudiés sur le plan de la performance.

E. Un exemple de framework : PrototypePHP[1]

La plupart des exemples sont tirés du framework PrototypePHP, qui intègre les projets PHP suivants :

- SMARTY
- ADODB
- OBJETBDD
- PHPGACL
- PHPCAS

et un fichier central XML de déclaration des modules.

1 http://prototypephp.sourceforge.net/

VI. Quelques liens...

SSO et CAS : http://www.esup-portail.org/consortium/espace/SSO_1B/cas/jres/cas-jres2003-article-web.htm

VII. Annexes

A. Classe Identification

```
2  <?php
3  /** Fichier cree le 4 mai 07 par quinton
4   *
5   *UTF-8
6   *
7   * Classe maÃ®trisant les aspects identification.
8   */
9
10 /**
11  * @class Identification
12  * Gestion de l'identification - rÃ©cupÃ©ration du login en fonction du type d'accÃ¨s
13  * @author Eric Quinton - eric.quinton@free.fr
14  *
15  */
16
17 class Identification {
18              var $ident_type = NULL;
19              var $CAS_address ;
20              var $CAS_port;
21              var $CAS_uri;
22              var $LDAP_address;
23              var $LDAP_port;
24              var $LDAP_rdn;
25              var $LDAP_basedn;
26              var $LDAP_user_attrib;
27              var $LDAP_v3;
28              var $LDAP_tls;
29              var $password;
30              var $login;
31              var $gacl;
32              var $aco;
33              var $aro;
34              var $pagelogin;
35
36              function setpageloginBDD($page) {
37                              $this->pagelogin = $page;
38              }
39              /**
40               * @param $ident_type string
41               */
42              function setidenttype($ident_type){
43                              $this->ident_type = $ident_type;
44              }
45              /**
46               * initialisation si utilisation d'un CAS
47               * @param $cas_address adresse du CAS
48               * @param $CAS_port port du CAS
49               * @return  none
50               */
51              function init_CAS($cas_address,$CAS_port, $CAS_uri) {
52                              $this->CAS_address = $cas_address;
53                              $this->CAS_port = $CAS_port;
54                              $this->CAS_uri = $CAS_uri;
55                              $this->ident_type = "CAS";
56
57              }
58              /**
59               * initialisation si utilisation d'un LDAP
60               * @param $LDAP_address adresse du CAS
61               * @param $LDAP_port port du serveur LDAP
62               * @param $LDAP_rdn chemin complet de recherche, incluant le login
```

```
63              * @param $login login qui sera retournÃ© Ã  l'application
64              * @param $password mot de passe Ã  tester
65              * @return  none
66              */
67
68             function init_LDAP($LDAP_address, $LDAP_port,$LDAP_basedn,
    $LDAP_user_attrib,$LDAP_v3,$LDAP_tls) {
69                             $this->LDAP_address = $LDAP_address;
70                             $this->LDAP_port = $LDAP_port;
71                             $this->LDAP_basedn = $LDAP_basedn;
72                             $this->LDAP_user_attrib = $LDAP_user_attrib;
73                             $this->LDAP_v3 = $LDAP_v3;
74                             $this->LDAP_tls = $LDAP_tls;
75                             $this->ident_type = "LDAP";
76             }
77
78             /**
79              * Retourne le login en mode CAS ou BDD
80              *
81              * @return login ou -1 - Le login est stockÃ© en variable de session si ok
82              */
83             function getLogin() {
84                             $ident_type = $this->ident_type;
85                             if (!isset($ident_type)){
86                                             echo "Cette fonction doit Ãªtre appelÃ©e aprÃ¨s soit
    init_LDAP, init_CAS, ou init_BDD";
87                                             die;
88                             }
89                             if (!isset($_SESSION["login"])) {
90                                             /*
91                                              * RÃ©cupÃ©ration du login selon le type
92                                              */
93                                             if ($ident_type == "BDD") {
94
95                                             }elseif ($ident_type == "CAS") {
96
97
    phpCAS::client(CAS_VERSION_2_0,$this->CAS_address,$this->CAS_port,$this-
    >CAS_uri);
98                                                                             //
                                phpCAS::isAuthenticated()==FALSE) {
99                                                                             phpCAS::forceAuthentication();
100                                                                            //
                                }
101                                                                            $_SESSION["login"] =
    phpCAS::getUser();
102                                            }
103                            }
104
105                            if (isset( $_SESSION["login"])) {
106                                            $this->login = $_SESSION["login"];
107                                            return $_SESSION["login"];
108                            }else{
109                                            return -1;
110                            }
111            }
112            /**
113             * Teste le login et le mot de passe sur un annuaire ldap
114             *
115             * @param string $login
116             * @param string $password
117             * @return string $password |int -1
118             */
119            function testLoginLdap ($login,$password) {
```

```
120                             if (!isset($this->ident_type)){
121                                     echo "Cette fonction doit Ãªtre appelee apres
    init_LDAP";
122                                     die;
123                             }
124                             $ldap = @ldap_connect($this->LDAP_address,$this->LDAP_port)
125                             or die("Impossible de se connecter au serveur LDAP.");
126             if ($this->LDAP_v3)
127                             {
128                                     ldap_set_option($ldap,
    LDAP_OPT_PROTOCOL_VERSION, 3);
129                             }
130                             if ($this->LDAP_tls)
131                             {
132                                     ldap_start_tls($ldap);
133                             }
134                             $dn = $this->LDAP_user_attrib."=".$login.",".$this->LDAP_basedn;
135                             $rep=ldap_bind($ldap,$dn, $password);
136                             if ($rep == 1)
137                             {
138                                     $_SESSION["login"] = $login;
139                                     return $login;
140                             }else{
141                                     return -1;
142                             }
143             }
144
145
146             /**
147              * DÃ©connexion de l'application
148              * @return 0:1
149              */
150             function disconnect($adresse_retour) {
151                             if (!isset($this->ident_type)) {
152                                     return 0;
153                             }
154                             if ($this->ident_type == "CAS") {
155                                     phpCAS::client(CAS_VERSION_2_0,$this-
    >CAS_address,$this->CAS_port,$this->CAS_uri);
156                                     phpCAS::logout($adresse_retour);
157                             }
158                             // DÃ©truit toutes les variables de session
159                             $_SESSION = array();
160
161                             // Si vous voulez dÃ©truire complÃ¨tement la session, effacez
    Ã©galement
162                             // le cookie de session.
163                             // Note : cela dÃ©truira la session et pas seulement les donnÃ©es
    de session !
164                             if (isset($_COOKIE[session_name()])) {
165                                     setcookie(session_name(), '', time()-42000, '/');
166                             }
167
168                             // Finalement, on dÃ©truit la session.
169                             session_destroy();
170                             return 1;
171
172             }
173}
```

VIII. Licence de Documentation Libre GNU (GNU Free Documentation License)

Free Software Foundation

This is an unofficial translation of the GNU Free Documentation License into French. It was not published by the Free Software Foundation, and does not legally state the distribution terms for software that uses the GNU FDL--only the original English text of the GNU FDL does that. However, we hope that this translation will help language speakers understand the GNU FDL better. Voici une adaptation non officielle de la Licence de Documentation Libre du projet GNU. Elle n'a pas été publiée par la Free Software Foundation et son contenu n'a aucune portée légale car seule la version anglaise de ce document détaille le mode de distribution des logiciels sous GNU FDL. Nous espérons cependant qu'elle permettra aux francophones de mieux comprendre la FDL.

A. DOMAINE D'APPLICATION ET DÉFINITIONS

La présente Licence s'applique à tout manuel ou travail contenant une mention placée par le détenteur du copyright indiquant que le document peut être distribué selon les termes de la présente Licence. Le terme « Document », ci-dessous, se réfère à tout manuel ou travail remplissant cette condition. Tout membre du public est bénéficiaire de la licence et se trouve ici désigné par « vous ».

Une « Version Modifiée » du Document signifie : tout travail contenant le Document, en intégralité ou en partie, aussi bien une copie verbatim ou avec des modifications qu'une traduction dans une autre langue.

Une « Section Secondaire » est une annexe ou un avant-propos du Document qui concerne exclusivement le rapport de l'éditeur ou des auteurs du Document avec le sujet général du Document (ou des domaines voisins) et ne contient rien qui puisse tomber directement sous le coup du sujet général (par exemple, si le Document est en quelque partie un manuel de mathématiques, une Section Secondaire n'enseignera pas les mathématiques). Le rapport peut être une connexion historique avec le sujet ou des domaines voisins, ou une précision légale, commerciale, philosophique, éthique ou politique les concernant.

Les « Sections Invariables » sont certaines Sections Secondaires désignées par leurs titres comme Sections Invariables dans la mention qui indique que le Document est couvert par la présente Licence.

Les « Textes de Couverture » sont certains courts passages du texte listés comme « Textes de Première de Couverture » ou « Textes de Quatrième de Couverture » dans la mention qui indique que le Document est couvert par la présente Licence.

Une copie « Transparente » du Document signifie : une copie lisible par une machine, réalisée dans un format dont les spécifications sont disponibles au grand public, et dont le contenu peut être directement visualisé et édité avec des éditeurs de texte génériques ou (pour les images composées de pixels) avec des programmes de composition d'images génériques ou (pour les figures techniques) un éditeur de dessin vectoriel largement diponible, et qui soit approprié aux logiciels qui mettent le texte en forme et le calibrent (formateurs de texte) ou au transcodage automatique vers un assortiment de formats appropriés aux formateurs de texte. Une copie réalisée dans un format de fichier habituellement Transparent mais dont le balisage a été conçu pour contrecarrer ou décourager des modifications ultérieures par le lecteur n'est pas Transparente. Une copie qui n'est pas « Transparente » est appelée « Opaque ».

Les formats appropriés aux copies Transparentes sont par exemple l'ASCII brut sans balises, le format Texinfo, le format LaTeX, SGML ou XML utilisant une DTD publiquement disponible, et l'HTML simple et conforme à la norme, conçu en vue d'une modification manuelle. Les formats Opaques incluent PostScript, PDF, les formats propriétaires qui ne peuvent être lus et édités que par des traitements de texte propriétaires, SGML et XML dont les DTD et/ou les outils de rendu ne sont pas généralement disponibles, et l'HTML généré automatiquement par certains traitements de texte à seule fin d'affichage.

La « Page de Titre » désigne, pour un livre imprimé, la page de titre proprement dite, plus les pages suivantes qui sont nécessaires pour faire figurer, lisiblement, les éléments dont la présente Licence requiert qu'ils apparaissent dans la Page de Titre. Pour les travaux dont le format ne comporte pas de page de titre en tant que telle, « Page de Titre » désigne le texte jouxtant l'apparition la plus marquante du titre de ce travail, qui précède le début du corps du texte.

B. COPIES VERBATIM

Vous pouvez copier et distribuer le Document sur tout support, aussi bien commercialement que non, pour autant que la présente Licence, les mentions de copyright, et les mentions de licence indiquant que la présente Licence s'applique au Document soient reproduites sur toutes les copies, et que vous n'ajoutiez aucune autre condition à celles de la présente Licence. Vous ne pouvez pas user de moyens techniques à des fins d'obstruction ou de contrôle de la lecture ou de la duplication des copies que vous réalisez ou distribuez. Vous pouvez cependant accepter des compensations en échange de la cession de copies. Si vous distribuez un assez grand nombre de copies, vous devez aussi suivre les conditions de la section Copies en quantité.

Vous pouvez aussi prêter des copies, selon les mêmes conditions que celles mentionnées ci-dessus, et vous pouvez exposer publiquement des copies.

C. COPIES EN QUANTITÉ

Si vous publiez des copies imprimées du Document à plus de 100 exemplaires, et que la mention de la licence du Document exige des Textes de Couverture, vous devez inclure les copies dans des couvertures où figurent, clairement et lisiblement, tous ces Textes de Couverture : les Textes de Première de Couverture sur la première de couverture, et les Textes de Quatrième de Couverture sur la quatrième de couverture. Les deux faces de la couverture doivent également clairement et lisiblement vous identifier comme étant l'éditeur de ces copies. La première de couverture doit présenter le titre complet, titre dont tous les mots doivent être également mis en valeur et visibles. Vous pouvez ajouter des éléments supplémentaires sur les couvertures. Toute copie avec des changements limités aux couvertures, pour autant qu'ils préservent le titre du Document et satisfont ces conditions, peut être considérée comme une copie verbatim à tous les autres égards.

Si les textes destinés à l'une ou l'autre page de couverture sont trop volumineux pour y figurer lisiblement, vous devez en mettre les premiers (autant qu'il est raisonnablement possible) sur la couverture proprement dite, et poursuivre sur les pages adjacentes.

Si vous publiez ou distribuez des copies Opaques du Document à plus de 100 exemplaires, vous devez soit inclure une copie Transparente dans un format lisible par une machine, adapté au traitement automatisé, en accompagnement de chaque copie Opaque, soit indiquer aux côtés de ou dans chaque copie Opaque une adresse de réseau électronique publiquement accessible, qui permette d'obtenir une copie Transparente du Document, sans éléments ajoutés, à laquelle le grand public puisse accéder pour téléchargement anonyme et sans frais en utilisant des protocoles de réseau publics et standard. Si vous retenez la dernière option, vous devez procéder prudemment et prendre les mesures nécessaires, lorsque vous commencez la distribution de copies Opaques en quantité, afin de vous assurer que cette copie Transparente demeurera accessible au public pendant au moins une année après le moment de la distribution (directement ou par l'intermédiaire de vos agents ou revendeurs) de la dernière copie Opaque de cette édition.

Il est souhaité, mais non exigé, que vous contactiez les auteurs du Document bien avant la redistribution de tout grand nombre de copies, afin de leur laisser la possibilité de vous fournir une version mise à jour du Document.

D. MODIFICATIONS

Vous pouvez copier et distribuer une Version Modifiée du Document selon les conditions des sections Copies verbatim et Copies en quantité qui précèdent, pourvu que vous diffusiez la Version Modifiée sous couvert précisément de la présente Licence, avec la Version Modifiée remplissant alors le rôle du Document, et ainsi autoriser la distribution et la modification de la Version Modifiée à quiconque en possède une copie. En complément, vous devez accomplir ce qui suit sur la Version Modifiée :

A. Utilisez dans la Page de Titre (et sur les couvertures, le cas échéant) un titre distinct de celui du Document et de ceux des précédentes versions (qui doivent, s'il en existe, être citées dans la

section « Historique » du Document). Vous pouvez utiliser le même titre qu'une version précédant la vôtre si l'éditeur original vous en donne la permission.

B. Indiquez sur la Page de Titre, comme auteurs, une ou plusieurs personnes ou entités responsables de l'écriture des modifications de la Version Modifiée, ainsi qu'au moins cinq des principaux auteurs du Document (ou tous les auteurs principaux, s'ils sont moins de cinq).

C. Apposez sur la Page de Titre de nom de l'éditeur de la Version Modifiée, en tant qu'éditeur.

D. Préservez toutes les mentions de copyright du Document.

E. Ajoutez une mention de copyright appropriée à vos modifications, aux côtés des autres mentions de copyright.

F. Incluez, immédiatement après les mentions de copyright, une mention de licence qui accorde la permission publique d'utiliser la Version Modifiée selon les termes de la présente Licence, sous la forme présentée dans la section Addendum ci-dessous.

G. Préservez dans cette mention de licence les listes complètes des Sections Invariables et des Textes de Couverture exigés, données dans la mention de licence du Document.

H. Incluez une copie non altérée de la présente Licence.

I. Préservez la section intitulée « Historique », et son titre, et ajoutez-y un article indiquant au moins le titre, l'année, les nouveaux auteurs, et l'éditeur de la Version Modifiée telle qu'elle apparaît sur la Page de Titre. Si le Document ne contient pas de section intitulée « Historique », créez-en une et indiquez-y le titre, l'année, les auteurs et l'éditeur du Document tels qu'indiqués sur la Page de Titre, puis ajoutez un article décrivant la Version Modifiée, comme exposé dans la phrase précédente.

J. Préservez, le cas échéant, l'adresse de réseau électronique donnée dans le Document pour accéder publiquement à une copie Transparente du Document, et préservez de même les adresses de réseau électronique données dans le Document pour les versions précédentes, sur lequelles le Document se fonde. Cela peut être placé dans la section « Historique ». Vous pouvez omettre l'adresse de réseau électronique pour un travail qui a été publié au moins quatre ans avant le Document lui-même, ou si l'éditeur original de la version à laquelle il se réfère en donne l'autorisation.

K. Dans toute section intitulée « Remerciements » ou « Dédicaces », préservez le titre de section et préservez dans cette section le ton et la substance de chacun des remerciements et/ou dédicaces donnés par les contributeurs.

L. Préservez toutes les Sections Invariables du Document, non altérées dans leurs textes et dans leurs titres. Les numéros de sections ou leurs équivalents ne sont pas considérés comme faisant partie des titres de sections.

M. Supprimez toute section intitulée « Approbations ». Une telle section ne doit pas être incluse dans la Version Modifiée.

N. Ne changez pas le titre d'une section existante en « Approbations » ou en un titre qui entre en conflit avec celui d'une Section Invariable quelconque.

Si la Version Modifiée inclut de nouvelles sections d'avant-propos ou des annexes qui remplissent les conditions imposées aux Sections Secondaires et ne contiennent aucun élément tiré du Document, vous pouvez, à votre convenance, désigner tout au partie de ces sections comme « Invariables ». Pour ce faire, ajoutez leurs titres à la liste des Sections Invariables dans la mention de licence de la Version Modifiée. Ces titres doivent être distincts de tout autre titre de section.

Vous pouvez ajouter une section intitulée « Approbations », pourvu qu'elle ne contienne rien d'autre que l'approbation de votre Version Modifiée par diverses parties -- par exemple, indication d'une revue par les pairs ou bien que le texte a été approuvé par une organisation en tant que définition de référence d'un standard.

Vous pouvez ajouter un passage de cinq mots ou moins en tant que Texte de la Première de Couverture, et un passage de 25 mots ou moins en tant que Texte de Quatrième de Couverture, à la fin de la liste des Textes de Couverture de la Version Modifiée. Toute entité peut ajouter (ou réaliser, à travers des arrangements) au plus un passage en tant que Texte de la Première de

Couverture et au plus un passage en tant que Texte de la Quatrième de Couverture. Si le Document inclut déjà un texte de Couverture pour la même couverture, précédemment ajouté par vous ou, selon arrangement, réalisé par l'entité pour le compte de laquelle vous agissez, vous ne pouvez en ajouter un autre ; mais vous pouvez remplacer l'ancien, avec la permission explicite de l'éditeur qui l'a précédemment ajouté.

Le ou les auteur(s) et le ou les éditeur(s) du Document ne confèrent pas par la présente Licence le droit d'utiliser leur nom à des fins publicitaires ou pour certifier ou suggérer l'approbation de n'importe quelle Version Modifiée.

E. MÉLANGE DE DOCUMENTS

Vous pouvez mêler le Document à d'autres documents publiés sous la présente Licence, selon les termes définis dans la section Modifications ci-dessus, traitant des versions modifiées, pour autant que vous incluiez dans ce travail toutes les Sections Invariables de tous les documents originaux, non modifiées, et en les indiquant toutes comme Sections Invariables de ce travail dans sa mention de licence.

Le travail issu du mélange peut ne contenir qu'une copie de cette Licence, et de multiples Sections Invariables identiques peuvent n'être présentes qu'en un exemplaire qui les représentera toutes. S'il existe plusieurs Sections Invariables portant le même nom mais des contenus différents, faites en sorte que le titre de chacune de ces sections soit unique, en indiquant à la fin de chacune d'entre elles, entre parenthèses, le nom de l'auteur original ou de l'éditeur de cette section s'il est connu, ou un numéro unique dans les collisions restantes. Pratiquez les mêmes ajustements pour les titres de sections, dans la liste des Sections Invariables de la mention de licence de ce travail mélangé.

Dans le mélange, vous devez regrouper toutes les sections intitulées « Historique » dans les divers documents originaux, afin de constituer une unique section intitulée « Historique » ; combinez de même toutes les sections intitulée « Remerciements », et toutes les sections intitulées « Dédicaces ». Vous devez supprimer toutes les sections intitulées « Approbations ».

F. RECUEILS DE DOCUMENTS

Vous pouvez réaliser un recueil regroupant le Document et d'autres documents publiés sous la présente Licence, et remplacer les diverses copies de la présente Licence figurant dans les différents documents par une copie unique incluse dans le recueil, pour autant que vous suiviez les règles de la présente Licence relatives à la copie verbatim pour chacun de ces documents, dans tous les autres aspects.

Vous pouvez n'extraire qu'un seul document d'un tel recueil, et le distribuer individuellement sous la présente Licence, pour autant que vous insériez une copie de la présente Licence dans le document extrait, et que vous suiviez la présente Licence dans tous ses autres aspects concernant la reproduction verbatim de ce document.

G. AGRÉGATION AVEC DES TRAVAUX INDÉPENDANTS

Une compilation du Document ou de ses dérivés avec d'autres documents ou travaux séparés et indépendants, ou bien sur une unité de stockage ou un support de distribution, ne compte pas comme une Version Modifiée de ce Document, pour autant qu'aucun copyright de compilation ne soit revendiqué pour la compilation. Une telle compilation est appelée une « agrégation », et la présente Licence ne s'applique pas aux autres travaux contenus et ainsi compilés avec le Document, sous prétexte du fait qu'ils sont ainsi compilés, s'ils ne sont pas eux-mêmes des travaux dérivés du Document.

Si les exigences de la section Copies en quantité en matière de Textes de Couverture s'appliquent aux copies du Document, et si le Document représente moins du quart de la totalité de l'agrégat, alors les Textes de Couverture du Document peuvent n'être placés que sur les couvertures qui entourent le document, au sein de l'agrégation. Dans le cas contraire, ils doivent apparaître sur les couvertures entourant tout l'agrégat.

H. TRADUCTION

La traduction est considérée comme un type de modification, de sorte que vous devez distribuer les traductions de ce Document selon les termes de la section Modifications. La substitution des Sections Invariables par des traductions requiert une autorisation spéciale de la part des détenteurs du copyright, mais vous pouvez ajouter des traductions de tout ou partie des Sections Invariables en sus des versions originales de ces Sections Invariables. Vous pouvez

inclure une traduction de la présente Licence pourvu que que vous incluiez la version originale, en anglais, de la présente Licence. En cas de désaccord entre la traduction et la version originale, en anglais, de la présente Licence, la version originale prévaudra.

I. RÉVOCATION

Vous ne pouvez copier, modifier, sous-licencier ou distribuer le Document autrement que selon les conditions expressément prévues par la présente Licence. Toute tentative de copier, modifier, sous-licencier ou distribuer autrement le Document est nulle et non avenue, et supprimera automatiquement vos droits relatifs à la présente Licence. De même, les parties qui auront reçu de votre part des copies ou des droits sous couvert de la présente Licence ne verront pas leurs licences révoquées tant que ces parties demeureront en pleine conformité avec la présente Licence.

J. RÉVISIONS FUTURES DE LA PRÉSENTE LICENCE

La Free Software Foundation (« fondation du logiciel libre ») peut publier de nouvelles versions révisées de la présente GNU Free Documentation License de temps à autre. Ces nouvelles versions seront similaires, dans l'esprit, à la présente version, mais peuvent différer dans le détail pour prendre en compte de nouveaux problèmes ou de nouvelles inquiétudes. Consultez http://www.gnu.org/copyleft/.

Chaque version de la Licence est publiée avec un numéro de version distinctif. Si le Document précise qu'une version particulière de la présente Licence, « ou toute version postérieure » s'applique, vous avez la possibilité de suivre les termes et les conditions aussi bien de la version spécifiée que de toute version publiée ultérieurement (pas en tant que brouillon) par la Free Software Foundation. Si le Document ne spécifie pas un numéro de version de la présente Licence, vous pouvez choisir d'y appliquer toute version publiée (pas en tant que brouillon) par la Free Software Foundation.