Gérer un projet de développement d'application

Méthodes, UML, bonnes pratiques en PHP

Copyright (c) 2010 - 2012 Éric Quinton. Tous droits réservés

Eric Quinton

Dernière modification : 13/09/2012 - révision 252

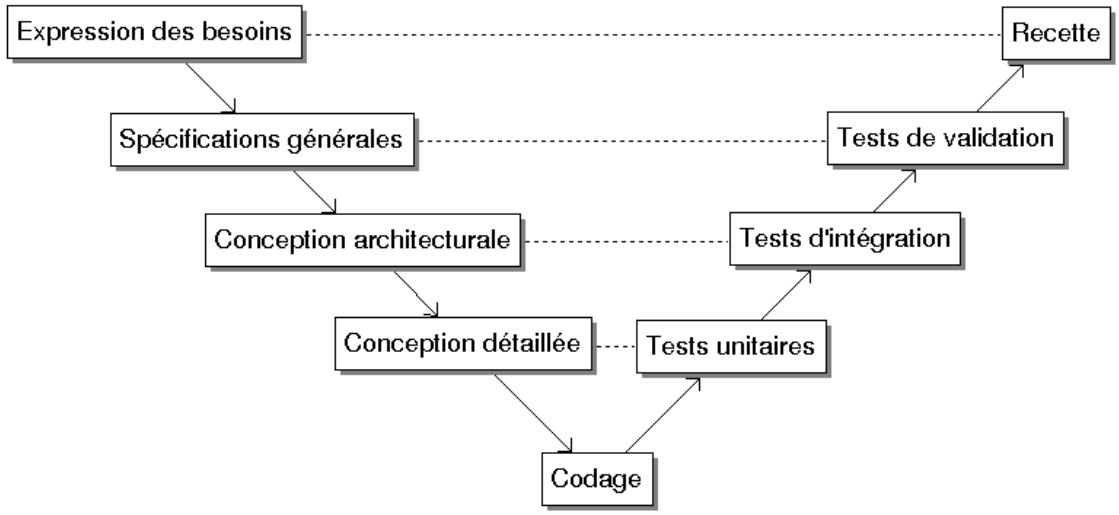
Table des matières

I. Les étapes d'un projet

A. Le développement traditionnel : le cycle en V

Le développement logiciel est mené traditionnellement selon un postulat : le demandeur (maître d'ouvrage) définit ses besoins, les décrit, et le développeur (maitre d'œuvre) réalise le logiciel correspondant. Le demandeur vérifie ensuite que le logiciel correspond bien à sa demande.

Le modèle du cycle en V est basé sur deux branches : la première descend depuis l'expression du besoin vers le codage, la seconde remonte du codage vers la mise en production, en passant par les phases de tests.



B. Maîtrise d'œuvre, maîtrise d'ouvrage

Le cycle en V nécessite l'intervention de deux acteurs majeurs : le maître d'ouvrage, celui qui définit ce qu'il veut, et le maître d'œuvre,celui qui réalise l'opération.

Cette dichotomie permet de séparer parfaitement les rôles : le maître d'ouvrage écrit le cahier des charges, le maître d'œuvre réalise ce qui a été demandé. Le maître d'ouvrage « paye » après avoir vérifié que ce qui lui a été livré correspond bien à sa demande.

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 : l'incompréhension peut vite s'installer, et les conflits survenir là où il aurait été préférable de travailler en bonne entente.

Pour pallier cet inconvénient, un troisième acteur intervient parfois, l'assistant à maitre d'ouvrage. Il 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 qui est livré correspond bien à ce qui a été demandé.

Pour formaliser le développement selon ces approches, de nombreuses méthodes ont vu le jour, dont la plus courante est la méthode **CMMI**[1] (Capability Maturity Model Integration). Il existe également une norme ISO, la norme 15504,

1 http://www.sei.cmu.edu/cmmi/

également connue sous le nom de **SPICE**[1] (Software Process Improvement and Capability Determination), les deux étant amenées à converger.

C. Les inconvénients des démarches traditionnelles

Les méthodes traditionnelles sont basées sur la rédaction du cahier des charges, qui est le « juge de paix » du projet. Le maître d'œuvre doit réaliser tout ce qui est prévu, et uniquement ce qui est prévu.

Si ce mécanisme est parfait sur le papier, il présente au moins deux inconvénients. Le premier tient à la difficulté de formaliser quelque chose qui n'existe pas (le logiciel) dans sa complétude, d'autant que cette formalisation est réalisée par des personnes qui ne sont pas des informaticiens, qui ne connaissent pas forcément l'état de l'art, ce qu'il est possible de faire, et ce qui ne l'est pas. Aujourd'hui, si un client réalisait un cahier des charges pour disposer d'un véhicule capable de se déplacer dans tous les environnements (terre, air, eau, espace), pouvant transporter 5 personnes, et à une vitesse de 10000 Km/h, il est probable qu'il n'obtiendrait pas le produit désiré... Très souvent, un assistant à maîtrise d'ouvrage est adjoint au maître d'ouvrage, qui l'aidera à formaliser la demande en fonction des contraintes informatiques.

Le second inconvénient tient à la durée du projet. Entre l'expression des besoins et la mise en production, beaucoup de temps peut s'écouler. Hélas, pendant cette période, les besoins peuvent évoluer, la concurrence s'exacerber, de nouveaux marchés apparaître... Le cahier des charges étant par définition intangible, sauf à multiplier des avenants, le logiciel mis en production risque fort d'être dépassé quand il sera terminé.

Il est également reproché aux méthodes traditionnelles d'être très consommatrices en papier – on parle de documentation pléthorique. La mise en place d'une démarche de ce type, par exemple CMMI, entraîne de nombreuses réunions à tous les niveaux du projet, avec rédaction systématique de comptes-rendus. Les incidents sont déclarés sur papier, confirmés, traités, puis leur correction vérifiée, le plus souvent dans le cadre d'une procédure d'assurance qualité. L'état d'avancement du projet est vérifié à partir de cette documentation, et non à partir du fonctionnement proprement dit du logiciel, le paradigme étant que la documentation reflète la réalité du développement. Sur de gros projets, cette documentation va finir par être très gourmande en temps, mobiliser plusieurs personnes uniquement pour dire que le développement du logiciel suit son cours ou qu'il fonctionne...

Pour éviter ces écueils, de nouvelles méthodes sont apparues ces dernières années, les méthodes agiles.

D. Les méthodes Agiles

Face à la complexité de mise en œuvre de projets et pour limiter les risques inhérents aux démarches traditionnelles, à savoir, effet tunnel, retards de mise en œuvre, 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.

1 http://www.sqi.gu.edu.au/SPICE/

1. 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.

a) 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.

b) 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.
- Livrez 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.
- 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.

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

- 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.

E. 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. SCRUM[2]

SCRUM (mêlée en français) : cette méthode fournit un cadre de gestion de projet, mais pas de règles d'ingénierie logicielle. Elle peut être utilisée pour tout projet (et pas uniquement informatique), mais doit être complétée, pour les aspects techniques, par d'autres méthodes « techniques ».

Principe général :

- l'organisation est divisée en petites équipes multidisciplinaires et auto-organisées
- le travail est divisé en une liste de petits livrables concrets, dont chaque élément est trié par priorité dont la taille relative est estimée
- le temps est divisé en petites itérations de taille fixe (en général, 2 ou 3 semaines), appelées sprint. A l'issue de chaque sprint, une démonstration est réalisée avec un produit potentiellement livrable
- le planning de la version doit être optimisé et les priorités mises à jour avec le client, sur la base de ce qui a été appris après chaque sprint.

1 Source : http://fr.wikipedia.org/wiki/Méthode_agile
2 http://www.scrum.org/

- Le processus est optimisé en organisant une rétrospective après chaque sprint.

Concrètement, chaque sprint démarre par une réunion qui permet de fixer les objectifs (ce qui va être réalisé pendant la durée du sprint). Chaque matin, une réunion de 10' maxi est organisée au sein de l'équipe, la mêlée (scrum). Tous les participants restent debout ! Enfin, en fin de sprint, une démo est présentée, suivie d'une réunion de rétrospective, qui permet de discuter sur ce qu'il s'est passé pendant le sprint.

Scum, méthode de gestion du projet, est souvent associée à l'extreme programming[1], méthode de programmation.

2. *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
   - 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

F. 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 œuvre qu'à condition de segmenter les modules en une taille raisonnable ;

1 http://www.extremeprogramming.org/
2 Source : 01 Informatique du 13/03/2008

sur le plan réglementaire, il est très difficile de fixer des conditions dans le cadre d'un marché ou d'un contrat. Un contrat vise à fournir un logiciel répondant à des besoins, à un coût fixé. Mais les objectifs ou les prescriptions pouvant évoluer au cours du développement, l'équilibre entre le contenu fonctionnel et l'investissement réalisé par le prestataire peut être difficile à trouver.

De plus, l'estimation des coûts est beaucoup plus complexe à faire dans ce cadre. Enfin, 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, leur mise en œuvre présente des intérêts indéniables, ne serait-ce que parce que l'implication obligatoire du demandeur augmente les chances que le logiciel sera utilisé *in fine*.

Sur des projets plus importants, il faut se tourner impérativement vers des méthodes éprouvées si l'on souhaite utiliser la méthodologie Agile, comme PUMA, ou SCRUM/XP, pour bénéficier au maximum de la normalisation qu'elles intègrent. Le passage à l'industrialisation et à la normalisation, au sein de l'organisation, devient alors obligatoire et ne peut être qu'un projet porté par l'entreprise (ou le service).

G. Les phases à retenir pour les petits projets

Mettre en place une norme, gérer des cahiers des charges et trois acteurs pour un même projet, cela fait « riche » pour de petits projets, souvent menés en interne ou par des équipes réduites.

En fait, sur un projet, quatre phases essentielles se rencontrent systématiquement :

- 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. C'est l'informaticien qui établit le cahier des charges : il devient alors, selon la terminologie classique, l'assistant à maitre d'ouvrage. C'est à cette étape que les exigences vont être définies (les contraintes imposées) ;
- 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 et 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...

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 concepteur 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'œuvre.

Les deux doivent travailler ensemble : un maître d'œuvre 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. Le maître d'œuvre 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'œuvre 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 ».

Pour 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 différences d'appréciations sur le cahier des charges, ou 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.

1. La demande initiale

Non formalisée, elle tient en général en une ligne ou une demi-page. L'archétype en 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, peut 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, on n'en aura pas besoin ». En général, à ce stade, il est bon de noter le point abordé, qui peut être parfois très structurant pour la suite du projet, même s'il n'est pas prévu de l'informatiser dans la phase initiale ;

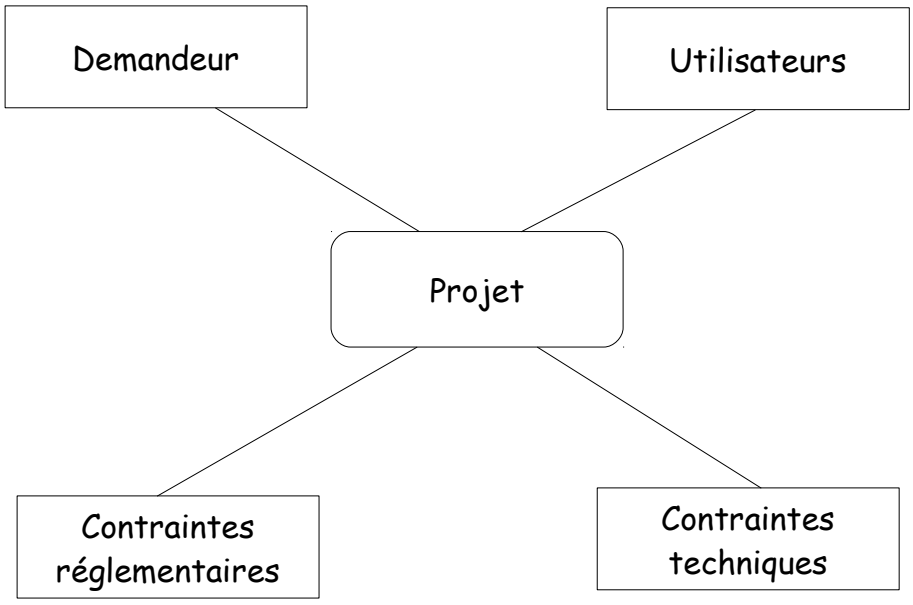
- d'autre part, porter un jugement sur la faisabilité (ou plutôt sur la non-faisabilité) d'une fonction ou d'une partie de la demande. C'est le rôle de l'analyse préalable de s'assurer dans quelles conditions la réalisation ou la mise en œuvre sont possibles, pas de la demande initiale.

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.

Un projet s'inscrit dans un contexte : on ne part pas de rien. Des contraintes existent, qu'elles soient implicites (l'application doit être utilisée avec un micro-ordinateur fonctionnant sous le système d'exploitation le plus courant) ou explicitées dans la demande initiale.

Un projet ne pourra aboutir que s'il prend en compte les contraintes suivantes :
- le demandeur paye le produit. Il est normal que le logiciel rende les services qu'il a demandé ;
- 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 à l'occasion de l'arrivée du nouveau logiciel) ;
- il existe une réglementation, et le logiciel doit s'y conformer. Très souvent, 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 (règles comptables, lois, décrets, circulaires d'application, normes en vigueur...) ;
- le projet s'insère dans un existant, composé de serveurs, de bases de données, d'outils... Si un logiciel est écrit en s'appuyant, par exemple, sur Java pour le code et MySql pour la base de données, mais que le client impose un développement avec WinDev et une base de données postgreSql, il est normal que le client refuse de payer l'application qui lui sera fournie...

Au final, le projet intègre quatre éléments : le demandeur, l'utilisateur, les contraintes techniques et les contraintes réglementaires.



Ne pas tenir compte d'une de ces contraintes, c'est s'exposer à de graves difficultés, soit lors de 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.

2. La restitution de la demande

Il s'agit de formaliser ce qui a été 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, il faudra se préoccuper de savoir d'où viennent les données, quel est leur format, quelle est la fréquence d'importation...
- de pouvoir estimer le « volume » du projet, et de définir les différentes phases et modules.

C'est lors de cette phase que la viabilité du projet va être étudiée.

Pour restituer la demande, et en partant du principe qu'un mauvais dessin vaut mieux qu'un grand discours, il est préférable de 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 qui va être réalisé ;
- 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 qui est faite de ces objets.

Ce document doit faire l'objet d'une validation de la part du demandeur (le maître d'ouvrage). Cette validation est essentielle : elle permet à chacun de partir sur une plate-forme commune, même si cette plate-forme peut évoluer au fil du temps.

3. 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.

4. 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.

5. 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 !

6. 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.

7. 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.

8. 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

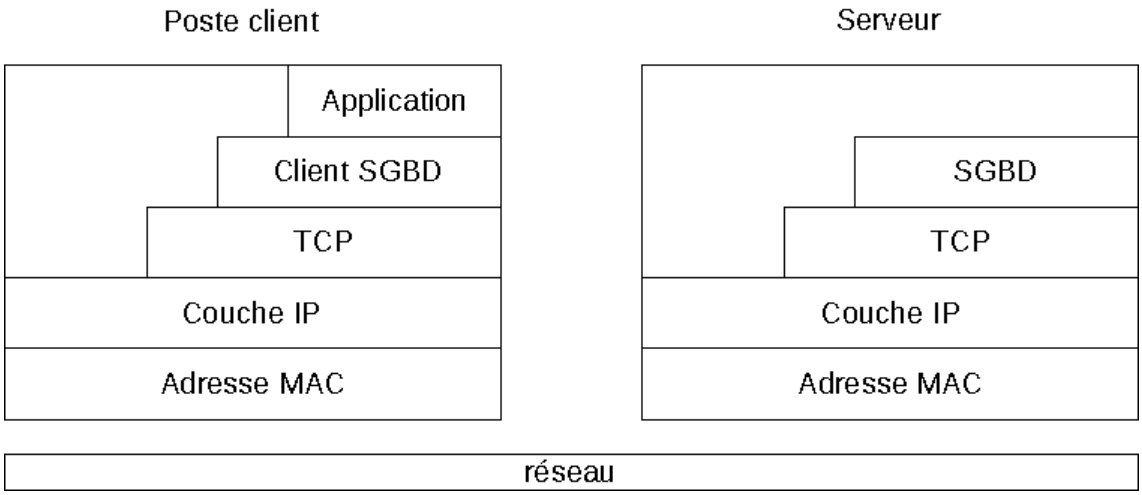
restreint d'utilisateurs qui maîtrisent bien leur procédure. Par contre, il est important de rédiger le guide « Administrateur », qui va décrire la solution technique retenue, l'implémentation de la base de données, le paramétrage, et la gestion des droits.

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

II. L'accès aux données

A. Les accès natifs

La communication entre un client, l'application, et le SGBD, le serveur, s'établit en utilisant les protocoles réseau classiques :



- l'application communique avec le logiciel client du SGBD, pour lui demander d'exécuter une requête ;
- le logiciel client SGBD établit une connexion avec le serveur, en envoyant une requête qui va contenir l'adresse IP de ce dernier, et le port TCP utilisé par le SGBD pour « écouter » le réseau (par exemple, pour MYSQL, le port TCP « standard » est le 3306, pour SYBASE ASE sous Windows ou Linux, c'est le port 5000). La requête contient également l'adresse IP du poste client, et le port TCP auquel il faudra retourner les résultats (le port TCP est choisi aléatoirement par le système) ;
- le serveur reçoit la requête, l'oriente vers l'application SGBD, grâce au port TCP indiqué dans la requête ;
- le SGBD traite la requête, puis renvoie le résultat de son exécution au client, grâce à l'adresse IP et au port TCP fourni ;
- le poste client reçoit la requête, la transmet au client SGBD ;
- le client SGBD fournit à l'application les résultats de la requête.

Les échanges avec le SGBD ne sont pas normalisés : chaque éditeur a mis en place ses propres stratégies pour établir le dialogue, formater les données... C'est pourquoi nous sommes obligés d'installer, sur chaque poste client, le logiciel qui permettra d'établir la communication avec le SGBD.

B. L'approche Objet dans les SGBD

Lorsque les mécanismes Objet ont été décrits, et notamment avec l'avènement de UML, certains chercheurs ont essayé d'abord de créer des bases de données objet, dont le résultat a été très mitigé : les bases de données manipulant des objets sont assez rares, tout simplement parce que le langage SQL, s'il n'est pas parfait, présente l'avantage d'être simple, efficace, et universel. Aujourd'hui, les mécanismes objets ne sont implémentés que pour des usages bien spécifiques. Certains auteurs considèrent que les annuaires à la norme LDAP peuvent être assimilés à des bases de données Objet. On retrouve également ce mécanisme implémenté dans des bases de données SQL classiques : ORACLE, par exemple, a introduit un objet appelé « cartouche spatial », qui permet de manipuler les informations utilisées dans les systèmes d'informations géographiques.

POSTGRESQL dispose, lui aussi, de son objet géographique, implémenté dans l'extension POSTGIS.

Le langage SQL, dans sa version 3 (SQL-3 ou SQL-99) a intégré quelques mécanismes issus de la recherche sur les objets, comme l'héritage, qui se retrouve aujourd'hui dans le SGBD POSTGRESQL.

1. L'héritage

POSTGRESQL implémente un concept intéressant, l'héritage de tables. Par exemple, une table Personne pourra être héritée dans une table **Beneficiaire**, cette dernière étant liée aux dossiers d'aide ou aux RIB utilisés pour le paiement. Ces deux tables, **Personne** et **Beneficiaire**, pourront être manipulées de façon très simple avec les extensions SQL de la norme SQL:1999.

Toutefois, la norme SQL:1999 n'a quasiment jamais été implémentée. Concevoir des bases de données selon ce système revient donc à travailler exclusivement avec la base de données POSTGRESQL : ce n'est pas forcément une mauvaise solution, mais il faut que les avantages apportés par le choix de ce gestionnaire de bases de données soient déterminants. Ce SGBD est souvent choisi d'ailleurs en raison de la disponibilité de son objet géographique, dont nous parlerons au paragraphe suivant.

Quant à l'héritage, ce mécanisme peut être contourné en travaillant avec des tables liées. Pour reprendre nos deux tables **Personne** et **Beneficiaire**, il suffit que **Beneficiaire** dispose de la même clé que Personne et qu'une jointure soit réalisée entre les deux tables pour arriver quasiment au même mécanisme.

2. L'objet géographique

La seconde approche a été mise en œuvre principalement pour gérer le stockage des informations géographiques. Pour stocker un polygone ou un point, il faut indiquer des coordonnées, des cheminements (le contour du polygone...), et il faut disposer également de fonctions pour les manipuler. ORACLE et POSTGRESQL, avec son extension POSTGIS, ont intégré depuis quelques années un objet géographique, qui permet de manipuler une information en une seule opération, ce qui simplifie grandement les opérations dans les systèmes d'informations géographiques.

C. Accéder à la base de données en développement Objet

1. Les concepts de base

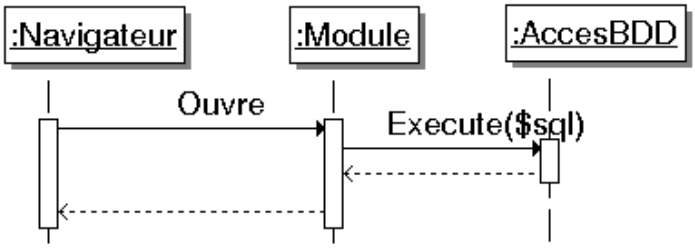
Ne pas confondre le modèle relationnel et l'application Objet !

Les modèles objet, et notamment le diagramme de classes, ressemblent fortement à un modèle relationnel (modèle conceptuel des données Merise, par exemple). La tendance est forte de dessiner un diagramme de classes, qui sera utilisé tel quel pour générer la base de données et, par extension, manipuler les données.

Cette approche présente un inconvénient majeur : elle ne tient pas compte de la nature intrinsèque des SGBD.

Les SGBD sont conçus pour répondre à une requête. Cette requête est émise dans un langage, le SQL, et le SGBD peut renvoyer des informations (une collection) et un code d'erreur. Le contenu de la requête n'influe en rien sur la manière dont les communications s'établissent entre l'application et le SGBD.

D'une manière générale, pour exécuter une requête SQL, c'est le fonctionnement suivant qui est le plus fréquemment mis en œuvre :



Dans cet exemple, le navigateur (l'utilisateur) demande l'ouverture d'un module (une page PHP). Ce dernier va demander à la classe d'accès à la base de données (AccesBDD) d'exécuter une requête SQL. C'est la classe AccesBDD qui va implémenter les commandes spécifiques du SGBD, et non le module directement.

Ce schéma est valable quelle que soit la requête, qu'elle soit de sélection, de modification ou de suppression... La méthode Execute va retourner une collection dans le cas d'une commande de sélection, ou simplement un code de retour en cas d'exécution d'une commande de mise à jour.

Il s'agit, ici, d'un exemple très simple : si ce schéma était conservé en l'état, il faudrait coder toutes les requêtes SQL dans tous les modules, ce qui irait à l'encontre du but recherché, à savoir un codage unique dans l'application.

Il existe plusieurs bibliothèques en PHP qui permettent d'accéder au données en intercalant une couche d'abstraction. Celle-ci va prendre en charge les différentes commandes utilisées pour se connecter et exécuter les requêtes, et permettre au développeur de s'affranchir du type de SGBD utilisé.

Plusieurs grandes bibliothèques sont disponibles en PHP actuellement : celle implémentée dans Zend Framework (classes Zend_db et connexes), qui s'appuie sur PDO[1] (Php Data Objects, une couche d'accès intégrée dans PHP depuis PHP5), celle présente dans PEAR, ou ADODB.

L'utilisation d'une couche d'abstraction ne doit pas faire oublier que seules les requêtes standard seront prises en charge quel que soit le SGBD : il ne faut pas oublier que chaque éditeur a rajouté ses propres extensions. Néanmoins, il est très souvent possible de les masquer en les intégrant dans des vues, codées directement dans la base de données.

2. Concevoir les objets de traitement

Nous l'avons vu, l'application d'un modèle relationnel n'a pas vraiment de sens en programmation objet. Néanmoins, un de ses objectifs est de ne coder qu'une seule fois, et de pouvoir réutiliser le code à d'autres endroits : des modules seront développés pour pouvoir retrouver une classe d'informations rapidement, quel que soit leur moment d'utilisation.

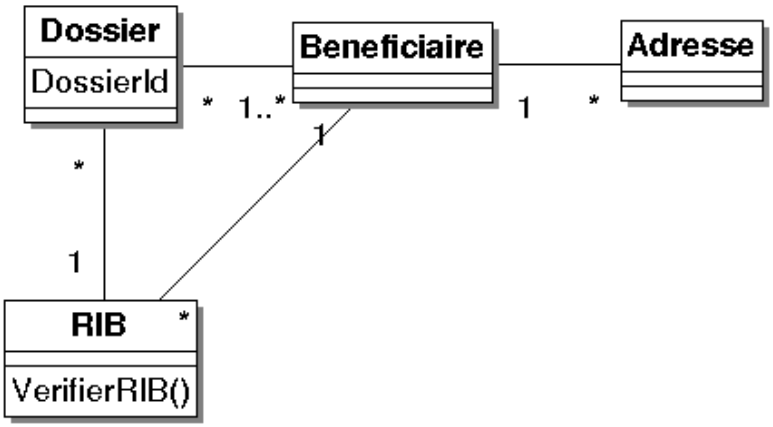
La conception de l'application va passer par les phases suivantes :

- description des objets de traitement nécessaires au sein de l'application ;
- déduction d'un schéma relationnel ;
- création de la base de données dans le SGBD choisi ;
- codage des objets de traitement.

1 http://php.net/manual/fr/book.pdo.php

Pour comprendre le mécanisme, nous allons prendre comme exemple la gestion d'un dossier de subvention. L'application doit permettre de stocker les informations concernant un dossier de subvention. La subvention concerne un bénéficiaire, qui peut disposer de plusieurs adresses (postales, siège social, atelier...), de plusieurs comptes bancaires représentés par des relevés d'identité bancaires (RIB) (…)

De cet exemple, nous pouvons en déduire le diagramme de classe suivant :



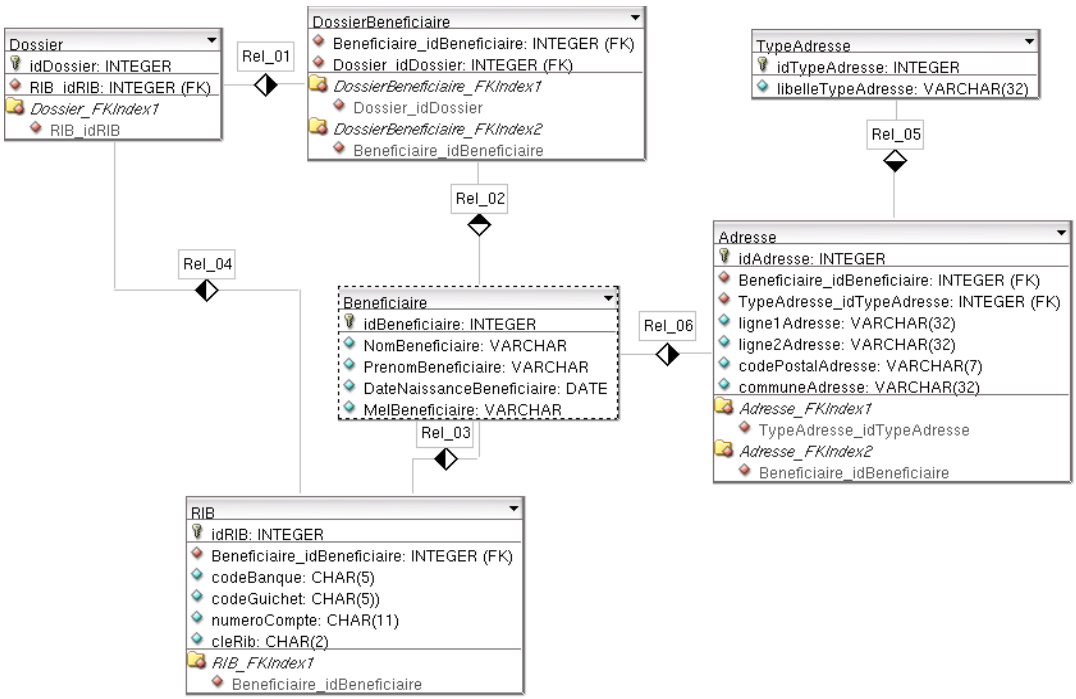
Ce schéma permet de préciser :

- qu'un dossier comporte au moins un bénéficiaire ;
- qu'un bénéficiaire peut avoir plusieurs adresses (adresse postale, adresse de résidence et, pour une société, siège social, atelier...) ;
- qu'un bénéficiaire peut disposer de plusieurs RIB (relevés d'identité bancaire), dont un seul sera utilisé dans le dossier.

En ce qui concerne la création des objets, si l'adresse avait été unique, nous n'aurions pas eu besoin de créer une classe spécifique : elle aurait fait partie des attributs de la classe Beneficiaire.

Il en est de même pour le RIB, qui pourrait être considéré comme un attribut de la classe Dossier. Mais si nous voulons rajouter un programme de vérification de la clé, afin de s'assurer que le RIB saisi soit bien correct, il vaut mieux créer alors une classe RIB qui contiendra alors la fonction permettant de vérifier sa validité.

A partir de ce diagramme, nous pouvons en déduire le schéma de la base de données :



Cette représentation permet de visualiser la table **DossierBeneficiaire**, qui est porteuse de la relation **n-n** entre les tables **Dossier** et **Beneficiaire** (cette relation particulière sera détaillée ultérieurement). Le RIB est stocké dans une table à part, pour pouvoir représenter les différents RIB possédés par un bénéficiaire. Il en est de même pour les adresses, qui sont typées (adresse postale, de résidence...).

3. Quelques règles d'écriture de la base de données

Il n'existe pas de norme particulière, qui va « dire » comment doit être construite une base de données. Au moment de la conception des tables, quelques concepts doivent être conservés à l'esprit :

- il faut faire simple : plus la base de données sera simple, plus elle sera facile à manipuler ;
- il faut rester dans le modèle relationnel : l'information est stockée une et une seule fois ;
- on n'introduit pas de « règles métier » (par exemple, le montant de la subvention ne doit pas dépasser 50 % du devis accepté) dans la base de données : c'est dans l'application qu'elles seront codées[1]. Elles peuvent évoluer au cours du temps, et il est beaucoup plus complexe de modifier une base de données qu'une application.

Les SGBD sont souvent sensibles à la casse (les minuscules et les majuscules sont reconnues comme des caractères différents). Cette caractéristique va pouvoir être utilisée pour rendre notre base de données lisible.

Voici un libellé de colonne, volontairement long, écrit dans trois styles différents :

releveidentitebancairedossierdepret

1 En fait, il est possible de stocker ce type d'informations dans la base de données, mais en utilisant des procédures stockées ou des triggers qui pourront être modifiés sans toucher à la structure des données. C'est un choix d'architecture qui est plus fréquent dans les grands systèmes.

RELEVEIDENTITEBANCAIREDOSSIERDEPRET
ReleveIdentiteBancaireDossierDePret

Le moins lisible est celui écrit en majuscule ; c'est, hélas, une habitude fréquente d'écrire les tables et les colonnes dans cette casse, en partie d'ailleurs à cause des outils de dessin des bases de données, qui proposent fréquemment par défaut la génération des tables dans ce mode. Le troisième est le plus lisible, simplement parce que chaque mot est isolé des autres en le commençant par une majuscule. C'est une notation appelée parfois « à l'allemande », qu'il est conseillé de privilégier.

Il faut faire également attention à ne pas utiliser d'accents ou de caractères spéciaux : selon le jeu de codage des caractères utilisé, le résultat peut être parfois désastreux.

4. En résumé...

Au moment de créer une base de données, il vaut mieux :

- préférer une notation « à l'allemande », en faisant commencer chaque partie du nom du champ d'une majuscule ;
- utiliser systématiquement des clés non significatives, numériques, auto-incrémentées ;
- pour les tables de type N-N, les nommer en concaténant le nom des tables liées, en commençant par la table principale de la relation ;
- gérer les différentes versions, pour s'assurer que le code de l'application sera compatible avec les schémas des tables ;

et, bien sûr, respecter les règles qui auront été fixées...

D. La couche d'abstraction

Chaque SGBD disposant de son propre jeu d'instructions, les programmeurs ont depuis longtemps cherché à s'affranchir le plus possible de la base de données pour réaliser un code portable : l'application peut être développée en s'appuyant sur une base de données PostGreSQL, puis porter l'application sous Oracle ou Sybase, par exemple.

En PHP, les initiatives pour s'abstraire du type de la base de données sont nombreuses. Dans la pratique, elles présentent toutes à peu près le même fonctionnement :

- un objet est créé, qui va gérer la connexion à la base de données ;
- les requêtes sont plus ou moins préparées automatiquement avant l'exécution.

Les bibliothèques d'abstraction les plus connues, aujourd'hui, sont les suivantes :

- ADODB : cette bibliothèque est une des plus anciennes. Elle a été créée en 2000, et est mature. Elle est très couramment utilisée, et fonctionne tant en PHP 4 qu'en PHP 5. On la retrouve intégrée dans la plupart des projets développés ces dernières années ;
- PEAR::MDB2 : c'est le composant d'accès aux données de PEAR (PHP Extension and Application Repository, ensemble de classes destinées à faciliter la vie du programmeur). Il couvre les bases de données les plus courantes ;

- PDO : (PHP Data Objects) : c'est une bibliothèque qui a été intégrée dans le moteur de PHP, et qui couvre la plupart des bases de données. Les requêtes sont en général exécutées ainsi :

```
$stmt=$bdd->prepare("INSERT INTO Beneficiaire (nom,prenom) VALUES (?,
?)");
$stmt->bindParam(1, $nom);
$stmt->bindParam(2,$prenom);

//insertion d'une ligne
$nom='Dupont'';
$prenom='Jean';
$stmt->execute();
```

- Zend_Db_Adapter : c'est un composant du framework Zend. Il s'appuie sur PDO pour les accès aux bases de données, mais en améliore l'approche objet. Il se distingue des autres classes par deux caractéristiques : d'une part, les requêtes SQL peuvent être montées « par morceau », en déclarant à part les clauses WHERE, ORDER..., et d'autre part, il est possible de décrire complètement la structure relationnelle de la base de données, et de lui demander de gérer les contraintes d'intégrité, tant pour les opérations d'écriture que de suppression.

1. *la bibliothèque ADODB*

ADODB[1] est une bibliothèque d'abstraction d'accès aux bases de données, disponible sous PHP et PYTHON. Elle encapsule les accès aux différentes bases de données, sans que le développeur ait besoin de connaître les différentes commandes spécifiques à chacune.

Elle permet d'accéder à quasiment toutes les bases de données disponibles : MySQL, PostgreSQL, Interbase, Firebird, Informix, Oracle, MS SQL, Foxpro, Access, ADO, Sybase, FrontBase, DB2, SAP DB, SQLite, Netezza, LDAP, ODBC, ODBTP. Il est également possible de rajouter ses propres connecteurs, si c'est nécessaire. C'est, à ce jour, très certainement la bibliothèque la plus riche disponible en PHP.

La gestion des exceptions est également disponible sous PHP5. Une version compilée, en langage C, peut également être installée sur le serveur web, ce qui permet d'accélérer nettement son exécution : la bibliothèque reconnaît automatiquement la présence du programme compilé et lui transmet l'exécution des instructions.

De façon simple, pour utiliser ADODB, nous avons besoin de quelques lignes de code.

Pour commencer, les informations concernant la base de données sont stockées dans des variables :

```
$BDD_type = "mysql"; // Base de données de type MYSQL
$BDD_server = "localhost"; // Adresse du serveur de base de données
$BDD_login = "proto"; // Login de connexion à la base de données
$BDD_passwd = "proto"; // Mot de passe associé au login
$BDD_database = "proto"; // Nom de la base de données
```

Puis les bibliothèques ADODB sont chargées, ici depuis le dossier plugins/adodb :

1 http://adodb.sourceforge.net/

```
include_once('plugins/adodb/adodb.inc.php');
```

et la connexion à la base de données est établie :

```
if (!isset($bdd)) {
// La connexion n'existe pas
          $bdd = ADONewConnection($BDD_type);
          $etatconn=$bdd->Connect($BDD_server, $BDD_login,
$BDD_passwd, $BDD_database);
          if ($etatconn == FALSE) {
                    echo( "La connexion a la base de donnees a
echoue" );
                    die ();
          }
}
```

L'application est maintenant connectée à notre base de données.

Nous allons exécuter maintenant une requête de sélection. Tout d'abord, la requête SQL est préparée :

```
$sql = "select * from Dossier";
```

Nous indiquons à notre objet ADOConnexion de travailler dans un mode particulier : chaque colonne sera identifiée par son nom (et non par le numéro de champ) :

```
$bdd->SetFetchMode(ADODB_FETCH_ASSOC);
```

La requête SQL est exécutée. Une instance de l'objet ADORecordSet, contenant le résultat de cette requête va être créé, $rs :

```
$rs = $bdd->execute($sql);
if(!$rs)
{
          $collection=false;
          print $bdd->ErrorMsg();
          } else {
          $collection = array();
          $collection = $rs->GetRows($rs->RecordCount());
}
```

$collection est un tableau indexé à deux dimensions. La première dimension correspond aux lignes retournées (numérotées de 0 à n), la seconde aux champs de chaque ligne.

E. Le couplage relationnel-Objet

1. Objectifs

Nous l'avons vu, la gestion de l'ensemble des opérations concernant une table, à savoir la lecture ou l'écriture des informations nécessite l'écriture d'un code particulièrement important, et qui est particulièrement fastidieux à rédiger.

Ainsi, pour une requête d'écriture, il faut :

- vérifier les données à intégrer dans la base de données, par exemple, s'assurer que les champs numériques ne contiennent pas des caractères ;
- transformer les dates issues du navigateur dans le format attendu par la base de données ;
- le cas échéant, s'assurer que les données ne présentent pas de risque de sécurité ;

- vérifier si c'est une requête d'insertion (INSERT) ou de modification (UPDATE) qu'il faut exécuter ;
- pour chaque champ à mettre à jour, vérifier s'il est du type numérique ou texte et, le cas échéant, l'entourer de guillemets ;
- exécuter la requête ;
- vérifier si celle-ci s'est bien exécutée ;
- et enfin, s'il s'agit d'une requête d'insertion, et que les enregistrements sont identifiés avec une clé automatique (clé auto-incrémentée), récupérer l'identifiant de l'enregistrement inséré.

La tentation est grande de vouloir encapsuler ces opérations dans une classe générique, qui réalisera toutes ces opérations à notre place.

Dans le cadre de la programmation objet, ce mécanisme est parfois appelé couplage relationnel-objet (ou ORM en anglais, Object Relational Mapping).

2. quelques bibliothèques connues

DOCTRINE : s'appuyant sur PDO pour les accès aux bases de données, cette bibliothèque gère les relations entre les tables et crée automatiquement les jointures, en les déduisant d'un schéma défini à partir des classes d'accès. Il faut décrire la structure et les relations dans les classes dérivées (par exemple, Beneficiaire).

EzComponents : il s'agit d'un ensemble de bibliothèques PHP, qui dispose d'un module d'accès aux données.

EZPDO : cette bibliothèque est basée elle aussi sur PDO. La structure de la base de données doit être décrite dans des zones de commentaires (balise @ORM), selon la norme de documentation PhpDocumentor (cf. chapitre 6, la structuration de l'application).

JDAO : intégré à Jelix, un atelier d'application 100 % objet. JDAO est basé sur une description de la base de données dans un fichier XML.

METASTORAGE : c'est une application qui va générer automatiquement les classes d'accès aux données, à partir d'un schéma XML.

PHPMYOBJECT : il fonctionne en natif avec les bases de données MySQL ou PostgreSQL, et peut s'appuyer sur les pilotes PDO. Il déduit intégralement la structure des tables en interrogeant la base de données (si la structure ne peut être déduite, elle peut être décrite dans une classe).

PROPEL : intégrée à Symphony un framework (squelette d'application - cf. chapitre 8, La conception selon le modèle MVC – les frameworks), elle s'appuie sur un fichier XML pour décrire la base de données, qui peut être généré automatiquement.

ZENDdbTable : ce composant est intégré à Zend Framework, s'appuie sur PDO pour les accès aux données, et dispose de fonctions avancées pour gérer les contraintes au sein de la base de données.

3. la classe OBJETBDD

La classe OBJETBDD[1], étudiée ici, présente l'avantage de s'appuyer sur ADODB pour gérer les accès à la base de données ; elle peut donc, en théorie, être utilisée quel que soit le moteur de base de données utilisé.

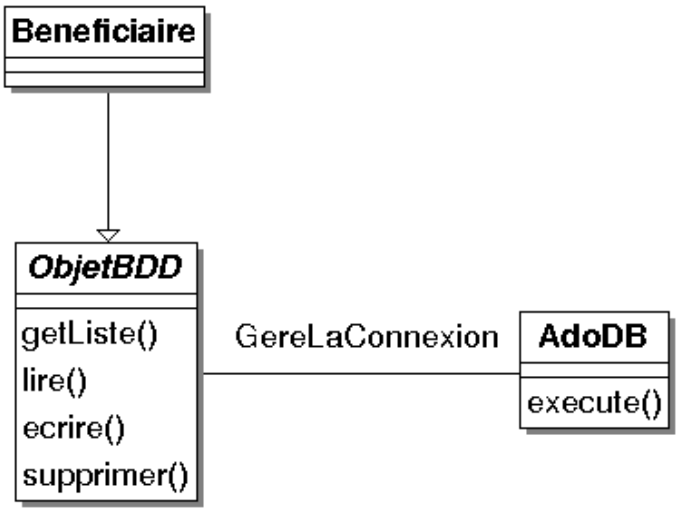
1 http://objetbdd.sourceforge.net

De plus, la plupart des classes ORM s'appuient sur des mécanismes qui récupèrent automatiquement le type des colonnes. Malheureusement, cela ne fonctionne pas avec toutes les bases de données ; ObjetBDD permet de contourner ce problème, mais impose en contrepartie une déclaration manuelle des colonnes qui ne sont pas de type texte pour fonctionner correctement.

Une autre différence tient au mode d'insertion des données. La plupart des ORM utilisent une assignation de variable directe (par exemple, **$maclasse->nom = 'Dupont'**), ObjetBDD travaille avec des tableaux (**$donnees = array('nom'=>'Dupont')**.

Enfin ObjetBDD réalise un certain nombre de contrôles avant l'insertion en base de données, gère automatiquement la transformation des dates depuis le format de la base de données vers l'affichage de l'utilisateur, et encode les caractères spéciaux HTML pour limiter les risques d'attaque par « cross site scripting ».

OBJETBDD est une classe non instanciable, qui doit donc par conséquent être héritée. Elle s'insère ainsi au sein de nos objets :



ObjetBDD utilise une instance ADOConnection pour réaliser les opérations sur la base de données.

L'ensemble des objets qui vont être utilisés pour manipuler les informations dans la base de données vont hériter de ObjetBDD.

exemple de création de la classe Beneficiaire :

Nom colonne	Type
IdBeneficiaire	Integer
NomBeneficiaire	Varchar
PrenomBeneficiaire	Varchar
DateNaissanceBeneficiaire	Date
MelBeneficiaire	varchar

Pour utiliser cette classe :

```
include_once('objetBDD/ObjetBDD.php');
class Beneficiaire extends ObjetBDD {
          function __construct($bdd, $param=NULL){
                    $this->table="Beneficiaire";
```

```
                    $this->colonnes = array(
                    "IdBeneficiaire"=>array("type"=>1,"requis"=>1),
                    "DateNaissanceBeneficiaire"=>array("types"=>2),

          "NomBeneficiaire"=>array("longueur"=>30,"pattern"=>"#^[a-
zA-Z]+$#","requis"=>1),
                    "PrenomBeneficiaire"=>array("longueur"=>20),
                    "MelBeneficiaire"=>array("pattern"=>"#^.+@.+\.
[a-zA-Z]{2,6}$#")
                    );
                  parent::__construct($link, $param);
```

Les différents types d'informations qui peuvent être déclarés pour chaque colonne sont les suivants :

Catégorie	Signification
type	Décrit le type de la colonne : 0 : champ texte (valeur par défaut, il est inutile de les déclarer) 1 : champ numérique 2 : champ date 3 : champ date/heure
longueur	Indique la longueur maximale que doit avoir la valeur avant d'être écrite en base de données
requis	Si vaut 1, le champ ne doit pas être vide
pattern	Contient l'expression régulière qui permet de vérifier la structure du contenu du champ. Dans l'exemple précédent, le pattern fourni pour le champ `MelBeneficiaire` (`#^.+@.+\.[a-zA-Z]{2,6}$#`) permet de s'assurer que l'utilisateur a bien saisi une adresse mel de type : monadresse@monsite.com

Par défaut, OBJETBDD adopte le comportement suivant :

- les identifiants des tables sont considérés comme des clés automatiques, auto-incrémentées ;
- les dates sont transformées sous la forme jj/mm/aaaa en lecture, et retraduites dans l'autre sens pour l'écriture ;
- les données sont systématiquement vérifiées avant d'être écrites ;
- les messages d'exécution SQL ne sont affichés qu'en cas d'erreur ;
- les caractères particuliers HTML (les signes <, >, &, guillemets doubles) sont transformés en leur équivalent HTML (<, $gt;, &, " respectivement) lors de la lecture, et inversement lors des opérations d'écriture.

Les principales méthodes disponibles :

Méthode	Description
lire(int $id)	Lit l'enregistrement `$id` en base de données, et retourne le résultat dans un tableau.
ecrire(array $listeVar)	Ecrit un enregistrement en base de données. Le tableau comprend la liste des colonnes à écrire, dont la clé, qui peut être nulle dans le cas d'une insertion avec clé automatique.
Supprimer(int $id)	Supprime l'enregistrement `$id`
supprimerChamp(int $id, string $colonne)	Supprime un ou plusieurs enregistrements, dont la colonne `$colonne` contient la valeur `$id`
GetListe()	Retourne la liste complète des enregistrements d'une table. Très pratique pour les tables de paramètres (table des civilités,

Méthode	Description
	des statuts...)
getListeParam(string $sql)	Retourne une liste correspondant à la requête SQL fournie

Exemple d'utilisation :

```
include_once('adodb.inc.php');
$bdd = ADONewConnection($BDD_type);
$bdd->Connect($BDD_server, $BDD_login, $BDD_passwd, $BDD_database);
include_once('ObjetBDD.php');
include('beneficiaire.class.php');
$beneficiaire = new Beneficiaire($bdd);
$id = 5;
$data = $beneficiaire->lire($id);
print_r($data);
```

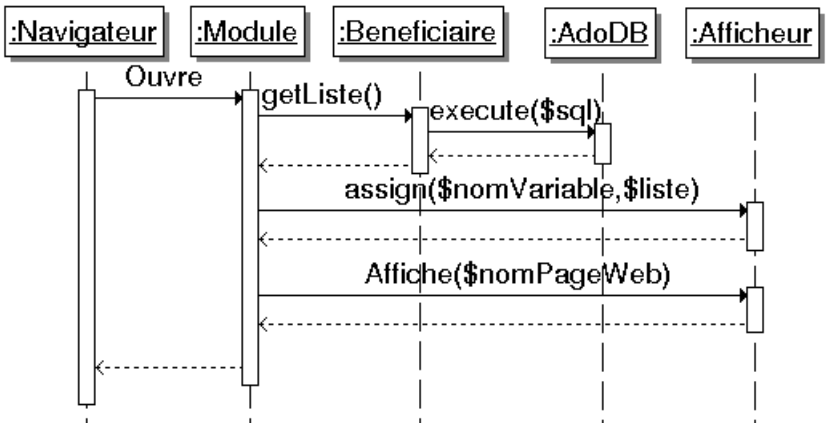
Comment intégrer la classe dans l'application ? En principe, la modification d'une information passe par trois phases :

- d'abord, la liste des enregistrements disponibles est affichée ;
- puis c'est au tour du détail de l'enregistrement qui nous intéresse, qui sera modifié dans un formulaire ;
- et enfin, les modifications sont écrites en base de données.

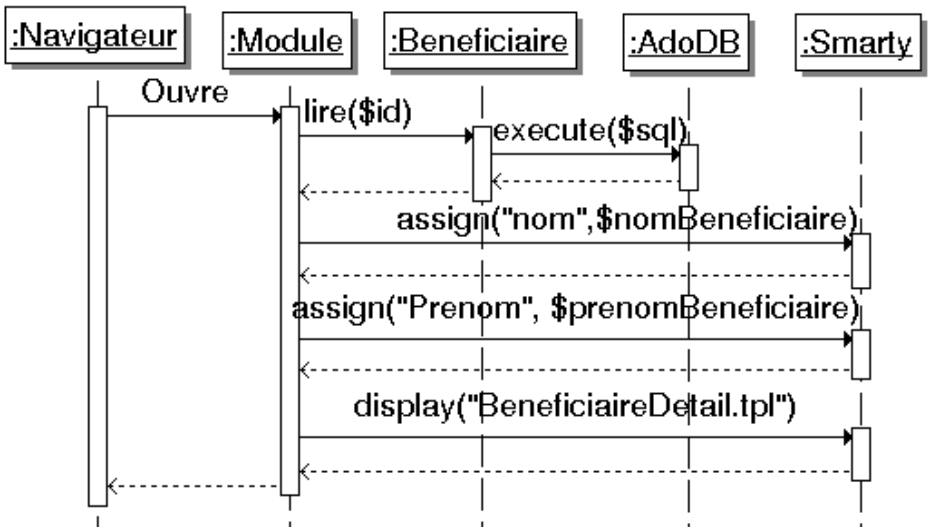
L'affichage de la liste des enregistrements va impliquer les opérations suivantes :

- le navigateur appelle le module PHP à exécuter (la page PHP) ;
- le module crée une instance de notre classe Beneficiaire, et lui demande d'exécuter la fonction getListe() ou getListeParam($sql) ;
- la classe Beneficiaire demande à l'instance de classe ADODB d'exécuter la requête, puis retourne le tableau contenant le résultat de la sélection au module PHP ;
- le module PHP prépare l'affichage ;
- et enfin, il envoie la page au navigateur, pour que l'utilisateur puisse visualiser le résultat.

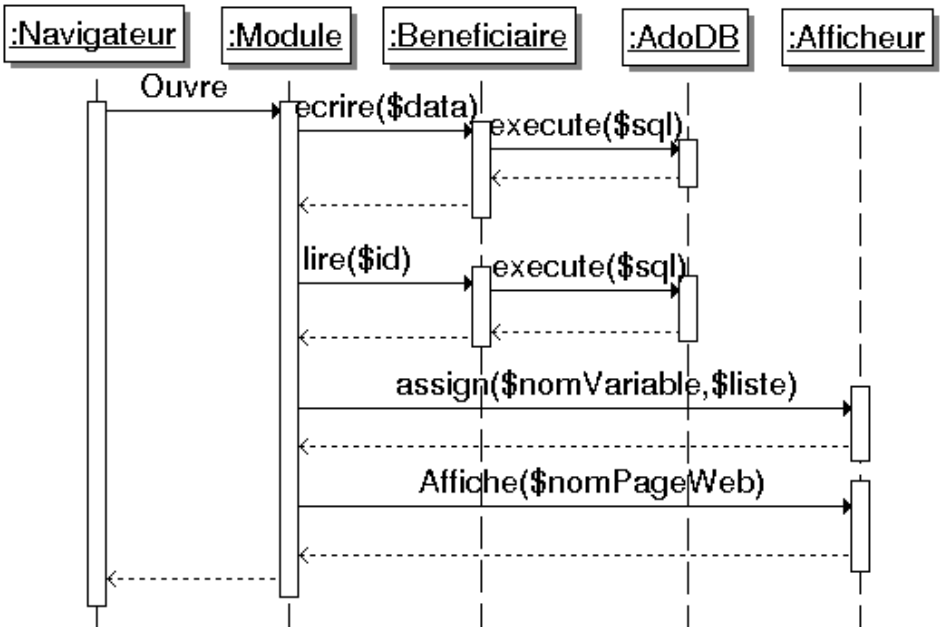
Voici d'abord l'affichage de la liste :



Puis la lecture d'un enregistrement :



Et enfin, la mise en fichier, et le réaffichage de l'information écrite :



III. L'affichage des informations

En PHP, il est possible d'intégrer dans la même page aussi bien du code HTML que JAVASCRIPT ou PHP. En général, les pages deviennent vite illisibles. Pour pallier ce problème, il est préférable de séparer les différentes logiques. Nous allons faire appel aux moteurs de gabarits, ou templates, pour gérer l'affichage, et plus particulièrement au plus connu d'entre eux, SMARTY[1].

Le traitement de l'affichage va être réalisé ainsi :

- le navigateur demande l'exécution d'un script PHP (1) ;
- le script fait appel à différents modules de l'application pour traiter les opérations demandées (appel de classes, lecture d'informations en bases de données...) (2) ;
- chaque fois que le script a besoin d'envoyer des informations à afficher, il fait appel à un module particulier qui a en charge l'affichage (3) ;
- et enfin, une fois l'exécution du script terminée, c'est le module d'affichage qui envoie la page HTML vers le navigateur (4).

Pour reprendre l'affichage de la liste des bénéficiaires, voici comment nous allons pouvoir coder, en utilisant SMARTY :

```
include_once('Smarty-2.6.26/libs/Smarty.class.php');
$SMARTY_template ='templates';
$SMARTY_template_c = 'templates_c';
$SMARTY_config = 'param/configs_smarty';
$SMARTY_cache_dir = 'smarty_cache';
$SMARTY_cache = FALSE;
if (!isset($smarty)){
          $smarty = new Smarty;
          $smarty->template_dir = $SMARTY_template;
          $smarty->compile_dir = $SMARTY_template_c;
          $smarty->config_dir = $SMARTY_config;
          $smarty->cache_dir = $SMARTY_cache_dir;
          $smarty->caching = $SMARTY_cache;
}

include('beneficiaire.class.php');
$beneficiaire = new Beneficiaire($bdd);
$list=$beneficiaire->lire($_REQUEST['id']);
$smarty->assign("nom",$list["Nom"]);
1 $smarty->assign("prenom",$list["Prenom"]);
$smarty->display("beneficiaireSaisie.tpl");
```

La page HTML, qui va permettre d'afficher le nom et le prénom, va ressembler à ceci :

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
15">
<title>Test d'affichage des informations sur le bénéficiaire</title>
</head>
<body>
Nom du bénéficiaire : {$nom}
<br>
Prénom : {$prenom}
</body>
</html>
```

1 http://www.smarty.net/

Les variables et commandes SMARTY sont encadrées par des accolades. Les variables sont précédées du signe $.

Pour afficher la liste des bénéficiaires, le code PHP sera modifié ainsi :

```
$list=$beneficiaire->getListe();
2 $smarty->assign("list",$list);
```

et le gabarit sera adapté, pour afficher la liste dans un tableau. Nous utilisons, pour parcourir l'ensemble du tableau, l'objet **section** :

```
<table id='listeBeneficiaire'>
            <tr>
                        <th>Nom du Bénéficiaire</th>
                        <th>Prénom</th>
            </tr>
{section name=boucle loop=$list}
            <tr>
                        <td>{$list[boucle].nom}</td>
                        <td>{$list[boucle].prenom}</td>
            </tr>
{/section}
</table>
```

Chaque item de notre tableau $list est identifié par $list[**boucle**], **boucle** étant le nom de notre section (attribut **name**), et correspond à l'occurrence courante.

Les différentes fonctions utilisables dans les gabarits :

Fonction	Rôle
`{if}`, `{elseif}`, `{else}`	Permet de réaliser des tests.
`{foreach}`, `{foreachelse}`	Permet de parcourir des tableaux associatifs simples. Cette fonction est en général peu utilisée ; on lui préfère la fonction `{section}`, qui est plus riche.
`{section}`, `{sectionelse}`	Permet de parcourir des tableaux associatifs complexes (à plusieurs dimensions). C'est l'outil par excellence de manipulation des tableaux.
`{include}`	Permet d'intégrer un autre gabarit dans le gabarit courant. Très pratique pour améliorer les mises en pages : il est possible, par exemple, de gérer les entêtes et les pieds de page sur des gabarits différents.
`{ldelim}`, `{rdelim}`	Utilisé pour « échapper » les caractères **{** et **}** respectivement. C'est une fonction indispensable quand il faut insérer du javascript directement dans le gabarit (les caractères { et } sont utilisés en javascript pour définir les fonctions). Dans la pratique, pour des questions de lisibilité, il est préférable de stocker le javascript dans des fichiers de code séparés.
`{literal}`	Même fonctionnement que précédemment, mais c'est l'ensemble du bloc encadré par les balises `{literal}` et `{/literal}` qui ne sera pas interprété par Smarty (et pas simplement les accolades qui seront échappées).
`{strip}`	C'est une fonction qui permet de supprimer tous les retours chariots, les tabulations, les espaces redondants, entre les deux balises `{strip}` et `{/strip}` Elle est utilisée quand des retours chariot sont insérés dans le code du gabarit, par exemple dans les balises HTML, pour en simplifier la lecture. C'est très pratique à utiliser quand le code des balises est très long.

Fonction	Rôle
	Attention toutefois à ne pas séparer deux termes d'une balise html par un retour chariot ou une tabulation (par exemple, `<option value...>`), sinon les deux termes seront accolés. Autrement dit, un retour chariot ne pas être employé à la place d'un espace.
`{html_checkboxes}`	Permet de générer directement une sélection type « cases à cocher » à partir de trois tableaux, le premier contenant les identifiants, le second les libellés à afficher, et le troisième les items sélectionnés par défaut. Il est également possible de remplacer les deux premiers tableaux par un tableau associatif.
`{html_options}`	Même fonctionnement que pour la fonction précédente, sauf que c'est une liste de sélection qui est générée.
`{html_radios}`	Même chose que précédemment, mais avec la génération de boutons radios.
`{mailto}`	Fonction permettant d'encoder l'adresse mel qui s'affiche sur l'écran, pour limiter leur collecte par les spammers.

Toutes les balises de type test, encadrement de bloc, gestion de boucle... doivent être fermées par des balises de fermeture. Ainsi, la balise {if} doit être fermée par {/if}.

Les fonctions Smarty peuvent être insérées n'importe où dans le gabarit, y compris au sein de balises HTML, y compris à l'intérieur de texte entre quottes.

IV. La sécurité

La sécurisation d'une application va s'appuyer sur plusieurs briques :

- l'identification : le login de l'utilisateur est vérifié ;
- l'habilitation : l'utilisateur doit disposer des droits adéquats pour accéder à un module, ou à un sous-ensemble de données définies, par exemple, le chiffre d'affaires de son service ;
- la cohérence : les données introduites dans le système doivent être conformes à ce qu'on attend ;
- et enfin, la sécurisation générale : la mise en œuvre de mécanismes adaptés rendront les attaques plus difficiles à mener.

A. Être en règle vis à vis de la loi

1. La CNIL

Tout développeur, avant de concevoir son application, doit vérifier si celle-ci est sujette à déclaration auprès de la CNIL. En effet, le fait de manipuler des informations personnelles, comme le nom ou l'adresse d'une personne, implique un certain nombre de contraintes, comme le droit de rectification et la non divulgation des informations à des tiers non autorisés. De même, le stockage de certaines catégories d'informations peut être refusé, comme celles afférentes à la religion, aux opinions...

Certains traitements informatisés sont exempts de déclaration, comme la comptabilité ou la gestion des fournisseurs. D'autres sont soumis à une déclaration simplifiée, comme la création d'un annuaire téléphonique au sein de l'entreprise ou la mise en place d'un fichier des prospects. Enfin, tous ceux qui ne rentrent pas dans les catégories pré-établies sont soumis à une déclaration normale. Celle-ci doit décrire dans le détail toutes les catégories d'informations manipulées, les échanges avec des tiers, les mesures de sécurité prises...

Le site de la CNIL (http://www.cnil.fr) propose un outil qui permet de savoir dans quel cas le projet se situe, et de préparer sa déclaration en quelques clics.

2. Les licences d'utilisation des logiciels

Lors de l'écriture d'un logiciel, il est fréquent d'utiliser des composants fournis par des développeurs tiers. Néanmoins, ces composants sont disponibles selon des conditions d'utilisation qui peuvent être contraignantes, sujets à royalties, ou qui peuvent conditionner l'usage qui sera fait du nouveau logiciel conçu.

Sur Internet, il existe pléthore de licences d'utilisation. Les deux plus fréquentes sont la licence GNU GPL (GNU General Public licence) et la licence GNU LGPL (GNU Lesser General Public licence). On trouve également la licence BSD, une des moins restrictives, et les licences CECILL, qui sont une transposition dans le droit français des licences GPL, LGPL et BSD.

La licence GPL (aujourd'hui en version 3) autorise tout utilisateur à recevoir une copie du logiciel publié sous celle-ci. Il est possible de modifier le logiciel, à condition que toutes les modifications, dès lors qu'elles ne sont pas conservées pour un usage privé, soient également publiées sous cette même licence. Elle autorise également l'usage commercial du logiciel. En principe, un logiciel qui inclut un composant GPL doit être publié également sous licence GPL. Ce n'est

pas le cas d'un composant sous licence BSD, qui peut être intégré à un logiciel non libre.

La licence LGPL est moins contraignante (*Lesser* signifie moindre en anglais) : le composant fourni selon celle-ci peut être associé à du code qui n'est pas publié sous cette licence ; l'aspect héritage de la licence GPL n'est plus obligatoire. La seule obligation consiste à publier les modifications du composant sous licence LGPL.

Les licences CeCill ont été créées à l'initiative du CEA, du CNRS et de l'INRIA, pour que les organismes publics puissent publier les logiciels qu'ils ont conçus. Il existe 3 licences CeCill : la licence CECILL-A, compatible avec la licence GPL, la licence CECILL-B, compatible avec la licence BSD, et la licence CECILL-C, destinée aux composants, est compatible avec la licence LGPL.

Certains composants sont fournis selon des licences particulières. Par exemple, certaines bibliothèques graphiques autorisent l'usage gratuit pour des applications destinées au projets humanitaires ou scolaires, mais payant pour tous les autres usages ; il est indispensable, avant d'utiliser un composant, de s'assurer des termes de la licence et de l'usage qui peut en être fait.

Enfin, si le logiciel que vous allez concevoir est destiné à être publié ou transmis à des tiers, pensez également à définir quelles sont les règles d'utilisation que vous souhaitez imposer en choisissant la licence qui vous semble adaptée.

Pour télécharger le texte des licences :

GPL : http://www.gnu.org/licenses/gpl.html, et sa traduction non officielle en français : http://www.april.org/files/groupes/trad-gpl/doc/GPLv3/www.rodage.org/gpl-3.0.fr.html

LGPL : http://www.gnu.org/licenses/lgpl.html et sa traduction non officielle en français : http://www.vadeker.net/humanite/copyleft/lgpl.html

CECILL : http://www.cecill.info/

BSD : http://www.freebsd.org/copyright/license.html

B. L'identification

Le login, porte d'entrée de l'utilisateur dans l'application, est associé en général à un mot de passe. L'utilisateur va être confronté, dans la majorité des cas, soit lors de l'ouverture de l'application, soit lorsqu'il souhaitera accéder à des informations dont l'accès est protégé, à un écran lui demandant de saisir son login et un mot de passe associé ; il existe néanmoins des identifications basées sur des mécanismes plus complexes, par exemple en utilisant des certificats numériques, ou en demandant à l'utilisateur de saisir des codes complémentaires, lus dans une grille fournie au préalable à l'utilisateur.

Aujourd'hui, trois grands modes d'identification sont utilisés :

- la base de données contient une table Login, dont un des champs correspond au mot de passe utilisé ;
- l'identification des utilisateurs est réalisée à partir d'une base de données centrale, et peut être réutilisée entre différentes applications. En entreprise, celle-ci s'appuie en général sur un annuaire, soit Active Directory de Microsoft, soit à la norme LDAP (Lightweight Directory Access Protocol). A noter que l'annuaire Active Directory est compatible

LDAP (hormis la toute première version) : nous ne traiterons ici que le cas LDAP ;

- l'identification des utilisateurs est sous-traitée à une application dédiée à cet effet, un serveur CAS (Central Authentication Service) – service d'authentification centralisé), qui va traiter de manière unique toutes les demandes de vérification de login. C'est en général la méthode la plus fiable et la plus sécurisée pour gérer l'identification dans les applications.

1. L'identification basée sur le login stocké en base de données

Une table Login :

Champ	Type
id	Int
login	varchar(32)
password	varchar(256)

la classe Login, basée sur ObjetBDD :

```
class Login extends ObjetBDD
{
          function Login($bdd)
          {
                    $this->table="Login";
                    ObjetBDD::ObjetBDD($bdd);
                    $this->id_auto = 1;
                    $this->types=array(
                    'id'=>1);
          }
          function verifLogin($login,$password){
                    $password = hash("sha256",$password);
                    $sql='select login from LoginGestion where login
="'.$login
                    .'" and password = "'.$password.'"';
                    $res=ObjetBDD::lireParam($sql);
                    if ($res["login"]==$login) {
                              return TRUE;
                    }else{
                              return FALSE;
                    }
          }
}
```

Il manque à cette classe l'enregistrement du login et du mot de passe. Le mot de passe doit être impérativement crypté, pour éviter qu'il puisse être consulté par qui que ce soit, administrateur ou pirate. Eviter le codage MD5, obsolète, et lui préférer sha256, whirlpool ou RIPEMD-160.

Son implémentation dans l'application :

```
$login = new Login($instance_adodb);
$res=$login->verifLogin($_REQUEST['login'],$_REQUEST['password']);
if ($res==TRUE){
          $_SESSION["login"] = $_REQUEST["login"];
}
```

2. L'identification basée sur un annuaire LDAP

La mise en place d'une identification basée sur l'annuaire LDAP est assez simple à réaliser. Elle vise à vérifier qu'il est possible de se connecter à l'annuaire en utilisant le couple login/mot de passe.

La classe LoginLdap :

```
class LoginLdap {
          function __construct($LDAP_address, $LDAP_port,
$LDAP_basedn, $LDAP_user_attrib,$LDAP_v3,$LDAP_tls) {
                    $this->LDAP_address = $LDAP_address;
- le port  (par défaut, 389) :
                    $this->LDAP_port = $LDAP_port;
                    $this->LDAP_basedn = $LDAP_basedn;
                    $this->LDAP_user_attrib = $LDAP_user_attrib;
                    $this->LDAP_v3 = $LDAP_v3;
                    $this->LDAP_tls = $LDAP_tls;
          }
                    function verifLogin($login,$password){
          $ldap = @ldap_connect($this->LDAP_address,$this->LDAP_port)
                    or die("Impossible de se connecter au serveur
LDAP.");
          if ($this->LDAP_v3)
                    {
                              ldap_set_option($ldap,
LDAP_OPT_PROTOCOL_VERSION, 3);
                    }
                    if ($this->LDAP_tls)
                    {
                              ldap_start_tls($ldap);
                    }
                    $dn = $this->LDAP_user_attrib."=".$login.",".
$this->LDAP_basedn;
                    $rep=ldap_bind($ldap,$dn, $password);
                    if ($rep == 1)
                    {
                              return true;
                    }else{
                              return false;
                    }
          }
}
```

Et un script qui permet d'utiliser la classe :

```
$LDAP_address = "localhost";
$LDAP_port = 389;
$LDAP_rdn = "cn=manager,dc=example,dc=com";
$LDAP_basedn = "ou=people,cn=manager,dc=example,dc=com";
$LDAP_user_attrib = "uid";
$LDAP_v3 = true;
$LDAP_tls = false;
$login = new LoginLdap($LDAP_address, $LDAP_port,$LDAP_basedn,
$LDAP_user_attrib,$LDAP_v3,$LDAP_tls);
$res=$login->verifLogin($_REQUEST['login'],$_REQUEST['password']);
if ($res==TRUE){
          $_SESSION["login"] = $_REQUEST["login"];
}
```

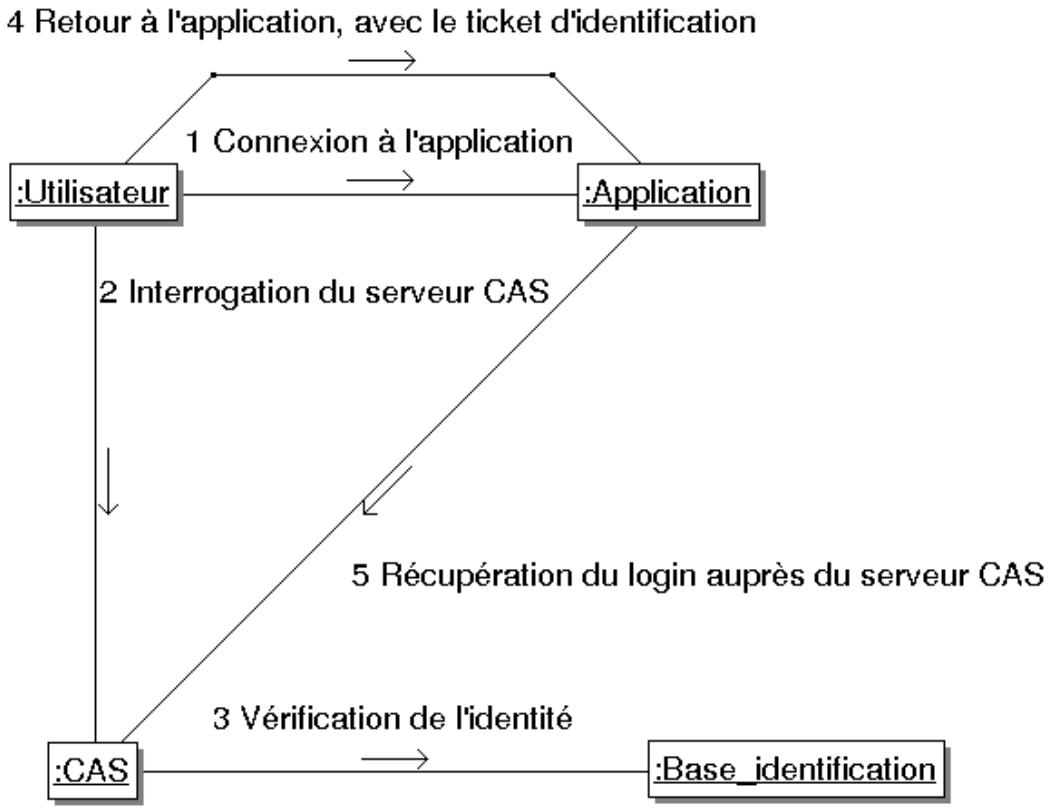
Avec cette approche, et si l'annuaire LDAP est celui utilisé pour l'identification dans le domaine Windows (Active Directory Service – ADS - ou domaine Samba/ldap), l'utilisateur n'aura pas besoin de retenir un nouvel identifiant.

3. L'identification basée sur un serveur d'identification CAS

Les deux méthodes d'identification que nous venons d'étudier, l'identification directe et l'identification via un annuaire LDAP, présentent deux écueils. Le premier, c'est que le mot de passe arrive en clair sur le serveur hébergeant l'application : un développeur peu scrupuleux peut mettre en place un « ramasse-miettes », qui va enregistrer dans un fichier à part tous les couples login/mot de passe, pour les réutiliser par la suite pour son compte personnel. Le second, c'est que si l'utilisateur doit travailler avec plusieurs applications, il est obligé de retaper son mot de passe chaque fois qu'il change de programme.

Avec l'avènement des applications web, un nouveau concept est apparu, le SSO (Single Sign On – identification unique). Son principe est simple : la vérification de l'identité de l'utilisateur est sous-traitée à une application particulière. Si l'utilisateur s'est déjà connecté à une des applications utilisant ce SSO, celui-ci fournit directement le login utilisé : l'utilisateur n'a plus besoin de retaper son mot de passe.

a) La première identification :



1 : l'utilisateur se connecte à l'application

2 : l'application a besoin d'identifier l'utilisateur : elle redirige le navigateur vers le serveur CAS, qui va être chargé de gérer l'identification. Lors de cette phase, le serveur CAS affiche un écran de saisie du login / mot de passe, en mode sécurisé via le protocole https.

3 : Au retour de la saisie du login/mot de passe, le serveur CAS vérifie l'identification auprès de sa base locale (annuaire LDAP, base de données...). Il retourne au navigateur un cookie de session, spécifique au serveur CAS ; l'échange de ce cookie ne se fait qu'en mode sécurisé (https).

4 : le serveur CAS redirige le navigateur de l'utilisateur vers l'application (paramètre passé lors de la phase 2), en lui joignant un ticket de session, utilisable une seule fois.

5 : l'application interroge le serveur CAS, en lui fournissant le ticket de session, et récupère, en retour, l'identifiant (le login) de l'utilisateur.

b) Les identifications suivantes :

Une fois que l'utilisateur s'est identifié une première fois, son navigateur contient un cookie provenant du serveur CAS.

Lorsqu'une nouvelle application a besoin de connaître le login de l'utilisateur, elle va, elle aussi, rediriger le navigateur vers le serveur CAS (début de la phase 2). Le serveur CAS détecte sur le navigateur un cookie (celui qui a été téléchargé lors de la première identification). Il le vérifie (il expire au bout d'un certain temps, de quelques minutes à quelques heures selon la politique de sécurité mise en place) : s'il est toujours valide, plutôt que d'afficher la page de saisie du login/mot de passe, il redirige directement le navigateur vers l'application en lui transmettant un nouveau ticket de session (phase 4). L'application n'aura plus qu'à récupérer le login (phase 5).

c) Le codage en PHP :

Nous allons utiliser la classe phpCAS, téléchargeable sur le site http://www.ja-sig.org/wiki/display/CASC/phpCAS.

Les informations à fournir sont les suivantes :

$CAS_address : adresse web du serveur CAS

$CAS_port : port d'interrogation (en général, 443, c'est le port utilisé par défaut par le protocole https)

$CAS_uri : chemin d'accès dans le serveur (par défaut, vide).

```
if (!isset($_SESSION["login"])) {
          include_once('esup-phpcas/source/CAS/CAS.php');
          phpCAS::client(CAS_VERSION_2_0,$CAS_address,$CAS_port,
$CAS_uri);
          phpCAS::forceAuthentication();
          $auth = phpCAS::getUser();
          $_SESSION["login"] = $auth;
}
```

Le login est stocké dans la variable de session $_SESSION["login"].

Deux mécanismes sont ainsi mis en place pour gérer l'identification de l'utilisateur :

- le login est stocké en variable de session. Tant que celle-ci n'a pas expiré, l'utilisateur est considéré comme connu ;
- si la session a expirée, ou s'il s'agit de la première connexion, le serveur CAS est interrogé pour connaître le login. Si l'utilisateur s'est déjà connecté sur le serveur CAS, ce dernier transmettra directement

l'information à notre application, sans repasser par une phase d'identification.

d) La gestion de la déconnexion avec phpCAS

Si l'utilisateur souhaite se déconnecter, il faut également en avertir le serveur CAS, qui doit révoquer son cookie d'identification.

L'opération va se dérouler en deux temps : d'abord, la suppression de la session, puis la déconnexion auprès du serveur CAS :

```
// Destruction de toutes les variables de session
$_SESSION = array();
// On efface également le cookie de session.
if (isset($_COOKIE[session_name()])) {
          setcookie(session_name(), '', time()-42000, '/');
}
// Finalement, on détruit la session.
session_destroy();
3 phpCAS::client(CAS_VERSION_2_0,$CAS_address,$CAS_port,$CAS_uri);
4 phpCAS::logout($adresse_retour);
```

C. Chiffrer la saisie du login/mot de passe

Quoi de plus facile que de capturer le login/mot de passe sur un réseau ! Un simple analyseur des trames IP (les messages échangés entre les ordinateurs) permet à toute personne mal intentionnée de lire en clair les mots de passe échangés entre le navigateur et le serveur.

Pour éviter ce problème, il n'y a qu'une solution : le chiffrement. Ce chiffrement peut être limité à la saisie du login/mot de passe, si les données saisies dans l'application n'ont pas de caractère confidentiel, ou étendu à l'ensemble de l'application.

Pourquoi mettre en place du https systématiquement dans les applications de gestion ?

le chiffrement comprend deux phases : l'initialisation de la connexion, qui se fait par échange de clés asymétriques, et qui est très gourmande en ressources, puis le reste de l'échange est effectué par des clés symétriques, plus efficaces. En principe, l'échange des clés asymétriques ne s'opère qu'à l'initialisation de la connexion SSL, et ne sera pas rejouée pour le reste de la session.

Google, en 2010, a activé par défaut le https pour tous les comptes GMAIL. Il considère que le TLS-SSL consomme moins de 1 % de la charge CPU, moins de 10 Ko par connexion, et 2 % de la charge réseau[1].

Enfin, les données manipulées dans une application de gestion peuvent avoir une valeur juridique importante. Ainsi, la divulgation d'un relevé d'identité bancaire (RIB) ou de données personnelles pourrait être reproché au gestionnaire de l'application s'il n'a pas pris les mesures nécessaires pour éviter que ces données ne puissent être récupérées facilement.

Il n'y a donc, aujourd'hui, plus de raison valable pour refuser de chiffrer son application en https.

1 Données extraites d'un document rédigé par Verisign, *SSL : la solution contre les attaques par Firesheep et les détournements de session HTTP*, téléchargeable depuis le site http://www.lemondeinformatique.fr/livre-blanc/ssl-la-solution-contre-les-attaques-par-firesheep-et-les-detournements-de-session-http-1093.html

Le chiffrement est mis en œuvre systématiquement dans les serveurs CAS. Il est également facile de l'activer sur un serveur web ; voici la procédure à suivre pour un serveur Apache (http://www.apache.org/) :

Dans le dossier où l'on souhaite activer le chiffrement, il faut créer un fichier `.htaccess`, qui comprend les lignes suivantes :

```
RewriteEngine on
RewriteCond %{REMOTE_PORT}  !^443$
RewriteRule ^/(.*) https://%{HTTP_HOST}/%{REQUEST_URI}
```

Pour que la redirection fonctionne, il faut impérativement que le serveur Apache l'autorise ! Sur un intranet, son fichier de configuration (**httpd.conf** ou **apache2.conf**, selon les distributions Linux), doit contenir la directive suivante :

```
<Directory />
Allowoverride All
</Directory>
```

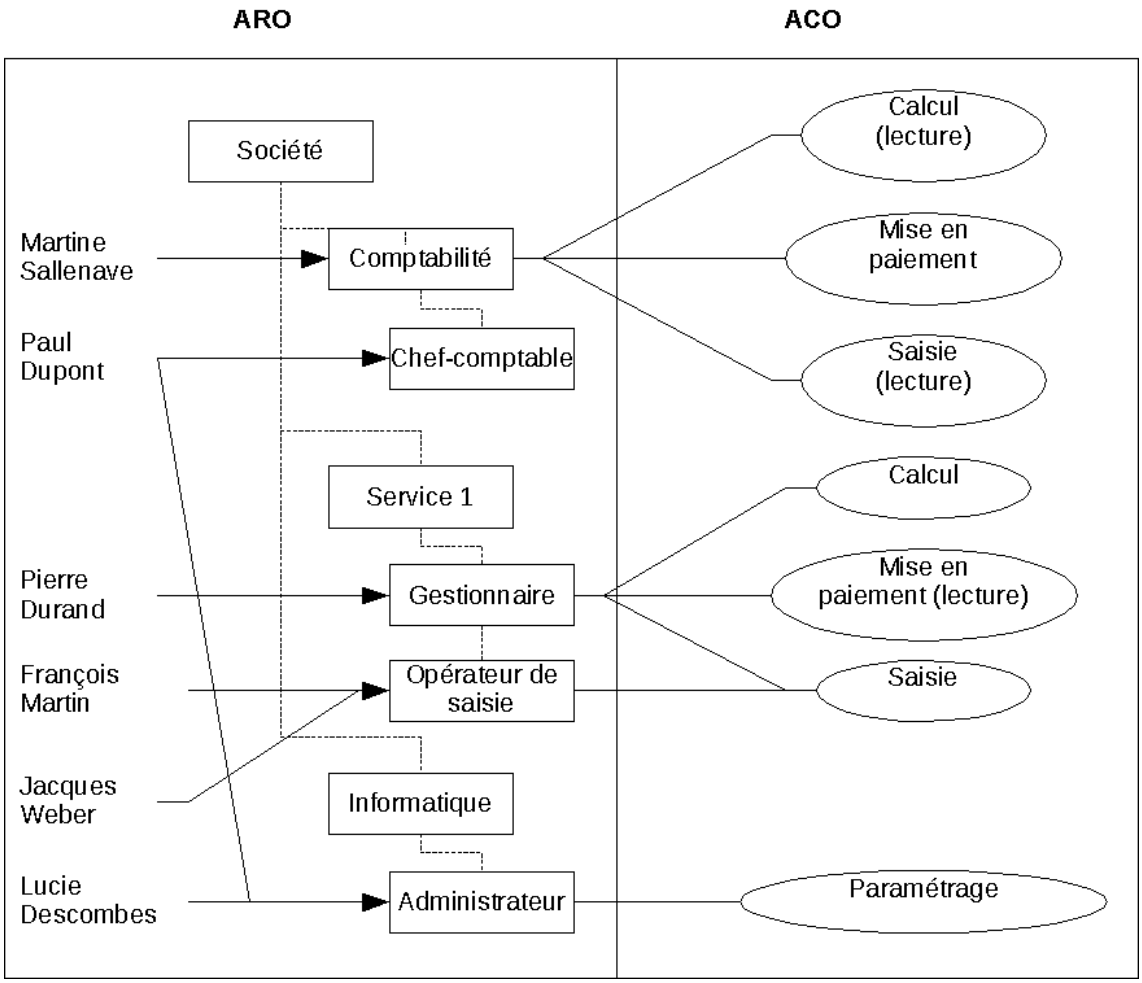
Dans le cas où il est souhaitable que la communication soit chiffrée pour l'ensemble de l'application, d'autres méthodes peuvent être utilisées pour l'implémenter (utilisation des *vhosts* notamment).

D. La gestion des droits avec PHPGACL

PhpGACL[1] est un projet de gestion des droits qui s'articule autour de quelques classes, et qui est probablement un des outils les plus complets disponibles en PHP. Il est utilisé dans plusieurs applications, dont **dotProject**[2], une application de gestion de projet en mode web qui a été intégrée dans Sourceforge, la forge la plus utilisée pour les projets OpenSource sur le web, et qui permet de générer des diagrammes de Gantt, de gérer les tâches de chaque participant..., et **Mambo**[3], un logiciel de gestion de contenu de site web. Il gère trois types d'objets principaux :

- les **ACO** (*Access Control Object*) : ce sont les objets qui vont permettre de contrôler les accès aux différents modules de l'application ;
- les **ARO** (*Access Request Object*) : ce sont les objets qui vont chercher à accéder aux modules de l'application, dans la pratique, les logins regroupés en groupes ;
- les **ACL**, (Access Common List) contiennent les droits d'accès. Ils vont faire la relation entre les ACO et les ARO. A noter qu'ils peuvent être positifs (j'ai le droit d'accéder à tel module) ou négatifs : j'ai le droit d'accéder à tel ensemble de modules, sauf à un en particulier. Cela permet de définir, par exemple, que toutes les personnes de tel service ont accès à tel module (droit positif pour le service), sauf les opérateurs de saisie (droit négatif pour ce groupe particulier).

1 http://phpgacl.sourceforge.net/
2 http://www.dotproject.net/
3 http://mambo-foundation.org/



Les ARO contiennent 2 objets : les **logins**, et les **groupes**. Dans cet exemple, François Martin est opérateur de saisie, Pierre Durand gestionnaire. Paul Dupont, chef-comptable, hérite des droits attribués au groupe comptabilité ; il est également administrateur informatique, et pourra ainsi accéder aux modules de paramétrage.

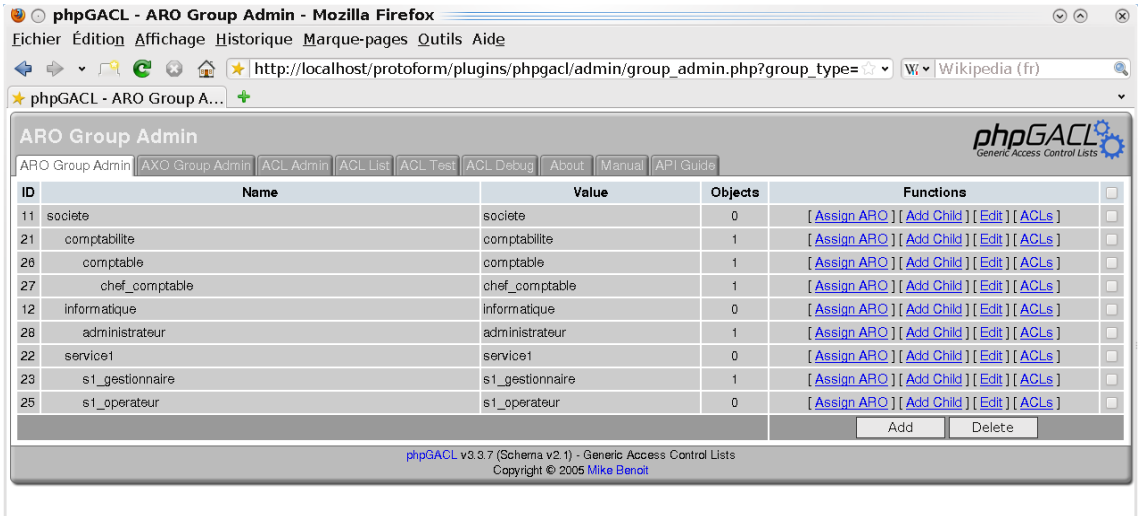
Il faut noter également la gestion arborescente des ARO : les groupes les plus à droite (de niveau plus élevé) comprendront également les droits attribués dans les groupes de niveau inférieur.

Les ACO (dans les ovales) correspondent aux droits qui seront testés dans l'application. Ainsi, pour accéder en écriture au module de Calcul, e programme vérifiera que le login de l'utilisateur a bien les droits attribués à l'ACO Calcul. La relation entre les groupes ARO et les ACO correspond aux ACL.

Le schéma a été volontairement simplifié pour en faciliter la lecture : le groupe Gestionnaire doit avoir les droits également pour Calcul (lecture) et Saisie (lecture), le groupe Opérateur de saisie sur Saisie (lecture), etc.

1. La saisie des ARO

Les groupes sont organisés de manière hiérarchique. Il ne peut y avoir qu'un seul groupe de niveau 0 (ici, la société).



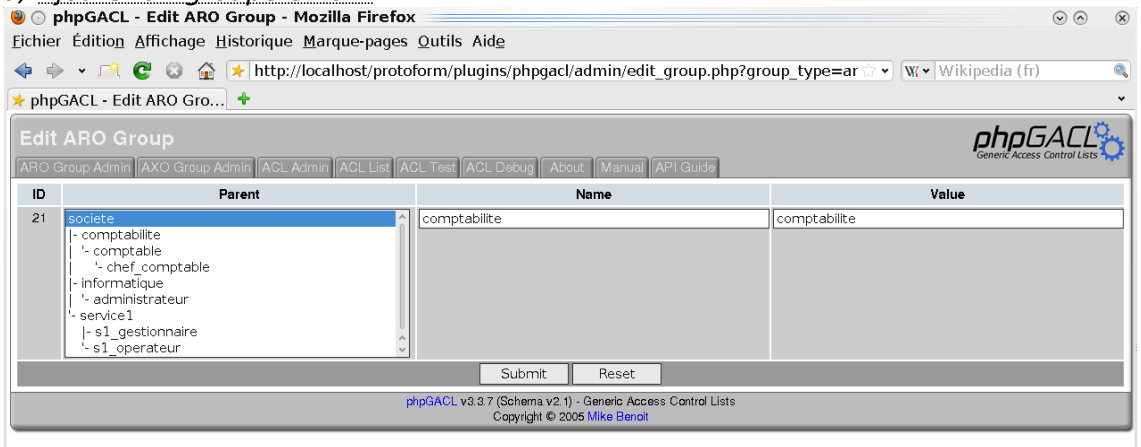
Il faut faire également attention à la notion de groupe enfant (`child`) : d'un point de vue des droits de l'application, ils ont un poids plus fort que les groupes parents. Ainsi, un `login` présent dans le groupe `chef_comptable` disposera également des droits attribués aux groupes `comptable`, `comptabilité` et `société`.

La colonne `Name` correspond au nom du groupe tel qu'il apparaîtra dans les autres écrans de l'application : c'est une zone de texte libre. La colonne `Value` correspond à la valeur que l'on utilise effectivement : elle doit être unique dans le référentiel des droits. Quant à la colonne *`Objects`*, elle affiche le nombre de logins présents dans chaque groupe.

Pour chaque groupe, plusieurs actions sont possibles :

- **Assign ARO** : c'est l'opération qui va permettre d'attribuer les logins au groupe ;
- **Add child** : cela va permettre de créer un sous-groupe ;
- **Edit** : permet de modifier les libellés du groupe ;
- **Acl** permet de visualiser les droits attribués au groupe.

a) Ajouter un groupe enfant

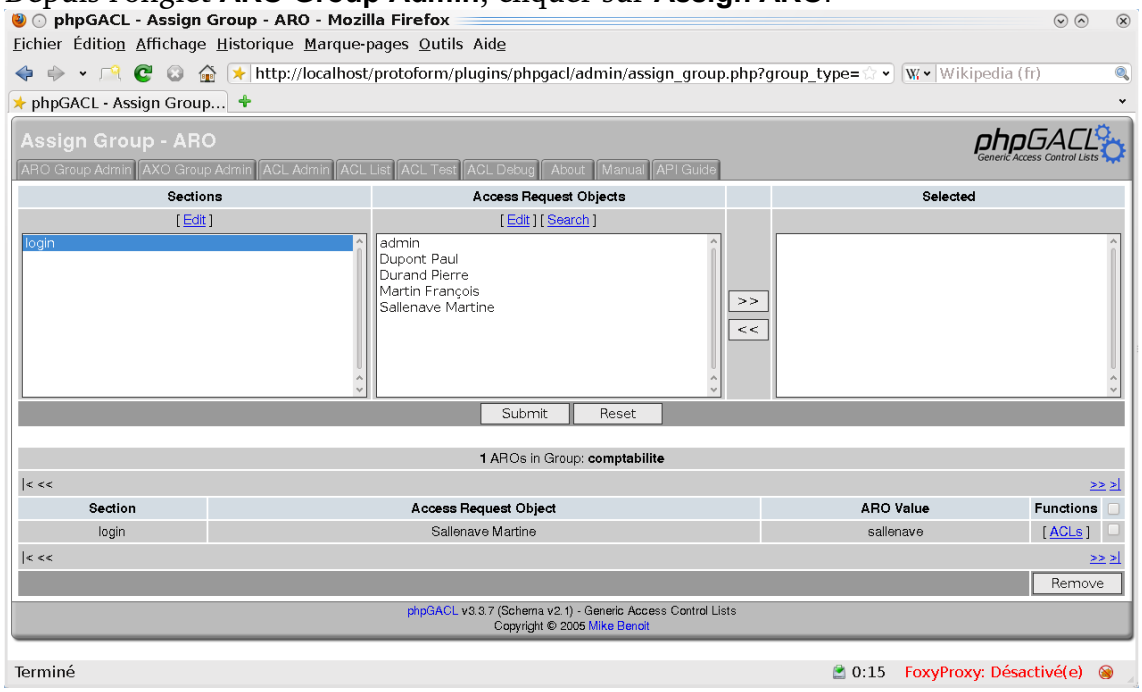


En cliquant sur **Add child**, un nouveau groupe enfant va pouvoir être saisi (l'opération **Edit** arrive au même écran, mais en mode modification).

Il suffit de renseigner les zones **Name** et **Value**. Dans la pratique, il est conseillé de mettre les mêmes valeurs.

b) Rattacher un login à un groupe

Depuis l'onglet **ARO Group Admin**, cliquer sur **Assign ARO**.

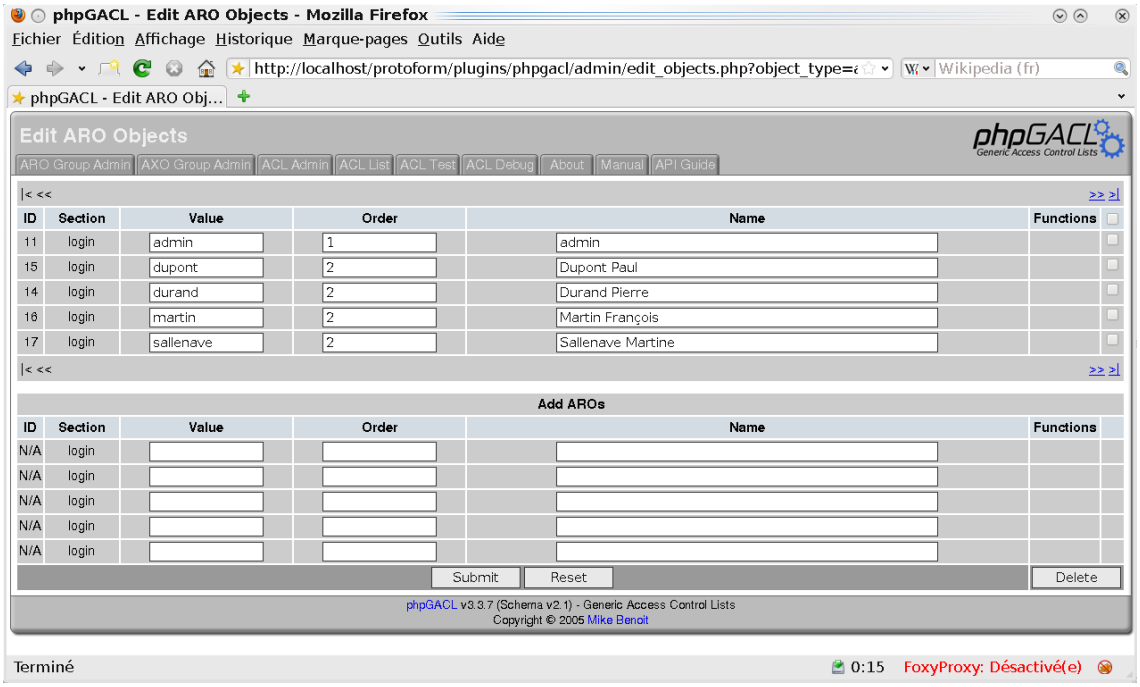


Dans phpGACL, les logins sont regroupés en sections : il est ainsi possible de gérer plusieurs catégories de logins, si l'application le nécessite. Dans la pratique, il vaut mieux se limiter à une seule section, ce qui sera nettement plus simple.

Dans la boite Sections, cliquez sur login (s'il n'est pas déjà sélectionné).

La boite Access Request Objects affiche la liste des logins existants.

Pour créer un nouveau login, cliquez sur Edit dans la boite Access Request Objects.



La partie haute permet de visualiser, voire de modifier, les logins existants. La partie basse permet de rajouter de nouveaux logins (5 maximum à la fois). Pour chaque login, il faut indiquer :

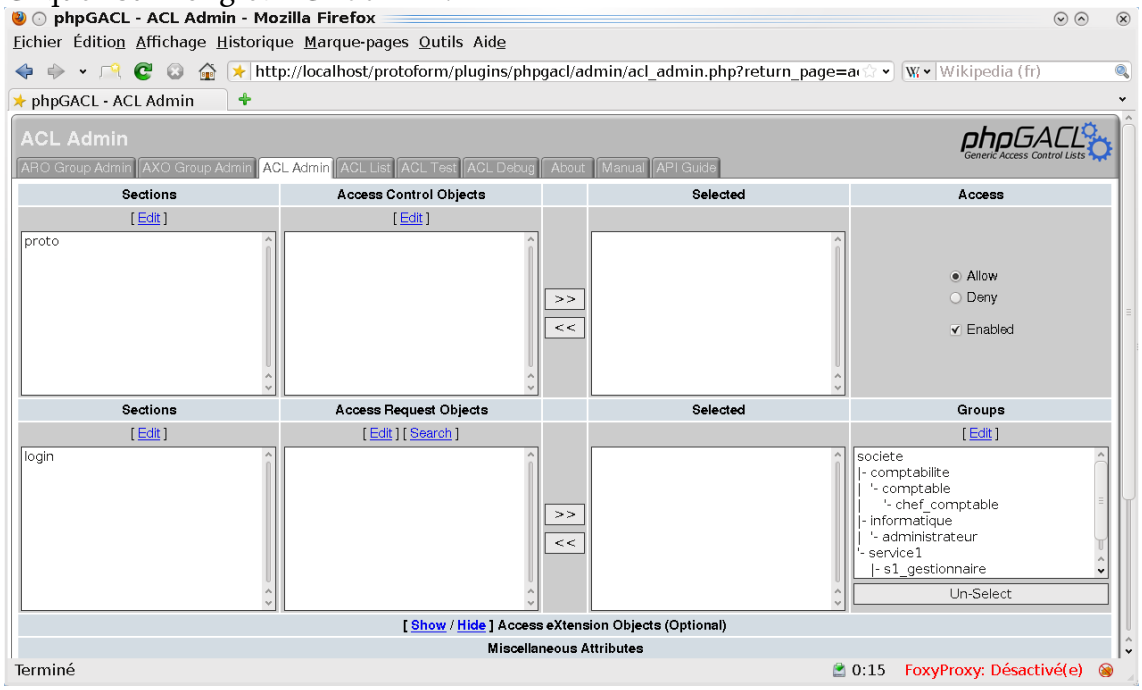
- value : le login qui sera effectivement testé dans l'application ;
- Order : l'ordre de tri d'affichage des logins dans l'interface phpGACL ;
- Name : un libellé plus facilement manipulable. Il est fortement conseillé de mettre d'abord le nom de la personne, puis son prénom (classement alphabétique) : avec plusieurs dizaines ou plusieurs centaines de logins, la recherche n'en sera que plus aisée.

Il est possible également de supprimer un login, en cochant la case située à droite de l'écran, puis en cliquant sur le bouton Delete.

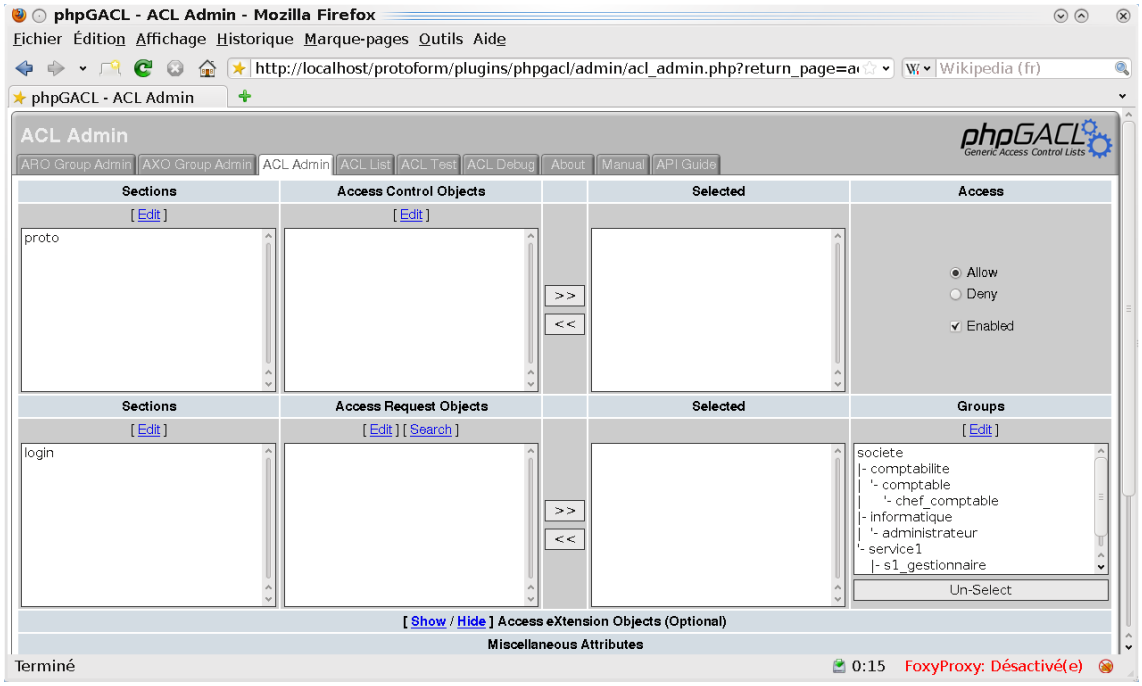
2. La création des ACO

Maintenant que les logins sont créés et organisés par groupes, nous allons pouvoir définir les objets de contrôle, puis attribuer les droits (ACL).

Cliquez sur l'onglet ACL admin :



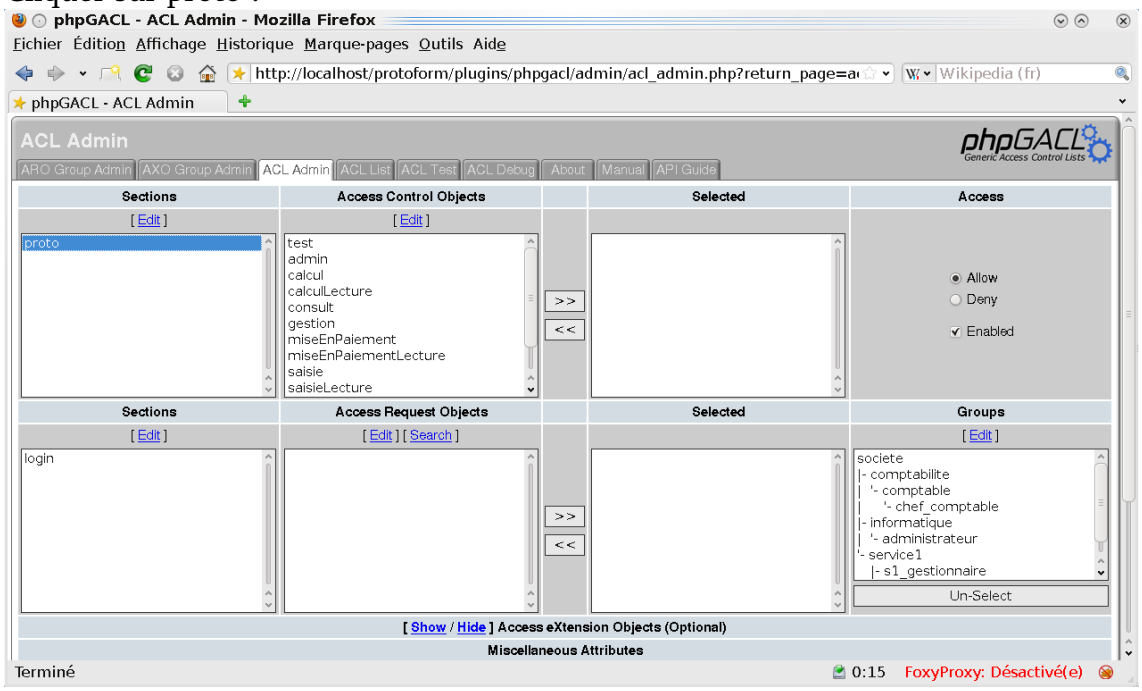
Nous accédons à l'écran de manipulation des ACO et des ACL :



Nous allons d'abord nous intéresser aux deux premières boites, en haut à gauche : Sections et Access Control Objects. Ces deux boites vont permettre de définir les droits de l'application.

La première boite, Sections, va permettre de définir des sous-ensembles d'ACO. C'est une possibilité qui sera utilisée pour mettre en œuvre une gestion unique des droits au niveau de l'entreprise : dans cette boite, un item sera créé par application. Dans l'exemple ci-dessus, nous n'avons qu'une application, appelée proto.

Cliquer sur proto :



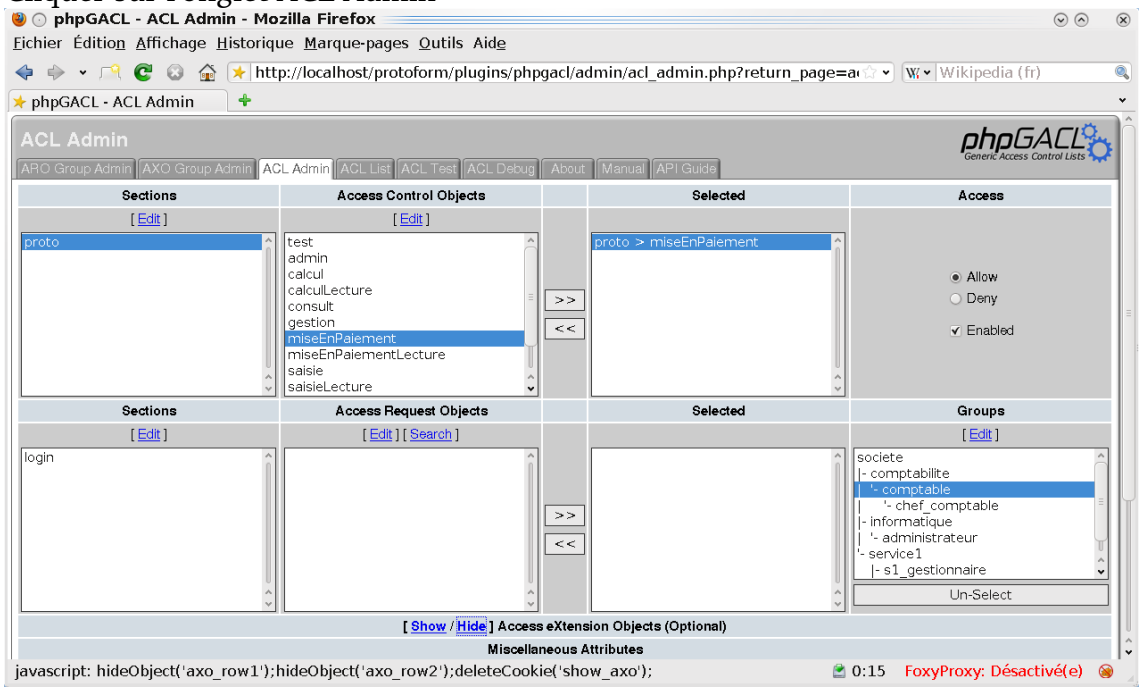
la liste des ACO déjà créés s'affiche pour cette application.

En cliquant sur Edit, nous pourrons accéder à la saisie des différents ACO ; le mode de saisie est identique à celui des logins.

3. Attribuer les ACL

Maintenant que nous avons créé d'une part nos logins et que nous les avons rattaché aux groupes, et d'autre part les ACO, il nous reste à les relier ensemble, c'est à dire donner les droits sur les ACO aux groupes ARO.

Cliquer sur l'onglet ACL Admin



Dans la première boite en haut à gauche (Sections), cliquer sur le nom de notre application (ici, proto).

Dans la seconde boite (Access Control Objects), sélectionner l'ACO auquel nous souhaitons rattacher un groupe, puis cliquer sur >>. Il est possible de sélectionner plusieurs ACO si nécessaire

Dans la boite Groups, sélectionner le où les groupes que nous souhaitons rattacher aux ACO sélectionnés

Valider en cliquant sur Submit

4. Utiliser phpGACL dans l'application

```
include_once ("plugins/phpgacl/gacl.class.php");
$phpgacl = new gacl();
$res = $phpgacl->acl_check("proto", "calcul", "login",
"pierre.durand");
```

$res contiendra 1 si le login pierre.durand dispose des droits pour accéder au module calcul.

E. Résister aux attaques

Voici quelques attaques fréquentes :

- l'injection de code SQL, qui consiste à faire exécuter du script SQL différent de celui prévu dans l'application,
- le cross-site scripting, qui vise à renvoyer l'internaute vers un autre site
- le cross-site-request-forgery, qui vise à faire exécuter du script par une personne qui dispose de droits étendus

- le détournement de session, qui s'appuie sur le vol de l'identifiant de session (en général stocké dans un cookie, un fichier permettant de stocker des informations dans le navigateur du poste client). Une fois l'identifiant de session récupéré, l'attaquant réalise toutes les opérations que peut faire l'utilisateur.

1. L'attaque par injection de code SQL

Prenons une requête de suppression d'un dossier. La classe Dossier dispose de la méthode suivante :

```
function supprimer($id) {
  $sql = 'delete from '.$table.'
  where Id = '.$Id;
  $resultat = $this->executeSQL($sql);
}
```

Si $Id vaut 5, l'enregistrement 5 sera alors supprimé. Par contre, si un attaquant modifie la valeur $Id, en lui substituant la chaîne de caractères Id, la requête va devenir :

```
delete from Personne where Id = Id;
```

L'exécution de la requête va alors vider la table Personne...

Dans ce cas de figure, la parade est assez simple : il suffit de vérifier que $id est bien un nombre, et non pas une chaîne de caractères.

Autre exemple, concernant, cette fois ci, la modification du mot de passe :

```
function setPasswd($login, $motDePasse){
  $sql = 'update Login
    set MotDePasse = "'.$motDePasse.'"
    where Login = "'.$login.'"';
  $resultat = $this->executeSQL($sql);
}
```

Nous allons modifier la valeur $login, en la remplaçant par :

```
jean.dupont" or "administrateur
```

Le code SQL de la requête devient alors :

```
update Login set ModDePasse = "AZERTYUIOP"
where Compte = "jean.dupont" or "administrateur";
```

A l'exécution, la requête va non seulement modifier le mot de passe du login jean.dupont, mais également celui du compte administrateur...

Heureusement, PHP est configuré par défaut pour interdire ce type d'attaques. Le fichier php.ini contient, dans quasiment tous les cas, la valeur suivante :

```
magic_quotes_gpc = on
```

Avec cette directive, PHP va préfixer les apostrophes ou les doubles apostrophes avec le caractère \. Notre requête va donc devenir :

```
update Login set ModDePasse = "AZERTYUIOP"
where Compte = "jean.dupont\" or \"administrateur";
```

et n'aboutira pas, aucun compte ne s'appelant jean.dupont\" or \"administrateur...

Si l'application fonctionne avec une base de données Sybase, cette directive ne fonctionnera pas, Sybase ne traitant pas correctement le caractère \. Il faudra alors modifier les directives suivantes dans le fichier php.ini :

```
magic_quotes_gpc = Off
magic_quotes_sybase = On
```

Si l'accès au fichier php.ini n'est pas possible, ces directives peuvent être introduites dans un fichier .htaccess, qui sera placé à la racine de l'application :

```
php_value magie_quotes_gpc off
php_value magic_quotes_sybase on
```

Dans ce cas de figure, l'apostrophe est « échappée » par une seconde apostrophe : Jame's devient Jame"s.

Ces directives ne sont pas modifiables en utilisant la fonction PHP ini_set().

a) Vérifier le type des données introduites

La première protection consiste à vérifier que les données renvoyées par le navigateur sont bien du type qui est attendu. C'est particulièrement vrai pour les nombres, d'autant plus s'ils sont utilisés comme argument dans une requête (cf. l'attaque par injection de code SQL, ci-dessus).

b) Vérifier si la donnée est bien un nombre

Par défaut, le navigateur renvoie toutes les données sous forme de chaîne de texte. Pour tester s'il s'agit d'une variable numérique, nous pouvons utiliser le script suivant :

```
if (strlen($var)>0) {
  if (is_numeric($var)) {
    // c'est un nombre
    ...
    }
}
```

A noter qu'il faut commencer par vérifier si la chaîne n'est pas vide.

Il est également possible de tester si la valeur entrée est booléenne ou non, avec la fonction is_bool(mixed var).

Pour les chaînes de caractères, il peut être également opportun de vérifier leur longueur. Ainsi, si un numéro de téléphone est attendu, mais que la chaîne passée par le navigateur dépasse 30 caractères, il y a probablement un problème quelque part (erreur de saisie par exemple). La vérification s'effectue avec la fonction strlen(var).

c) Vérifier le masque – expressions régulières

Dans un certain nombre de cas de figure, il est préférable de s'assurer que les informations fournies par le navigateur respectent bien une forme prédéfinie. Nous utiliserons alors une fonction de vérification de masque, ou motif (pattern en anglais).

Les fonctions de masque s'appuient sur les expressions régulières, qui se retrouvent dans quasiment tous les langages. Ces masques sont bâtis sur un certain nombre de méta-caractères standards. Néanmoins, chaque langage a rajouté ses propres extensions (shells Unix, norme Posix, langage Perl...).

PHP implémente, dans ses dernières versions, les extensions POSIX et PERL, grâce à sa bibliothèque PCRE (Perl Compatible Regular Expressions). Voici la liste des méta-caractères les plus courants utilisables dans les expressions régulières en PHP :

Méta-caractère	Usage	Type d'opérateur
^	Début de chaîne (uniquement en début de masque). Sinon, signifie la valeur contraire à ce qui suit – utilisé dans les classes.	Standard
$	Fin de chaîne	standard
.	N'importe quel caractère	standard
*	Répète de 0 à n le caractère précédent. Ainsi, .* indique une chaîne vide ou contenant plusieurs caractères indéterminés	standard
+	Répète de 1 à n le caractère précédent. .+ indique une chaîne d'au moins un caractère	POSIX
?	Vérifie la présence ou l'absence du caractère précédent (0 ou 1)	standard
\	Caractère d'échappement, qui inhibe le fonctionnement normal du caractère qui suit. Par exemple, pour rechercher le caractère *, il faudra écrire : *	standard
[ ]	Encadre une classe. Une classe permet de tester un seul caractère. Par exemple, pour tester la présence d'une voyelle, on utilisera [aeiouy]. Si le test porte sur tout caractère qui ne soit pas une voyelle, le masque aura la forme [^aeiouy]	standard
( )	Encadre un sous-masque	POSIX
{ }	Quantificateur. {2,6} autorise une chaîne qui a une taille de 2 à 6 caractères. Se positionne après le caractère ou la classe.	PCRE
-	Indique un intervalle de caractères. Par exemple, 0-9 permet de vérifier la présence d'un chiffre.	standard
|	Permet une alternative. (dupont|durand) sera valide si dupont ou durand sont présents.	Standard
\n	Nouvelle ligne	PCRE
\r	Retour chariot	PCRE
\t	Tabulation	PCRE
\f	Saut de page	PCRE
\d \D	Tout caractère décimal tout caractère autre que décimal	PCRE
\h \H	N'importe quel espace horizontal n'importe quel caractère qui n'est pas un espace horizontal	PCRE
\s \S	Tout caractère blanc tout caractère qui n'est pas un blanc	PCRE
\w \W	Tout caractère pouvant faire partie d'un mot (alphabétique, chiffre, tiret...) tout caractère ne pouvant faire partie d'un mot	PCRE

Le masque suivant pourrait être utilisé pour tester la validité d'une adresse mél :

^.+@.+\.[a-zA-Z]{2,6}$

ce qui signifie :

- une chaîne qui commence par au moins un caractère (^.+),
- qui contient le signe @
- qui contient au moins un caractère (.+),
- suivi d'un point \. ,
- et qui se termine par une chaîne de 2 à 6 caractères alphabétiques, en majuscules ou en minuscules ([a-zA-Z]{2,6}$

En PHP, Ce masque va pouvoir être vérifié avec la fonction suivante :

```
$var = "jean.dupont@monserveur.com";
$pattern = "#^.+@.+\.[a-z]{2,6}$#";
if (preg_match($pattern, $var) == 1) {
  //Il s'agit d'une adresse mel
  ...
}
```

En PHP, les masques doivent être encadrés par les deux mêmes caractères, qui les délimitent. C'est pourquoi le contenu de la variable $pattern est encadrée par le signe #.

2. L'attaque par cross-site scripting

Le principe de cette attaque est relativement simple. En HTML, il existe une balise, META http-equiv, qui permet de rediriger une page vers une autre, ou de la rafraîchir. Ainsi, le code suivant va déclencher l'affichage d'une nouvelle page au bout de 10 secondes, vers le site *http://pirate.com/* :

```
<META http-EQUIV='Refresh' CONTENT='10; url=http://pirate.com/'>
```

Il est également possible d'utiliser les balises javascript pour rediriger l'application vers une autre page :

```
<script>document.location='http://pirate.com'</script>
```

Une personne mal intentionnée peut pouvoir introduire ce script dans un champ, par exemple une adresse mél. A l'affichage, le navigateur exécutera l'instruction saisie... Il suffit alors au pirate de construire un site d'aspect identique pour induire l'internaute en erreur et lui faire faire n'importe quoi !

Pour éviter ce type d'attaque, nous allons utiliser la fonction PHP **htmlspecialchars**, qui va transformer les caractères spéciaux HTML en leur équivalent textuel, et ce, avant l'envoi vers le navigateur :

```
$var = htmlspecialchars($var);
echo ($var);
```

Les caractères suivants sont transformés :

Caractère	Code de transformation
&	&amp;
" (guillemets doubles)	&quot;
<	&lt;
>	&gt;

Ainsi, le code

```
<META http-EQUIV='Refresh' CONTENT='10; url=pirate.com/'>
```

devient :

```
&lt;META http-EQUIV='Refresh' CONTENT='10;
url=http://pirate.com/'$gt;
```

et ne sera plus traité par le navigateur comme une instruction.

3. Le Cross site request forgery

Le principe général de cette attaque est de faire déclencher du script par l'utilisateur, dans une application où il dispose de droits élevés. Par exemple, un utilisateur souhaite supprimer une fiche, mais il ne dispose pas des droits adéquats. Pour cela, il va faire exécuter le script par un administrateur, et à son insu. Le plus simple, c'est de lui envoyer un mél contenant un lien de type image. En fait, l'URL n'est pas une image, mais un lien vers le script à exécuter sur le serveur, par exemple :

```
<href=index.php?module=supprimerFiche&id=10>
```

L'administrateur disposant des droits d'exécution, le script s'exécutera et la fiche sera supprimée.

Pour se prémunir de ce type d'attaque, il n'y a pas de solution miracle. La première, c'est que l'application vérifie systématiquement la requête de l'utilisateur, en lui demandant de confirmer l'opération. C'est souvent fastidieux, et en général utilisé uniquement lors des opérations de suppression.

La seconde approche vise à générer, dans chaque formulaire, une donnée aléatoire, qui sera vérifiée lors de la validation. C'est la technique du jeton, qui doit, de plus, avoir une durée de vie limitée.

La troisième consiste à s'assurer que la page qui déclenche l'action (*Referer*) est bien celle qui est attendue. Ainsi, cela évite que l'action soit déclenchée depuis une page quelconque, voire hors de l'application (cas du lien fourni dans un mél).

4. Le détournement de session

La communication entre un serveur web et un navigateur s'effectue en mode asynchrone : chaque nouvelle requête envoyée vers le serveur s'apparente à une nouvelle demande.

Pour pouvoir gérer la continuité de la connexion, les serveurs web mettent en place un mécanisme de session, et transmettent au navigateur un identifiant de session, soit dans une variable, soit dans un cookie (cas le plus fréquent). La session contient des variables, qui permettent notamment de savoir qui est connecté, les droits dont l'utilisateur dispose, etc.

Si un pirate récupère l'identifiant de session, il a alors la possibilité de faire exécuter du code en se faisant passer pour l'utilisateur. Cette récupération peut s'effectuer en redirigeant, par l'intermédiaire d'un lien, l'utilisateur vers un faux site.

A noter que depuis les points d'accès publics en wifi, il est très facile de récupérer les cookies qui circulent en clair sur le réseau (une extension Firefox permet de récupérer tous les cookies « qui passent » en quelques clics) : le risque de piratage est de fait très grand. Pour éviter ce vol lié à l'écoute du réseau, seul le cryptage complet de la session est nécessaire (et non pas seulement la phase d'identification).

Sans rentrer dans les mécanismes détaillés de ce mode d'attaque, il est fortement conseillé, pour s'en protéger, de régénérer l'identifiant de session juste après que l'utilisateur se soit identifié. La fonction `session_regenerate_id()`[1] sera alors utilisée.

Il est également possible de vérifier que l'adresse IP de la machine qui a ouverte la session est toujours la même. Ce n'est pas une garantie absolue, mais cela peut limiter les risques.

5. Protéger les accès à certains dossiers de l'application

Lors de l'écriture d'une application web, un grand nombre de pages sont créées, et sont organisées dans une arborescence. Avec le principe du langage HTML, toute page est accessible nativement, en indiquant le chemin complet d'accès, par exemple :

`html://monsite.com/monapplication/parametres/monfichierparam.php`

.

Dans un certain nombre de cas, il est souhaitable de ne pas autoriser l'accès direct à un dossier, comme celui qui contient les fichiers de paramètres, ou ceux qui reçoivent des fichiers en téléchargement.

En général, il est conseillé de déplacer ces dossiers sur le serveur à un endroit où ils ne sont pas directement accessibles. Malheureusement, ce n'est possible que si le programmeur maîtrise complètement le serveur, ce qui est rarement le cas. De plus, il faut bien connaître la structure de l'application pour savoir quels dossiers déplacer, surtout s'il s'agit d'un logiciel récupéré sur Internet.

Avec le serveur Web Apache, la solution pour gérer cette protection est assez simple. Il suffit de créer un fichier nommé `.htaccess` à la racine du dossier à protéger, qui comprendra les instructions suivantes :

```
order deny,all
deny from all
```

Ces deux lignes vont interdire l'accès direct au dossier depuis le navigateur. Cela n'empêchera pas l'application d'y accéder, par l'intermédiaire de la commande `include_once`, qui permet de charger le fichier.

Avec le serveur Web de Microsoft (IIS, Internet Information Server), ce mécanisme n'est pas pris en charge. La protection des dossiers s'effectue depuis la console d'administration, en indiquant, pour chaque dossier, les droits à attribuer.

Force est de constater que l'approche d'Apache est la plus simple : un copier-coller de l'application entre deux serveurs intègre également la copie des fichiers `.htaccess` : sur le nouveau serveur, le mécanisme de protection décidé par le développeur est immédiatement opérationnel.

---

1 Le changement de l'identifiant de session permet d'éviter que la connexion soit rejouée par un pirate. Dans le cas où la communication est chiffrée, le pirate ne pourra pas connaître le mot de passe utilisé. Toutefois, s'il arrive à capturer la trame qui contient le mot de passe, il peut la réexpédier au serveur, en modifiant simplement l'adresse IP de retour (non chiffrée). Pour éviter cela, il faut transmettre au serveur non seulement le mot de passe, mais également une autre information, qui change à chaque fois, le tout étant chiffré.
Le navigateur n'envoie pas uniquement le formulaire au serveur : il transmet également d'autres informations, comme le navigateur, la langue utilisée, etc., et également le numéro de session PHP fourni par le serveur.
En régénérant le numéro de session juste après une identification réussie, même si le pirate a pu capturer la trame, il ne pourra la rejouer, celle-ci contenant alors une information obsolète, à savoir l'ancien identifiant de session.

Avec IIS, l'administrateur doit déclarer les droits dossier par dossier, ce qui n'est pas forcément très simple, et il faut surtout que ceux-ci soient décrits : ce n'est que très rarement le cas pour les applications PHP. Par expérience, c'est une information qui ne figure jamais dans les documentations d'installation.

Il est toutefois possible d'installer des modules complémentaires dans les serveurs IIS pour que les fichiers `.htaccess` soient pris en compte. Des solutions commerciales existent sur Internet, comme, par exemple, http://www.iistools.com/ ou http://www.isapirewrite.com/ (solutions non testées).

Une autre solution, plus complexe, vise à créer une double arborescence pour l'application. On crée deux dossiers de base, par exemple `code` et `www`. Seul le dossier `www` est accessible depuis un navigateur, et l'ensemble du code de l'application est stocké dans le dossier `code`. Les pages accessibles, stockées dans `www`, appellent, par l'intermédiaire de fonctions `include()`, les pages contenant la logique de l'application. Cette approche impose un paramétrage spécifique du serveur web pour qu'il redirige les requêtes vers le dossier `www`. Ce mécanisme est très souvent implémenté dans les frameworks modernes.

F. Quelques rappels concernant le chiffrement

1. Principes généraux

Le chiffrement est une opération qui vise à éviter qu'un échange puisse être lu par une personne tierce. Deux types de chiffrement existent, le chiffrement symétrique et le chiffrement asymétrique.

Dans le chiffrement symétrique (ou chiffrement à clé secrète), une même clé est utilisée pour crypter et décrypter le message. C'est une technique rapide à mettre en place, mais elle implique que les deux correspondants aient la même clé. Actuellement, l'algorithme **AES256** est largement utilisé pour mettre en place un tel chiffrement. Il est notamment utilisé dans les cartes bancaires.

Le chiffrement symétrique est réalisé par bloc[1]. Le message à chiffrer est découpé en plusieurs blocs, de 64, 128 ou 256 bits en général. Un bloc est choisi aléatoirement pour commencer le chiffrement. Ce bloc est chiffré avec la clé de chiffrement. Le bloc suivant est transformé, avant chiffrement, en l'additionnant (addition bit à bit, sans retenue) avec le résultat du chiffrement du bloc précédent, puis est chiffré avec la clé. Les autres blocs sont chiffrés selon le même mécanisme.
La sécurité du chiffrement symétrique tient d'une part à la longueur de la clé, et d'autre part à la taille des blocs : plus les blocs sont grands, meilleure sera la sécurité. De plus, en raison du mode de chiffrement, la perte d'un bloc peut être compensée en recalculant sa valeur, à partir des deux blocs continus. On peut également rajouter une valeur de hachage, calculée à partir d'une clé différente et prenant en compte l'intégralité des blocs transmis, pour s'assurer de l'intégrité du message.

Le chiffrement asymétrique est d'invention assez récente. Il est basé sur le calcul du produit de deux nombres premiers. Sa résistance tient à l'impossibilité

1 Il est possible de réaliser un chiffrement en continu, chaque bit étant recalculé à partir de la clé de chiffrement. Cette technique est imparable, il est impossible de retrouver le message d'origine, dès lors que la clé est de la taille du message à chiffrer. Néanmoins, comme le message est chiffré « au fil de l'eau », un pirate qui s'introduit au milieu du système (attaque dite de Man in the middle) peut facilement modifier un bit dans le message et, si la structure du message est connue (flux de paiement, par exemple), modifier des informations facilement. De fait, le chiffrement en continu ne permet pas de s'assurer de l'intégrité du message.

actuelle de factoriser le nombre produit autrement que par la « force brute »[1], c'est à dire en testant toutes les combinaisons possibles. Le temps de calcul croît de façon exponentielle avec la taille de la clé.

L'algorithme le plus connu est le **RSA**, décrit en 1977 par Ronald Rivest, Adi Shamir et Leonard Adleman. Son principe est simple. Deux clés sont générées, un message crypté avec une des clés ne peut être décrypté qu'avec la seconde, et vice-versa. Dans la pratique, une des deux clés générées reste en la possession exclusive de son propriétaire (la clé privée), la seconde étant transmise à tous les correspondants (la clé publique). Selon que l'on veuille envoyer un message chiffré ou s'assurer de l'identité de l'expéditeur, on utilisera soit la clé publique du correspondant pour chiffrer le message, soit la clé privée. Nous verrons plus loin différents cas d'utilisation du chiffrement asymétrique.

Pour de plus amples informations concernant le détail du fonctionnement de l'algorithme, consultez l'article de Wikipedia *Rivest, Shamir, Adleman*[2].

2. Les fonctions de calcul d'empreinte (hachage)

Le calcul d'empreinte permet de calculer, à partir d'un texte ou d'un fichier, un condensat de longueur fixe, qui a les caractéristiques suivantes :

- il est unique (il n'existe pas de possibilité pour que deux entrées différentes puisse aboutir à la même empreinte – dans ce cas, on parle de collision) ;
- le texte ou le fichier d'origine ne peut être retrouvé à partir de l'empreinte.

Ce mécanisme va être utilisé principalement dans deux cas :

- d'une part, quand on veut s'assurer que la copie d'un fichier est bien identique à l'original, par exemple dans le cas du téléchargement d'une image ISO[3] ;
- d'autre part, pour encoder un mot de passe, qui ne pourra pas être décodé.

Pour vérifier la cohérence d'un fichier téléchargé, nous appliquerons le processus suivant :

- le fournisseur du fichier calcule une empreinte, et la publie ;
- une fois le fichier récupéré, l'empreinte de celui-ci est calculée ;
- si les deux empreintes sont identiques, alors on considère que les deux fichiers sont identiques.

Pour stocker les mots de passe, le mécanisme suivant va être utilisé :

- lors de la création du mot de passe, son empreinte est calculée, et c'est celle-ci qui va être stockée en base de données ;
- au moment où le mot de passe est utilisé, c'est à dire quand son détenteur le saisit, le programme de vérification calcule l'empreinte du texte saisi, puis la compare à la valeur stockée dans la base de données. Si les deux empreintes sont identiques, alors le mot de passe est correct.

1 Il existe toutefois des méthodes mathématiques qui permettent de gagner un peu de temps
2 http://fr.wikipedia.org/wiki/Rivest_Shamir_Adleman
3 Image d'un disque, qui pourra être reproduit à l'identique

Voici un exemple de calcul d'empreinte du mot **Bonjour** en utilisant l'algorithme **sha-256** (commande **sha256sum**) :

```
8dc2a6966f1be1644ec6b1f7223f47e53de5ad05e1c976736d948e7977a13dd3
```

Il suffit de modifier une seule lettre, en tapant par exemple **bonjour** (sans majuscule), pour que le résultat soit totalement différent :

```
9cec0af545144159bac85c7b908d5e0b9b0ef961497401c5ad8da26f065ad926
```

Pour qu'un algorithme de hachage soit considéré comme suffisamment sûr, il faut :

- qu'il ne soit pas possible de créer des collisions (même si, comme le nombre de combinaisons est nécessairement fini – la taille du condensat étant fixe – cette possibilité ne puisse jamais être totalement exclue) ;
- que la longueur de stockage soit suffisamment élevée pour que l'on ne puisse pas créer de dictionnaire inverse, c'est à dire une base de données qui permettrait, à partir du condensat obtenu, d'obtenir l'original. Un condensat de 256 bits entraîne $1{,}158 \times 10^{77}$ combinaisons possibles (2^{256}), ce qui, mathématiquement, rend impossible toute recherche exhaustive.

L'utilisation de SHA-256 est donc considéré comme sûr.

3. Le pseudo-cryptage

Si le chiffrement permet de garantir la confidentialité d'une information échangée, il présente un inconvénient : un fichier crypté se reconnaît très facilement, et certains gouvernements totalitaires pourraient être tentés de bloquer les échanges cryptés, par l'utilisation de logiciels qui vont analyser les messages échangés et repérer ceux dont les chaînes de caractères sont totalement aléatoires.

Pour éviter cela, il est possible de transformer les messages envoyés en appliquant des transformations suffisantes pour les rendre incompréhensibles, mais qui ne seront pas considérés comme aléatoires par les systèmes de surveillance automatiques. Bien sûr, ces messages pourraient être décryptés en quelques jours, à condition qu'ils soient reconnus comme tels, interceptés, et qu'on y consacre des ressources de calcul importantes, mais cela peut être suffisant : certaines informations n'ont pas besoin de rester confidentielles trop longtemps (scoop journalistique, par exemple). On est typiquement dans le rapport entre les moyens à mettre en œuvre pour décoder une information et sa valeur intrinsèque : dans beaucoup de cas, le « jeu n'en vaudra pas la chandelle ».

De plus, si le chiffrement est autorisé en France, ce n'est pas le cas dans un certain nombre de pays, qui tiennent à pouvoir déchiffrer tous les messages échangés (lutte contre le terrorisme, par exemple). Des technologies de chiffrement classique comme AES ne pourront pas être utilisées, seules celles permettant un déchiffrement seront permises[1].

1 Mais ça ne signifie pas que le déchiffrement soit facile et à la portée de tous !

La bibliothèque PERSEUS[1] répond à ces besoins : elle garantit qu'un message échangé ne peut être déchiffré qu'à condition d'y consacrer des ressources très importantes, hors de portée des pirates habituels.

Concrètement, l'idée est de chiffrer les données échangées à l'aide de codes convolutifs[2]. Une fois chiffré, le code est brouillé à l'aide d'un « bruit » numérique. Avant transmission, les paramètres nécessaires au déchiffrement, générés aléatoirement pour chaque séquence, sont envoyés lors d'une courte session https standard. Le destinataire commence donc par recevoir ces codes à partir desquels il obtiendra sans difficulté la suite de l'échange.

Pour accéder aux informations transmises, tout observateur placé sur la ligne, doit de son côté commencer par déchiffrer la première séquence, avant d'opérer une seconde opération de déchiffrement. Comme les codes changent en permanence de façon aléatoire, le pirate doit fournir une puissance de calcul importante pour ne rien perdre d'une communication. L'augmentation des ressources consommées finirait alors par trahir la machine infectée. Autrement dit, l'écoute « systématique » deviendrait prohibitive, alors qu'il reste possible d'intervenir ponctuellement sur un échange donné, en allouant à cette tâche des machines dédiées.

En raison de sa faible complexité technique, le pseudo-cryptage peut également être mis en place pour protéger des sauvegardes de données : les temps de calcul ne ralentiront pas significativement l'opération, ce qui pourrait se produire avec un chiffrement classique.

4. Quel chiffrement utiliser ?

Le chiffrement asymétrique est très gourmand en ressources, en raison du mode de calcul, mais il permet de s'assurer de l'identité du correspondant. A contrario, le chiffrement symétrique s'exécute rapidement, mais impose que la clé de chiffrement/déchiffrement soit partagée entre les deux correspondants.

Plusieurs cas d'utilisation peuvent être identifiés :

- l'échange de fichiers : le fichier est chiffré au préalable, puis envoyé, par exemple par courriel. L'important, c'est que seul le destinataire puisse déchiffrer le fichier, pas les personnes qui pourraient l'intercepter. On utilisera alors le chiffrement asymétrique.
- La connexion permanente, par exemple pour se connecter au système d'informations de l'entreprise. On crée alors un tunnel chiffré. La procédure de chiffrement va être réalisée en deux étapes :
  - le chiffrement asymétrique va être utilisé pour initier la connexion, pour s'assurer de l'identité de chacun des correspondants ;
  - une clé de chiffrement symétrique va alors être échangée entre les deux correspondants, qui ne sera donc connue que d'eux seuls ;
  - le reste de la communication est chiffré en mode symétrique, ce qui sera plus performant et moins coûteux en temps machine.

1 Pour plus d'informations concernant PERSEUS, un article dans Linux Magazine N° 135 : *PERSEUS : protéger des communications avec du bruit* (http://www.gnulinuxmag.com/index.php/2011/01), ou une page web : http://www.clubic.com/actualite-308400-perseus-chiffrement-donnees-responsable.html

2 Un code convolutif est un code qui utilise les valeurs obtenues précédemment pour continuer le codage (effet mémoire). Ils sont très souvent utilisés comme codes de détections d'erreurs, notamment dans les transmissions.

- le stockage de mots de passe, par exemple dans une base de données. Dans ce cas, soit on utilise une fonction de chiffrement symétrique classique, qui permet de recouvrer le mot de passe, soit on préférera l'utilisation d'une fonction de calcul d'empreinte (hachage), qui garantira la confidentialité du mot de passe[1].

5. Quelques algorithmes de chiffrement

a) RSA

Cf. le début de ce chapitre.

b) Blowfish[2]

Blowfish est un algorithme de chiffrement symétrique (i.e. *« à clef secrète »*) par blocs conçu par Bruce Schneier en 1993. Il tire son nom du poisson globe japonais (ou *fugu*), qui en est également l'emblème.

Blowfish utilise une taille de bloc de 64 bits et la clé de longueur variable peut aller de 32 à 448 bits. Elle est basée sur l'idée qu'une bonne sécurité contre les attaques de cryptanalyse peut être obtenue en utilisant de très grandes clés pseudo-aléatoires.

Blowfish présente une bonne rapidité d'exécution excepté lors d'un changement de clé, il est environ 5 fois plus rapide que Triple DES et deux fois plus rapide que IDEA. Malgré son âge, il demeure encore solide du point de vue cryptographique avec relativement peu d'attaques efficaces sur les versions avec moins de tours. La version complète avec 16 tours est à ce jour entièrement fiable et la recherche exhaustive reste le seul moyen pour l'attaquer.

Il a été placé dans le domaine public par son créateur ; il n'est protégé par aucun brevet, et son utilisation n'est pas soumise à licence. Cela explique en partie son succès, car ce fut un des premiers algorithmes de chiffrement dont l'utilisation était libre. Il est utilisé dans de nombreux logiciels propriétaires et libres (dont **GnuPG** et **OpenSSH**).

c) DES et triple DES[3]

Le **Data Encryption Standard** (**DES**) est un algorithme de chiffrement symétrique (chiffrement par bloc) utilisant des clés de 56 bits. Son emploi n'est plus recommandé aujourd'hui, du fait de sa lenteur à l'exécution et de son espace de clés trop petit permettant une attaque systématique en un temps raisonnable. Quand il est encore utilisé, c'est généralement en Triple DES, ce qui ne fait rien pour améliorer ses performances. DES a notamment été utilisé dans le système de mots de passe UNIX.

d) AES[4]

Advanced Encryption Standard ou AES (soit « standard de chiffrement avancé » en français), aussi connu sous le nom de *Rijndael*, est un algorithme de chiffrement symétrique. Il remporta en octobre 2000 le concours AES, lancé en

1 Pour éviter que les mots de passes puissent être décryptés par un pirate, s'il se procure la base de données, on utilise en général la technique du salage, qui consiste à rajouter une chaîne de caractères au mot de passe stocké (cf. IV.G, *Compliquer la tâche des pirates : le salage*, page 61).
2 Source : wikipedia - http://fr.wikipedia.org/wiki/Blowfish
3 Source : wikipedia - http://fr.wikipedia.org/wiki/Data_Encryption_Standard
4 Source : wikipedia - http://fr.wikipedia.org/wiki/Standard_de_chiffrement_avancé

1997 par le NIST et devint le nouveau standard de chiffrement pour les organisations du gouvernement des États-Unis.

e) Autres...

On peut également citer **Serpent** et **Twofish**, que l'on retrouve dans des logiciels de chiffrement comme **Truecrypt** (http://www.truecrypt.org/).

6. Quelques algorithmes de calcul d'empreintes

a) MD5[1]

L'algorithme **MD5**, pour **Message Digest 5**, est une fonction de hachage cryptographique qui permet d'obtenir l'empreinte numérique d'un fichier (...). Il a été inventé par Ronald Rivest en 1991.

Cette fonction de hachage produit des condensats de 64 bits, dont seuls 56 sont réellement utilisés, les autres servant de code de correction.

En 2004, une équipe chinoise découvre des collisions complètes. **MD5** n'est donc plus considéré comme sûr au sens cryptographique. On suggère maintenant d'utiliser plutôt des algorithmes tels que **SHA-256**, **RIPEMD-160** ou **Whirlpool**.

Cependant, la fonction MD5 reste encore largement utilisée comme outil de vérification lors des téléchargements et l'utilisateur peut valider l'intégrité de la version téléchargée grâce à l'empreinte. Elle a été également utilisée par GNU/Linux pour stocker les mots de passe (en utilisant une fonction complémentaire pour compliquer le déchiffrement)[2].

b) Whirlpool[3]

Whirlpool est une fonction de hachage cryptographique conçue par Vincent Rijmen et Paulo Barreto pour le projet NESSIE. Elle a été nommée d'après la galaxie M51. La fonction utilise une architecture de type *Miyaguchi-Preneel* connue pour sa résistance à la cryptanalyse, cette structure produit des empreintes de 512 bits (…).

En interne, l'algorithme travaille sur 512 bits grâce à une fonction similaire à celle de l'algorithme de chiffrement symétrique AES (auquel Vincent Rijmen a également participé et qui à l'origine s'appelle *Rijndael*). L'utilisation d'une version modifiée du bloc de chiffrement de AES (appelée W) garantit un système robuste et fiable.

c) SHA-1

SHA-1 a été conçu par la *National Security Agency* des États-Unis (NSA), et a été publié en 1995. Cette fonction produit un condensat de 160 bits.

Une attaque basée sur le paradoxe des anniversaires (cf. IV.J.2, *Le paradoxe des anniversaires1,* page 69) permet de trouver une collision complète sur le SHA-1 avec un nombre d'opérations de l'ordre de 2^{69}opérations. Même s'il est moins sûr (au sens cryptographique) que SHA-256, et si des collisions restent possibles, il est encore largement assez résistant pour être utilisé, en remplacement de MD5.

1 Source : wikipedia - http://fr.wikipedia.org/wiki/MD5

2 Aujourd'hui, les distributions Linux utilisent des méthodes de chiffrement bien plus sûres. Vous pouvez consulter le mécanisme utilisé dans votre distribution en consultant, dans le fichier **/etc/login.defs**, la valeur de **ENCRYPT_METHOD**. Les dernières versions d'Ubuntu utilisent SHA512, qui produit des condensats de 512 bits.

3 Source : wikipedia - http://fr.wikipedia.org/wiki/Whirlpool_(algorithme)

C'est d'ailleurs cette fonction qui est utilisée dans les annuaires OPENLDAP pour chiffrer les mots de passe.

d) SHA-256[1]

SHA-256 (*Secure Hash Algorithm*) est une fonction de hachage cryptographique conçue par la *National Security Agency* des États-Unis (NSA), et publiée en 2000. Elle produit un résultat (appelé « *hash* » ou *condensat*) de 256 bits et dérive du **SHA-1**.

SHA-256 est devenu le nouveau standard recommandé en matière de hachage cryptographique après les attaques sur MD5 et SHA-1.

7. Algorithme de signature

a) DSA

Le **Digital Signature Algorithm** a été inventé alors que le RSA était encore breveté. Il est basé sur le même principe de clés asymétriques, et est utilisé uniquement pour la signature.

En pratique, on le retrouve associé à SSH, notamment pour identifier les serveurs susceptibles de se connecter automatiquement à un autre serveur sans fournir de mot de passe.

G. Compliquer la tâche des pirates : le salage

Nous venons de voir qu'une empreinte correspond à une et une seule chaîne, et qu'il n'est pas possible de retrouver la chaîne à partir de l'empreinte. Ça, c'est le cas général.

Normalement, un mot de passe sera stocké sous forme d'empreinte, pour qu'on ne puisse pas le retrouver si on connait la clé de chiffrement.

Malheureusement, les utilisateurs sont rarement inventifs dans le choix des mots de passe. Une étude américaine[2] a permis de relever le top 25 des mots de passe les plus utilisés, et c'est assez navrant... Voici le début de la liste :

- password
- 123456
- 12345678
- 1234
- qwerty

Vous remplacez **qwerty** par **azerty**, et vous avez l'équivalent en français !

Maintenant, si je calcule l'empreinte de **azerty**, j'obtiens la valeur suivante :

```
70b2e56b49d7264191925cea3da97a5213b511e68ae905a077cfec6f6a257af8
```

Si, par hasard, la base de données contenant les empreintes est piratée, ou simplement publiée par erreur (ou accessible...), il suffit de rechercher dans celle-ci si une empreinte contenant la chaîne précédente pour retrouver le mot de passe : **azerty**. Ainsi, un pirate pourra facilement retrouver une bonne partie des mots de passe à partir d'un dictionnaire comprenant les combinaisons les plus fréquentes.

1 Source : wikipedia - http://fr.wikipedia.org/wiki/SHA-256
2 http://www.zdnet.fr/actualites/123456-un-mot-de-passe-toujours-trop-repandu-39772721.htm

Pour pallier cette éventualité, on utilise aujourd'hui la technique dite du « salage ». Elle consiste à rajouter une chaîne (non publiée, bien sûr) avant de calculer l'empreinte.

Ainsi, si j'insère la chaîne **9d8c47b** après le 3ème caractère du mot de passe, je vais calculer l'empreinte de **aze9d8c47brty**, qui vaut :

```
05f91406b3665ac5ec8f63599afa50ea786405fd2b9be76732e84d9319d61cd8
```

Les pirates ne pourront alors plus utiliser un dictionnaire pour rechercher les mots de passe les plus fréquents.

H. Le chiffrement asymétrique

1. Comment s'utilise le chiffrement asymétrique ?

Paul et Julie disposent chacun de clés asymétriques, une clé privée, qu'ils conservent en leur possession, et une clé publique, qu'ils transmettent à leurs correspondants[1].

a) Pour chiffrer un échange

Paul veut envoyer un fichier chiffré à Julie. Voici les différentes étapes qu'il va suivre :

- Julie transmet sa clé publique à Paul ;
- Paul chiffre son fichier avec la clé publique de Julie, puis lui transmet son fichier chiffré;
- Julie déchiffre le fichier avec sa clé privée.

b) Pour signer un fichier

Paul veut transmettre un message, et Julie veut être sûre que c'est Paul qui lui envoie :

- Paul transmet sa clé publique à Julie ;
- Paul chiffre son message avec sa clé privée ;
- Julie reçoit le message, et le déchiffre avec la clé publique de Paul : si le déchiffrement fonctionne, c'est que c'est bien Paul qui a envoyé le message.

2. Le tiers de confiance

Bien sûr, cela ne peut fonctionner que :

- si Paul est sûr que la clé publique fournie par Julie est bien la sienne ;
- si Julie est sûre que la clé publique fournie par Paul lui appartient bien...

Pour cela, Paul et Julie vont demander à un tiers de confiance, c'est à dire quelqu'un dont l'identité est reconnue par tous, de valider leur clé publique. Ce tiers de confiance va chiffrer l'empreinte de la clé publique avec sa propre clé privée. Par exemple, pour valider la clé publique de Paul, ce tiers de confiance va donc :

- récupérer la clé publique de Paul ;
- s'assurer qu'il s'agit bien de Paul, et pas de quelqu'un d'autre[2] ;
- calculer l'empreinte de la clé de Paul ;

1 Par défaut, les logiciels de messagerie, comme Thunderbird, transmettent systématiquement les clés publiques aux correspondants, ce qui permet d'initier les échanges et de faciliter les échanges cryptés futurs.

2 Cela peut être en vérifiant les papiers d'identité, un code fourni dans un document, comme la déclaration d'impôts ou le montant de la dernière facture associé à un numéro de client...

- chiffrer cette empreinte avec sa propre clé privée.

Pour s'assurer que la clé de Paul est bien la sienne, Julie va :

- calculer l'empreinte de la clé de Paul ;
- utiliser la clé publique du tiers de confiance, pour déchiffrer l'empreinte chiffrée.

Si les deux informations sont identiques, alors Julie sera sûre qu'elle est bien en possession de la clé de Paul.

Le tiers de confiance est donc une autorité reconnue par tous, qui dispose d'une clé privée, et d'une clé publique qu'il va distribuer. Mais comment éviter que quelqu'un crée son propre certificat, en se faisant passer pour un autre ? Après tout, chacun peut facilement créer une clé privée et un certificat, et se faire passer pour une société comme Verisign, par exemple[1]. Il est possible également d'émettre un certificat comprenant le nom d'une banque, de créer un site ressemblant à s'y méprendre au site institutionnel de celle-ci, puis, au moyen de pages web malicieuses, rediriger les internautes vers ce site pirate.

Normalement, l'internaute devrait avoir ceci :

- certificat racine Verisign
  - certificat banque XXX
    - site banque XXX

À partir du site pirate, l'architecture sera la suivante :

- faux certificat Verisign
  - faux certificat de la banque XXX
    - faux site de la banque XXX

C'est sur les « épaules » du navigateur que va porter la responsabilité de valider ou non le certificat racine, c'est à dire toute la chaîne de certification. Si le navigateur ne contient pas le certificat racine, il va émettre une alerte :

Le navigateur a vérifié plusieurs points :

1 Il est facile d'intégrer n'importe quelle information dans un certificat auto-généré, et pourquoi pas, le nom d'une société...

- est-ce que le nom du site correspond exactement au nom présent dans le certificat ?
- Est-ce qu'il existe un certificat racine connu pour ce site ?

Les navigateurs modernes intègrent les certificats des tiers de confiance. Vous pouvez les consulter, par exemple dans Firefox, à partir du menu **Outils → Options** (ou **Édition → préférences** sous Windows), bouton **Avancé**, onglet **Chiffrement**, **Afficher les certificats**, puis **Autorités**.

N'intégrez jamais des certificats d'autorités qui ne seraient pas présents, à moins que vous ne soyez sûrs de ce que vous faites (autorité interne à une entreprise, par exemple). De même, ne passez jamais outre les alertes concernant la sécurité d'un site, sauf s'il s'agit d'un site interne à votre entreprise (et qu'on vous en a informé au préalable), ou s'il s'agit d'un site internet où vous n'avez pas besoin de vous identifier (sites d'informations qui utilisent systématiquement le protocole **https**).

3. Les certificats chaînés

Nous l'avons vu, le point faible de la chaîne de certification est le certificat racine initial. Dans la pratique, les sociétés de certification n'utilisent (ou ne devraient utiliser) le certificat racine que très rarement, et uniquement pour créer des autorités de certification intermédiaires. Ainsi, générer un certificat pour un serveur ou pour une personne entraîne souvent une chaîne de certification relativement longue. Voici par exemple la chaîne de certification concernant un site du ministère de l'agriculture :

- **IGC/A** : certificat racine de l'État français
  - **Agriculture AC Racine** : certificat racine, validé par l'IGC/A, pour le ministère de l'agriculture
    - **Agriculture AC Serveurs** : autorité de certification qui délivre les certificats pour les serveurs
      - **eap.agriculture.gouv.fr** : certificat du site éponyme

Pour vérifier le certificat **eap.agriculture.gouv.fr**, il faudra donc vérifier chacune des signatures des certificats précédents, jusqu'à remonter au certificat racine **IGC/A**. Le logiciel qui voudra s'assurer de la validité du certificat devra donc :

- vérifier l'empreinte du certificat **eap.agriculture.gouv.fr**, cryptée avec la clé privée de **Agriculture AC Serveurs**, à l'aide de la clé publique correspondante ; cette clé publique est intégrée dans le certificat **Agriculture AC Serveurs** ;
- vérifier de la même façon l'empreinte du certificat **Agriculture AC Serveurs**, cryptée avec la clé privée de **Agriculture AC Racine**, à partir de la clé publique contenue dans le certificat correspondant ;
- enfin, vérifier l'empreinte du certificat **Agriculture AC Racine**, cryptée avec la clé privée de **IGC/A**, à partir de la clé publique présente dans le certificat éponyme.

Si un seul des certificats présentés n'est pas valide (expiré, révoqué...), le certificat **eap.agriculture.gouv.fr** ne pourra pas être considéré comme sûr.

La plus grande difficulté de cette vérification tient au fait qu'il faut disposer de tous les certificats intermédiaires. En général, les certificats racines considérés comme sûrs sont intégrés dans les logiciels, notamment les navigateurs ou les clients de messagerie. Par contre, les certificats intermédiaires doivent être

récupérés, soit automatiquement[1], soit manuellement, pour que la vérification puisse se faire.

Dans les certificats personnels de type **p12**, les certificats des autorités intermédiaires sont en principe intégrés. Ce n'est pas toujours le cas pour d'autres certificats : il importe donc soit de les importer manuellement, soit de demander au logiciel qui génère le chiffrement (serveur web Apache, par exemple), de présenter la liste des certificats intermédiaires.

Il est également important de mettre à jour régulièrement les logiciels qui utilisent les certificats, pour qu'ils puissent intégrer les nouvelles autorités racines, ou pour qu'ils soient purgés des autorités racines qui ne sont plus considérées comme fiables, suite à une attaque se traduisant par une corruption de la chaîne de certification, par exemple.

4. La révocation des certificats

Un certificat est émis pour une durée déterminée. Toutefois, il peut arriver que ce certificat ne soit plus sûr : il suffit que la clé privée qui lui est attachée soit divulguée pour qu'il ne puisse plus être utilisé. C'est bien le principe même des clés asymétriques : si la clé privée est détenue par quelqu'un d'autre que son propriétaire, la confidentialité des informations échangées ne peut plus être garantie.

Ainsi, les autorités de certification éditent régulièrement des listes des certificats qui sont révoqués avant leur date d'expiration (fichiers au format **crl** principalement). L'adresse où l'on peut trouver les listes de révocation est souvent intégrée dans les certificats.

Suite à des attaques informatiques de grande ampleur, il est même arrivé que le certificat racine d'une autorité de certification soit volé, ou utilisé pour générer des certificats frauduleux : ainsi, ce sont tous les certificats émis par cette autorité qui n'ont plus de valeur et qui doivent être révoqués.

5. En résumé...

Le **chiffrement asymétrique** est basé sur la mise en place d'un couple de clés (également appelé **bi-clés**) ; une information cryptée avec une clé ne peut être décryptée qu'avec l'autre. Une des deux clés est nommée la **clé privée**, l'autre la **clé publique**.

La **clé privée** est conservée précieusement par son propriétaire, et ne doit en aucun cas être divulguée.

La **clé publique** est transmise à l'ensemble des correspondants, qui l'utilisent soit pour s'assurer de l'identité de son propriétaire, soit pour lui envoyer des informations cryptées.

Pour garantir l'identité du détenteur de la clé publique, celle-ci est signée par une **autorité de certification** reconnue. La signature, ainsi que la clé publique, sont stockées dans un **certificat numérique**.

Un **certificat racine** est un certificat auto-signé par une autorité de certification, qui contient donc la clé publique de cette autorité.

1 Les serveurs web comme Apache peuvent fournir les certificats intermédiaires, à condition d'être correctement configurés (il faut qu'ils disposent d'un certificat « bundle » comprenant tous les certificats utilisés)

Pour vérifier un certificat, il faut impérativement disposer du certificat racine de l'autorité de certification, voire des certificats des autorités intermédiaires.

Un certificat a une durée de validité limitée. Il peut être révoqué avant sa date d'expiration ; des listes de certificats révoqués sont régulièrement publiées par les autorités de certification.

I. Chiffrer les accès aux bases de données

Remarque préalable : les informations décrites dans ce chapitre n'ont pas été testées...

On y pense peu : la communication entre un serveur de bases de données et un client est réalisée, par défaut, en clair. Un pirate qui enregistrerait tout ce qui transite sur le réseau pourrait, d'une part, récupérer tous les résultats des requêtes et, d'autre part, dans certaines conditions, obtenir le mot de passe de l'utilisateur.

Avec les versions récentes des SGBD, le mot de passe est toujours transmis de façon cryptée, au minimum en MD5, avec rajout systématique d'un « grain de sel » à usage unique pour la session courante, histoire de complexifier la recherche du mot de passe (cf. IV.G, *Compliquer la tâche des pirates : le salage,*page 61).

Dans les petites configurations, le serveur web est souvent hébergé dans la même machine que le serveur de bases de données : on pourrait penser que le chiffrement de la liaison entre les deux serveurs est inutile.

Il faut toutefois se rappeler que la base de données est parfois interrogée directement depuis un poste de travail, avec un outil quelconque (module de LibreOffice, par exemple). Et dans ce cas, toutes les informations circulent en clair.

1. Configurer les SGBD pour activer le mode SSL

Aujourd'hui, les SGBD peuvent être configurés pour s'appuyer sur la couche SSL, dans tous les échanges avec les clients.

Quelques références concernant l'activation du mode SSL dans les SGBD :

- pour MySQL : http://www.chicoree.fr/w/Connexion_sécurisée_à_MySQL
- Pour PostGreSQL : http://docs.postgresqlfr.org/8.3/libpq-ssl.html et http://docs.postgresqlfr.org/8.3/ssl-tcp.html, qui présente la configuration spécifique du serveur.

2. Configurer les clients pour qu'ils puissent se connecter en mode SSL

a) Utiliser le pilote JDBC PostGreSQL

En principe, il suffit d'activer le mode SSL pour que celui-ci soit pris en compte. Cela se passe en rajoutant un paramètre dans la ligne de commande[1] :

```
jdbc:postgresql://localhost/test?user=fred&password=secret&ssl=true";
```

Si le certificat utilisé par le serveur a été généré par une autorité reconnue, alors vous n'aurez probablement rien d'autre à faire. Par contre, si vous devez importer le certificat (clé publique, ou certificat de l'autorité de certification),

1 http://jdbc.postgresql.org/documentation/80/connect.html

consultez les instructions décrites ici :
http://jdbc.postgresql.org/documentation/91/ssl-client.html.

Si vous ne voulez pas vérifier le certificat du serveur, il suffit de rajouter, dans la ligne de commande du pilote, la valeur :

```
jdbc:postgresql://localhost/test?
user=fred&password=secret&ssl=true&sslfactory=org.postgresql.ssl.NonValidatingFactory
```

b) Utiliser le pilote JDBC MySQL

C'est le même principe que pour PostGreSQL : il suffit de rajouter le paramètre ad-hoc dans l'URL de connexion[1] :

```
useSSL=true
```

Les aspects concernant la reconnaissance du certificat du serveur sont identiques : il faudra intégrer la clé publique ou la clé de l'autorité de certification dans le *keystore* Java, en suivant les consignes décrites dans la documentation MySQL (cf. lien cité).

c) Utiliser d'autres mécanismes de connexion

Les outils natifs fournis directement par les éditeurs permettent de se connecter en SSL, en indiquant les certificats adéquats.

Depuis OpenOffice/LibreOffice, il vaut mieux utiliser les pilotes JDBC plutôt que les pilotes SDBC, qui ne semblent pas proposer de paramétrage spécifique pour activer le SSL.

3. Se connecter en SSL depuis PHP

Nous allons simplement regarder la configuration en utilisant ADODB[2]. Si vous utilisez une autre couche d'abstraction, consultez la documentation de celle-ci ou les paramètres possibles dans les commandes PHP.

a) PostGreSQL[3]

La connexion va pouvoir s'établir ainsi :

```
$db = NewADOConnection('host=serveur port=5432 dbname=base_de_donnees
user=login password=mot_de_passe sslmode=allow') ;
```

Si le fichier **~/.postgresql/root.crt** est présent, le programme client **libpq** va vérifier le certificat du serveur à partir des certificats intégrés à ce fichier. Dans le cas d'un serveur Apache, le dossier racine du login **www-data** est en principe **/var/www**, c'est à dire la racine du site. C'est donc dans ce dossier qu'il faudra créer le sous-dossier **.postgresql** et mettre le certificat racine.

sslmode peut prendre d'autres valeurs, dont :

- **disable** : pas de chiffrement ;
- **verify-ca** : le certificat du serveur va être vérifié ;
- **verify-full** : le certificat du serveur va être vérifié, le nom présenté dans le certificat devra correspondre au nom du serveur. C'est le mode le plus protégé.

1 http://dev.mysql.com/doc/refman/5.1/en/connector-j-reference-using-ssl.html
2 Source : http://mbrisby.blogspot.fr/2008/06/adodb-php-mysql-ssl.html
3 Source : http://www.postgresql.org/docs/9.2/static/libpq-ssl.html

Si le serveur vérifie l'identité du client (présentation du certificat par le client), le sous-dossier **.postgresql** doit contenir les deux fichiers de certificats : **postgresql.key** (clé privée, modulo 600), et **postgresql.crt** (clé publique, modulo 644).

b) MySql

Dans un premier temps, indiquez le certificat racine du serveur dans le fichier **/etc/mysql/my.cnf** (ou **/etc/my.cnf**) :

```
ssl-ca=/chemin_vers_cacert.pem
```

Puis établissez la connexion dans ADODB ainsi :

```
dsn = 'mysqli://ssluser:sslpass@dbhost/test?clientflags=2048';
$dbh = NewADOConnection($dsn);
```

J. Appendice sur la sécurité

La plupart des informations citées ici sont issues des documents présents dans le site de l'agence nationale de la sécurité des systèmes d'informations (ANSSI), et plus particulièrement depuis la page contenant ses recommandations et guides : http://www.ssi.gouv.fr/fr/bonnes-pratiques/recommandations-et-guides/

La lecture de ces documents permettra de mieux comprendre le fonctionnement général de la cryptographie.

1. Quelle longueur pour les clés de chiffrement ?

Les chiffres indiqués ici proviennent du document : http://www.ssi.gouv.fr/IMG/pdf/RGS_B_1.pdf

Quelques rappels d'ordres de grandeur :

2^n	10^n	Ordre de grandeur
2^{32}	4 294 967 296	Nombre d'hommes sur terre
2^{46}	$7{,}036874418\times10^{13}$	Distance Terre – Soleil, en millimètres
2^{55}	$3{,}602879702\times10^{16}$	Nombre d'opérations effectuées en une année à raison d'un milliard par seconde (1 Ghz)
2^{90}	$1{,}237940039\times10^{27}$	Nombre d'opérations effectuées en 15 milliards d'années à raison d'un milliard d'opérations par seconde (1 Ghz)
2^{128}	$3{,}402823669\times10^{38}$	
2^{256}	$1{,}157920892\times10^{77}$	Nombre d'électrons dans l'univers

Pour les chiffrements à clés secrètes (clés symétriques), la taille minimale des blocs doit être au minimum de 64 bits (128 bits à partir de 2020). La longueur de la clé de chiffrement doit être d'au minimum 128 bits. On considère qu'une clé d'une longueur de 256 bits ne pourra jamais être cassée par la force brute.

En chiffrement asymétrique, les règles de déchiffrement sont totalement différentes : on utilise des mécanismes de factorisation, qui sont plus facilement réalisables. La taille minimale des modules premiers, c'est à dire ceux qui sont utilisés pour la génération des clés, ne doit pas être inférieur à 2048 bits (4096 bits à partir de 2020).

2. Le paradoxe des anniversaires[1]

Question : combien faut-il regrouper de personnes pour qu'on ait 50 % de chances que 2 d'entre elles aient la même date d'anniversaire ?

La réponse, donnée par un calcul de probabilité, est de 23... Avec 80 personnes, la probabilité est de 99,99 % !

Ce paradoxe (qui contredit l'intuition) est utilisé pour le calcul des collisions dans les fonctions de hachage. Une probabilité égale à $p=\frac{1}{2}$ (50%) s'obtient, pour n bits, avec environ $2^{\frac{n}{2}}$ hachés d'éléments. Ainsi, on peut considérer que les collisions deviennent quasiment impossibles avec des clés de hachage de 256 bits, puisque la probabilité de survenue d'une collision est de 2^{128} .

3. Longueur des mots de passe

La puissance de calcul augmentant régulièrement (cf. Loi de Moore), les risques de casse d'un mot de passe augmentent également régulièrement. Si, il y a quelques années, on considérait qu'une longueur d'un mot de passe devait être d'au minimum 8 caractères, choisis dans 3 des 4 jeux disponibles (chiffres, minuscules, majuscules, caractères spéciaux), aujourd'hui, on considère que les mots de passe doivent avoir une longueur de 12 caractères pris dans les 4 jeux disponibles pour que le cassage exhaustif ne soit plus possible.

Mais, dans ce cas, le code de la carte bancaire, sur 4 chiffres, est-il sûr ?

On estime qu'un mot de passe est correct dès lors qu'il présente un risque inférieur à $1/2^{11}$ *d'être découvert (soit environ 1 sur 2000). Un code PIN à 4 chiffres, avec 3 essais possibles (et verrouillage ensuite) répond à ce critère.*

a) Verrouiller les mots de passe

Pour limiter la recherche exhaustive, on peut rajouter un blocage du mot de passe au bout de quelques essais infructueux. Néanmoins, un pirate peut décider de ne tester que quelques combinaisons, puis d'attendre que l'utilisateur rentre son mot de passe, ce qui va entraîner une réinitialisation du nombre d'essais. Ainsi, si le blocage intervient au bout de 10 essais infructueux, il est possible, pour un pirate, de tester uniquement 5 à 9 combinaisons par jour, en espérant raisonnablement que l'utilisateur va saisir son bon mot de passe tous les jours. Le pirate va pouvoir ainsi tester tous les jours quelques mots de passe judicieusement choisis, et si les règles de base de complexité des mots de passe ne sont pas respectées, aura de fortes chances de tomber sur le bon en quelques semaines.

Les systèmes d'identification proposent en général deux modes de verrouillage : un mode avec verrouillage permanent, et un mode avec déverrouillage au bout d'un certain temps.

Avec verrouillage permanent, la recherche par essais successifs est quasiment impossible, surtout si le mot de passe ne figure pas dans la liste des mots de passe les plus utilisés. Par contre, cela peut poser un problème de « déni de service » : un pirate peut très bien vouloir verrouiller les mots de passe, particulièrement ceux des administrateurs, pour bloquer les accès au système et gêner le fonctionnement de la structure attaquée.

1 http://fr.wikipedia.org/wiki/Paradoxe_des_anniversaires

Le verrouillage avec déverrouillage facilite, lui, la recherche des mots de passe : si le déverrouillage intervient au bout de 10', le pirate pourra tester 60 mots de passe à l'heure, soit 250 000 en 6 mois. Là encore, si le mot de passe est complexe, il est peu probable que celui-ci puisse être découvert. Mais s'il peut être déduit facilement (prénom de l'enfant + chiffre, par exemple, ou transformation d'une minuscule en majuscule), la probabilité de découvrir le bon mot de passe augmente fortement. Un des bons moyens pour détecter ce type d'attaque consiste à tracer tous les blocages de mots de passe, et donc de mettre en place un mécanisme qui signale tous les blocages qui se produisent. Le blocage régulier d'un même mot de passe pourra, ainsi, indiquer qu'une tentative de cassage est en cours.

Pour se prémunir contre une recherche exhaustive, et dans le cas de mots de passe aléatoires (non inclus dans un dictionnaire), l'ANSII recommande l'utilisation d'au minimum 12 caractères, comprenant les quatre types de caractères (minuscules, majuscules, chiffres ou caractères spéciaux).

b) Quelques préconisations

En résumé :

- imposez des mots de passe comprenant au minimum 3 types de caractères différents, d'un longueur minimale de 12 caractères, et 16 caractères pour les comptes sensibles (ou 12 caractères et 4 types différents) ;
- mettez en place des mécanismes de verrouillage ;
- imposez le changement du mot de passe au moins une fois tous les 3 à 6 mois.

Bien sûr, on est là dans le cas d'une attaque externe : si un pirate a accès à la base de données contenant les mots de passe, le problème est tout autre, surtout si le mécanisme utilisé pour le chiffrement est connu. Il peut alors rejouer toutes les combinaisons en très peu de temps si le mot de passe est court.

Voici les recommandations de l'ANSSI[1] :

- **R1** Utilisez des mots de passe différents pour vous authentifier auprès de systèmes distincts. En particulier, l'utilisation d'un même mot de passe pour sa messagerie professionnelle et pour sa messagerie personnelle est à proscrire impérativement.
- **R2** Choisissez un mot de passe qui n'est pas lié à votre identité (mot de passe composé d'un nom de société, d'une date de naissance, etc.).
- **R3** Ne demandez jamais à un tiers de créer pour vous un mot de passe.
- **R4** Modifiez systématiquement et au plus tôt les mots de passe par défaut lorsque les systèmes en contiennent.
- **R5** Renouvelez vos mots de passe avec une fréquence raisonnable. Tous les 90 jours est un bon compromis pour les systèmes contenant des données sensibles.
- **R6** Ne stockez pas les mots de passe dans un fichier sur un poste informatique particulièrement exposé au risque (exemple : en ligne sur internet), encore moins sur un papier facilement accessible.
- **R7** Ne vous envoyez pas vos propres mots de passe sur votre messagerie personnelle.

1 http://www.ssi.gouv.fr/fr/bonnes-pratiques/recommandations-et-guides/securite-du-poste-de-travail-et-des-serveurs/mot-de-passe.html

- **R8** Configurez les logiciels, y compris votre navigateur web, pour qu'ils ne se "souviennent" pas des mots de passe choisis.

Par rapport à la règle **R8**, si vous souhaitez tout de même enregistrer vos mots de passe quand vous surfez sur Internet ou que vous relevez vos messages, pensez à définir un mot de passe général pour votre navigateur[1] ou votre client de messagerie totalement différent des autres mots de passe que vous utilisez, et qui respecte les prescriptions précédentes. Les mots de passe que vous utilisez, les certificats que vous stockez, seront alors cryptés. N'utilisez cette technologie que dans des ordinateurs « sûrs », qui ne sont pas en libre-service, et ne divulguez jamais ce mot de passe (pas d'inscription dans un carnet, etc...).

1 Dans Firefox : Outils > Options (ou Édition > Préférences sous Linux), sécurité, et cochez : utiliser un mot de passe principal

V. Aller plus loin...

A. L'internationalisation de l'application

1. Pour quoi 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.

Il existe plusieurs techniques pour gérer les traductions. La plus simple, c'est d'utiliser un fichier de correspondance, ou fichier de langue.

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 :

```
fichier locales/fr_FR.php
<?php
$LANG= array();
$LANG["menu"][0] = "Gestion";
$LANG["menu"][1] = "Opérations de gestion";
$LANG["menu"][2] = "Liste des comptes";
$LANG["menu"][3] = "Liste des logins - identification via la base de
données";
$LANG["menu"][4] = "Administration";
$LANG["menu"][5] = "Administration de l'application";
$LANG["menu"][6] = "Déconnexion";
$LANG["menu"][7] = "Déconnexion de l'application";
$LANG["menu"][8] = "A propos";
$LANG["menu"][9] = "A propos de prototypePHP";
$LANG["menu"][10] = "Aide";
$LANG["menu"][11] = "Quelques conseils...";
$LANG["menu"][12] = "Gestion des droits";
$LANG["menu"][13] = "Gestion des droits d'accès aux modules de
l'application";
$LANG["menu"][20] = "Aide";


$LANG["message"][0] = "Bienvenue";
$LANG["message"][1] = "prototypephp - titre de l'application";
$LANG["message"][2] = "Module administration";
$LANG["message"][3] = "Module gestion";
$LANG["message"][4] = "Suppression effectuée";
$LANG["message"][5] = "Enregistrement effectué";
$LANG["message"][6] = "Vidage effectué";
$LANG["message"][7] = "Vous êtes maintenant déconnecté";
```

```
$LANG["message"][8] = "Vous n'êtes pas connecté";
$LANG["message"][9] = "Vous n'avez pas le droit d'accéder à cette
fonction";
$LANG["message"][10] = "Identification réussie !";
$LANG["message"][11] = "Identification refusée";
$LANG["message"][12] = "Problème lors de la mise en fichier...";
$LANG["message"][13] = "Problème lors de la suppression";
$LANG["message"][14] = "Vous ne pouvez pas accéder directement à
cette page";
$LANG["message"][15] = "oui";
$LANG["message"][16] = "non";
$LANG["message"][17] = "Attention :";
$LANG["message"][18] = "Confirmez-vous l'opération ?";
$LANG["message"][19] = "Valider";
$LANG["message"][20] = "Supprimer";
$LANG["message"][21] = "Rechercher";
$LANG["message"][22] = "Echec de connexion à la base de données";

$LANG["login"][0] = "Login";
$LANG["login"][1] = "Mot de passe";
$LANG["login"][2] = "Veuillez utiliser votre login du domaine pour
vous identifier";
$LANG["login"][5] = "Nouveau login";
$LANG["login"][6] = "login";
$LANG["login"][7] = "Nom - prénom";
$LANG["login"][8] = "Mél";
$LANG["login"][9] = "Nom";
$LANG["login"][10] = "Prénom";
$LANG["login"][11] = "Date";
$LANG["login"][12] = "Répétez le mot de passe";

?>
```

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.

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

3. L'intégration des libellés avec SMARTY

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

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

puis, dans les templates smarty :

```
<form method="POST" action="index.php">
          <table class="tablesaisie">
          <tr>
          <input type="hidden" name="module" value={$module}>
          <td>{$LANG.login.0} :</td><td> <input name="login"
maxlength="32"></td>
          </tr>
          <tr><td>
          {$LANG.login.1} :</td><td><input name="password"
type="password" maxlength="32"></td>
          </tr>
          <tr>
          <td><input type="submit"></td><td> <input
type="reset"></td>
          </tr>
```

```
        </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 modèle (template) **documentation/index.html**. Ce modèle va contenir au minimum la ligne suivante :

```
{$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...

B. 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.
@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.

a) Exemple

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

```
/**
 * Valide un mouvement de jeu d'Echecs.
 * @param leDepuisFile   File de la pièce à déplacer
 * @param leDepuisRangée Rangée de la pièce à déplacer
 * @param leVersFile     File de la case de destination
 * @param leVersRangée   Rangée de la case de destination
 * @return vrai(true) si le mouvement d'échec est valide ou
faux(false) si invalide
 */
 boolean estUnDéplacementValide(int leDepuisFile, int leDepuisRangée,
int leVersFile, int leVersRangée)
 {
     ...
 }
```

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

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

```
/**
 * Classe de gestion d'étudiant
 * @author John Doe
 * @version 2.6
 */
 public class Etudiant
 {
   ...
 }
```

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 :

3. Les tags de PHPDOCUMENTOR[4]

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.

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

Tag	Usage	Description
@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 ou au manuel utilisateur... Elle peut également être complétée par une documentation générale décrivant le fonctionnement global de l'application.

C. Gérer les différentes versions de l'application

Avec des applications en mode « client lourd », comme les traitements de textes, une nouvelle version se traduit par l'exécution d'un programme de mise à jour. Avec les applications web, la mise à jour s'effectue sur le serveur directement.

En général, les versions successives des applications sont stockées dans un dossier, dont le nom contient le numéro de version : smarty-2.6.26, par exemple. Sur un serveur web, plutôt que d'accéder à une application en décrivant le chemin complet (par exemple, http://monserveur.masociete.com/monappli-1.0.1/), il est fréquent de mettre en place une redirection et d'accéder à l'application par un lien direct : http://monappli.masociete.com.

Cette redirection s'effectue en deux étapes :

- d'une part, une entrée DNS est créée (un nom de site web est déposé),
- d'autre part, un alias est créé sur le serveur web.

Avec Apache, le fichier contenant l'alias a souvent la forme suivante :

```
<VirtualHost *:80>
ServerName monappli.masociete.com
DocumentRoot /var/www/html/monappli-1.0.1
</VirtualHost>
```

Lors de la mise en production de la nouvelle version, il est alors possible de modifier le fichier d'alias pour le faire pointer vers le dossier qui la contient, par exemple `monappli-1.1.0` : il suffit simplement de changer la directive `DocumentRoot`.

Cela permet ainsi de disposer de plusieurs versions sur le serveur, mais seule celle qui est en production est accessible par les internautes.

Dans un certain nombre de cas, les administrateurs systèmes préfèrent stocker l'application dans une arborescence différente de celle de base du serveur, par exemple /var/lib plutôt que /var/www. Cela limite les risques d'accès indésirables.

L'autre approche, sous Linux, consiste à créer un lien vers le dossier contenant la version :

```
ln -s /var/www/monappli-1.0.1 /var/www/monappli
```

L'accès à l'application sera réalisé en pointant vers monappli. Lors du changement de version, il suffit de supprimer le lien, puis de le recréer vers la nouvelle version.

D. Les avancées du HTML et du JAVASCRIPT

Le langage HTML a été créé pour afficher simplement des informations, et sa grande force était de promouvoir la notion de lien hypertexte, à la base du « surf ».

Néanmoins, ces possibilités étaient, au départ, très limitées, les interface de saisie ou d'affichage étant très frustres.

Heureusement, HTML a été complété par de nouvelles fonctionnalités, comme la gestion des blocs ou les feuilles de style CSS (Cascading Style Sheets, feuilles de style en cascade, en français), et un moteur d'exécution de code, qui fonctionne sur le navigateur, a été rapidement implémenté, le Javascript.

Grâce à ces innovations, les interfaces web ont gagné en qualité, et sont même capable de rivaliser avec nombre d'applications développées en client lourd : il existe aujourd'hui des applications de gestion de courrier électronique qui gèrent le clic droit de la souris, le glisser-déplacer... comme le font les applications traditionnelles comme Thunderbird, par exemple.

Quel que soit l'outil mis en place du côté du navigateur, il ne faut pas perdre de vue que le programmeur, de par la conception même des applications web, n'a pas la possibilité de maîtriser complètement l'information qui est transmise. Ainsi, si des contrôles de cohérence des données peuvent être réalisés en Javascript sur le navigateur, ils devront impérativement être rejoués sur le serveur pour s'assurer que l'utilisateur n'a pas contourné les tests réalisés.

E. Regrouper les pages action en une seule

Prenons l'affichage de la liste des bénéficiaires d'un dossier. L'application devra permettre les opérations suivantes :

- afficher la liste ;
- afficher le détail d'un bénéficiaire ;
- passer en mode modification avec l'affichage d'un formulaire ;
- enregistrer les modifications ;
- supprimer une fiche.

Rien que pour gérer cet aspect là du programme, une première approche nécessite la création de cinq pages PHP :

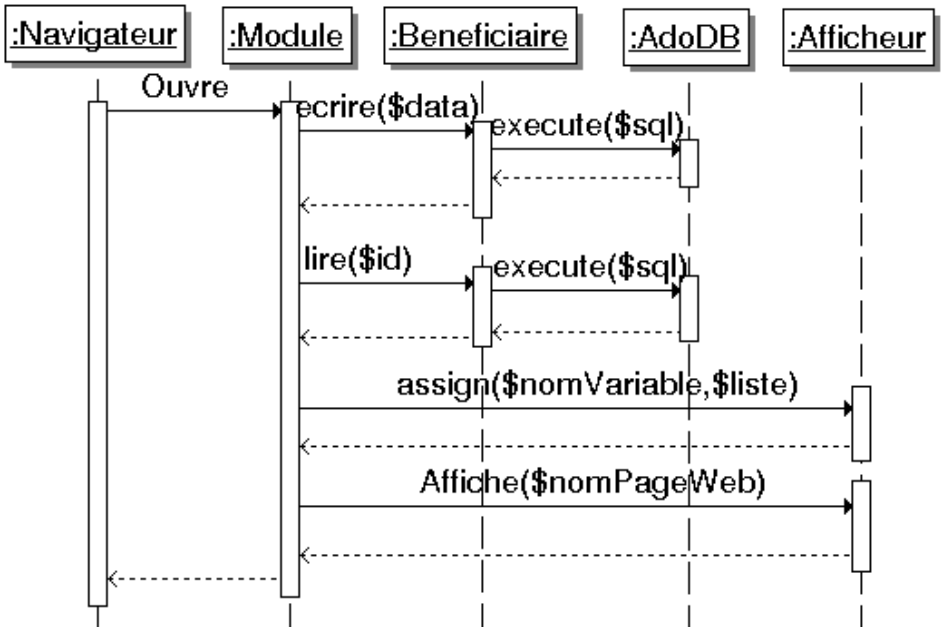
- BeneficiaireListe.php
- BeneficiaireDetail.php
- BeneficiaireModif.php
- BeneficiaireEcrire.php

- BeneficiaireSupprimer.php

Nous allons essayer de limiter le nombre de pages, pour simplifier la structure de notre application.Pour cela, reprenons l'analyse UML de ces modules.

Notre première page sert à afficher la liste des bénéficiaires :

Maintenant, intéressons-nous à l'affichage du détail d'un bénéficiaire. Le diagramme de séquence intègre maintenant trois actions, récapitulées dans un seul module : l'affichage du détail, la saisie des modifications (affichage d'une page contenant le formulaire de saisie), et la mise en fichier, suivie d'un réaffichage du détail.



Le choix du traitement est réalisé par l'utilisation d'une variable `action`, qui va permettre de savoir ce qui doit être réalisé. Le code correspondant à cette page va être maintenant le suivant :

```
<?php
include_once('common.inc.php');
include_once('beneficiaire.class.php');
$beneficiaire = new Beneficiaire($bdd);
/*
 * Gestion de la mise en fichier
 * La classe Beneficiaire preparera les donnees fournies
 * dans le tableau $_REQUEST avant d'effectuer
 * l'operation d'ecriture
 */
if ($_REQUEST['action'] == 'ecrire') {
          $beneficiaire->ecrire($_REQUEST);
}
$detail = $beneficiaire->lire($_REQUEST['IdBenef']);
$smarty->assign ('beneficiaire',$detail);
// choix du gabarit a afficher
if ($_REQUEST['action'] == 'modif') {
          $template = 'BeneficiaireModif';
} else {
          $template = 'BeneficiaireDetail';
}
$smarty->display($template);
```

```
?>
```

Il est également possible de gérer la suppression de la fiche, mais il faut maintenant tenir compte du fait qu'il n'est plus possible de réafficher la fiche une fois qu'elle a été effacée. En général, le programme va alors appeler l'affichage de la liste.

Attention : le regroupement n'a de sens que si les droits sont identiques entre chaque fonction. Avec le modèle MVC, on préfère aujourd'hui créer des modules séparés pour l'affichage, la modification, et la suppression. C'est parfois un peu plus long à développer, mais la gestion des droits est alors beaucoup plus simple, et on maîtrise mieux ce qu'il se passe.

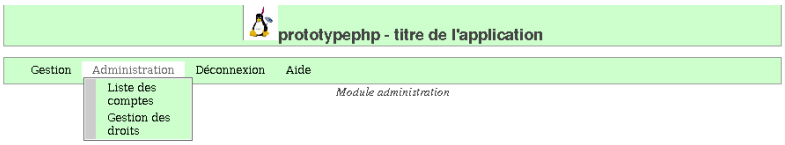
F. Générer des menus avec les feuilles de styles

Il y a quelques années, gérer les menus dans une page HTML était complexe. De nombreux scripts écrits en Javascript permettaient de les générer, avec fenêtres popups et effets variés.

Aujourd'hui, la solution la plus simple (une fois qu'elle est mise en place) consiste à s'appuyer sur les feuilles de styles CSS.

Le CSS (Cascading Style Sheet, feuilles de style en cascade) est un langage qui permet de décrire l'affichage de documents HTML ou XML, par l'intermédiaire de styles définissant comment l'information doit être présentée. Nous allons voir comment gérer un menu en s'appuyant sur les mécanismes que le CSS propose.

Notre menu va être réalisé en préparant une liste non ordonnée (balise html <ul>). Nous allons prendre comme exemple le menu général de l'application prototypePhp (html://prototypephp.sourceforge.net). Voici à quoi ressemble le menu une fois formaté :



06e04.png

Ce menu a comme caractéristique de proposer quatre grande rubriques principales, à savoir Gestion, Administration, Déconnexion, Aide. Deux sous-menus sont disponibles sous les rubriques Administration et Aide.

Ces menus sont constitués de listes HTML imbriquées. Ces listes (listes non ordonnées) sont créées avec les balises suivantes :

- **<ul>**, qui délimite la liste
- **<li>**, qui délimite chaque item de la liste.

Les sous-menus sont réalisés en intégrant une nouvelle liste dans une balise <li> :

```
<link rel="stylesheet" href="CSS/proto.css" type="text/css">
<div class="menu">
<ul>
<li id="gestion"><a href="index.php?module=gestion" title="Opérations
de gestion">Gestion</a></li>
<li id="administration"><a href="index.php?module=administration"
title="Administration de l'application">Administration</a>
```

```
<ul>
<li id="loginliste"><a href="index.php?module=loginliste"
title="Liste des logins - identification via la base de
données">Liste des comptes</a></li>
<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>
</ul>
</li>
<li id="identdisconnect"><a href="index.php?module=identdisconnect"
title="Déconnexion de l'application">Déconnexion</a></li>
<li id="docindex"><a href="index.php?module=docindex" title="Utiliser
PrototypePHP">Aide</a>
<ul><li id="docinstallation"><a href="index.php?
module=docinstallation" title="Installer
PrototypePHP">Installation</a></li>
<li id="docdroits"><a href="index.php?module=docdroits" title="Gérer
les droits dans l'application">Gestion des droits</a></li>
<li id="docmodulelecture"><a href="index.php?module=docmodulelecture"
title="Créer un module pour lire ou écrire un enregistrement">Module
de lecture-écriture</a></li>
<li id="docparammodule"><a href="index.php?module=docparammodule"
title="Configurer le module dans le fichier xml">Paramétrage du
module</a></li>
<li id="docclasseaccess"><a href="index.php?module=docclasseaccess"
title="Créer la classe permettant d'accéder à une table">Classe
d'accès aux données</a></li>
<li id="docgestiondoc"><a href="index.php?module=docgestiondoc"
title="Gérer les pages de la documentation">Gestion de la
documentation</a></li>
<li id="docinternationalisation"><a href="index.php?
module=docinternationalisation" title="Gestion des différentes
langues, internationalisation de l'application">Langues</a></li><li
id="docdescriptionorganisation"><a href="index.php?
module=docdescriptionorganisation" title="Organisation de
PrototypePHP">Structure</a></li>
<li id="apropos"><a href="index.php?module=apropos" title="A propos
de prototypePHP">A propos</a></li>
</ul>
</li>
</ul>
</div>
```

Et voici ce qui serait affiché, si aucune feuille de style n'était définie :

06e05.png

Tout l'enjeu est maintenant d'afficher notre liste pour la transformer en menu dynamique.

Nous l'avons vu, notre menu commence par une balise , qui définit la classe menu. Cette classe va être déclarée dans notre feuille de style `proto.css` :

```
div.menu {
  border: #999999 1px solid;
  background: #ccffcc;
  font-size: 12px;
  padding:0.5em;
  margin-left: 15%;
  margin-right: 15%;
}
div.menu:after {
  content: "";
  clear: both;
  display: block;
}
```

Les arguments indiqués permettent de définir une bordure (border), la couleur de fond (background), la taille de la police (font-size), l'espace intérieur entre la marge et le texte (padding), et les marges gauches et droites (margin-left, margin-right), le menu n'occupant, dans cet exemple, que 70 % de la largeur de l'écran du navigateur.

La balise `after` permet de définir que l'affichage de ce qui suit la balise `<div>` sera réalisée dans un bloc, et les hauteurs recalculées.

La structure de notre menu ressemble à ceci, une fois enlevé le texte affiché :

```
<ul>
          <li>
                    <ul>
                              <li> </li>
                              <li> </li>
                    </ul>
          </li>
          <li> </li>
</ul>
```

Le menu principal doit être horizontal, les sous-menus seront verticaux. Nous allons utiliser les capacités d'enchaînement du CSS pour décrire le comportement de chaque balise. Nous allons combiner deux mécanismes, l'enchaînement parent>enfant, et les sélecteurs descendants. Une balise enfant est définie par le symbole >, les sélecteurs descendants étant simplement séparés par un espace du sélecteur précédent.

Voici l'ensemble des éléments de la feuille de style utilisés pour afficher notre menu :

```
div.menu > ul {
  list-style-type: none;
  padding: 0;
  cursor: pointer;
  margin: 0 0 0 1em;
}
div.menu > ul span {
  margin: 0;
  padding: 0 1em 2px 1em;
}
div.menu > ul > li {
  margin-top: -2px;
```

```
  float: left;
}
div.menu > ul li:hover span {
  background: #FFFFFF;
    color: #ccffcc;
}
div.menu > ul li ul {
  border: 1px solid;
  border-color: #666;
  margin: -1px 0 0 2px;
  padding: 1px;
  width: 10em;
  height: auto;
  background: #FFFFFF;
    color: #ffffff;
  list-style-type: none;
  position: absolute;
  display: none;
}
div.menu > ul li:hover ul {
  display: block;
}
div.menu > ul li ul li {
  margin: 0;
  padding-left: 1px;
  background: #ccffcc;
  border-left: solid 1em #CCC;
    color: #ffffff;
}
div.menu a:link, div.menu a:visited, div.menu a:active {
  color: #000000;
  display: block;
  margin: 0;
  padding: 2px 1em 2px 1em;
}
div.menu a:hover {
  background: #FFFFFF;
    color: #666666;
}
```

a) Le code permettant d'extraire les informations nécessaires pour gérer le menu

```
          /**
           * Fonction lisant l'arborescence sur 2 niveaux du fichier
xml
           * Lecture depuis la racine du fichier xml, des noeuds de
niveau 1
           * et des attributs associes
           *
           * @param string $racine
           * @return array
           */
          function lireGlobal() {
                    $root = $this->dom->getElementsByTagName($this-
>dom->documentElement->tagName);
                    $root = $root->item(0);
                    $noeuds = $root->childNodes;
                    $g_module = array();
                    foreach ($noeuds as $noeud){
                              // Exclusion du modele
```

```
                                if ($noeud->hasAttributes()&&$noeud-
>tagName<>"model") {
                                        foreach ($noeud->attributes
as $attname=>$noeud_attribute) {
                                                $g_module[$noeud-
>tagName][$attname] = $noeud->getAttribute($attname);
                                        }
                                }
                        }
                        return $g_module;
                }

/**
 * Fonction preparant un tableau multi-niveaux, contenant tous les
items
 * necessaires pour generer un menu a partir du fichier xml.
 * Gere 2 niveaux de menu
 * Le tableau recupere doit ensuite etre trie (via ksort), et les
droits
 * verifies (menuloginrequis et menudroits)
 *
 * @return array
 */
                function genererMenu() {
                        $gmenu = array();
                        foreach ($this->g_module as $key => $value ) {
                                if (isset($value["menulevel"])) {
                                        // Recuperation des
informations sur le menu
                                        $menu = array();
                                        // print_r($value);
                                        $menu["menuvalue"] =
$value["menuvalue"];
                                        $menu["module"] = $key;
                                        if
(isset($value["menudroits"]))
                                        $menu["menudroits"] =
$value["menudroits"];
                                        if
(isset($value["menuloginrequis"]))
                                        $menu["menuloginrequis"] =
$value["menuloginrequis"];
                                        $menu["menutitle"] =
$value["menutitle"];
                                        // Recherche si on est en
menu principal ou secondaire
                                        if ($value["menulevel"]==0)
{
                                                foreach ($menu as
$key1 => $value1) {

$gmenu[$value["menuorder"]][$key1]=$value1;
                                                }
                                        } else {

$gmenu[$value["menuparent"]][$value["menuorder"]]=$menu;
                                        }
                                }
                        }
                        return $gmenu;
```

```
        }
```

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.

b) Le code générant la liste des menus

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

          // Preparation menu niveau 0
          if ($ok) {
                    $menu .= '<li id="'.$value["module"].'"><a
href="index.php?module='.$value["module"].'" title="'
                    .$LANG["menu"][$value["menutitle"]].'">'.
$LANG["menu"][$value["menuvalue"]].'</a>';
                    $flag=0;
                    // Gestion sous-menu
                    ksort($value);
                    foreach($value as $key1 => $value1){
                              if (is_array($value1)) {
                                        $ok1 = true;
                                        // Verification des droits
                                        if
($value1["menuloginrequis"]==1 && !isset($_SESSION["login"])) $ok1 =
false;
                                        if
(strlen($value1["menudroits"])>1&& !$phpgacl->acl_check($GACL_aco,
$value1["menudroits"],$GACL_aro,$_SESSION["login"]))
                                        $ok1 = false;
                                        if ($ok1) {
                                                  // Preparation du
sous-menu
                                                  if ($flag==0) {

$menu .="<ul>";
                                                            $flag =
1;
                                                  }
                                                  $menu .= '<li
id="'.$value1["module"].'"><a href="index.php?module='.
$value1["module"].'" title="'
```

```
                                                                .$LANG["menu"]
[$value1["menutitle"]].'">'.$LANG["menu"]
[$value1["menuvalue"]].'</a></li>';
                                              }

                                    }
                        }
                        if ($flag == 1) $menu .="</ul>";
                        $menu .="</li>";
          }
}
$menu .="</ul>";
?>
```

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...

c) 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 :

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

Et, dans la page SMARTY :

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

G. Imprimer des documents depuis l'application

Les applications informatiques stockent les informations sous forme numérique. Il y a toutefois des moments où il devient nécessaire d'imprimer un courrier, un contrat, une facture... L'impression depuis un navigateur n'est en général pas de grande qualité, la mise en forme ne pouvant être vraiment respectée.

Lors de la réalisation d'impressions formatées, deux cas se présentent. Soit le document n'a pas vocation à être modifié, soit il doit être retravaillé dans un éditeur de texte. Par exemple, un contrat ou un courrier peuvent être préparés depuis l'application, celle-ci fournissant les informations techniques issues de la base de données. Ils seront ensuite complétés manuellement par l'utilisateur pour les adapter à un contexte particulier.

Dans le premier cas, l'impression est dans la plupart des situations réalisée en recourant au format PDF. Dans le second cas, l'application génère un fichier dans un format de traitement de texte, qui sera ensuite modifié manuellement.

Pour générer des fichiers PDF, deux classes sont actuellement largement répandues : fpdf[1] et tcfpdf[2]. Pour générer des fichiers ODF, on pourra recourir à odtphp[3] pour générer des fichiers ODT (traitement de texte), qui permet

1 http://www.fpdf.org/
2 http://www.tecnick.com/public/code/cp_dpage.php?aiocp_dp=tcpdf
3 http://www.odtphp.com)

notamment de remplir des tableaux, ou OpenOffice_generation_document[1] pour créer des fichiers ODS (feuille de calcul).

Quant à l'insertion de graphiques, si de nombreuses classes existent, très peu sont réellement opensource et libres de droits. Une des plus fréquentes est pChart[2].

H. Les services Web

Dans une entreprise, il est rare qu'une application fonctionne de manière totalement autonome. Il est fréquent que la base d'identification des utilisateurs soit utilisée tant pour l'ouverture des sessions Windows que pour l'accès aux différents logiciels, et que les droits d'accès soient gérés de façon unique.

Certains processus peuvent également être mutualisés entre différents logiciels. Cela pourrait être le cas, par exemple, de la vérification de la validité d'un relevé d'identité bancaire ou d'une adresse postale : plutôt que de confier ce contrôle à chaque application, c'est une application unique au sein de l'entreprise qui serait chargée de cette tâche. La maintenance en est facilitée : toute erreur est corrigée pour l'ensemble des applications, toute évolution est immédiatement disponible.

Plusieurs techniques différentes sont utilisées pour faire dialoguer les serveurs et les applications entre eux. Parmi celles-ci, on retrouve CORBA, SOAP, et, plus récemment (bien qu'il s'agisse de la technologie originelle du web), REST.

La norme CORBA (Common Object Request Architecture) a été créée en 1992. Largement implémentée, elle a été utilisée pour faire dialoguer entre elles différentes applications, ou pour créer des applications à partir de de différentes briques bâties selon ce principe. Elle est surtout utilisée sur les gros systèmes.

SOAP (au départ, acronyme de Simple Object Access Protocol – protocole simple d'accès à des objets), est devenu un standard du W3C en 2003, dans sa version 1.2. Ce protocole est basé sur des échanges réalisés sur le web, tant en HTML qu'en SMTP, les données transitant dans un format XML normalisé. Le protocole prévoit également la mise en œuvre d'annuaires de services.

PHP5 intègre aujourd'hui des fonctions natives de gestion des échanges, grâce à l'extension SOAP, qui propose différentes classes natives pour gérer les échanges.

Comme les échanges de données s'effectuent en XML, il est parfois reproché à SOAP d'être un peu trop « verbeux », surtout si le volume de données à échanger est important.

REST (Representational State Transfert) n'est pas une norme à proprement parlé, mais plutôt un concept architectural. Il part du principe que toute information est une ressource (une image, une base de données, mais également le détail d'une fiche...). Il s'agit non seulement d'utiliser le protocole HTTP pour accéder à une ressource, mais également pour en créer une nouvelle, la modifier ou la supprimer. De nombreuses sociétés proposent des services web utilisables selon ce protocole : c'est le cas par exemple de Yahoo, qui fournit un service de géo-localisation pour les Etats-Unis[3].

REST est basé sur le principe que « tout est ressource », et que toute information peut être obtenue par l'intermédiaire d'une requête HTTP.

1 http://membres.lycos.fr/tafelmak/,
2 http://pchart.sourceforge.net/)
3 http://developer.yahoo.com/maps/rest/V1/geocode.html

Depuis un navigateur, l'appel à une ressource, par exemple une image, va être réalisé par l'intermédiaire d'une commande HTTP GET. Il en est de même pour récupérer le détail d'une fiche ou une liste de personnes.

S'il doit réaliser une mise à jour ou une création, le programmeur va très certainement envoyer son formulaire avec la commande POST vers le serveur.

En fait, HTTP gère d'autres commandes. Quatre sont utilisées généralement avec REST :

- GET : lit une information (une fiche, une liste...) ;
- POST : crée une nouvelle ressource (une nouvelle fiche, par exemple) ;
- PUT : met à jour une ressource (modification d'une fiche) ;
- DELETE : supprime une ressource (suppression d'une fiche).

Il est ainsi possible d'échanger des messages entre deux applications en utilisant ce mécanisme. En reprenant l'exemple précédent, pour obtenir notre racine carrée, il suffirait d'envoyer un message HTTP qui pointerait vers une page dédiée à cet effet, par exemple getRacineCarree.php, grâce à la classe PHP HttpRequest().

Cette page getRacineCarree.php devrait alors analyser la requête PHP, et renvoyer ensuite le résultat du traitement, comme le ferait n'importe quel service web.

Cette technologie est de plus en plus utilisée, en raison notamment de sa simplicité, même si le codage est un peu plus complexe (des exemples de classes tant pour le côté serveur que le côté client foisonnent sur le web).

I. Quelques conseils d'ergonomie[1]

Une application web, ce n'est pas un site... Un site web doit être « joli » et accrocheur, pour conserver le visiteur. Une application web n'a besoin que d'être esthétique. Par contre, elle doit être efficace, pour permettre aux utilisateurs de travailler rapidement. Elle doit également être intuitive, même s'il est possible de former les utilisateurs.

D'une manière générale, il vaut mieux conserver l'ergonomie générale du web et éviter de copier l'interface Windows. Cela peut se traduire ainsi :

- une page web est toujours déconnectée, peut être abandonnée, ou on peut y accéder par l'intermédiaire d'un raccourci. Cela implique que les informations qui sont saisies doivent être conservées, quel que soit le contexte, et notamment lorsque la saisie impose de nombreuses pages successives. De même, il est impossible de prévoir à l'avance par où va passer l'utilisateur pour accéder à une information ;
- l'ergonomie générale doit être de la forme : critères de sélection, liste de résultats, accès au détail, modification, suppression, ou rajout d'un élément ;
- privilégiez le mode « chemin de fer » quand c'est possible, c'est à dire une présentation qui serait de la forme :

Critères > liste > détail fiche > modification

ce qui permet à l'utilisateur de savoir où il se trouve, et de revenir facilement à une situation antérieure. On retrouve d'ailleurs là les mécanismes présentés dans les diagrammes de séquence dans ce document.

- Utilisez de préférence les formulaires en mode « GET », pour que la fonctionnalité RETOUR du navigateur puisse fonctionner. Ne réservez le mode

1 La plupart des informations sont issues du livre blanc : *Conception d'applications web : efficacité et utilisabilité*, édité par SMILE (http://www.smile.fr/Livres-blancs/Culture-du-web/Conceptions-d-applications-web)

« POST » que pour les enregistrements de fiches, et dès lors que le retour arrière peut avoir des répercussions gênantes dans l'application ;

- ne multipliez pas les fenêtres : au pire, utilisez des fenêtres popup pour sélectionner des informations particulières concernant la fiche principale. Seule exception : les pages d'aide, qui ont tout intérêt à être séparées de l'application ;
- en mode saisie, pensez que l'utilisateur va manipuler son clavier : complétez les listes déroulantes avec des saisies au clavier, plus rapides... Ainsi, plutôt que de chercher dans une liste déroulante une commune, autant pouvoir la saisir au clavier, et l'associer ensuite à la fiche correspondante dans la base de données ;
- Une page, une information...
- utilisez tout l'écran : ne mettez pas de marge à gauche ou à droite. Si l'utilisateur dispose d'un grand écran, il ne pourra pas l'utiliser, ce qui serait dommage...
- si le scrolling vertical est un grand classique, l'ascenseur horizontal est à bannir.
- Préférez les liens aux boutons : leur symbolique peut échapper à l'utilisateur occasionnel.
- Pour les saisies de dates, utilisez une zone de texte couplée à un calendrier. Ne découpez pas la date en trois zones, c'est une catastrophe d'un point de vue ergonomique, même si certains outils, comme Smarty, le propose.
- Pour les saisies multiples (plusieurs lignes rattachées à un dossier), soit vous réalisez la saisie dans une page dédiée, soit vous utilisez un formulaire situé sous la liste, à condition qu'il n'y ait pas trop d'informations à saisir.
- Évitez d'utiliser trop de Javascript : conservez simplement le code utile, évitez les fioritures.
- Conservez les liens hypertextes soulignés, avec le code couleur par défaut bleu/violet (ou noir/gris) : l'utilisateur ne sera pas dérouté, et pourra identifier immédiatement les liens dans la page ;
- ne touchez pas aux tailles de caractères (sauf cas particulier, comme le pied de page, par exemple). L'utilisateur doit pouvoir augmenter ou diminuer la taille d'affichage, en fonction de son poste de travail, de sa vue, …
- si vous utilisez des onglets, considérez que les onglets correspondent à une page web indépendante. Ne gérez pas les onglets dans une seule page, qui contiendrait tout le contenu du dossier (code DHTML). Chaque onglet doit pouvoir être enregistré de façon indépendante. L'utilisateur doit également savoir qu'il doit enregistrer chaque onglet, avant de passer au suivant (bouton « enregistrer ») ;
- Si vous le pouvez, utilisez un graphiste web pour vous aider à générer votre modèle d'application : les bons développeurs ne sont pas forcément des bons graphistes...

On pourrait continuer longtemps mais, pour résumer : faites simple, évitez les fioritures ou les « astuces », mettez vous à la place de l'utilisateur. Pensez également que tout ce qui est complexe à écrire est complexe à maintenir.

VI. 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).

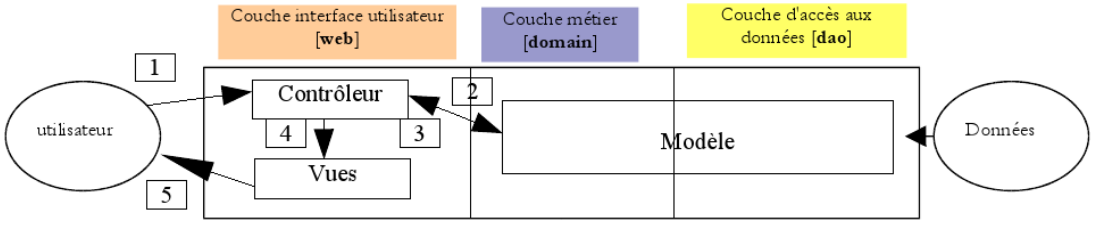


Illustration 1: Le modèle vue-contrôleur. Source : http://tahe.developpez.com/web/php/mvc/?page=introduction

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

a) Le fichier XML

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

b) La classe de manipulation du fichier XML

```
<?php
/** Fichier cree le 10 juil. 08 par quinton
 *
 *UTF-8
 */

/**
 * @class Navigation
 * @author Eric Quinton
 *
 */
class Navigation {
          var $doc;
```

```
          var $module;
          var $nommodule = "";
          var $t_module = array();


          /**
           * Fonction de creation de la classe
           * lecture du fichier xml contenant la navigation de
l'application
           *
           * @param string $nomfichier
           * @return Navigation
           */
          function Navigation($nomfichier) {
                    $this->doc = new DOMDocument();
                    $this->doc->load($nomfichier);
          }
          /**
           * Retourne un tableau contenant tous les attributs du
module
           *
           * @param string $module
           * @return array
           */
          function getModule($module) {
                    $this->lireModule($module);
                    return $this->t_module;
          }
/**
 * Fonction lisant les attributs du module, et les restituant
 * dans le tableau t_module
 * Operation effectuee que si le module n'a pas encore ete charge
 *
 * @param string $nommodule
 */
          function lireModule($nommodule) {
                    if ($this->nommodule<>$nommodule) {
                              $this->module = $this->doc-
>getElementsByTagName($nommodule);
                              $this->t_module = array();
                              foreach ($this->module as $noeud){
                                        if ($noeud->hasAttributes())
{
                                                  foreach ($noeud-
>attributes as $attname=>$noeud_attribute) {
                                                            $this-
>t_module[$attname] = $noeud->getAttribute($attname);
                                                  }
                                        }
                              }
                              $this->nommodule = $nommodule;
                    }
          }
          /**
           * Fonction retournant la valeur d'un attribut pour le
module considere
           *
           * @param string $nommodule
           * @param string $attribut
           * @return string
           */
```

```
        function getAttribut($nommodule, $attribut){
                $this->lireModule($nommodule);
                return $this->t_module[$attribut];
        }
}

?>
```

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.

c) 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.

Voici maintenant l'utilisation dans le modèle Smarty :

```
<li id="critereliste"><a href="index.php?module=critereliste"
title="Consultations par groupe ou par imprimante">Consultations
globales</a></li>
<li id="listeindividuel"><a href="index.php?module=listeindividuel"
title="Consultations personnelles">Consultations
individuelles</a></li>
<li id="administration"><a href="index.php?module=administration"
title="Opérations d'administration">Paramétrage</a>
<ul>
<li id="ldapinsertform"><a href="index.php?module=ldapinsertform"
title="Intégration des groupes et des logins depuis l'annuaire
LDAP">Import LDAP</a></li>
```

```
<li id="printerlist"><a href="index.php?module=printerlist"
title="Affichage de la liste des imprimantes">Saisie des
imprimantes</a></li>
<li id="queuelistenonaffecte"><a href="index.php?
module=queuelistenonaffecte" title="Liste des files d'impression non
affectées">Files non affectées</a></li>
<li id="purgeform"><a href="index.php?module=purgeform"
title="Suppression des éditions anciennes">Purge</a></li>
<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>
</ul>
</li>
<li id="identdisconnect"><a href="index.php?module=identdisconnect"
title="Déconnexion de l'application">Déconnexion</a></li>
<li id="apropos"><a href="index.php?module=apropos" title="A propos
de JASMINE2">A propos</a>
<ul>
<li id="aide"><a href="index.php?module=aide" title="Quelques
conseils...">Aide</a></li>
</ul>
</li>
</ul>
</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.

a) 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.

b)

B. Les URL conviviales

Nous l'avons vu, l'appel à un module va être réalisé systématiquement en utilisant le code :

```
http://localhost/proto/index.php?module=listePersonne
```

Pour simplifier l'écriture du lien, et le rendre plus facile à manipuler, nous allons le modifier ainsi :

```
http://localhost/proto/module-listePersonne
```

Bien évidemment, cette page n'existe pas... Nous allons utiliser une fonction d'Apache, qui permet de redéfinir les adresses demandées.

Pour cela, créons le fichier .htaccess à la racine de notre application, qui contiendra les lignes suivantes :

```
RewriteEngine on
RewriteRule ^module-(.*)&*(.*)$ /proto/index.php?module=$1&$2
```

Il faut auparavant avoir activé le mode Rewrite dans Apache (sous Ubuntu ou Debian) :

```
a2enmod rewrite
service apache2 restart
```

Cette méthode permet également de fournir des variables complémentaires. Nous pourrions ainsi taper le lien suivant :

```
http://localhost/proto/module-lirePersonne&id=5&type=I
```

qui serait équivalent à :

```
http://localhost/proto/index.php?module=lirePersonne&id=5&type=I
```

Ces URL sont plus faciles à manipuler et permettent de masquer en partie le fonctionnement global de l'application. D'autres redirections sont possibles, les possibilités de configuration sont étendues.

C. 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.

VII. Les outils pour développer

A. Introduction

Quand on apprend un langage de programmation, un simple éditeur de texte (voir évolué comme Notepad ++) suffit. Mais dès que le programme prend de l'ampleur, il est largement préférable de recourir à des outils adaptés, parfois un peu plus complexes à utiliser, mais beaucoup plus riches et qui vont faciliter le travail.

Si on pense tout de suite à un éditeur de code, les besoins se font vite ressentir d'abord pour modéliser la base de données, puis pour dessiner les schémas UML. Dans un second temps, surtout si on travaille à plusieurs, des outils plus sophistiqués peuvent rendre de grands services, comme les gestionnaires de version, les outils de test, voir les outils de gestion de projets ou de suivi des incidents (bugs).

B. Logiciel libre ou propriétaire ?

… qu'il ne faut pas traduire par : gratuit ou payant...

Quels sont les avantages des produits propriétaires ?

- ils sont gérés par une société, qui a tout intérêt à les faire évoluer, pour pouvoir vendre des versions et corriger les bogues ;
- le support est en principe plus facile à obtenir.

Mais :

- comme ils sont détenus par une société, ils peuvent être abandonnés (souvent suite à un rachat d'entreprise, d'ailleurs), ce qui laissera les clients dans une certaine perplexité ;
- si la société va mal, là aussi, le support peut s'en ressentir ;
- l'innovation est portée uniquement par la société, ce qui réduit les capacités d'évolution si les ressources ne sont pas suffisantes.

Les logiciels libres, eux, sont réputés pour :

- la capacité d'innovation : la communauté de développement est virtuellement illimitée, toute personne ayant besoin d'une nouvelle fonctionnalité peut l'intégrer au produit ;
- si le produit disparaît ou est abandonné, en général, des moulinettes sont prévues pour transférer les données dans un autre logiciel, plus récent et plus performant ;
- leur interopérabilité, liée à l'utilisation de standards, à la disponibilité du code source, ou à la publication des informations adéquates pour pouvoir interagir facilement avec le produit.

Mais :

- le support est souvent dépendant de la communauté des utilisateurs ; les réponses ne sont pas parfois aussi rapides que l'on peut l'espérer. Toutefois, pour les produits les plus fréquents, il est souvent possible de négocier des contrats de support avec soit la société qui gère le développement du produit, soit avec des sociétés de services spécialisées.

Nous nous intéresserons ici qu'aux logiciels sous licence OpenSource.

Attention également aux versions gratuites de certains logiciels propriétaires (éditions « communautaires ») : non seulement elles sont bridées en terme de

fonctionnalités (ce n'est pas forcément très gênant), mais souvent également en nombre d'items pouvant être gérés ; dans ce cas de figure, l'outil est inutilisable !

C. Les outils indépendants les uns des autres

1. Les éditeurs

a) Éditeurs de code

Pour éditer le code, Eclipse[1] est, sans conteste, le produit le plus complet et le plus utilisé.

Doté de nombreux plugins, il permet de gérer toutes les phases de codage, y compris la gestion de version (CVS) ou la génération de documentation.

L'inconvénient majeur d'Eclipse tient à sa gestion de versions du produit lui-même : il est très compliqué de faire cohabiter dans le même logiciel les composants pour un développement Java ou pour un développement PHP. De plus, à chaque nouvelle version d'Eclipse, il faut attendre que les plugins soient adaptés à celle-ci pour pouvoir les utiliser. Enfin, la plupart des modules étant sous forme de plugins, il n'est pas rare de rencontrer des régressions de fonctionnement de certains modules, pas forcément corrigées dans la version courante d'Eclipse.

Dans la pratique, il n'est pas rare de devoir disposer de plusieurs versions dans son poste de travail, surtout si on gère des projets dans différents langages.

Pour réaliser des projets en PHP, il est conseillé d'installer la version Eclipse-php[2], associée (au moins) avec les outils complémentaires suivants :

- **smartypdt**[3], qui rajoute des fonctions d'édition des fichiers *tpl smarty* ;
- **Doxigen**[4], qui permet de générer automatiquement la documentation technique de l'application (classes, etc.)
- et de configurer le serveur PHP pour qu'il supporte le mode **Zend Debug** (cf. VII.E, *Configurer Eclipse pour activer le mode debug*, infra).

b) Éditeurs UML

La solution **Rational Rose**, connue depuis les débuts d'UML, est une solution complète, mais propriétaire.

ArgoUML[5] est un outil développé en Java, très populaire qui semble maintenant mature.

BOUML, développé initialement sous linux, était sous licence OpenSource jusqu'à la version 4. Depuis la version 5, le produit, par ailleurs performant, est maintenant payant. La dernière version disponible dans les distributions Linux est donc la version 4.21.

c) Éditeurs de bases de données

Ce qu'on attend d'un éditeur de base de données, c'est qu'il permette :

- de réaliser un *reverse-engeneering* d'une base de données existante ;
- de pouvoir créer des nouvelles tables, index, etc.
- de générer les scripts de modification.

1 http://www.eclipse.org
2 http://www.eclipse.org/projects/project.php?id=tools.pdt
3 http://code.google.com/p/smartypdt/
4 http://www.stack.nl/~dimitri/doxygen/index.html
5 http://argouml.tigris.org/

Cette fonctionnalité est largement couverte par les outils propriétaires. Dans le monde OpenSource, c'est beaucoup plus rare.

Si vous ne manipulez que des bases de données MySql, vous pouvez utiliser **Mysql Workbench**[1], fourni par Mysql, qui fait à peu près ce que l'on attend de ce type de produits.

Si vous voulez manipuler d'autres bases de données, tournez-vous alors vers **SQLPower Architect**[2], produit développé en Java et dont la version communautaire est sous licence GPL. Ce produit permet de manipuler des bases de données Oracle, PostgreSql et MySql notamment.

2. Les outils de gestion du projet

a) Gestion des versions

Dès lors qu'on réalise versions multiples d'un ou plusieurs projets, qu'on intègre des composants externes, et (ou) qu'on travaille à plusieurs, un outil de gestion de versions devient indispensable.

Ces outils permettent, au moins :

- de gérer les différentes versions du code ;
- de télécharger l'ensemble du code dans un ordinateur indépendant pour pouvoir réaliser des modifications locales, et ensuite les publier ;
- de gérer des « branches » différentes, pour préparer des versions livrables ou tester de nouvelles approches.

Deux outils sont principalement connus : **Subversion**[3] (SVN), qui est basé sur un serveur centralisé (utilisé dans Sourceforge), et **GIT**[4], décentralisé, qui permet les échanges inter-membres sans passer par le serveur principal.

b) Gestion de projets

L'objectif est de quantifier le travail et de suivre l'avancement du projet. Les restitutions sont en général réalisées sous forme de diagrammes de Gantt, mais d'autres approches existent, selon les métodes utilisées.

Parmi les produits simples et OpenSource, signalons **GanttProject**[5], un logiciel écrit en Java qui, comme son nom l'indique, permet de générer des diagrammes de Gantt.

c) Compilateurs, intégrateurs

Dans certains projets, un « build » quotidien est généré (exécutable créé à partir de la version du jour). Cette compilation est réalisée automatiquement, par des outils qui vont déclencher les tâches adéquates. La plupart des séquenceurs de tâches ont été écrits en Java et sont en général dédiés à Java. Parmi ceux-ci, ont peut citer **Ant**[6], un des plus connus.

Comme ces outils sont surtout utilisés pour générer des exécutables, ils sont rarement utilisés dans le cadre de projets PHP...

1 http://www.mysql.fr/products/workbench/
2 http://www.sqlpower.ca
3 http://subversion.apache.org/
4 http://git-scm.com/
5 http://www.ganttproject.biz/
6 http://ant.apache.org/

d) Suivi d'incidents

Dès lors que l'on souhaite amener un peu de qualité dans son développement, un gestionnaire de tickets d'incidents devient nécessaire. Un des outils les plus connus est **MantisBT**[1], écrit en PHP et s'appuyant sur une base de données MySql.

On peut également citer **BugZilla**[2], lui aussi très connu, qui a été créé par la fondation Mozilla pour ses propres besoins.

e) Automatisation des tests, intégration continue

Tests de non régression à chaque nouvelle version, lancement de jeux de tests prédéfinis..., tout cela est utile et est utilisé dans les gros projets.

De même, savoir ce qui a été modifié dans le gestionnaire de version, voir les impacts au niveau des tests, tout cela peut devenir nécessaire.

Trois outils signalés par *SMILE*[3] dans son livre blanc concernant les solutions ALM (Application Life Management) : **Continuum**[4], **Jenkins**[5], **Hudson**[6].

D. Les outils intégrés de gestion de projets

Plutôt que d'avoir plusieurs produits différents, et donc des plates-formes différentes, une gestion des utilisateurs, des droits, séparées, il peut être tentant de recourir à des outils qui intègrent tout dans une seule interface. Le livre blanc de SMILE (cité ci-dessus) présente plusieurs outils :

- **FusionForge**[7], anciennement Gforge, probablement le plus ancien : « *FusionForge propose des outils pour faciliter la collaboration au sein de votre équipe de développement, avec des forums et des listes de diffusion, et des outils pour créer et contrôler l'accès à des systèmes de gestion de code source comme CVS et Subversion. FusionForge crée automatiquement les dépôts et règle le contrôle d'accès en fonction des rôles définis à l'intérieur du projet.* » ;
- **Tuleap**[8], développé par la société ENALEAN (société basée en Savoie) : « *Tuleap, une Suite ALM complète : tracking, gestion de version, intégration continue, gestion documentaire, collaboration* »[9]
- **Redmine**[10] : *« Redmine is a flexible project management web application. Written using the Ruby on Rails framework, it is cross-platform and cross-database »*.

Les deux premiers produits sont écrits en PHP, le dernier en Ruby.

Pour savoir quel produit utiliser, consultez le livre blanc de Smile !

1 http://www.mantisbt.org/
2 http://www.bugzilla.org/
3 http://www.smile.fr/Livres-blancs/Systeme-et-infrastructure/ALM-open-source
4 http://continuum.apache.org/
5 http://jenkins-ci.org/
6 http://hudson-ci.org/
7 https://fusionforge.org/
8 http://www.tuleap.com/
9 http://www.enalean.com/produits/tuleap/
10 http://www.redmine.org/

E. Configurer Eclipse pour activer le mode debug

Quand vous codez, il n'est pas rare de devoir mettre des points d'arrêt pour vérifier le contenu d'une variable, ou s'assurer du bon déroulement d'un script.

Vous pouvez, bien évidemment, et c'est souvent la facilité, parsemer votre code de *echo $variable* ou *print_r($tableau)*, suivis ou non de *die()* pour arrêter le script.

Vous pouvez également installer le débogueur PHP dans votre serveur web, qui vous permettra d'avoir une vision nettement meilleure, et de suivre, pas à pas, l'évolution de vos variables et le cheminement de vos scripts.

L'installation a été réalisée à partir de ce document : http://static.zend.com/topics/Zend-Debugger-Installation-Guide-050211.pdf.

1. Installer les librairies dans le serveur

Téléchargez le Studio web debugger de Zend depuis le site http://www.zend.com/en/community/pdt/downloads. Vous pouvez choisir la version correspondant à votre serveur (Linux, Windows, en 32 ou 64 bits).

Dans votre serveur, vérifiez la version de PHP :

```
php -v
PHP 5.3.2-1ubuntu4.17 with Suhosin-Patch (cli) (built: Jun 19 2012
01:35:33)
Copyright (c) 1997-2009 The PHP Group
Zend Engine v2.3.0, Copyright (c) 1998-2010 Zend Technologies
```

Décompressez l'archive, puis :

- recopiez le fichier **dummy.php** à la racine de l'espace de publication de votre serveur web :

```
scp dummy.php root@svweb:/var/www
```

- recopiez le fichier **ZendDebugger.so** correspondant à votre version PHP vers un dossier accessible à votre serveur web, par exemple :

```
scp 5_3_x_comp/ZendDebugger.so root@svweb:/usr/local/bin/
```

- modifiez le fichier **php.ini** utilisé par votre serveur web, par exemple :

```
vim /etc/php5/apache2/php.ini
```

et rajoutez les directives suivantes :

```
zend_extension = /usr/local/bin/ZendDebugger.so
zend_debugger.allow_hosts=10.33.32.0/21
zend_debugger.expose_remotely=allowed_hosts
```

Pour calculer les adresses autorisées (séparées par une virgule), vous pouvez utiliser le programme disponible ici : http://cric.grenoble.cnrs.fr/Administrateurs/Outils/CalculMasque/

Le paramètre **zend_debugger.expose_remotely** peut prendre d'autres valeurs : *allways*, pour montrer que le débogueur est actif à tous les postes, ou *never*, pour inhiber sa présentation.

- Redémarrez alors le serveur web :

```
service apache2 restart
```

2. Configurer Eclipse pour pouvoir utiliser le débogueur

Dans Eclipse, l'activation du débogueur est très simple : il suffit de lui indiquer l'adresse du serveur web.

Allez dans **Windows > Preferences**, puis **PHP > PHP executables > PHP server**

Éditez **Default PHP Web Server**, et indiquez l'adresse web de votre serveur, par exemple : *http://svweb*.

Pour lancer le débogage, ouvrez le fichier que vous voulez exécuter (en principe, index.php), puis lancez le programme : **Run > Debug**. Eclipse vous proposera de changer de perspective, pour passer dans le mode *Debug*, puis s'arrêtera à la première instruction. À vous ensuite de définir vos points d'arrêt !