

Réalisation d'un Intranet : Cohérence d'un Ensemble Réparti et Communicant, autour d'une Architecture Réflexive

John Seraphin

▶ **To cite this version:**

John Seraphin. Réalisation d'un Intranet : Cohérence d'un Ensemble Réparti et Communicant, autour d'une Architecture Réflexive. domain_stic.hype. Université René Descartes - Paris V, 1998. Français. NNT : . tel-00009831v3

HAL Id: tel-00009831

https://theses.hal.science/tel-00009831v3

Submitted on 29 Apr 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Thèse présentée en vue de l'obtention du grade de Docteur de l'Université René Descartes, Paris V

Discipline : Sciences de la Vie et de la Matière
Spécialité : Informatique
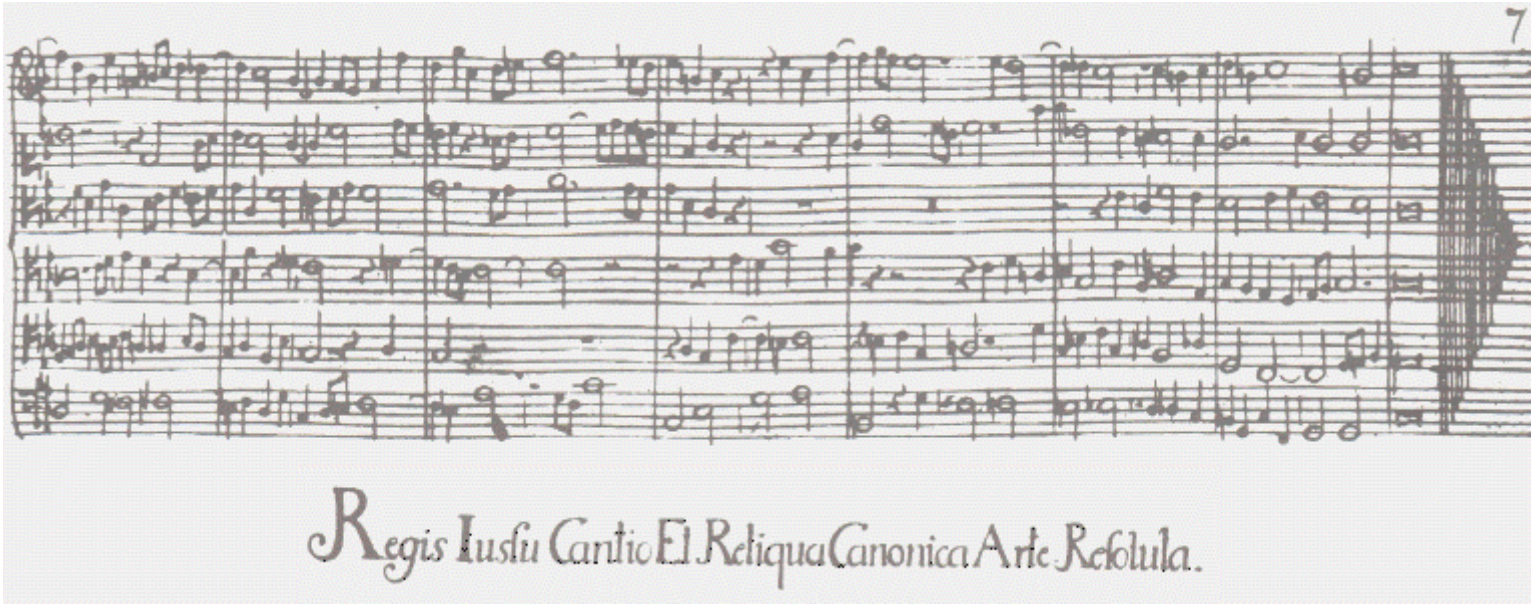
par **John SÉRAPHIN**

Sujet : Réalisation d'un Intranet : Cohérence d'un Ensemble Réparti et Communicant, autour d'une Architecture Réflexive

Soutenue le 6 février 1998 devant le jury composé de :

Richard CASTANET	- *Rapporteur*
Jacques PITRAT	- *Rapporteur*
Guy PUJOLLE	- *Président*
Dominique SERET	- *Directeur*
Léo CACCIARI	- *Examinateur*
Dominique PASTRE	- *Examinateur*

0. Méta-thèse



0.1 Autour d'un titre *3*
0.2 Remerciements *4*
0.3 Préambule *5*
0.4 Organisation de la thèse *7*
0.4.1 Polymorphisme 7
0.4.2 Conventions de présentation 9
0.4.3 Sommaire de la Thèse « manuscrite » 10
0.4.4 Sommaire de l'hyper-Thèse 15

0.1 Autour d'un titre

Résumée en quelques lignes, cette Thèse trouve sans doute son origine dans une fable, extraite d'un livre de textes d'école, qui relatait les exploits d'un ordinateur apprenant à composer du Mozart au fur et à mesure qu'il en intégrait les partitions.

Ainsi amené aux ordinateurs, j'ai eu la chance de faire partie des pionniers de la micro-informatique, avec l'acquisition du fameux ZX-80 de *Sinclair*, premier d'une longue lignée.

Cette découverte a aussi marqué le début de l'apprentissage d'une interminable série de langages de programmation.

Reprenant, dans chacun de ces langages, un même programme qui me sert à faire « des gammes » j'enrageais, mû par un souci viscéral d'économie de l'effort, que d'aucuns qualifieraient de paresse, de devoir reproduire sans fin des écrans et des procédures similaires. Après tout, n'avais-je pas sous les doigts une interface avec un objet qui ne connaît que des « 0 » et des « 1 », que la répétitivité ne dérange manifestement pas ? Pourquoi ne pas lui soumettre un programme qui comporterait une description du problème et le laisser faire le travail, tout seul ? Je ne vous con(mp)terai pas ici le nombre d'échecs cuisants, fruits de programmes maladroits, implémentés dans des langages inadaptés à une telle entreprise.

Éludant quelques années plus tard la préparation de concours, jugés incompatibles avec cette « paresse » naturelle, j'ai exercé mon premier choix en m'inscrivant en informatique, à l'Université, au grand dam de mes proches qui me coiffaient d'un bicorne.

C'est ainsi que j'ai acquis la traduction, qui venait de paraître, du « *G*ödel *E*scher *B*ach » de Douglas Hofstadter. J'y découvris, étonné, la réflexivité dans tous ses états : le théorème de Gödel, la diagonale de Cantor, Magritte, Turing et surtout une grille de lecture de Bach que le wagnérien que je suis trouvait jusqu'alors hermétique.

Initiée au hasard de lectures puis dans le cadre d'un DEA en Intelligence Artificielle, la rencontre avec *Jacques Pitrat*, devait achever ma (dé)formation aux systèmes et architectures auto-référentiels qui continuent à me fasciner.

Réalisée puis implémentée en contrepoint à une expérimentation à la RATP, cette thèse consacre un amorçage Récurrent depuis vingt ans, achevé grâce aux critiques, relectures, encouragements et conseils incessamment renouvelés de *Dominique Seret* et d'*Olivier Houdé*. Il était donc normal qu'elle réunît, **ici**, ses différents inspirateurs, au travers du thème central du *GEB* et de l'*Offrande Musicale.*

0.2 Remerciements

Ce qui est enfin devenu une thèse n'aurait jamais été réalisé sans l'intervention d'une pléiade d'acteurs qui, chacun à un moment plus ou moins important de sa conception, de sa réalisation puis de sa rédaction ont suggéré des améliorations, prodigué le mot de soutien, la critique ou l'encouragement qui m'ont finalement permis d'aboutir.

Un grand merci à Richard Castanet qui a eu la gentillesse d'accepter de lire et de rapporter cette thèse et ce, même *après* que j'ai eu exposé à Pau.

Je ne remercierai jamais assez Léo Cacciari, Dominique Pastre et Guy Pujolle d'avoir respectivement accepté de faire partie du jury et d'en être le président.

Merci aussi à Madeleine Bonnet qui m'a accueilli dans son laboratoire où j'ai pu disposer de tous les moyens techniques, de l'aide et du soutien de Jean-Pierre Heymann, de Christian Le Guen et de Thierry Raedersdorff.

Plusieurs stagiaires ont eu l'occasion de m'en vouloir en essayant soit de comprendre soit d'améliorer RICERCAR. En tout état de cause, Laurent Courtois, Éric Denoual, Alassane Diedhiou et Brice Meudec se sont efforcé de suivre les méandres des évolutions du système avec un stoïcisme remarquable.

Claude Bonnot a, dans un premier temps, bien voulu me consacrer plusieurs heures pour m'expliquer des subtilités de la programmation d'Access. Il a ensuite été volontaire pour développer, à ma demande, la première maquette réalisant la connexion d'un SGBD au Web. Il a enfin, sans hésiter, adopté et utilisé RICERCAR pour ses besoins propres sans exiger d'en connaître les arcanes. Son aide et ses conseils auront été précieux.

Un grand merci à Joël Nicolas et Philippe Pallu qui ont su m'écouter et me prodiguer leurs encouragements dans les moments de doute et qui, surtout, ont bien voulu accepter de lire et de commenter les épreuves décousues de cette thèse.

Francis Robino a, quant à lui, assisté à la naissance de RICERCAR. C'est grâce à ses conseils et critiques que le système s'est rapidement recentré sur les fonctionnalités concrètes d'un intranet.

Michel Raison m'a accueilli dans son équipe à un moment où rares étaient ceux qui auraient voulu d'un pareil canard. Il m'a fait confiance et m'a laissé l'occasion de prouver que j'étais capable de réaliser. Qu'il en soit remercié.

Ma déjà longue présence à la RATP ne peut, enfin, être dépeinte sans la présence contrastée de Janick Taillandier dont j'ai fait et continue à faire partie des équipiers les moins faciles à manœuvrer. Bien qu'il n'ait pas partagé des options et des choix critiques de la thèse, ce n'est toutefois qu'avec son accord que j'ai pu la mener à bien, ménager mon emploi du temps, disposer d'un matériel et des logiciels adaptés, participer à des manifestations scientifiques et avoir la liberté d'esprit de l'achever. Merci.

0.3 Préambule

Le client-serveur, fonctionnel depuis de nombreuses années, s'est néanmoins heurté au problème aigu du coût du déploiement d'une application, de sa configuration sur des milliers de postes de travail hétérogènes (type, mémoire, capacité, niveau de système d'exploitation, *etc.*). L'appropriation des technologies du *World Wide Web* par l'entreprise lui ouvre un nouvel avenir.

Ce nouveau paradigme, autour duquel semble devoir s'organiser le système d'information, n'est pas sans poser à son tour un ensemble de questions dont l'acuité est à l'échelle du succès attendu de l'intranet. Tant techniques que structurels et organisationnels [Martin_97, Schneider_97], ces problèmes sont d'autant plus difficiles à appréhender qu'ils concernent peu l'Internet dont le polymorphisme, parfois chaotique, ne fait en réalité que refléter la diversité de culture de ses différents acteurs.

La réussite d'un intranet suppose en effet qu'une *qualité de service* soit assurée par une architecture et des outils modulaires, robustes mais simples. Il s'agit, à l'instar de toute application ayant pour cible l'ensemble de l'entreprise [1], de :

- garantir la validité, la cohérence et la fraîcheur des informations publiées ;
- assurer la pérennité de données issues d'applications plus anciennes ;
- définir les règles permettant à un ensemble d'acteurs, de profils très variables, d'enrichir le système en minimisant les contraintes ;
- fournir à l'utilisateur final, peu formé par définition, une ergonomie simple et évolutive ;
- gérer les profils et droits éventuels de milliers d'utilisateurs ;
- gérer des données dynamiques, réparties sur l'ensemble de l'entreprise ;
- disposer de statistiques d'accès aux différents services, fiables et adaptatives ;
- fournir aux exploitants des outils de gestion, de surveillance et d'audit.

Établi par [Black_94, Berners-Lee_94a], le parallèle entre le Web et un système d'exploitation orienté objet souligne bien l'adéquation de ce modèle à la diversité, la modularité et la « transversalité » de l'Internet (et par restriction, de l'intranet). Il implémente en effet naturellement la *persistance* des objets (création, destruction et stockage des documents), leur *invocation* (un protocole - HTTP - compris de tous les objets, capable de leur transmettre des messages avec des paramètres - les méthodes GET, POST, *etc.*), un *nommage uniforme* (les URL

[1] Cette thèse a été réalisée à la RATP et à l'Université René Descartes qui ont offert des terrains de discussion et d'expérimentation privilégiés mais variés. La grande homogénéité de la perception de l'intranet en France, vérifiée au travers de contacts réguliers nous amènera cependant à utiliser ici le terme « entreprise » dans son acception la plus générale.

[Berners-Lee_94b]) et une *extensibilité* du modèle sans recompilation d'un noyau (les CGI et API).

Par ailleurs, le foisonnement actuel des recherches autour des nombreux concepts d'objets et de composants réutilisables tels CORBA [OMG_91], DCOM [Brown_96] ou JavaBeans [Colan_97] montre clairement l'intérêt du mariage de ces approches. Il masque cependant l'absence de maturité des standards et surtout le fossé technologique qui existe entre des groupes ayant une vision prospective évoluée de l'informatique et d'autres, plus traditionnels, où la taille et la culture d'entreprise induisent des lourdeurs et une inertie inévitables.

Nous présentons ici une **a**rchitecture de **b**ase, **r**éflexive, ***o**rientée*-documents et **c**ommune à un intranet qui associe dans sa phase actuelle le Web, les bases de données, un métalangage de manipulation de description et des techniques issues de l'Intelligence Artificielle, pour proposer aux utilisateurs connectés au réseau d'entreprise un accès fiable et uniforme à un ensemble facilement extensible de données locales ou transversales.

RICERCAR met en place un ensemble de bases de méta-données fédérées qui décrivent et référencent les objets disponibles. Les serveurs Web associés à ces bases composent ainsi dynamiquement les documents correspondants, indépendamment du serveur interrogé ou de la localisation effective de ces données. Cette architecture garantit la qualité de service en assurant notamment la permanence des URL publiées et la génération dynamique de la structure (l'arborescence) d'un serveur. Elle propose un modèle de navigation uniforme, gère l'authentification et les accès des utilisateurs et, enfin, autorise une surveillance d'ensemble ainsi que des statistiques de fréquentation modulaires et significatives.

À la suite de ses différents amorçages (*bootstrap*), RICERCAR enregistre, dans cette même base répartie, la description et les références de ses propres données ainsi que celle des méta-scripts utilisés pour générer dynamiquement les documents de l'intranet. Cette réflexivité, qui lui permet de manipuler et d'enrichir ses structures, en fait ainsi un système ouvert et *adaptatif*.

Nous analyserons donc les spécificités techniques et organisationnelles qui singularisent à notre sens l'intranet par rapport à l'Internet ainsi que leur implémentation dans RICERCAR dont nous présenterons ensuite les applications concrètes ainsi que les perspectives d'évolution.

0.4 Organisation de la thèse

0.4.1 Polymorphisme

Il nous a semblé qu'un minimum de cohérence dans la présentation d'une thèse proposant une *architecture réflexive* pour assurer la *cohérence* d'un *intranet* imposait que sa rédaction même respectât les principes qui la sous-tendent. Nous avons donc rapidement décidé, avec l'accord de Mme le Professeur Seret, de *communiquer* de façon interactive ce travail en assurant sa publication sur un serveur HTTP.

Ainsi, à l'ombre d'illustres prédécesseurs [Hofstadter_85, Pitrat_96], avons-nous été séduit par l'idée d'ajouter un niveau *méta* supplémentaire à RICERCAR, en mettant en place une instance auto-référentielle dont le contenu et le fonctionnement soient totalement dévolus à la défense et à l'illustration de son propre fonctionnement.

L'exercice *a priori* banal, consistant à publier une thèse sur l'Internet, s'est cependant avéré à l'usage bien plus périlleux que ne le laissait supposer notre enthousiasme initial.

Il est en effet normal que la conception et la présentation d'un travail universitaire, en vue d'être « manuscrit », offrent un confort de lecture indiscutable au prix d'une souplesse de lecture et d'une interactivité moindres que celle d'un parcours hypertextuel. Il est cependant avéré que l'accès à des textes longs et parfois ardus à partir d'un écran est généralement fastidieux (et ce, en dépit de nos efforts constants de simplification) .

Une thèse n'étant pas encore un *objet polymorphe* (*cf.* § 2.2.6, p.40), nous avons donc dû nous livrer à une gymnastique constante, envisageant d'une part une lecture « aléatoire » au gré des liens hypertextes et, d'autre part, une consultation plus traditionnelle, linéaire, de sa version imprimée.

Aussi, par crainte d'appauvrir le tout en cherchant un dénominateur commun à ces deux rédactions, en avons-nous privilégié résolument la forme hypertextuelle, à défaut de pouvoir éliminer toute version imprimée [2].

L'indépendance des différents chapitres et sections ainsi que la « navigabilité » de l'ensemble ont imposé l'introduction d'une certaine redondance afin que chaque partie puisse être éventuellement lue séparément. Nous nous sommes néanmoins efforcé d'exploiter ces redondances pour préciser les notions présentées, en fonction du contexte.

De façon pragmatique, la disponibilité sur des serveurs RICERCAR, de l'ensemble varié de documents que constitue cette Thèse, est une démonstration en grandeur réelle des fonctionnalités proposées.

[2] Laquelle est d'une part, à ce jour, la seule qui fasse foi administrativement et devrait être, d'autre part, à l'épreuve de tout acte de malveillance (notre serveur étant directement sur l'Internet) !

Le lecteur pourra notamment s'assurer de l'indépendance et de la transparence à la localisation, de la cohérence des liens, consulter le code des méta-scripts, vérifier l'implémentation des modèles et le respect d'une charte graphique, *etc.*

Tout au long de la rédaction de cette Thèse, nous attirerons son attention par des liens hypertextes dont l'apparence varie en fonction de la signification :

- l'activation d'un lien vers une [référence bibliographique] provoque l'ouverture d'une fenêtre secondaire du navigateur. Celle-ci génère la liste bibliographique et l'affiche à la référence demandée ;
- un lien hypertexte en *italique* signale la « première » apparition d'un *STL* dont l'explicitation à partir du glossaire est, de même, proposée dans une fenêtre séparée ;
- les renvois vers des documents internes (au sens de RICERCAR) sont signalés par un lien hypertexte normal ;
- la référence à un document accessible ailleurs sur l'Internet (ou non géré par le système) est visualisée sous la forme d'un pictogramme  suivi du lien hypertexte;
- les références à des URL absolues, citées en tant que telles, respectent le format recommandé dans la RFC-1738 [Berners-Lee_94b], c'est-à-dire *<URL:url_affichée>* avec un lien hypertexte associé, précédée éventuellement de  ;
- les références à des URL relatives à RICERCAR, citées en tant que telles, respectent (l'esprit de) la RFC-1808 [Fielding_95] *<RURL:url_relative_affichée>* avec le lien hypertexte associé ;
- nous citons, pour bonne fin, le cas d'un document indisponible (détruit, dont l'identifiant a été mal orthographié ou non encore publié), qui est visualisé par un lien hypertexte, précédé de , qui illustre son inaccessibilité.

Cette Thèse est donc disponible sous sa forme imprimée et sous deux formats électroniques : en tant que fichier *Postscript*,

<URL:http://ricercar.math-info.univ-paris5.fr/ricercar/ricercar_ps> ,

et surtout sous sa forme hypertextuelle,

<URL:http://ricercar.math-info.univ-paris5.fr/ricercar/ricercar_these>

(plus exactement *<RURL:ricercar_ps>* et *<RURL:ricercar_these>* respectivement, pour vous qui lisez ces lignes à partir d'un navigateur) .

En tout état de cause, nous enjoignons le lecteur à se référer, dans la mesure du possible, à cette dernière version [3] qui répercute les modifications et mises à jour les plus récentes.

[3] Un exemple caractéristique de mauvaise réflexivité consisterait à indiquer ici un renvoi hypertextuel.

0.4.2 Conventions de présentation

D'une façon générale, l'*emphase* (que la plupart des navigateurs représentent par des *italiques*) met en valeur un terme ou un concept nouveau alors que les caractères **gras** attirent l'attention sur un élément **important**.

Nous utiliserons la notation UML [Booch_95b] pour les différents diagrammes de classe, de collaboration, de cas, *etc*.

De plus, la présentation des éléments suivants mérite d'être précisée :

- Les balises HTML et CFML seront indiquées en MAJUSCULES NORMALES, placées entre <>
  Ex: <HTML>
- Les noms de *programmes*, *méthodes*, *URL*, *fichiers* ou *répertoires* seront en *italiques, minuscules* (balise <I>)
  Ex: *application.html*
- Les noms des identifiants des documents *RICERCAR* seront en caractères *italiques minuscules* sauf les identifiants d'administration, dont le premier caractère sera en majuscule
  Ex: *ricercar_conv_typo*, *Sty_meta*
- Les noms des tables de la base centrale (et de ses réplications) seront en *MAJUSCULES, ITALIQUES*
  Ex: *KEYS*
- Les noms des tables des bases locales seront en *italique*, le premier caractère en Majuscule, les suivants en minuscules
  Ex: *Dictphys*;
- Les champs des tables seront en MAJUSCULES de type « `machine à écrire` » (balise <CODE>)
  Ex: `KEY`
- Les objets (et leurs attributs) seront en minuscules, *italique* et les classes auront le premier caractère en Majuscule
  Ex: *object.name*, *Class*
- Les noms des variables seront écrits en caractères de type « *`machine à écrire, italique`* »
  Ex: *`erreur`*
- Les listings des programmes, pages HTML, CFML, SQL, *etc*. utiliseront le marqueur <LISTING>
  Ex: <HTML> ... <BODY> ... </BODY> </HTML>

0.4.3 Sommaire de la Thèse « manuscrite »

0. Méta-thèse ***1***

0.1 Autour d'un titre *3*

0.2 Remerciements *4*

0.3 Préambule *5*

0.4 Organisation de la thèse *7*

0.4.1 Polymorphisme 7

0.4.2 Conventions de présentation 9

0.4.3 Sommaire de la Thèse « manuscrite » 10

0.4.4 Sommaire de l'hyper-Thèse 15

1. L'Internet et le World Wide Web ***17***

1.1 La Genèse *19*

1.2 De l'Internet au Web *22*

1.2.1 Le nommage 22

1.2.1.1 IRL (Internet Resource Locator) 23

1.2.1.2 URN (Uniform Resource Name) 23

1.2.1.3 URI (Universal Resource Identifier) 23

1.2.1.4 URL (Uniform Resource Locator) 24

1.2.2 Les langages de description 24

1.2.2.1 Le métalangage SGML 24

1.2.2.2 Le langage HTML 26

1.2.3 Le protocole HTTP 27

1.2.4 Les limites du modèle HTTP/HTML 29

1.2.4.1 Les SSI (Server Side Includes) 29

1.2.4.2 Les CGI (Common Gateway Interface) 29

1.2.4.3 Les API (Application Programming Interface) 31

1.3 Les objets, de base *31*

2. Introduction aux objets et aux systèmes distribués ***33***

2.1 Évolution des représentations *35*

2.2 Les objets *36*

2.2.1 Classes 36

2.2.2 Héritage 37

2.2.3 Métaclasses 38

2.2.4 Envoi de message 39

2.2.5 Encapsulation 40

2.2.6 Polymorphisme 40
2.2.7 Agrégation - Délégation 41
2.3 De l'objet aux systèmes distribués 43
2.4 Différentes architectures distribuées 45
2.4.1 Pourquoi CORBA ? 47
2.5 CORBA 48
2.5.1 L'architecture de l'OMG 48
2.5.2 IDL 50
2.5.3 Architecture générale de l'ORB 51
2.5.4 Activation d'une implémentation 54
2.5.4.1 Indépendance des accès à la localisation 54
2.5.4.2 Interopérabilité des ORB 55
2.5.5 CORBA/IIOP: le futur du Web ? 56
2.5.6 CORBA : synthèse 58
3. RICERCAR : de l'Internet à un intranet 61
3.1 L'intranet : une « application » d'entreprise normale 63
3.2 « L'intranet : un chantier social » 66
3.3 Réaliser un intranet : construction d'un ensemble réparti et cohérent autour d'une architecture réflexive 68
3.3.1 Panorama des besoins 68
3.3.2 Cohérence des liens hypertextes 69
3.3.2.1 Cohérence intra-serveur 72
3.3.2.2 Cohérence inter-serveurs 73
3.3.3 Exploitation du dictionnaire 75
3.3.3.1 Interface des « pages » 75
3.3.3.2 Interface des « index » 75
3.3.3.3 Charte graphique et cohérence de la navigation 77
3.3.4 Cohérence de l'exploitation d'un intranet 78
3.3.4.1 Authentification des utilisateurs 78
3.3.4.2 Métrologie des accès 79
3.4 Administration d'un intranet 81

4. RICERCAR par le Menu 83

4.1 Réflexions et méta-choix 85
4.1.1 Amorçages 85
4.1.2 Réflexions autour d'un intranet (convaincre l'entreprise de se risquer dans la conception d'architectures réparties) 87

4.2 Intranet : modélisation 91

4.3 Choix des outils 94
4.3.1 Le serveur HTTP : WebSite© de O'Reilly 95
4.3.2 Bases de Données 96
4.3.3 Le métalangage Cold Fusion 97
4.3.3.1 Visite rapide des fonctionnalités du langage 98

4.4 Équivalence aux Objets dans un SGBD 102
4.4.1 Modèles finals 102

4.5 Réalisation du modèle présenté 107
4.5.1 Conventions de représentation et de lecture 107
4.5.2 Premiers tours de manivelle 107
4.5.2.1 Correspondance entre URL et envoi de message 108
4.5.2.2 Cohérence des liens d'un serveur 109
4.5.2.3 Génération automatique des index 111
4.5.2.4 Généralisation de la démarche 113
4.5.3 Le modèle 117
4.5.3.1 Méta-interprétation 117
4.5.3.2 Enrichissement des classes 118
4.5.3.3 Modèles : identification entre classe et méta-objet 122
4.5.3.4 Uniformisation de la représentation des données 124
4.5.3.5 Les sous-classes de Documents 127
4.5.3.6 Les classes non « Documentées » 128

4.6 Caractéristiques 131
4.6.1 Les principales méthodes 131
4.6.1.1 L'interpréteur 131
4.6.1.2 Les méthodes « internes » au système 132
4.6.1.3 Les macro-méthodes 134
4.6.1.4 Les Documents 135
4.6.2 L'architecture 136

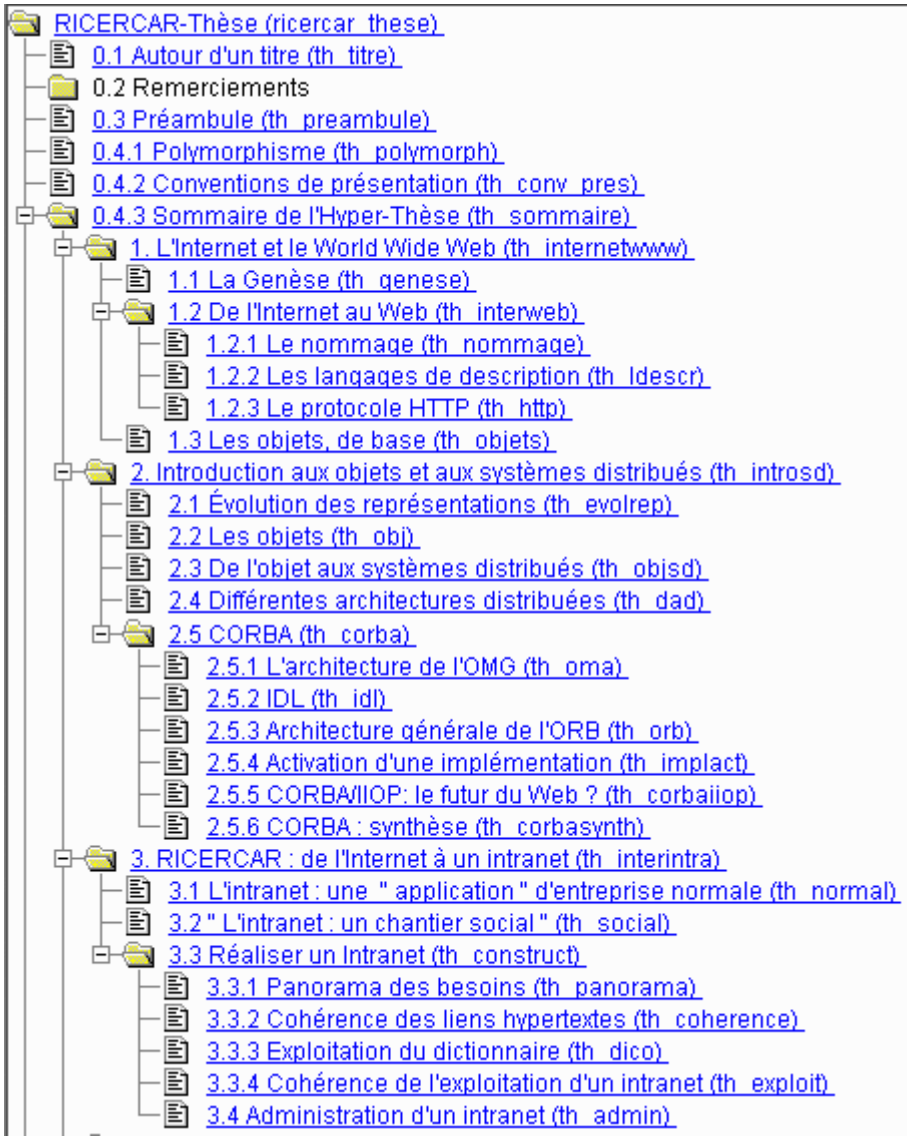
4.7 Application : génération de Documents 138
4.7.1.1 Cas simple (le plus fréquent) 138
4.7.1.2 Exemple de cas simple 141
4.7.1.3 Le pire des cas 142
4.7.1.4 Exemple du pire des cas 149

4.8 RICERCAR, un ORB 150
4.8.1 Les « Services » et « Objets Applicatifs » en bref 153
4.8.1.1 « Services Communs » 153
4.8.1.2 « Objets Applicatifs » à la RATP 155

4.8.1.3 « Objets Applicatifs » à l'Université René Descartes..... 156
5. Conclusion ***157***
5.1 Quo Vadis ? *159*
6. Annexes ***163***
6.1 Bibliographie *165*
6.2 Glossaire *179*
6.3 La RATP *185*
6.3.1 Quelques chiffres 185
6.3.1.1 Le trafic et le parc 185
6.3.2 Historique 186
6.3.3 La structure de la RATP 187
6.3.3.1 Organisation Générale 187
6.3.3.1.1 Les Unités Décentralisées ou Spécialisées 187
6.3.3.1.2 Les départements 188
6.3.3.1.3 La direction générale 188
6.3.3.1.4 Le département des Systèmes d'Information et de Télécommunications 189
6.3.4 Les réseaux informatiques de la RATP 190
6.3.4.1 RAMEAU (Réseau d'Acheminement Multi-protocole inter-Établissement à hAUt débit.) 190
6.3.4.2 Internet à la RATP 192
6.3.4.3 La passerelle ARTS 192
6.4 Rapport du MCD *195*
6.5 Le code des méta-scripts de RICERCAR *208*
6.6 Exemples de Documents *générés par RICERCAR* *209*
6.6.1 Un cas simple : requête de l'objet « flash » 209
6.6.1.1 « Flash.html », tel qu'enregistré par son auteur (la DGC). 209
6.6.1.2 Le document résultant de l'exécution de la méthode Sty_page 210
6.6.1.3 Les traces de l'exécution de l'interpréteur 211
6.6.1.4 Les traces de l'exécution de Sty_page (invoquée par l'interpréteur) 212
6.6.2 Le pire des cas : requête de « jsbookmark », d'accès protégé, à partir d'un autre serveur 214
6.6.2.1 Redirection sur le serveur d'authentification 214
6.6.2.2 Les traces de l'exécution de l'interpréteur (et service de sécurité) 215
6.6.2.3 Accès authentifié à « jsbookmark » 220
6.6.2.4 Les traces de l'exécution de l'interpréteur 221

6.6.3 Services Communs 225
6.6.3.1 Événements 225
6.6.3.2 Cycle de Vie 229
6.6.3.3 Statistiques 231
6.6.4 Les Objets Applicatifs 233
6.6.4.1 Antilope 234
6.6.4.2 Infocentres 237
6.6.4.3 95 PPP 238
6.6.4.4 Annuaire des UFR 240
6.7 Table des figures 243
6.8 Récapitulatif et index des citations in extenso de Ricercar, constituant des auto-références 245

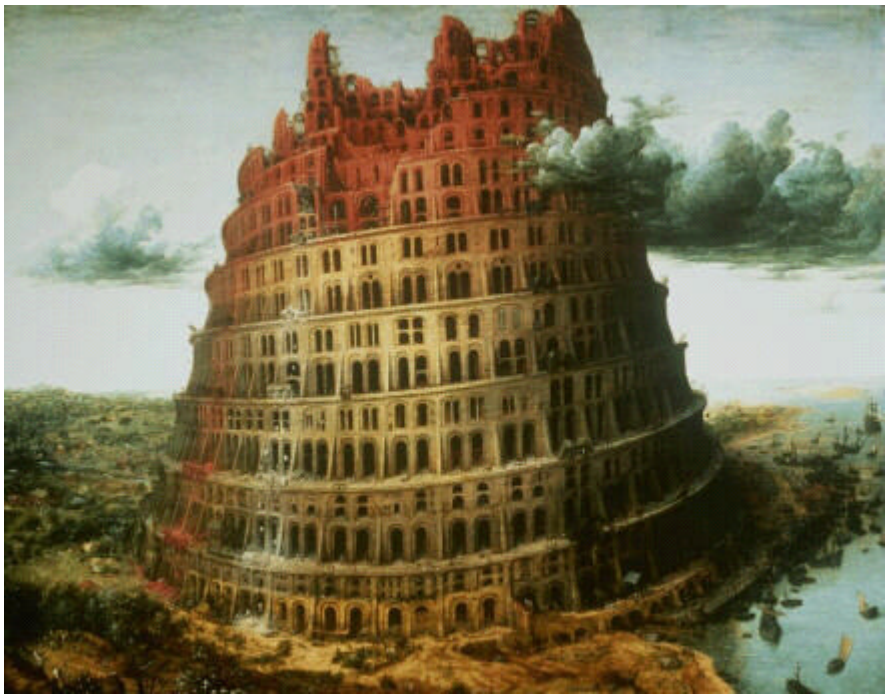
0.4.4 Sommaire de l'hyper-Thèse



RICERCAR-Thèse (ricercar_these)
0.1 Autour d'un titre (th_titre)
0.2 Remerciements
0.3 Préambule (th_preambule)
0.4.1 Polymorphisme (th_polymorph)
0.4.2 Conventions de présentation (th_conv_pres)
0.4.3 Sommaire de l'Hyper-Thèse (th_sommaire)
1. L'Internet et le World Wide Web (th_internetwww)
1.1 La Genèse (th_genese)
1.2 De l'Internet au Web (th_interweb)
1.2.1 Le nommage (th_nommage)
1.2.2 Les langages de description (th_ldescr)
1.2.3 Le protocole HTTP (th_http)
1.3 Les objets, de base (th_objets)
2. Introduction aux objets et aux systèmes distribués (th_introsd)
2.1 Évolution des représentations (th_evolrep)
2.2 Les objets (th_obj)
2.3 De l'objet aux systèmes distribués (th_objsd)
2.4 Différentes architectures distribuées (th_dad)
2.5 CORBA (th_corba)
2.5.1 L'architecture de l'OMG (th_oma)
2.5.2 IDL (th_idl)
2.5.3 Architecture générale de l'ORB (th_orb)
2.5.4 Activation d'une implémentation (th_implact)
2.5.5 CORBA/IIOP: le futur du Web ? (th_corbaiiop)
2.5.6 CORBA : synthèse (th_corbasynth)
3. RICERCAR : de l'Internet à un intranet (th_interintra)
3.1 L'intranet : une " application " d'entreprise normale (th_normal)
3.2 " L'intranet : un chantier social " (th_social)
3.3 Réaliser un Intranet (th_construct)
3.3.1 Panorama des besoins (th_panorama)
3.3.2 Cohérence des liens hypertextes (th_coherence)
3.3.3 Exploitation du dictionnaire (th_dico)
3.3.4 Cohérence de l'exploitation d'un intranet (th_exploit)
3.4 Administration d'un intranet (th_admin)


4. RICERCAR par le Menu (th_ricercarmenu)
4.1 Réflexions et méta-choix (th_meta)
4.1.1 Amorçages (th_bootstrap)
4.1.2 Réflexions autour d'un Intranet (th_reflexions)
4.2 Intranet : modélisation (th_modelisation)
4.3 Choix des outils (th_outils)
4.3.1 Le serveur HTTP : WebSite de O'Reilly (th_website)
4.3.2 Bases de Données (th_sgbd)
4.3.3 Le métalangage Cold Fusion (th_cfml)
4.4 Équivalence aux Objets dans un SGBD (th_oosgbd)
4.5 Réalisation du modèle présenté (th_realisation)
4.5.1 Conventions de représentation et de lecture (th_convpres)
4.5.2 Premiers tours de manivelle (th_manivelle)
4.5.3 Le modèle (th_modele)
4.6 Caractéristiques (th_caract)
4.6.1 Les principales méthodes (th_methodes)
4.6.1.1 L'interpréteur (th_interpret)
4.6.1.2 Les méthodes " internes " au système (th_interne)
4.6.1.4 Les macro-méthodes (th_macros)
4.6.1.4 Les Documents (th_docs)
4.6.2 L'architecture (th_architect)
4.7 Application : génération de Documents (th_application)
4.7.1.1 Cas simple (le plus fréquent) (th_simple)
4.7.1.3 Le pire des cas (th_pire)
4.8 RICERCAR, un ORB (th_ricercarorb)
5. Conclusion (th_conclusion)
6. Annexes (th_annexes)
6.1 Bibliographie (ricercar_biblio)
6.2 Glossaire (ricercar_glossaire)
6.3 La RATP (th_ratp)
6.3.1 La RATP : Quelques chiffres (th_ratp_chiffres)
6.3.2 La RATP: Historique (th_ratp_hist)
6.3.3 La Structure de la RATP (th_ratp_struct)
6.3.4 Les réseaux informatiques de la RATP (th_ratp_inf)
6.4 Rapport du MCD
6.5 Le code des métascripts de RICERCAR (ricercar_pgm)
Sty_Goto (sty_goto)
Sty_Page (sty_page)
Sys_Application (sys_application)
Sty_Menu (sty_menu)
Sys_Declare (sys_declare)
Sty_Vpage (sty_vpage)
Sty_Meta (sty_meta)
Sty_Raw (sty_raw)
Sty_PageNph (sty_pagenph)
Sys_Erreur (sys_erreur)
Sty_Img (sty_img)
Sys_ACL (sys_acl)
Sys_VerifACL (sys_verifacl)
Sty_HtmlHead (sty_htmlhead)
Sty_Header (sty_header)
Sty_Signe (sty_signe)
Sys_Mai (sys_mai)
Sty_WWW (sty_www)
Sys_Crypt (sys_crypt)
Sys_Form (Sys_Form)
Sty_Tree (Sty_tree)
Sys_AHref (sys_ahref)
Sys_Img (Sys_Img)
Sys_Fig (Sys_Fig)

1. L'Internet et le World Wide Web



Et Yahvé les dispersa de là, à la surface de toute la terre,
et ils cessèrent de bâtir la ville.

(Genèse, 11-8)

1.1 La Genèse 19
1.2 De l'Internet au Web..... 22
1.2.1 Le nommage..... 22
1.2.1.1 IRL (Internet Resource Locator)..... 23
1.2.1.2 URN (Uniform Resource Name) 23
1.2.1.3 URI (Universal Resource Identifier)..... 23
1.2.1.4 URL (Uniform Resource Locator)..... 24
1.2.2 Les langages de description 24
1.2.2.1 Le métalangage SGML 24
1.2.2.2 Le langage HTML 26
1.2.3 Le protocole HTTP 27
1.2.4 Les limites du modèle HTTP/HTML 29
1.2.4.1 Les SSI (Server Side Includes) 29
1.2.4.2 Les CGI (Common Gateway Interface) 29
1.2.4.3 Les API (Application Programming Interface)..... 31
1.3 Les objets, de base 31

1.1 La Genèse

Nous ne reproduirons pas ici le sempiternel historique de l'Internet depuis sa genèse au début des années soixante. Le lecteur désirant retracer les pas des pionniers de l'époque est invité à consulter [Archimbaud_95, Dufour_96] qui en font un remarquable récapitulatif.

Il nous semble plutôt intéressant de tenter de cerner les raisons qui ont permis à un ensemble de technologies informatiques disparates de cristalliser autour du *World Wide Web*, inventé par Tim Berners-Lee [Berners-Lee_90], dont l'ambition d'origine était d'améliorer l'accès des physiciens du CERN à des documents hétérogènes.

Il a fallu qu'un ensemble de technologies soient simultanément matures pour que leur synergie puisse rendre le Web possible et faire connaître à l'Internet (et par delà lui, à l'intranet) son essor actuel après plus de vingt ans d'existence paisible et quasi-exclusivement universitaire :

- la **micro-informatique** est sans doute le moteur principal du dynamisme de l'informatique des deux dernières décennies. Elle a en effet mis à la disposition d'une large population une gamme étendue et hétérogène de micro-ordinateurs, de moins en moins coûteux, de plus en plus puissants. Dotés de capacités mémoire et disque réservés quelques années auparavant aux systèmes centraux, ces ordinateurs ont apporté à une génération entière un moyen de production efficace, souple et d'un confort jusqu'alors inconnu ;
- après une première phase d'individualisme, les entreprises se sont souciées de mettre un terme à l'éparpillement et à la fragilisation de leurs données. Elles ont organisé la communication et la capitalisation des informations dispersées ainsi que la rationalisation des dépenses. Les **réseaux locaux** ont ainsi permis de mettre un terme à cette atomisation. Ils ont aussi offert des espaces et des services d'échange communs par le biais d'outils réseaux (messagerie, agendas, services d'impression partagés, *etc.*) ;
- pendant des années, plusieurs propositions de transports se sont trouvé [4] en concurrence (TCP/IP, IPX/SPX, OSI/ISO, *Appletalk*). Le modèle **TCP/IP**, dopé par la percée d'UNIX comme système d'exploitation des serveurs mais aussi grâce au dynamisme de l'Internet, s'est imposé en tant que standard *de facto*. Sa relative simplicité alliée à sa robustesse [Pujolle_88], comparées à celles de ses concurrents, ont ainsi favorisé sa dissémination. Il s'est d'abord imposé comme une solution de choix pour interconnecter (par passerelles) les différents réseaux hétérogènes, puis a remplacé les réseaux propriétaires au fur et à mesure de leur évolution ;

[4] *Cf.* la règle d'accord de Marot [Grévisse_88, § 907].

- l'émergence et la généralisation d'une micro-informatique hétérogène et anarchique a placé les entreprises devant le risque réel de voir leurs informations prisonnières d'une *Tour de Babel* de systèmes et de progiciels de génération de documents incompatibles. Aussi, plusieurs groupes de travail indépendants se sont-ils formés afin de proposer des normes de **formats pivots pour produire des documents** (*SGML* et *ODA* notamment [DSA_91]), offrant la possibilité de créer des conversions avec les formats propriétaires. Ces normes ont connu des réussites et des applications diverses telles DSSSL et Cals et, bien que ne s'étant pas imposées de façon universelle, ont préparé le terrain pour une norme de présentation simple et générale telle HTML ;

- parallèlement, l'éclosion de la micro-informatique a automatiquement engendré un besoin de stockage, d'organisation et de gestion de données locales. **Le modèle relationnel qui s'est imposé pour les *SGBD*** [Codd_70, Chen_76], connaît ses premières implémentations robustes au début des années quatre-vingts et propose rapidement des modèles de stockage de données qui s'en inspirent sur des micro-ordinateurs (Dbase, Open Access, MS Access, *etc.*). Le succès de *SQL* [IBM_81] puis d'*ODBC* [Microsoft_92] comme interface d'accès a ainsi permis aux utilisateurs de disposer du même modèle relationnel et du même langage d'interrogation pour accéder à un grand nombre de SGBD fussent-ils sur des systèmes centraux, des serveurs Bureautique® ou sur le poste de travail ;

- l'usage croissant de la micro-informatique, des bases de données et des réseaux locaux a permis la **maturation fonctionnelle du modèle client-serveur** qui découple les données d'entreprise de leurs traitements. Les applications sensibles, qui nécessitaient jusqu'alors des émulations de terminal, ont pu être enrichies en bénéficiant de la puissance de calcul locale utilisée pour la manipulation et le traitement des données dont la cohérence et l'intégrité restaient garanties de façon centralisée [Zakaria_95];

- quoique séduisant, **le modèle client-serveur pose dans la pratique un ensemble de difficultés aiguës**. En effet, la cohérence et l'intégrité des données garanties de façon centrale reporte la totalité des difficultés sur les postes de travail clients. Il faut installer un logiciel spécifique, pour chaque application développée, sur l'ensemble des postes concernés de l'entreprise. Ces installations se heurtent rapidement à l'hétérogénéité du parc installé (*Macintosh*, PC, Station X, différents niveaux d'OS, différentes capacités mémoire, disque, *etc.*).
  Le développement d'une application client-serveur se transforme en un vrai parcours du combattant puisqu'il faut concevoir autant de versions clientes qu'il y a d'OS en place, les adapter pour fonctionner sur les différents réseaux utilisés et, enfin, les déployer sur un ensemble de postes, fragilisés par ces installations successives. L'opération se répétant pour toute mise à niveau de l'application, les promesses effectuées de réduire les coûts de développement grâce au client-serveur n'ont généralement pas pu être tenues devant la combinatoire réalisée. Bien qu'un ensemble de solutions aient été mises en place (télé-installation, homogénéisation du parc et des réseaux locaux, *etc.*),

l'étendue des problèmes posés par le déploiement concret du client-serveur a singulièrement réduit son intérêt ;

- c'est ainsi que **les modèles objet ont commencé à intéresser l'entreprise**. Les applications client-serveur sont de plus en plus complexes et nécessitent des mises à jours régulières sur les postes de travail. Les suites Bureautique se succèdent et doivent être déployées dans des délais qu'une grande entreprise ne peut pas maîtriser.
  Il s'avère qu'une grande partie des ces applications sont communes ou sont identiques d'une version à l'autre. Le mythe du code réutilisable, des composants logiciels modulables et de l'héritage des propriétés et méthodes séduit successivement les grands distributeurs de logiciels, avides de factoriser leurs développements, puis les entreprises qui essayent de reprendre ces principes à leur compte.

C'est dans ce contexte, d'une informatique un peu chaotique, que des tentatives de rationalisation et d'homogénéisation commencent à éclore autour du monde UNIX :

- l'Internet a mis en place un ensemble de protocoles (*SMTP*, *FTP*, *NNTP*, *Telnet* , *etc.*) qui proposent des moyens de partage et de transport de différents types d'informations ;
- le concept flou de renvois automatiques entre livres, sommaires, dictionnaires, *etc.*, rêvé à la fin de la seconde guerre mondiale par Vernon Bush [Bush_45] a été concrétisé depuis, baptisé hypertexte [Nelson_67] et a connu des implémentations remarquées, notamment par *Apple* (Hypercard) ;
- la première tentative de proposer une architecture distribuée pour les applications, *DCE*, ne connaît qu'un succès d'estime mais consacre les mécanismes de *RPC*, exploités par la suite dans *CORBA*/*IIOP* et *COM*/*DCOM* (qui s'appuie sur DCE).

L'invention du Web arrive alors, comme la plupart des processus menant aux grandes découvertes, à ce moment particulier où les technologies de l'informatique ont atteint une masse critique [Pitrat_85, Pitrat_95] qui permet à une vision originale de s'imposer.
Par itérations rapides, ce qui commence par être un système de navigation hypertextuel spécifique [Berners-Lee_89] s'enrichit pour devenir un **modèle client-serveur simple et généralisable**, basé sur un protocole, *HTTP* (serveur), et un langage, *HTML* (client), qui connectent les grandes bases documentaires réparties et hétérogènes par des liens normalisés, les *URL*. Le client (navigateur), quel qu'il soit et où qu'il soit, peut (théoriquement) parcourir le graphe des documents disponibles sans jamais se soucier de leur localisation ni du type de serveur qui les abrite.

Le *big-bang* du Web a depuis laissé place à une *Soupe primitive* à partir de laquelle la communauté de l'Internet essaye de définir les architectures de demain.

1.2 De l'Internet au Web

1.2.1 Le nommage

En près de trente ans de développement, l'Internet a été fécond en systèmes et protocoles engendrés pour apporter des solutions originales aux problèmes de partage et de communication d'informations réparties. L'hégémonie initiale de serveurs de type UNIX a naturellement suggéré que les nommages soient calqués sur l'arborescence du système de fichiers utilisé.

Le service de nommage est cependant au cœur de tout système distribué. C'est grâce à lui que les références aux objets servis sont résolues. C'est lui qui assure l'interopérabilité et permet qu'un client adresse une requête semblable à deux objets situés sur deux machines distantes et d'architecture interne totalement distincte.

La difficulté à réaliser un service de nommage efficace réside dans la nécessité d'offrir un mécanisme général pour retrouver les éléments atomiques de façon *non ambiguë* et *stable,* à partir d'éléments caractéristiques invariants.

S'il est relativement aisé de développer de nouvelles passerelles pour s'accommoder de nouveaux formats de données, le caractère *public* d'une adresse limite, en revanche, fortement les possibilités de mettre en place des systèmes de conversion (à moins de maintenir plusieurs niveaux d'adressage simultanés).

Le nommage issu d'une arborescence physique de fichiers a ainsi montré ses limites lors de l'explosion du Web : un système en pleine effervescence dont les éléments ont des durées de vie relativement courtes, ou qui se restructure très régulièrement, supporte assez mal une métrique basée sur une localisation physique dans des arborescences.

Par ailleurs, la tendance croissante de proposer sur le Web des documents composites, créés dynamiquement à la demande de l'utilisateur et ayant une durée de vie éphémère par définition, a renforcé l'inadéquation du schéma jusqu'alors établi.

La communauté de l'Internet a réalisé cette lacune assez rapidement. À défaut de pouvoir remettre « à plat » l'adressage en vigueur pour d'évidentes raisons de maintien de l'existant, elle s'est efforcé [5] d'introduire une sémantique nouvelle, considérant une adresse comme un chemin *logique* en *injection* avec une représentation physique, pour lesquels toute *bijection* serait le pur fruit du hasard (qui assure la transition avec le passé).

[5] *ibid.*

Plusieurs terminologies ont été définies ou précisées afin de tenir compte d'une sémantique plus riche et plus souple.

1.2.1.1 IRL (Internet Resource Locator)

Un ensemble de méta-règles de nommage, les IRL [Kunze_95], ont été rendu [6] nécessaires par l'explosion du Web. Elles ont été définies dans le but de permettre à un client (humain ou machine) de référencer de façon transparente et stable (indépendante de la durée de vie ou de la mobilité) un ensemble d'objets (appelés *ressource*) provenant de systèmes hétérogènes (ex : FTP, Gopher, Telnet, HTTP).

Ce nommage ne présuppose **en aucun cas** l'accessibilité, l'existence, l'unicité, la réplication ou l'accès déterministe aux objets référencés. Il se contente d'offrir un formalisme *universel*, indépendant de la localisation et décomposable en sous-éléments interprétables, par des algorithmes éventuellement différents.

1.2.1.2 URN (Uniform Resource Name)

Les URN [Sollins_94] identifient quant à eux une *ressource* ou une unité d'information, le plus généralement possible, indépendamment de sa localisation ou de sa durée de vie.

Un URN est **unique** (ne peut désigner deux objets différents), persistant (dépasse la durée de vie de l'objet nommé) et doit pouvoir englober des schémas de nommage très distincts. Un numéro ISBN, une lettre postale, un numéro de RFC ou de norme ISO, un code barre ou une URL sont donc des URN !

1.2.1.3 URI (Universal Resource Identifier)

Les URI [Berners-Lee_94a] sont des identifiants normalisés définissant une syntaxe extensible et assurant un isomorphisme avec tout système de nommage. Cette syntaxe ne présuppose aucune connaissance quant à la façon dont les adresses sont effectivement calculées dans chacun des référentiels représentés. La syntaxe d'un URI indique, d'une part, la méthode d'accès à la ressource (HTTP, FTP, FILE, MAILTO, NEWS, *etc.*) et, d'autre part, un *chemin* dépendant de cette méthode.

En l'occurrence, le protocole HTTP considère les URI comme des chaînes de caractères formatées qui identifient - par leur nom, localisation ou **autre caractéristique quelconque** - une ressource disponible sur le réseau.

---

[6] *Ibid.*

« Toute similarité entre un *chemin* et une arborescence physique telle que décrite dans un quelconque système d'exploitation (UNIX ou autre) doit être considérée comme une coïncidence et **ne doit pas être utilisée** pour interpréter un URI comme un nom de fichier. »

1.2.1.4 URL (Uniform Resource Locator)

Enfin, une URL [Berners-Lee 94b] est un URI particulier qui peut être utilisé algorithmiquement pour exprimer l'adresse d'une ressource disponible sur le réseau (l'Internet notamment).

Les URL sont généralement considérées dans la littérature comme des synonymes d'adresses Web, d'Identifiants Universels de Documents ou une combinaison d'URI et d'URN.

Concrètement, chaque objet accessible par le réseau est identifié par une chaîne de caractères ASCII (son URL) dont la forme, qui dépend du protocole utilisé, peut être schématisée par

<protocole>://<partie-dépendant-du-protocole>

Ainsi, l'URL d'un objet accessible par un serveur HTTP a-t-elle pour forme

```
http://serveur:port/chemin?param1=val1&param2=...
```

alors qu'un objet servi par le protocole FTP a pour URL

```
ftp://utilisateur:mot-de-passe@serveur:port/chemin;type=commande
```

1.2.2 Les langages de description

1.2.2.1 Le métalangage SGML

Devant la grande complexité d'extraire l'information structurelle et physique d'un document formaté, un groupe de chercheurs, à l'instigation d'IBM, mit au point au milieu des années soixante-dix un langage de balisage généralisé (GML). Un groupe de travail international fut créé par la suite qui aboutit à la mise au point de *SGML*.

L'intérêt principal de SGML est de définir des méthodes pour le balisage de documents qui préservent les informations relatives à leur structure. Il est possible à un groupe de personnes échangeant régulièrement des informations de définir une structure à leurs documents et de la modéliser. Ce modèle, une *DTD*, est une feuille de style qui va décrire la syntaxe et les balises (« *tags* » en anglais) qui seront utilisées. Cette description comprend la structure *logique* des documents qui

seront liés à la DTD (hiérarchie et niveaux des titres, types de listes, types de paragraphes, *etc.*).

La création d'un document revient à l'attacher à sa DTD (la première ligne du document spécifie la DTD utilisée), puis à baliser le texte en fonction de la structure retenue.

Le succès de SGML est principalement dû au découplage total effectué entre la structure logique d'un document et ses différentes présentations. L'auteur se focalise sur la structuration de ses informations sans jamais se soucier de connaître le *medium* utilisé pour leur consultation. Chaque *medium* compatible SGML dispose d'un interpréteur (*parser*) qui, à la réception d'un document, le lie à sa DTD (jointe ou stockée dans une bibliothèque), le décode et interprète chacune de ses balises en fonction de ses spécificités [Bryan_88, Goldfarb_90].

C'est ainsi qu'un balisage <SOS>Attention, danger !</SOS>, pourra être interprété par une imprimante en un texte gras, centré et en 30 points,

Attention, danger !

alors qu'un écran le représentera, par exemple, par un texte rouge, centré et clignotant ou qu'un système pour handicapés physiques actionnera une alarme sonore ou visuelle !

L'auteur, lui, s'est simplement [7] chargé d'actionner une balise de détresse, laissant à chaque élément terminal physique le loisir de l'interpréter d'une manière adéquate, sans être obligé de prévoir l'ensemble des usages possibles de son avertissement.

Cette structuration des documents permet d'automatiser l'extraction des éléments de structure, de créer des tables de matières automatiques, des index, *etc.*

SGML *n'est pas* un langage de description de documents. C'est un *méta*langage normalisé qui crée des schémas syntaxiques de balisage qui seront, eux, utilisés pour décrire des documents structurés.

Au fil des années, plusieurs grands organismes tels l'« *Association of American Publishers* », le « *Department of Defense* » ou « *Her Majesty's Stationary Office* » ont mis au point des DTD pour leur usage particulier. Des variations de ces DTD ont ensuite été utilisées par différentes maisons d'édition telles que *McGraw-Hill*, *Prentice-Hall*, *etc.*

[7] Ce qui n'est bien entendu pas le cas ici. Ni le traitement de texte, ni l'imprimante que nous utilisons, ni le navigateur du lecteur ne savent interpréter une DTD SGML. Nous avons donc dû adapter « à la main » les différentes représentations de la définition que nous prêtons à cette balise <SOS> !

1.2.2.2 Le langage HTML

Le langage HTML [Berners-Lee_95] n'est qu'une DTD SGML particulière et relativement simple. Elle définit, notamment, six niveaux de titres, un ensemble de types de listes (définitions, puces, numérotées) et quelques propriétés logiques (emphase, renforcé) ou physiques (italique, gras, souligné).

Il est possible d'associer une étiquette à un texte, de signaler un lien vers un autre document ou encore d'inclure une image, *etc*.

C'est la simplicité même de cette DTD, la mettant à la portée du plus grand nombre, qui est à l'origine de son succès (et de celui du Web) depuis plusieurs années. Il s'avère néanmoins que l'accroissement de la complexité des applications et des présentations faites sur le Web a nécessité des propriétés de plus en plus en sophistiquées.

Une première version du langage, aux propriétés très frustres, a été rapidement suivie d'autres plus sophistiquées qui ont successivement introduit les tables et surtout les formulaires, offrant ainsi un moyen simple d'interagir avec les utilisateurs finals (interrogations, galeries marchandes virtuelles, *etc*.).

Cette évolution est loin d'être terminée. De nouveaux éléments ont été introduits ces derniers mois, dont le nombre et la complexité croissante laissent entrevoir l'avènement d'un Web beaucoup moins à la portée de l'« utilisateur moyen », notamment :

- des propriétés pour définir et rendre public le type de contenu des documents [Miller_96]. Ces propriétés sont, par exemple, destinées à informer du caractère particulier de certaines pages (pornographie, violence, publicité, *etc.*) et permettre - à des parents - de limiter l'accès en fonction de critères bien définis ;
- les feuilles de style [Lie_96] donnent la possibilité d'associer à des éléments logiques (titres, paragraphes) des indications *physiques* de présentation (police, taille, couleur, centrage, *etc.*) ;
- ces feuilles de styles sont intégrées dans la nouvelle DTD de HTML (v4.0) [Ragget_97] en cours de préparation par le W3C. Cette DTD cherche par ailleurs à offrir des algorithmes pour internationaliser le langage [Yergeau_97] (par une gestion des langages orientaux et la combinaison des sens de lecture - gauche à droite ou droite à gauche - dans un même document). Elle vise aussi à augmenter l'ergonomie du langage en vue de permettre aux handicapés physiques de bénéficier d'une adaptation de ses fonctionnalités ;
- des extensions normatives à HTML sont en cours d'élaboration (plus d'une dizaine actuellement). Ainsi le W3C s'intéresse-t-il à *XML* [Bray_97], *PEP* [Nielsen_97] ou *MathML* [Ion_97] dont la visée est de compléter HTML dans des domaines mal couverts à ce jour (au prix de la simplicité originelle de HTML, cependant).

1.2.3 Le protocole HTTP

HTTP [Berners-Lee_96] est un protocole applicatif basé sur un paradigme requête/réponse. Un client établit une connexion avec un serveur et lui émet une requête composée de la méthode à invoquer, une URI, une version de protocole et un message de type MIME [Borenstein_93] comportant des informations sur le client, des paramètres et un éventuel contenu de message. Le serveur renvoie un message comportant la version du protocole utilisé, un code d'erreur ou de réussite de la requête, suivis d'un message MIME contenant des informations sur le serveur, des méta-informations sur la requête et un éventuel contenu de message (le code HTML, interprété localement).

Sur l'Internet, les communications HTTP utilisent généralement des connexions TCP/IP, mais ce protocole n'est en aucun cas comminatoire : HTTP nécessite l'usage d'une couche transport fiable ; n'importe quel protocole de communication garantissant cette fiabilité peut être utilisé [8].

Le protocole HTTP s'est largement inspiré du protocole FTP, adapté aux documents hypertextes (d'où **HT**TP au lieu de **F**TP). La volonté de gérer un très grand nombre d'accès simultanés, présente dès l'origine de HTTP, s'est concrétisée par un mode non connecté.

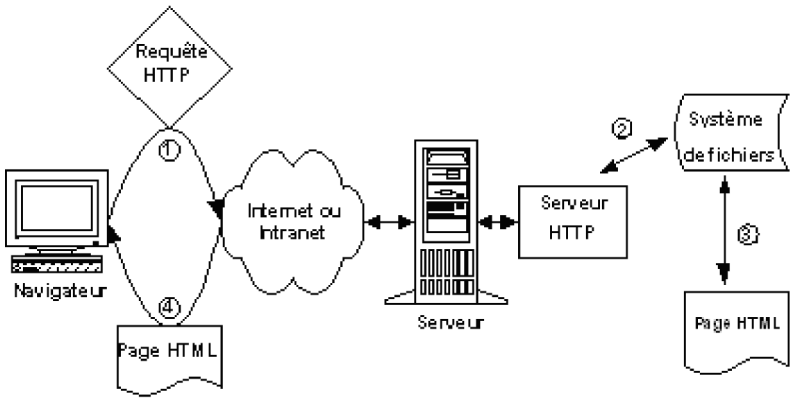


Figure 0. 1. Un serveur Web « passif » inspiré du modèle FTP.

La session du client n'est établie avec le serveur que pendant la brève durée de transfert du fichier spécifié par l'URL. Ce mode non connecté, efficace dans le cadre d'accès concurrents vers des documents (hyper) textuels, est pénalisant dans certains cas :

[8] Nous ne connaissons cependant aucune implémentation de HTTP sur une autre couche transport que TCP.

1. un même client accédant à plusieurs documents d'un même serveur (ce qui englobe le cas du document comportant des images incluses) établit autant de sessions que de documents requis, d'où une perte de temps pour ré-établir une session à chaque accès. Cela est d'autant plus dommageable que le protocole TCP, conçu pour optimiser les accès au réseau en fonction du temps de connexion, pénalise du fait de leur brièveté les sessions HTTP, intermittentes [Comer_96] ;
2. il est difficile de gérer une persistance de l'utilisateur pour simuler une session transactionnelle et maintenir un contexte de l'utilisateur (ou des variables globales d'une application) entre deux requêtes.

La version 1.1 du protocole HTTP [Fielding_97, Franks_97], dont la normalisation se heurte à un certain nombre de problèmes d'implémentation depuis près d'un an, essaye - entre autres - de pallier ces défauts :

1. un mode *keep-alive* est proposé qui associe un *timeout* [9] au mode non connecté. Ce mode établit une session TCP/IP entre un client et le serveur, pendant un temps paramétrable, dans l'éventualité d'accès ultérieurs ;
2. la gestion d'un *contexte utilisateur* est quant à elle effectuée par le biais de variables rémanentes (les *cookies*) stockées dans un fichier sauvegardé sur le poste du client. Ces variables permettent d'établir une cinématique transactionnelle entre un client et le serveur [Kristol_97].

Ces *cookies*, introduits par *Netscape* en 1996, offrent désormais un moyen normalisé, efficace et sécurisé d'établir des transactions complètes entre un client et un serveur HTTP, d'une part, et de sauvegarder les préférences des utilisateurs d'une session à l'autre, d'autre part (il n'est pas possible à une application d'accéder aux variables d'une autre application, écartant dès lors le spectre du *Big Brother* tant craint sur l'Internet).

Ces variables rémanentes peuvent être instanciées très finement et associées à l'accès d'un sous-domaine DNS complet (*.intra.ratp* par exemple), d'un chemin absolu ou relatif, ou même d'une URL particulière. Elles peuvent être enregistrées pour des durées très variables allant de la session de l'utilisateur (jusqu'à sa sortie du navigateur) ou pour des durées précisées en jours/heures/minutes ou en temps absolu.

Il est laissé à la charge des applications concernées d'utiliser ces variables comme sémaphores dans les mécanismes complexes de gestion de transactions.

[9] Nous avons renoncé à utiliser les termes français de « garder vivant » ou de « minuterie ». *Cookies*, quant à lui, est absolument intraduisible (« petits biscuits » ?).

1.2.4 Les limites du modèle HTTP/HTML

Le principal défaut de l'association HTTP-HTML réside dans ses faibles capacités d'adaptation au contexte, issues de son héritage de FTP notamment.

Une fois épuisée la nouveauté introduite par la navigation dans des documents hypertextes passifs, le besoin s'est fait sentir de définir des mécanismes pour assurer un minimum d'interactivité.

Il s'agissait d'abord de transmettre à l'utilisateur des informations sur les données auxquelles il accède (dates de création ou modification, taille des fichiers à télécharger, nombre d'accès, *etc.*), puis de réaliser des interfaces pour accéder en lecture ou en écriture à des données plus structurées.

Ce manque d'extensibilité du modèle HTTP est une des faiblesses originelles de la métaphore avec un système d'exploitation orienté-objet [Black_94]. Les différentes extensions proposées, que nous résumons ci-après, tentent bien d'apporter des solutions opérationnelles mais manquent toutefois de réaliser une intégration sans heurt au modèle.

1.2.4.1 Les SSI (Server Side Includes)

Les serveurs HTTP ont mis au point un certain nombre de *macro*-commandes (*SSI*) qui accèdent aux variables du serveur ainsi qu'à d'autres données statistiques [NSCA_95].

Ces commandes sont incluses dans un document HTML sous la forme de commentaires HTML, ce qui permet à un autre serveur d'ignorer leur interprétation :

<!-#commande tag1="valeur1" tag2="valeur2" -->

Lors du chargement d'une page HTML, le serveur exécute ces macro-commandes et les remplace par le résultat de leur interprétation.

1.2.4.2 Les CGI (Common Gateway Interface)

L'adoption des formulaires dans HTML a précipité le besoin d'une interface programmatique. Le navigateur, à l'origine conçu pour la *consultation* de documents distribués, est utilisé pour requérir/renseigner des bases de données.

La validation d'un formulaire correspond à l'appel d'une URL, qui indique le nom d'un programme, à laquelle sont joints les paramètres d'entrée (concaténés à l'URL pour une méthode GET, inclus dans un champ MIME pour un POST).

L'interface *CGI* [McCool_94, Gundavaram_96] définit un ensemble de variables normalisées que le serveur met à la disposition du programmeur, ainsi que le format des paramètres qu'il accepte en retour.

L'exécution du programme CGI, implémentant les traitements dans n'importe quel langage (Perl, C, TK/TCL, Visual Basic, Lisp, *etc.*), est donc initiée par l'appel de l'utilisateur qui référence un programme dans une URL. Le serveur HTTP, qui ne sait que renvoyer du code HTML, s'adresse à son interface CGI à laquelle il envoie en entrée les paramètres d'exécution transmis par l'utilisateur. L'exécution du programme renvoie, en résultat, du code HTML formaté que le serveur HTTP réémet en destination de l'utilisateur.

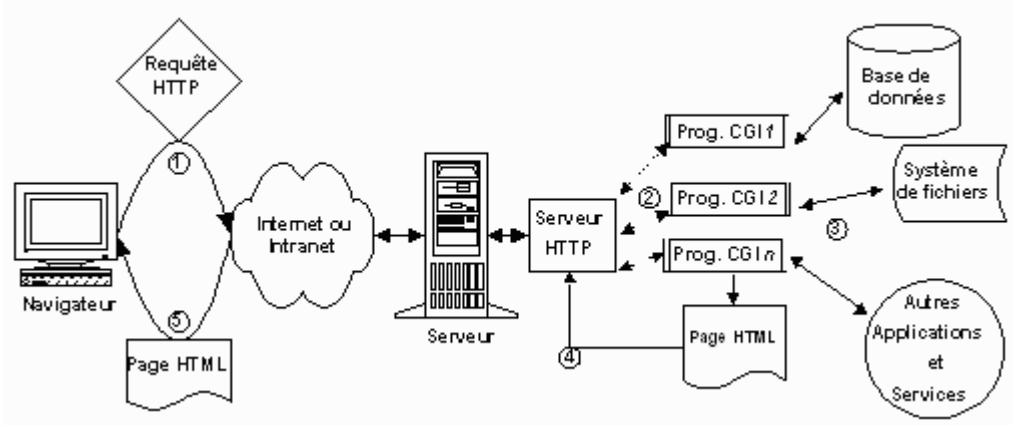


Figure 0. 2. Cheminement d'une requête au travers d'un programme CGI.

Aux premières interfaces, réalisées à l'aide de programmes spécifiques en C, C++, Perl ou TK/TCL, ont succédé de nombreux *méta*langages structurés qui permettent désormais de concevoir des applications modulaires rendant aisés l'accès et la mise à jour de bases de données issues d'applications traditionnelles.

Toutefois, et en dépit de leur standardisation, les CGI présentent des inconvénients tels que leur usage est incompatible avec un service performant et complexe. À l'instar du protocole HTTP, non connecté et sans état, chaque appel à une CGI crée un nouveau processus dont la durée de vie est limitée à son bref temps d'exécution.

Ce mécanisme, coûteux en ressources (CPU, temps et mémoire), initialise son contexte à chaque appel et rend acrobatique la gestion de transactions. Il s'avère être un goulet d'étranglement pour un serveur qui doit parfois exécuter plusieurs dizaines de copies simultanées d'un même programme.

1.2.4.3 Les API (Application Programming Interface)

La généralisation des CGI pour servir des pages dynamiques ayant mis en évidence leur manque de performance et de fiabilité, les fournisseurs de serveurs HTTP ont publié un ensemble d'API spécifiques qui en circonviennent les limites : un processus constamment actif sur le serveur, gère un contexte global d'exécution.

Chaque appel de l'utilisateur crée un processus fils (ou un *thread*) qui hérite du contexte global et est susceptible de l'enrichir. Il devient possible d'exécuter des transactions complexes.

1.3 Les objets, de base

Nous avons ainsi parcouru quelques décennies de l'histoire de l'informatique qui, au travers de la naissance du « réseau des réseaux », ont conduit à la création d'un ensemble de protocoles de communication et de propositions de nommage universel.

La diffusion de micro-ordinateurs de plus en plus performants et connectés en réseau, liée au besoin croissant d'accéder à des bases de données transversales, *multimedia* et partagées, a abouti à l'emballement des services, technologies et architectures dont nous sommes aujourd'hui témoins.

L'ensemble des standards élaborés pour assurer un format d'échange et des protocoles de communication de ces données a érigé une véritable *Tour de Babel*, finalement sans architecture ni organisation.

Si cette hétérogénéité correspond bien à celle de l'Internet elle répond mal aux besoins des entreprises qui voient enfin avec l'émergence de l'intranet la possibilité de réaliser les promesses, non tenues jusqu'ici, du client-serveur.

Le problème de l'existence et du déploiement d'un *client universel* est en passe d'être résolu. L'hétérogénéité des *serveurs* d'application et leur incompatibilité vont sans doute, à leur tour, poser des problèmes d'autant plus aigus qu'un nombre croissant d'utilisateurs privilégiera ce mode de connexion et exigera une cohérence et l'interconnexion des données proposées.

Les applications verticales et cloisonnées qui formalisent dans des représentations incompatibles les mêmes données vont vite se révéler des freins à l'évolution des systèmes d'information.

De nombreux modèles déclinent des architectures de *bus logiciels* qui, s'interfaçant entre clients et serveurs, visent à assurer l'interopérabilité, la transparence, l'efficacité, la distribution et la *réutilisabilité* des différents *composants*.

Nous rappellerons, dans le chapitre 2, les propriétés qui font de l'objet un modèle de représentation des connaissances désormais incontournable. Nous présenterons ensuite les grandes lignes de *CORBA*, seule proposition d'objets distribués qui ait à ce jour franchi les différentes étapes de normalisation.

Nous distinguerons, dans le chapitre 3, les multiples facettes d'un intranet qui s'intègrent naturellement dans une architecture distribuée, alors décrite fonctionnellement.

Nous montrerons, dans le chapitre 4, qu'un modèle objet réalisant une implémentation concrète dans l'entreprise de la *réutilisation* de *composants*, apporte une réponse qui intègre l'intranet au Système d'Information traditionnel.

Nous établirons ensuite un parallèle fonctionnel entre notre solution, proposée ici, et un ORB. Nous montrerons qu'il est possible, pour accompagner les évolutions de l'entreprise vers une architecture normalisée, de formuler un ensemble d'hypothèses qui, simplifiant celles qui guident l'architecture commune de l'OMG, permettent de proposer une approche incrémentale pour réaliser un bus logiciel (quel qu'il soit).

2. Introduction aux objets et aux systèmes distribués

Trop souvent, l'élaboration des logiciels reste une activité parcellaire et dispersée. Si l'informaticien construisait des meubles, il commencerait par planter des glands, débiterait les arbres en planches, creuserait le sol à la recherche de minerai de fer, fabriquerait une forge pour faire ses clous et finirait par assembler le tout pour obtenir un meuble.

[Muller_97, p. 196]

2.1 Évolution des représentations *35*
2.2 Les objets *36*
2.2.1 Classes 36
2.2.2 Héritage 37
2.2.3 Métaclasses 38
2.2.4 Envoi de message 39
2.2.5 Encapsulation 40
2.2.6 Polymorphisme 40
2.2.7 Agrégation - Délégation 41
2.3 De l'objet aux systèmes distribués *43*
2.4 Différentes architectures distribuées *45*
2.4.1 Pourquoi CORBA ? 47
2.5 CORBA *48*
2.5.1 L'architecture de l'OMG 48
2.5.2 IDL 50
2.5.3 Architecture générale de l'ORB 51
2.5.4 Activation d'une implémentation 54
2.5.4.1 Indépendance des accès à la localisation 54
2.5.4.2 Interopérabilité des ORB 55
2.5.5 CORBA/IIOP: le futur du Web ? 56
2.5.6 CORBA : synthèse 58

2.1 Évolution des représentations

Il pourrait sembler à un observateur sceptique que la renaissance et l'engouement croissant pour les langages et les méthodes orientés objet ne sont que l'expression d'une tendance éphémère, basée sur des modélisations artificielles et peu concrètes. Il n'en est rien cependant : la détermination de modèles pour représenter les connaissances a de tous temps été au cœur des réflexions des philosophes puis des psychologues cognitivistes longtemps avant d'intéresser la communauté des informaticiens.

Sans nécessairement remonter aux syllogismes des grecs anciens, les pionniers de l'informatique et de la cybernétique [Rosenblueth_43, Neumann_48, Newell_57] s'interrogent déjà sur la meilleure manière d'*identifier* et de *classifier* les comportements humains afin de les *modéliser* et de les *reproduire* à l'aide de robots ou d'ordinateurs. Il est en effet rapidement apparu aux chercheurs que le mode d'acquisition des connaissances, leur organisation et surtout leur représentation étaient à la base des raisonnements qui permettaient de résoudre des classes entières de problèmes (*cf.* les problèmes NP-complets, [Kayser_98]).

Déjà les premiers langages informatiques (Algol, Simula, Modula-2, Pascal, *etc.*) offraient-ils un mode de représentation de structures de données. Il a fallu cependant attendre les premiers travaux en *IA* de la fin des années cinquante [Newell_61] qui introduisirent la représentation déclarative des connaissances organisées par domaines d'*expertise*.

Ces représentations *analytiques* sous forme d'assertions logiques (ex : « un oiseau a deux ailes ») s'avèrent toutefois rapidement insuffisantes, dans la mesure où elles ne s'intéressent qu'aux connaissances sans se soucier de la manière dont elles sont développées ni à leur organisation. Seul un rapport *ensembliste* est établi avec le réel.

Une approche *structuraliste* [Quillian_68] permet ensuite de considérer les connaissances comme un réseau de concepts liés entre eux (des réseaux sémantiques) puis comme des schémas (*frames*) [Minsky_74], utilisés pour déterminer un cadre général au problème posé (un contexte) et décrire le déroulement de *scripts* qui mettent en œuvre une succession de connaissances pour le résoudre.

Parallèlement, les travaux conjoints en IA et en Recherche Opérationnelle [Laurière_76] confirment que la résolution d'un certain nombre de problèmes est étroitement liée à la capacité d'un système (*ALICE* en l'occurrence) d'en choisir une bonne représentation interne et de pouvoir la modifier, le cas échéant. Ils montrent notamment qu'un système généraliste peut utiliser des heuristiques (qui adaptent la représentation) pour apporter des solutions au moins aussi efficaces que des algorithmes spécifiques, câblés.

Les différents modèles étudiés mettent systématiquement en avant d'une part l'importance de la structuration des connaissances et d'autre part l'importance de

disposer et de structurer aussi les connaissances nécessaires pour manipuler correctement ces connaissances [Pitrat_66, Tello_89, Simon_91, Steier_96]. Il apparaît que ces *métaconnaissances* sont intrinsèquement de même nature que les connaissances qu'elles manipulent et ne justifient pas la définition de modèles de représentation spécifiques [Greiner_80, Goldberg_83, Pitrat_90].

2.2 Les objets

L'importance de la structuration des données est devenue tellement prégnante qu'un nouveau type de programmation, dirigé par les données (*data driven*), a été conceptualisé. Les données sont regroupées par « paquets » qui sont une modélisation informatique (appelée *réification*) des *objets* du monde réel ainsi que des connaissances (elles mêmes des objets) nécessaires pour les manipuler [Cointe_85, Briot_87]. Une taxonomie de ces objets est réalisée, regroupant dans une arborescence de *généralisations* les ensembles d'objets (*classes*) ayant des propriétés et des comportements semblables [Barthes_90].

Un ensemble de propriétés et caractéristiques différencient les multiples implémentations et acceptions d'objets. Il nous a semblé utile ici de reprendre les points essentiels bâtis ou simulés par la plupart des modèles d'objets.

2.2.1 Classes

Une classe est l'abstraction qui spécifie la structure (attributs) et les propriétés (méthodes) communes à un ensemble d'objets qui en sont les instances. Elle correspond à un prototype qui va être utilisé pour créer de nouveaux objets (souvent par l'appel d'une méthode *new*) disposant automatiquement des attributs et méthodes de leur classe. Cette schématisation centralise la définition des objets dans un endroit unique (la classe) ce qui permet de ne définir le code des méthodes qu'une seule fois. Les instances pointent alors vers leur classe et ne contiennent que les valeurs des attributs.

La plupart des implémentations d'objet considèrent les attributs comme des propriétés privées *encapsulées* dans leur classe alors que les méthodes sont des propriétés publiques accessibles par l'ensemble des objets.

2.2.2 Héritage

L'héritage est le mécanisme qui généralise l'abstraction introduite par la définition de classes d'objets. Il permet de construire une hiérarchie de classes d'objets similaires, allant du plus général au plus spécifique. Chaque classe fille « hérite » de la structure et des méthodes de sa classe mère (ou super-classe), qu'elle peut redéfinir (surcharger) localement. Elle complète (spécialise) éventuellement sa structure et rajoute des méthodes qui seront à leur tour transmises à d'éventuelles sous-classes.

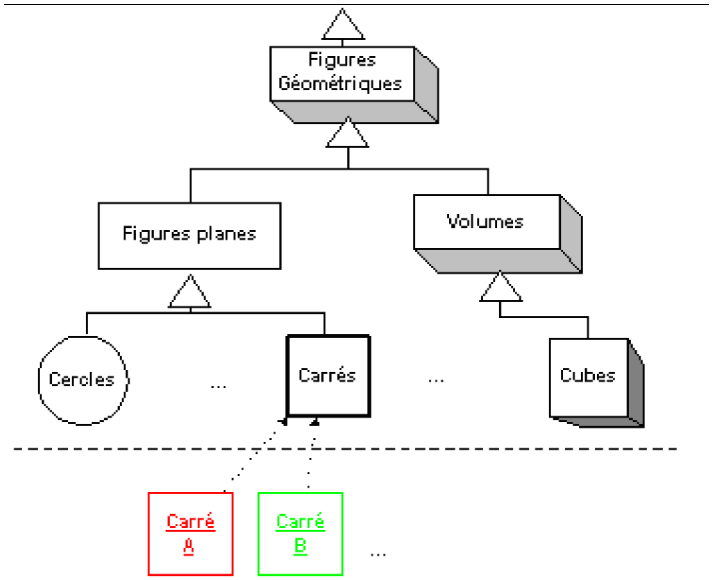


Figure 2. 1. L'héritage permet de créer une hiérarchie de classes de plus en plus spécialisées.

En fonction des modèles, un héritage peut être *simple* ou *multiple* selon qu'une sous-classe hérite d'une ou de plusieurs classes. Ce dernier cas, utile au demeurant, n'est cependant pas sans poser de nombreux problèmes de cohérence de structure pour lesquels il n'y a pas à ce jour de solution satisfaisante [Gardarin_90, Tisseau_96, Muller_97]. C'est pourquoi la plupart des langages orientés-objet tels SmallTalk, CLOS, Ada ou Java limitent l'héritage à sa forme simple pour éviter de devoir implémenter des algorithmes de résolution particuliers (C++, Eiffel).

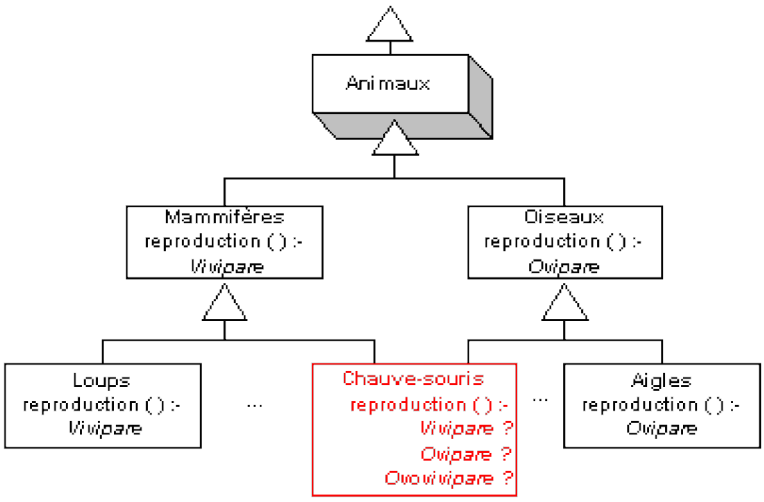


Figure 2. 2. L'héritage multiple peut entraîner des incohérences dans la modélisation.

2.2.3 Métaclasses

Beaucoup de langages objets figent l'arborescence des classes en séparant leur représentation de celle des objets (C++ notamment). Cette dichotomie impose cependant de recompiler le schéma des classes à chaque modification ou création de classe. L'introduction d'une *métaclasse*, classe de classes, permet alors de considérer un monde « tout objet » où l'ensemble des classes deviennent des objets ordinaires, instances de leur métaclasse et créés dynamiquement par l'appel d'une méthode *new* de leur métaclasse. L'ensemble des opérations applicables aux classes sont alors « classiquement » décrites dans cette métaclasse.

Afin d'éviter une récursivité infinie dans la hiérarchie des métaclasses (si une classe est un objet dont la classe est une métaclasse, alors il existe une *méta*-métaclasse qui est la classe des métaclasses, *etc.*), *ObjVlisp* [Briot_87] modélise une double hiérarchie où :

1. toutes les classes sont sous-classes d'une classe sauf une classe -*Object* - qui est la racine unique de l'arbre d'héritage ;
2. tous les objets sont instances d'une classe et, par *bootstrap*, une classe-mère - *Class* - est instance d'elle-même.

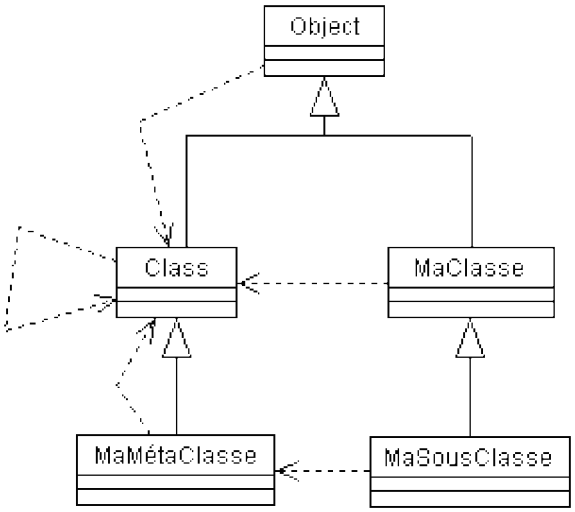


Figure 2.3. Le modèle ObjVlisp permet une représentation finie du « tout objet », où la classe « Class » est la matrice de toutes les métaclasses du système.

2.2.4 Envoi de message

Les objets communiquent entre eux par l'envoi et la réception de *messages* composés d'informations structurées sous la forme d'un sélecteur de méthode (son nom) et des paramètres passés à la méthode. Selon le type du langage, la *liaison*, qui consiste à associer le code effectif à exécuter lors de la réception d'un message par un objet, est réalisée à la compilation (C++) ou à l'exécution (Smalltalk, Java).

La majorité des langages orientés objet ne supportent cependant que l'envoi de messages *séquentiels* et *synchrones* qui correspondent à l'exécution par l'objet d'une fonction définie localement et du renvoi du résultat à l'appelant.

La possibilité d'émettre des messages *asynchrones* et *parallèles* donne aux objets un rôle d'*acteurs* [Briot_89, Ferber_89, Ferber_95] justifié par l'autonomie apparente dont disposent alors ces objets. En effet, le résultat d'un message asynchrone n'est plus renvoyé à l'expéditeur mais spécifie les coordonnées *statiques* ou *dynamiques* d'un objet destinataire, appelé *continuation locale*.

D'autres modèles de communication ont aussi été définis pour tenter de pallier les limitations des envois synchrones. Le modèle du tableau noir (*blackboard*) [Hayes-Roth_85] offre aux objets un espace en libre accès, le tableau noir (lui-même un objet), que l'ensemble des objets concourant à la résolution d'un problème peuvent consulter pour se tenir informés de l'avancement de certaines tâches ou dans lequel ils peuvent enregistrer les sous-buts qu'ils ont résolus, ceux qu'ils se proposent d'étudier, *etc.*

2.2.5 Encapsulation

L'encapsulation est une des propriétés les plus importantes des objets, dans la mesure où elle leur *permet* de masquer leurs données privées au travers d'un ensemble de méthodes qui vont constituer leur interface publique.

Seule la classe d'un objet peut ainsi accéder à ses attributs (son interface privée), l'ensemble des autres interactions se font nécessairement par le biais de l'interface publique.

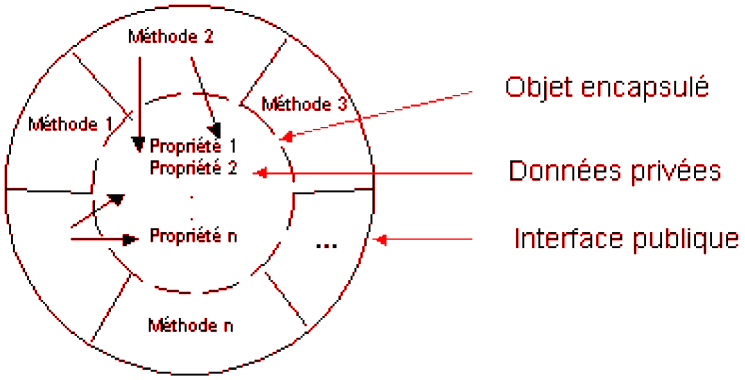


Figure 2.4. L'encapsulation isole la structure de l'objet des méthodes qu'il implémente

Cette encapsulation équivaut à un contrat passé par une classe et le reste du monde qui consiste à laisser l'objet gérer librement ses propres ressources. Il indique, en échange de cette liberté, les méthodes qu'il honore ainsi que les paramètres qu'il en attend. La classe de l'objet se réserve ainsi le droit de modifier l'implémentation effective des méthodes publiées mais aussi de modifier sa propre structure (ajout ou destruction de propriétés privées).

2.2.6 Polymorphisme

Le polymorphisme est en fait une conséquence immédiate de l'encapsulation. Une fois l'interface d'un objet publiée, il n'est absolument plus possible d'en déterminer l'implémentation exacte. Rien n'empêche donc plus deux classes d'objets différentes, de publier deux mêmes méthodes avec des signatures identiques (nom et arguments semblables), sans pour autant que ces deux méthodes aient la même implémentation effective.

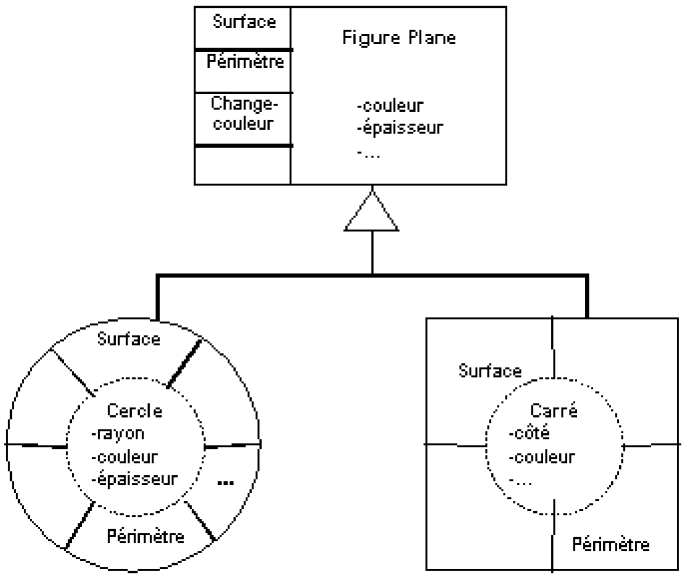


Figure 2.5. Le polymorphisme permet d'implémenter une même méthode avec des algorithmes différents

Classiquement, les classes « carré » et « cercle » définiront chacune une méthode de calcul de « surface » et il est évident que l'algorithme sera différent selon qu'il s'applique à un cercle ou un carré. L'intérêt du polymorphisme est de pouvoir s'adresser de façon transparente à une catégorie d'objets que l'on sait partager des propriétés semblables (la classe des « figures planes » par exemple).

2.2.7 Agrégation - Délégation

L'**agrégation** et la **délégation** sont deux façons commodes de réaliser un héritage dans des modèles qui n'en prévoient pas de structurel. Elles réduisent le couplage entre les classes (plus de transmission automatique des propriétés et des attributs) et permettent notamment à une classe d'être construite à partir de composants agrégés dans une relation du type maître-esclave, l'agrégation pouvant être vue comme une relation « se compose de ».

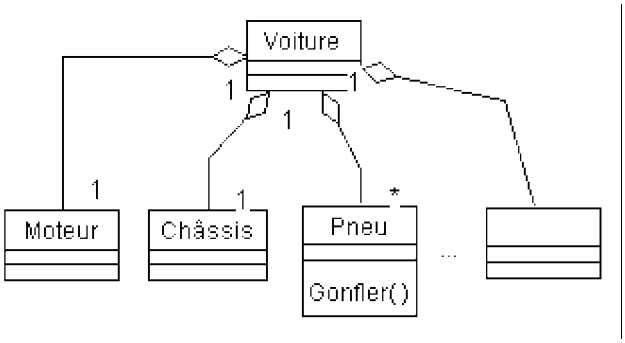


Figure 2.6. L'agrégation permet de décrire une « voiture » comme l'agrégation d'un « moteur », d'un « châssis », etc. La classe « pneu » délègue ainsi sa méthode « gonfler » à la classe « voiture ».

La **délégation** met à la disposition de la classe délégataire un ensemble de pointeurs vers les propriétés des classes déléguées (il est possible qu'une délégation se fasse par transmission du code des méthodes). Il devient possible de réaliser des schémas d'héritage complexes qui gèrent de façon dynamique les propriétés à utiliser (grâce à une implémentation par pointeurs) et de simuler un héritage multiple si nécessaire. C'est ainsi que X Window [Young_90], réalisé en C, implémente totalement « à la main » l'héritage grâce à un ensemble de pointeurs vers les structures des classes « parentes ». Le modèle OLE2/COM [Microsoft_93a] institutionnalise quant à lui l'agrégation-délégation comme **seul mécanisme d'héritage connu**.

2.3 De l'objet aux systèmes distribués

Les langages objets ont prouvé depuis plusieurs années (SmallTalk-72 [Kay_76]) que la réutilisabilité des composants est réelle et que les temps de modélisation et de réalisation des applications peuvent être réduits par des facteurs importants (de l'ordre de 50% [Bouzeghoub_97]).

[Coad_91] a aussi montré qu'une conception « objet » est possible et garde son intérêt même avec des implémentations traditionnelles, dans la mesure où elle a l'avantage d'accroître la productivité des concepteurs et de faciliter la documentation et la maintenance des logiciels.

Plusieurs méthodes objets ont par ailleurs été adaptées à partir de méthodes existantes (OOD [Booch_91], OOM [Bouzeghoub_94], OMT [Rumbaugh_96], UML [Booch_95b], *etc.*). Elles déclinent des modèles de représentation expressifs dans des langages et des outils habituels à l'entreprise.

Si la modélisation d'une application peut désormais être appréhendée par une quelconque de ces méthodes avec des avantages indiscutables, le choix de l'architecture sous-jacente est loin d'être aussi clair :

- les objets distribués ne suffisent pas à réaliser une application distribuée. Ils doivent être combinés, utiliser des canaux et des mécanismes de communication standardisés, assurer la persistance des données qu'ils modélisent, gérer des transactions, un nommage transversal , *etc*. [Coulouris_95].
  Il faut, en l'occurrence, se soucier d'homogénéiser la *sémantique* de données après que celle de leur *structure* a été assurée ;
- le stockage et la gestion des données constituent un point critique auquel sont confrontés les développeurs lorsqu'ils utilisent des bases de données relationnelles dans des environnements de programmation orientée objet. Les SGBD ne sont pas conçus pour stocker des objets mais des *relations* entre objets. Une multiplication de jointures de tables est nécessaire pour apporter le niveau de souplesse des objets aux structures de données bi-dimentionnelles et statiques (au prix, de surcroît, d'une perte sémantique).

Il est fréquent que les contraintes de programmation propres à l'accès aux bases de données relationnelles annulent les avantages que confère l'utilisation des objets ;

- l'intégration d'applications hétérogènes est un autre problème crucial auquel sont confrontées les entreprises, désireuses d'améliorer et d'enrichir leur système d'information tout en maintenant et en faisant vivre des applications plus anciennes, généralement monolithiques.

Les différentes stratégies de maintenance qui nécessitent la maîtrise de techniques complexes (langages de programmation, réseau, bases de données et micro-informatique) ont, à ce jour, été résolues pratiquement par l'adoption d'interfaces propriétaires ou spécifiques, complexes et difficilement évolutives.

Ces solutions *ad hoc* limitent souvent les marges de manœuvre des entreprises qui se retrouvent devant le dilemme de figer leur architecture ou de suivre indéfiniment les évolutions technologiques de leurs fournisseurs sans planification ni grande maîtrise de leur déploiement.

Cette complexité croissante des technologies [Bengrid_96a] a amené la plupart des acteurs du marché à définir un *middleware* qui s'appuie sur des technologies objet, en repensant profondément les architectures et les méthodes de programmation utilisées.

Le but poursuivi est de mettre en place des interfaces maîtrisables et évolutives auxquelles puissent être rattachées, de façon incrémentale, les applications, en fonction des besoins de l'entreprise.

Ces interfaces, s'appuyant sur un « bus » commun, encouragent une factorisation et une homogénéisation des données qui ne nécessitent pas une refonte complète de l'existant [Schmidt_95].

Elles ouvrent par ailleurs la possibilité de « composer » des applications à partir de sous-ensembles développés indépendamment qui simplifient la modélisation, l'implémentation et la maintenance d'ensemble en réduisant de surcroît les cycles de développement.

2.4 Différentes architectures distribuées

Ces dix dernières années ont vu naître plusieurs familles d'architectures : *DCE*, *CORBA*, *COM* et, plus récemment, JavaBeans.

- **DCE** [Rosenberry_92, OSF_93a, OSF_93b] a sans doute été la première concrétisation, inspirée par l'*OSF*, de la distribution d'applications (procédurales) sur un ensemble de processus distants. Basée sur les *RPC*, elle offre, parmi de nombreuses autres fonctionnalités, un système de nommage (local *CDS* et global *GDS*, basé sur X.500 [CCITT_X.500]), d'authentification et de sécurité utilisant *Kerberos* V5 [Steiner_88, Kohl_93] et décrit un ensemble de passerelles vers d'autres serveurs de ressources.

  Son semi-échec s'explique par sa complexité, par ses faibles performances au vu des charges CPU et réseau induites et surtout par l'usage de RPC comme mécanisme général de communication. Ces procédures, synchrones, nécessitent en effet que l'ensemble des serveurs, simultanément actifs, traitent une requête pour que l'application puisse continuer à s'exécuter.

  Une faiblesse structurelle de DCE réside par ailleurs dans la conception procédurale de son *IDL*, inspiré du C, qui n'implémente pas l'héritage des interfaces et définit un espace de nommage unique (à plat) [Brando_95b].

  DCE a néanmoins inspiré les deux grandes architectures actuellement en compétition, *CORBA* et *DCOM*. De nombreuses améliorations apportées ces dernières années [Leddy_93, Viveney_94, Dilley_95] ont aussi défini des passerelles vers l'architecture de l'*OMG*.

- L'**OMG** est, à l'instar de l'OSF, un consortium international créé en 1989, qui regroupe à ce jour plus de 700 membres dont des concepteurs de logiciels, des architectes de systèmes et des utilisateurs. Il s'est fixé pour mission de promouvoir la théorie et la pratique des technologies orientées objet dans la conception de logiciels.

  La démarche suivie est complexe dans la mesure où elle consiste à lancer des appels d'offre (*RFP*) pour chaque norme, suivis d'une concertation pour faire converger les différentes propositions. Une période d'évaluation des solutions puis l'étude des problèmes soulevés est couronnée par une série de votes qui officialisent par consensus les normes proposées.

  Ces itérations, lourdes et coûteuses en temps (sans compter les oppositions issues d'intérêts commerciaux divergents), ont conduit à l'élaboration de la série de normes dont est issue CORBA (ORB, CORBAServices, CORBAfacilities, *etc.*).

- En dépit de son appartenance à l'OMG, ***Microsoft*** a construit, au fur et à mesure de ses besoins, des solutions ponctuelles aux problèmes de communication rencontrés par ses différents outils. Cette construction « anarchique » se décline par les *DDE*, *OLE* [Microsoft_93a], *OLE Automation*, contrôles OLE, COM , DCOM, *ActiveX* et autre DirectX.

  L'absence de transparence de *Microsoft* (qui publie des spécifications incomplètes de ses protocoles), ainsi qu'une politique focalisée sur le marketing autour de ses produits en rendent l'analyse très confuse.

  L'exemple de DCOM est ainsi une illustration de ces comportements singuliers : *Microsoft* implémente une version propriétaire de DCE pour étendre à une architecture répartie les mécanismes établit pour COM (rajout de paramètres aux fonctions standards, changement de nom des fonctions, reprise des RPC de DCE comme « couche de transport » et enrichissement de son IDL en *ODL*, *etc*.).

  La tentative de normalisation (et de portage) de DCOM a par ailleurs fait long feu. La version du protocole proposée à l'*IETF* (*draft*) [Brown_96] a expiré en mai 1997 sans connaître de suite.

  Le quasi monopole des systèmes *Microsoft* sur les postes de travail ainsi que sa grande versatilité ont néanmoins abouti à un ensemble de solutions propriétaires. Celles-ci n'offrent aucune des garanties de généralisabilité [10] que promeuvent les architecture de l'OSF ou de l'OMG mais sont, en revanche, simples à mettre en œuvre par les utilisateurs, avec un investissement organisationnel et financier minimal à court terme [11].

- Le succès du **Web** a bouleversé l'ensemble des architectures distribuées de ces dernières années en s'imposant *de facto* comme l'architecture distribuée la plus répandue, bien que basée sur un protocole rustique et *a priori* peu efficace.

  C'est ainsi, qu'après avoir échoué en 1995 dans sa tentative de se l'approprier, *Microsoft* a procédé à un revirement stratégique total et met désormais l'Internet au cœur de ses nouvelles architectures DCOM et ActiveX.

  L'OMG a, en revanche, rapidement saisi l'ampleur du phénomène « Web » et a intégré *IIOP* à CORBA en 1996 pour montrer sa volonté de tenir compte de ce succès. Il s'est aussi associé au *W3C* pour contribuer à l'élaboration des nouvelles architectures.

  Le souci de consensus de l'OMG l'a amené à proposer un standard pour l'interopérabilité de COM et CORBA, en 1995. Ce standard, approuvé en 1996, fait désormais partie intégrante des spécifications de CORBA 2 [OMG_96a].

---

[10] Nous n'avons pas trouvé dans la littérature une traduction satisfaisante du terme anglais *scalability.*

[11] Arguments qui séduisent généralement les utilisateurs, peu enclins à procéder à des études d'architectures globales.

- Le dernier venu, **Java** de *Sun* [Flanagan 96], a un grand impact (surtout médiatique, à défaut de réalisation probante à ce jour). Il pourrait apporter au Web la diversité, la souplesse et la dynamique qui lui manquent. Le langage, interprété ou compilé à la volée (***J**ust **I**n **T**ime*) par une machine virtuelle, incluse dans un navigateur, ouvre la porte à la distribution d'applications portables sur une majorité de plates-formes.

Son inspiration du C++ mais aussi son indépendance par rapport à un système d'exploitation et à un navigateur (quoique ?) représente pour la communauté de l'Internet une dernière alternative au monopole de *Microsoft*. Cette alternative rend son succès quasiment incontournable dans l'ensemble des architectures de l'Internet en dépit de son manque de maturité, d'erreurs de conception, de performances encore discutables et de l'état de prototype de son modèle de composants distribués, RMI-JavaBeans.

2.4.1 Pourquoi CORBA ?

Le principe de la répartition des objets ou des composants est aujourd'hui acquis. L'architecture de l'OMG s'est appuyée sur l'expérience accumulée par l'OSF avant de se spécialiser, pour proposer une interopérabilité entre clients et serveurs.

Elle réunit aujourd'hui autour d'elle une quasi-unanimité. Les propositions concurrentes que sont DCOM de *Microsoft* et RMI-JavaBeans de *Sun* offrent au minimum des passerelles vers CORBA. Elles disposent souvent, en plus de leurs mécanismes propres, des possibilités d'interconnexion natives à un ORB et se sont souvent inspirées, directement ou non, des réflexions menées par l'OMG.

Ce chapitre aborde plus de trente ans d'informatique vus, cette fois, sous l'angle de la lente maturation de la structuration des données puis des différents mécanismes assurant leur répartition. Cette répartition des objets, composants et autres agents mobiles, foisonne depuis quelques années et trouve dans l'Internet et ses déclinaisons un vecteur de propagation idéal.

Aussi notre propos n'est-il pas de dresser ici un tableau comparatif - qui ne risquerait pas d'être exhaustif - des différentes architectures en compétition ni d'établir un pronostic sur leur pérennité respective. Il nous paraît cependant important de montrer, par le biais de la première architecture répartie normalisée, l'état des réflexions de centaines de chercheurs, spécialistes de chacun des points traités et qui commence à recueillir en 1997 le fruit de ses efforts en termes de fiabilité et de réalité commerciale.

2.5 CORBA

2.5.1 L'architecture de l'OMG

Depuis 1989, l'*Object Management Group* s'est attelé à définir une norme pour l'intégration d'applications distribuées, basée sur l'approche objet.

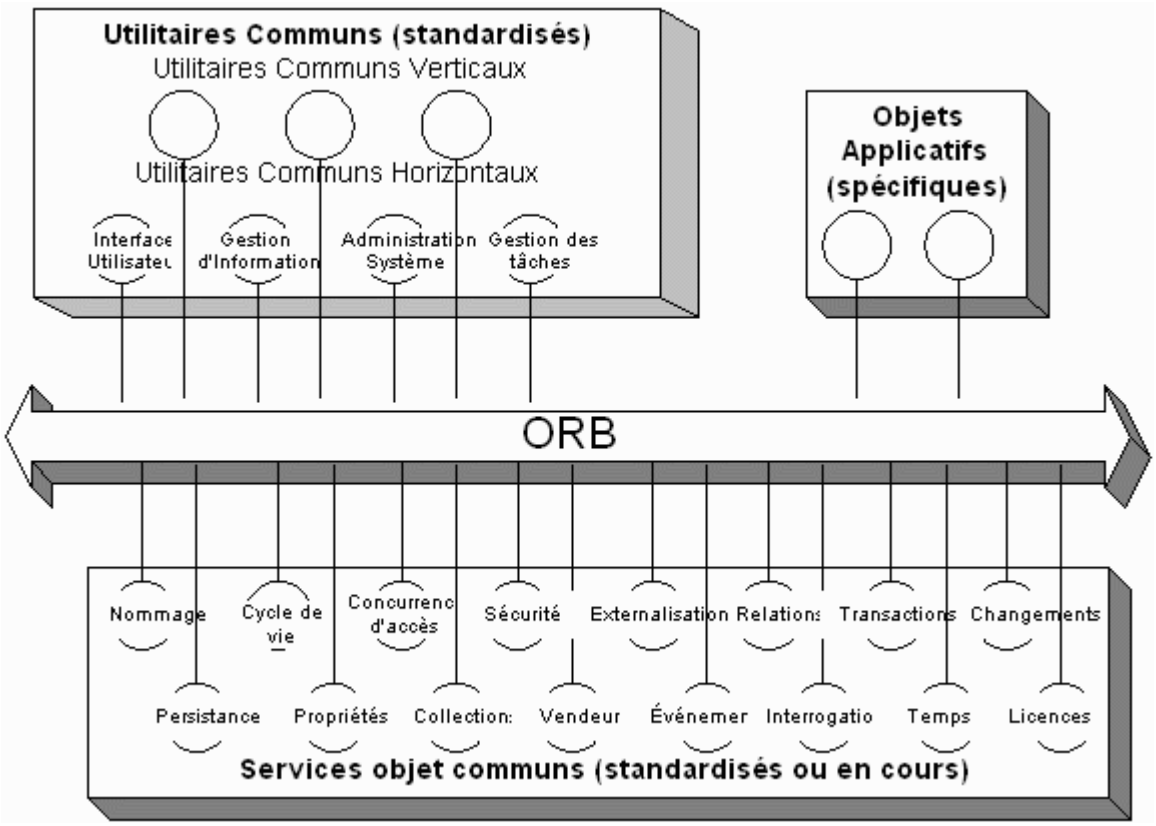


Figure 2. 7 L'architecture définie par l'OMG.

La première phase de travail de l'OMG a abouti à la définition d'un modèle de référence, l'*OMA*, qui décrit les concepts utilisés, dans le but d'aplanir les différences d'implémentation de ces concepts entre ses nombreux membres [OMG_92].

Ce modèle de référence s'articule autour de plusieurs composants :

- un « bus logiciel », le **négociateur des requêtes objet** (*ORB*) dont la fonction est d'assurer que des applications, écrites par des fournisseurs différents pour des systèmes d'exploitation divers, communiquent entre elles indépendamment de leur localisation [Mowbray_95] ;
- un langage de définition d'interfaces, l'*IDL*, qui permet de décrire de façon déclarative l'ensemble des objets et leurs interactions ;
- des **services communs** (*CORBAservices*) [OMG_93a, OMG_94b, OMG_94c] tels le nommage, la sécurité, le cycle de vie, *etc.* enrichissent l'architecture en précisant les spécifications de concepts de niveau *système.*
  Seize services de base ont été identifiés à ce jour [OMG_95b, OMG_96b] ;
- des **utilitaires communs** (*CORBAfacilities*) [OMG_94d, OMG_96c] fournissent (en IDL) des *canevas* applicatifs indépendants, horizontaux (une interface utilisateur, la gestion des tâches, *etc.*) et verticaux (gestion documentaire, comptabilité, *etc.*) directement intégrés par les objets de métier (à la manière de composants dans un circuit intégré, d'où le terme « canevas ») ;
- des **objets applicatifs** (*Object Applications*) peuvent aussi être définis (toujours en IDL). Ce sont des composants spécifiques à une application et qui ne font donc pas l'objet de standardisation.

Un des buts principaux visés dès l'origine de cette architecture est de permettre aux clients d'invoquer le bus logiciel en appelant statiquement ou dynamiquement des méthodes qui appartiennent à des objets distants.

L'Architecture Commune pour l'ORB, *CORBA* (1.1) [OMG_91] déterminée en 1991, regroupe autour de ce bus un ensemble d'API qui spécifient les interactions entre un ORB donné et des clients/serveurs différents.

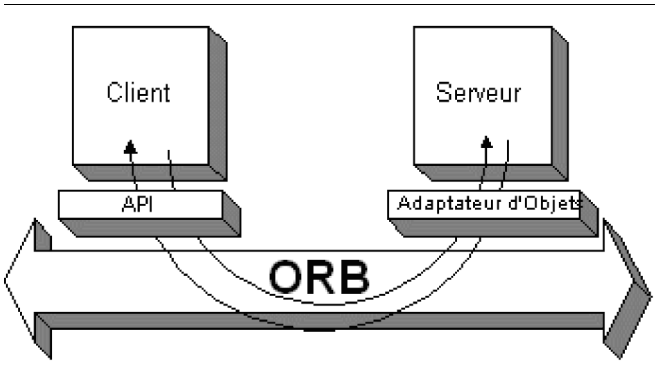


Figure 2. 8 Une requête client-serveur transitant par l'ORB.

Ce n'est qu'avec CORBA 2.0, adopté en décembre 1994 [OMG_96a], qu'est comblé un ensemble de lacunes constatées dans ses versions 1.1 et 1.2 et que sont posées les bases d'une vraie interopérabilité, en spécifiant des protocoles de communication standards entre ORB de fournisseurs différents.

2.5.2 IDL

Un ensemble réparti d'applications se décompose - naturellement - en objets implémentés dans différents langages de programmation. La stratégie classique pour permettre aux différents sous-ensembles de communiquer consiste généralement à concevoir des passerelles (ou demi-passerelles) pour partager un noyau commun de données. Cette multiplicité des langages entraîne toutefois une *fragilisation* de la construction globale, alors tributaire de la défaillance ou de la modification de ces implémentations particulières.

La solution générale, adoptée par l'OMG, consiste à abstraire les fonctionnalités et les paramètres des différents composants en mettant au point un **L**angage de **D**éfinition d'**I**nterface standardisé.

Inspiré de C++ auquel il emprunte la syntaxe l'IDL décrit, dans une grammaire spécifique, la représentation des classes d'objets (appelées *interfaces* dans la terminologie utilisée), précisant les *opérations* (les méthodes) qu'elles peuvent effectuer ainsi que leurs signatures, les classes dont elles héritent, les exceptions qu'elles génèrent, *etc.*

L'IDL n'est pas un langage de programmation en ce sens qu'il ne permet pas d'affectation de variables, d'exécution de boucles ou de branchements. C'est un langage qui sert à définir les structures - complexes - qui modélisent les différentes applications qui vont interopérer.

Ces définitions rendent alors possible la conception de projections standardisées (*mappings*) de ces objets de l'IDL vers des langages traditionnels. Certaines projections telles C [OMG_93b], C++ [OMG_94a], SmallTalk [OMG_96a] ou Ada [OMG_95a] ont été normalisées par l'OMG alors que d'autres sont en cours de normalisations ou sont étudiées en fonction des besoins exprimés (Java, Cobol, Visual Basic, Eiffel, *etc.*).

En pratique, chaque fournisseur d'ORB met à la disposition de ses clients une boîte à outils qui comprend, entre autres, un compilateur effectuant la projection de l'IDL vers des langages de programmation traditionnels.

L'avantage de la définition en IDL d'un objet est évident : le programmeur est totalement libre d'implémenter concrètement cet objet dans le langage de son choix. Il a l'assurance que les requêtes effectuées vers des objets distants se font indépendamment du langage.

De façon analogue, il peut requérir un quelconque objet distant dans son langage favori, sans être au courant des détails d'implémentation de sa classe. Il lui suffit de prendre connaissance de la définition de l'objet en IDL, qui peut être effectivement librement implémenté.

Il est de la responsabilité de l'ORB de transmettre correctement la requête entre ces différents objets et de les informer de sa complétion.

2.5.3 Architecture générale de l'ORB

Le développement traditionnel d'applications client-serveur s'appuie sur une architecture *ad hoc,* qui met en place des protocoles de communication spécifiques entre les différents sous-systèmes. Ces protocoles sont généralement dépendants des langages de programmation utilisés, du type de transport, du système d'exploitation, *etc.*

La mise en place d'un ORB ramène toutefois ces protocoles à la description en IDL des différents composants. Il devient loisible au programmeur de choisir le système d'exploitation ou l'environnement d'exécution qui lui conviennent et même le langage de programmation de chacun des composants du système en cours d'élaboration.

L'intégration de composants existants par ailleurs est ainsi rendue possible, dans la mesure où il « suffit » de modéliser en IDL un ancien composant (de manière identique à la modélisation d'un nouvel objet) puis d'écrire des procédures qui traduisent les requêtes entre l'ORB et l'application d'origine (l'opération est toutefois loin d'être triviale).

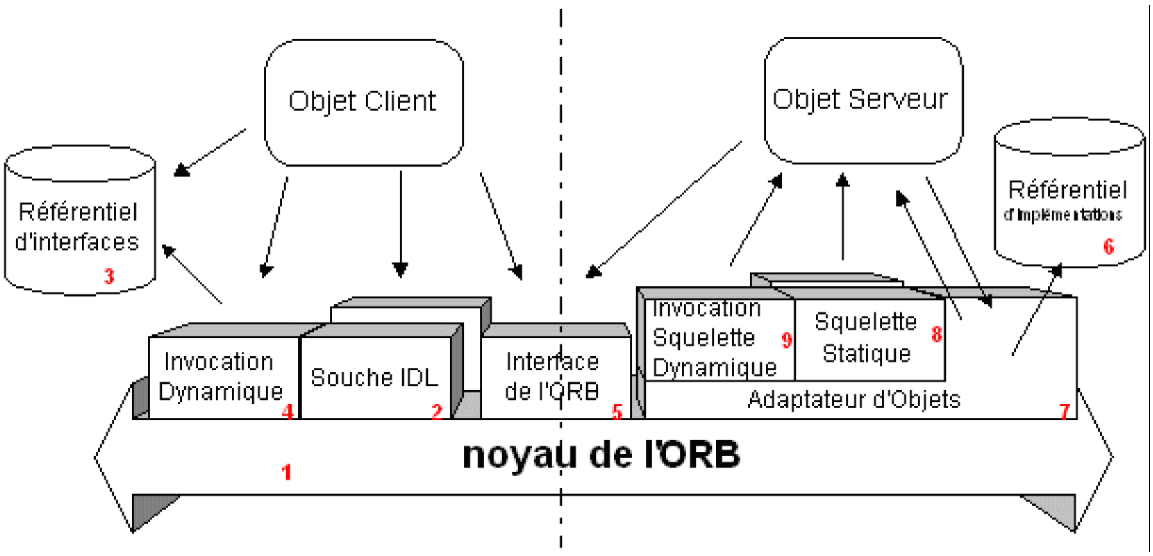


Figure 2. 9 Architecture schématique d'un ORB.

L'architecture schématisée ici, souligne bien - en dépit de sa complexité - l'indépendance à la localisation de CORBA.

Une *implémentation* (classe serveur) peut utiliser le même processus que le client ou se trouver sur la même machine, comme elle peut en être distante. L'ORB se charge de transmettre la requête du client au serveur, où qu'ils se trouvent respectivement.

Les différents services de l'ORB sont instanciables à partir du client comme du serveur (les notions de client ou de serveur sont toutes relatives puisqu'un objet peut jouer les deux rôles indifféremment) :

1. le **noyau de l'ORB** est le *middleware* proprement dit qui établit des relations (*via* une couche transport) entre objets-*clients* et objets-*serveurs*. Un client peut invoquer de façon transparente une méthode sur un objet faisant office de serveur, fût-il local ou distant sur le réseau. Il est de la responsabilité de l'ORB d'intercepter l'appel et de se charger de trouver un objet qui puisse implémenter la requête. Une fois l'objet trouvé, il lui transmet les paramètres d'appels, invoque la méthode correspondante et renvoie le résultat. Le client n'a pas à se soucier de la localisation de l'objet, du langage de programmation qu'il utilise, du système d'exploitation ou d'une quelconque propriété ne faisant pas partie de l'interface de cet objet ;
2. l'**Interface d'Invocation Statique** (**IIS**) se compose des *souches* (*stubs*) issues de la compilation (projection) des descriptions en IDL vers le langage cible (C++, SmallTalk, *etc.*). Ces souches assurent la transparence de l'invocation d'une méthode en offrant au client la possibilité d'intégrer les appels de méthodes à partir de son langage de programmation ; la requête étant traduite par la souche dans un format de message transmis au serveur ;
3. le **Référentiel d'Interfaces (IR)** est une base de *méta*-données (potentiellement distribuée) qui référence et détaille l'ensemble des interfaces IDL d'un ORB. C'est un moyen efficace et puissant dont dispose tout objet pour obtenir la description de l'ensemble des objets avec lesquels il peut interagir. L'IR est accédé et mis à jour grâce à un ensemble d'API standardisées ;
4. l'**Interface d'Invocation Dynamique (DII)** sert à construire dynamiquement les requêtes vers des objets. Des API permettent en effet d'interroger l'IR pour construire une requête à partir de sa signature et de ses paramètres.
   Cette interface est cependant bien plus délicate à mettre en œuvre qu'une souche. Elle nécessite de spécifier, à l'exécution, l'objet cible, l'opération ainsi que les paramètres, renvoyés par l'IR. Il faut ensuite générer la requête puis récupérer et traiter le résultat.

   La complexité de la démarche (et accessoirement sa moindre performance) offre néanmoins la souplesse nécessaire à l'ORB qui peut invoquer les objets qu'il veut sans avoir à connaître les souches correspondantes ;
5. l'**interface de l'ORB** (propriétaire et spécifique à chaque ORB) fournit, entre autres, un ensemble de primitives d'initialisation et de paramétrage du bus, de description de l'implémentation des objets ainsi que des fonctions de conversion des références d'objets (conversion entre chaînes de caractères et objets notamment) ;
6. le **Référentiel d'implémentations** décrit l'ensemble des classes d'un serveur, les objets qui y sont instanciés, les droits d'accès ainsi que toute information nécessaire à l'administration ou à la sécurité des objets ;

7. l'**Adaptateur d'Objets**, appelé par le serveur, est le « canif suisse » qu'utilise une implémentation pour accéder aux fonctions d'un ORB. C'est lui qui est chargé de faire croire à un client que l'objet requis est actif et prêt à lui répondre. Il interroge le Référentiel d'implémentations et fournit les méthodes (opérations) pour instancier les objets du serveur, leur affecter un identifiant et leur transmettre les paramètres d'exécution.

   Théoriquement, il devrait y avoir autant d'adaptateurs d'objets qu'il y a de familles de requêtes (accès à des SGBDR, SGBDO, *etc.*). L'OMG a préféré imposer un seul adaptateur d'objets élémentaire, le *BOA* qui s'adapte à une grande variété d'implémentations d'objets. Il fournit un environnement suffisant pour qu'une application serveur s'exécute en incluant notamment des interfaces pour générer et gérer les références aux objets, enregistrer des implémentations (des classes) qui constituent un ou plusieurs programmes, activer des implémentations (instancier les classes) par le biais de souches et authentifier des requêtes. Il fournit aussi un service limité de persistance aux objets qui peut être connecté à un service de nommage plus global (un service d'annuaires X.500 par exemple).

Les spécifications du BOA sont toutefois trop générales et incomplètes. Elles omettent notamment de préciser l'association des squelettes aux serveurs et de définir des API standards pour enregistrer les serveurs implicitement (à l'invocation du serveur) ou explicitement (par appel d'une méthode du BOA). Ces zones d'ombre ont abouti à des interprétations différentes du BOA qui limitent largement la portabilité inter-ORB des programmes.

L'OMG est actuellement (depuis mars 1997) en train de valider la définition d'un nouvel adaptateur d'objets, le *POA* [Schmidt_97], qui reprend les principes d'activation du BOA au travers d'un « gestionnaire d'instances » lié à chaque implémentation d'interface d'objet. Il gère ainsi des problèmes spécifiques mal couverts jusqu'ici tels la persistance des objets, la gestion et le contrôle des identifiants, l'activation de l'objet sur de longues périodes, la gestion d'accès concurrents, la répartition des charges ou encore la récupération des erreurs [Cattel_94].

Il est prévu que le POA remplacera le BOA dès son adoption en 1998, renforçant l'interopérabilité des ORB.

8. le **squelette statique (*Skeletton*)** est l'équivalent d'une souche au niveau du serveur : il est généré par le compilateur IDL et a pour fonction de récupérer, par le biais de l'Adaptateur d'Objets (d'où les problèmes de portage), les paramètres d'une requête et d'en renvoyer les résultats ;

9. l'**Invocation de Squelette Dynamique (*DSI*)** est l'équivalent, du côté serveur, du DII. Elle permet à des composants qui n'ont pas de squelette compilé de retrouver dynamiquement à partir du message entrant, l'objet-cible et les paramètres d'appel de la requête. Cette interface est notamment utilisée pour implémenter des ponts inter-ORB ou des interpréteurs de commandes qui génèrent à l'exécution des implémentations d'objets.

2.5.4 Activation d'une implémentation

L'activation d'une implémentation par le biais de l'ORB, se fait schématiquement en cinq étapes :

1. le client communique les paramètres d'appel de la méthode par le biais de la souche ;
2. l'ORB transmet la requête au BOA qui active l'implémentation (le serveur) ;
3. l'implémentation informe le BOA qu'elle est active et disponible ;
4. le BOA transmet la requête à l'implémentation par le biais du squelette ;
5. l'implémentation renvoie le résultat ou l'exception à l'ORB qui l'achemine au client.

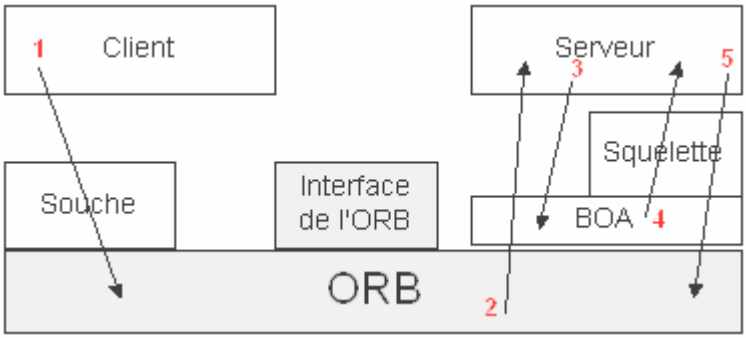


Figure 2. 10 Activation d'une implémentation.

2.5.4.1 Indépendance des accès à la localisation

Un des principaux buts cité par l'OMA est d'offrir l'accès aux objets de manière totalement indépendance de leur localisation. En l'occurrence, dans CORBA, les attributs relatifs à la localisation d'un objet sont cachés. Le client requiert un objet en spécifiant son identifiant (unique). L'invocation de l'objet renvoie sa référence, utilisée ensuite par le client pour invoquer des méthodes sur cet objet.

Si, en l'occurrence l'objet s'avérait distant, la référence renvoyée dénote l'usage d'un objet « mandataire » (*proxy*) qui se charge concrètement de véhiculer les requêtes vers l'objet distant. Cependant, ni le programmeur ni l'objet client n'ont besoin d'invoquer explicitement le *proxy*, toute requête se comportera de façon identique que l'objet soit local ou distant, exécuté par un même processus ou pas.

Seuls des effets de bords peuvent mettre cette transparence en défaut dans la mesure où l'accès à un objet distant peut provoquer des erreurs asynchrones qui laissent une requête dans un état indéterminé (comment savoir en cas d'erreur si l'objet à bien reçu le message ou s'il l'a traité avant que l'erreur soit intervenue ?).

CORBA spécifie ainsi l'état d'une requête qui retourne selon le cas « complète », « incomplète » ou, pour indiquer l'indétermination, « peut-être complète » [OMG_96a, Orfali_97a].

2.5.4.2 Interopérabilité des ORB

La mise en place de l'architecture de l'OMG s'est effectuée en plusieurs étapes. Les spécifications de CORBA 1.1 (1991) ne définissaient que l'IDL, les liens avec les langages et les API d'accès à l'ORB. La grande liberté laissée pour l'interprétation de l'implémentation des ORB rendait ainsi leur interopérabilité totalement utopique [Orfali_95].

CORBA 2 (1994) améliore notablement cette interopérabilité en décrivant les services communs et en imposant un protocole réseau. Le GIOP (*General Inter-ORB Protocol*) précise un ensemble de règles de représentation de données (CDR - *Common Data Representation*) qui interfacent les types définis en IDL. Ces données sont ensuite utilisées par le GIOP pour décrire un ensemble de formats de messages (sept exactement) qui gèrent les requêtes, les connexions inter-ORB et localisent les implémentations des objets.

Les messages GIOP peuvent théoriquement utiliser n'importe quel protocole de transport tels que TCP/IP, Novell SPX, SNA, *etc.* mais vue l'importance croissante de l'Internet, l'OMG a décidé d'imposer TCP/IP dans les spécifications de CORBA 2. Cette spécification prend le nom de IIOP (*Internet Inter-Orb Protocol*) [OMG_96].

D'autres ESIOP *(Environment Specific Inter-ORB Protocols*) pourraient être aussi utilisés pour réaliser l'inter-fonctionnement sur des réseaux particuliers (comme IPX ou SNA ou les RPC de DCE) mais, à ce jour, seul HP propose un ESIOP-DCE.

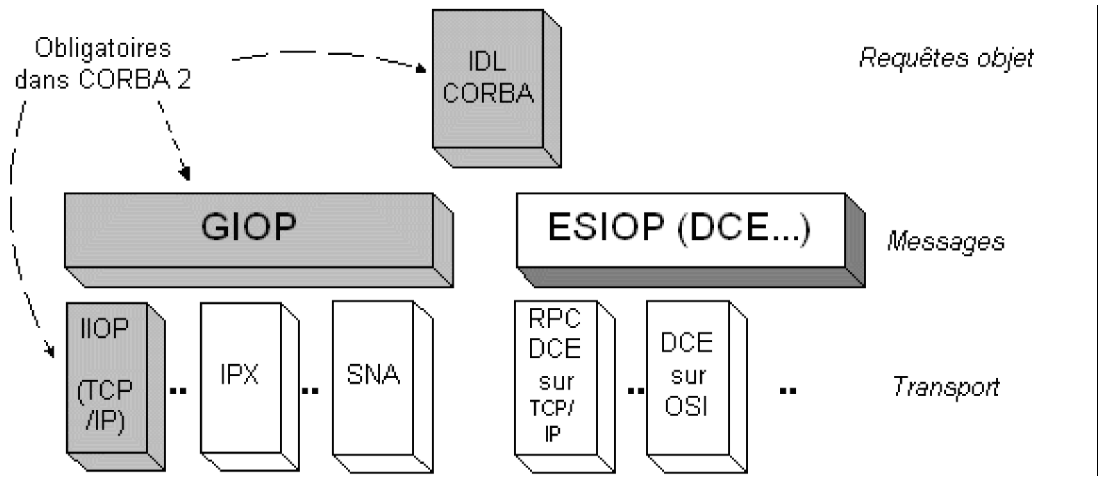


Figure 2. 11 L'architecture Inter-ORB de CORBA 2.

Les objets publient leurs références (identifiant et localisation) par des IOR (*Interoperable Object References*). Un IOR est constitué par un ensemble de profils dont chacun décrit, pour un protocole donné, comment cet objet doit être contacté et le format des requêtes qu'il attend (sachant que IIOP est le seul protocole obligatoire).

C'est grâce à ces IOR, spécifiques au protocole en cours, qu'un ORB a l'assurance de pouvoir déterminer la localisation de tout objet. Le profil IIOP contient, par exemple, l'adresse Internet du serveur qui héberge l'objet ainsi qu'une valeur-clef utilisée par ce serveur pour retrouver l'objet passé en référence.

La représentation externe d'un IOR est arbitraire et peut être transmise sous forme d'une chaîne de caractères comme une URL, un message SMTP, un enregistrement dans une base de données, *etc.* C'est l'application CORBA qui convertit ce format en IOR et l'utilise ensuite pour instancier l'objet [Orfali_97b].

2.5.5 CORBA/IIOP: le futur du Web ?

Au fur et à mesure du succès de l'Internet et de l'évolution des réseaux distants, de nouveaux protocoles ont été rendu nécessaires pour permettre à différents types d'applications de communiquer entre elles. Ces protocoles conçus séparément font qu'aujourd'hui l'Internet supporte simultanément la messagerie SMTP, le transfert de fichiers FTP, les conférences NNTP, l'hypertexte HTTP, *etc.* Il est à craindre que cette démarche aboutisse à créer un nouveau protocole pour tout nouveau service non encore prévu sur l'Internet.

HTTP, ainsi destiné à transmettre des documents *hypermedia* sur l'Internet, ne se soucie pas de leur format (texte, image, *video*, son, *etc.*). Le protocole ne gère pas l'information, il se contente de véhiculer un contenu du client au serveur et réciproquement, à charge aux interlocuteurs de définir une sémantique éventuelle aux informations transmises.

Ce protocole, justifié pour l'affichage d'informations, atteint néanmoins ses limites au fur et à mesure que la complexité des applications qui y sont servies croît (gestion de transactions, optimisation des connexions, *etc.*). Ces applications dynamiques, puisque nécessitant la saisie d'informations par l'utilisateur, posent des problèmes de programmation non triviaux et imposent surtout des développements *ad hoc* pour chaque problème traité. Les mécanismes offerts (CGI) ne sont généralement pas orientés objets et n'offrent pas de mécanismes de sécurité ni de transaction intrinsèques. Les données sont transmises par le client sous forme de chaînes de caractères ou de contenu de message, à charge pour le serveur de les interpréter au cas par cas (sans format normalisé).

Il faut rajouter à ces inconvénients l'absence de méthode standard pour référencer les différents services offerts ce qui, en plus de capacités d'introspection très faibles, rend complexe la possibilité de créer des applications à partir de modules combinables et réutilisables.

Il est alors tentant d'opposer CORBA/IIOP comme une alternative à HTTP pour gérer la complexité des applications afin de proposer une architecture réseau client-serveur « intergalactique » [Orfali_95]. Un des attendus de CORBA n'est-il pas justement de faire communiquer des applications (modulaires) où qu'elles soient et quels que soient leurs concepteurs ? CORBA et IIOP prennent d'office en considération les détails d'implémentation des interfaces ainsi que les problèmes de gestion des paramètres et résultats de requêtes, décrits très précisément et de façon non ambiguë en IDL. Le passage et l'interprétation des paramètres, typés et contrôlés, sont générés automatiquement au lieu d'être traités « manuellement » comme en HTTP. La décomposition d'une application en classes d'objets renforce la modularité et permet que la modélisation d'une ressource (une base de données par exemple) puisse être reprise telle qu'elle pour une autre application. La complexité de l'architecture de CORBA a pour contrepartie de décharger le programmeur de la plupart des détails de communication et d'interface entre objets et de porter les efforts de conceptions sur la modélisation de l'application.

Enfin, plusieurs stratégies plus ou moins douces de migration de HTTP vers CORBA/IIOP peuvent être évoquées [Orfali_97b]:

1. écrire des passerelles bidirectionnelles CGI-CORBA [Edwards_94] qui véhiculent des requêtes, encodées dans les URL (et les formulaires joints) et transmises vers un ORB. Cette méthode, efficace, se heurte au goulet d'étranglement qu'est l'interface CGI d'un serveur HTTP ;
2. mettre en place un jeu de passerelles HTTP/IIOP ou des langages de script implémentés sur le DII et l'IR [Merle_97] ;
3. utiliser HTTP pour télécharger des pages HTML contenant des *applets* Java incluant un *ORBlet* écrit en Java ; ces *applets* initialisant à leur chargement la communication avec des objets distants via IIOP.

   Cette solution permet un usage mixte de HTTP et IIOP, transparent pour l'utilisateur, HTTP étant utilisé pour récupérer les pages de texte, les images, *etc.* et IIOP étant le protocole privilégié pour faire du client-serveur efficace ;
4. généraliser, à l'instar de *Netscape*, l'intégration d'un ORB à tout navigateur HTML/HTTP [Visigenic_97, Dreyfus_97, Gulesian_97] et définir de nouveaux formats d'URL (comme `iiop://`) qui référenceront les objets et les requêtes. Les langages utilisés, JavaScript, VBScript pourront alors s'interfacer directement avec l'ORB et communiquer *via* IIOP.

Rien n'empêcherait par exemple qu'un objet décrit en JavaScript découvre et instancie dynamiquement un objet décrit en VBScript !

CORBA 2 (donc IIOP) a été pensé dès son origine pour faire communiquer entre elles des applications diverses et distantes en garantissant « par construction » la robustesse, la sécurité et l'intégration de chacun de ses éléments. Il y a donc lieu de penser que cette architecture s'imposera à terme (dès que les limitations actuelles de CORBA 2 seront levées) dans le futur du Web.

2.5.6 CORBA : synthèse

L'année 1997, à défaut d'être l'année de l'ORB dans l'entreprise [Orfali_96] plante le décor de la guerre prochaine des composants : *Microsoft* contre le reste du monde.

Sans nous prononcer sur les chances de victoire du « reste du monde », il nous semble important de lister en quelques points l'essentiel des avantages et défauts de CORBA, qui devraient s'avérer critiques à court terme.

Ses principaux atouts :

- le standard est réellement ouvert : il est aujourd'hui possible de trouver des ORB de plusieurs dizaines de fournisseurs qui fonctionnent sur tout type de plate-forme (NT et UNIX notamment);
- l'architecture distribuée est efficace et basée sur un nommage robuste. Celui-ci, à la différence des RPC, permet de localiser un *objet* (et donc l'ensemble de ses propriétés et méthodes) au lieu de pointer sur une *fonction* ;
- l'indépendance à la localisation : deux objets communiquent de la même façon, qu'ils soient sur une même machine ou séparés de milliers de kilomètres, grâce aux différentes fonctions supportées par IIOP (gestion de mandataires) ;
- les objets peuvent être instanciés dynamiquement et peuvent découvrir leurs propriétés en cours d'exécution (introspection - peu disponible cependant) ;
- une excellente intégration de CORBA et de Java [Orfali_97a, Orfali_97b, Vogel_97], due aux projections quasi triviale des structures en IDL, qui devrait bénéficier du succès actuel de Java (sans compter le nombre d'ORB qui sont désormais eux-mêmes écrits en Java, tel VisiBroker).

Les faiblesses résiduelles ne sont pas tant dans des défauts de l'architecture (qui ont été réglés depuis CORBA 2) mais sont plutôt à chercher dans la pauvreté des services et des délais d'implémentation, encore trop importants :

- l'interopérabilité entre ORB de fournisseurs différents commence seulement à être possible grâce à IIOP ;
- la portabilité est toujours discutable : autant celle d'un client CORBA est quasiment assurée depuis CORBA 2 autant celle du serveur est limitée par l'absence de transparence de l'implémentation du BOA.

  Par ailleurs, il faut vraisemblablement attendre que CORBA 3 introduise les SFA (*Server Framework Adapters*) qui gèrent un niveau d'abstraction meilleur que celui du BOA mais spécifique à chaque langage de programmation utilisé (C++, Java, *etc*.) ;
- les services tardent à être offerts : seul le *nommage* est uniformément offert par tous les fournisseurs d'ORB. Les autres services les plus courants sont les *événements* ou le *cycle de vie*.

Il est probable que la généralisation de Java accélérera la disponibilité prochaine de la dizaine de services absents ;

- il n'y a toujours pas de système de messagerie qui permette de gérer dans des files d'attente (client et serveur) des requêtes asynchrones ;
- il n'y a pas de mécanisme d'équilibrage des charges ;
- il n'y a pas de tolérance aux pannes ;
- il n'y a pas de gestion des versions d'objets alors que le nommage en prévoit une explicitement.

Ces listes n'épuisent pas les avantages (et défauts) de CORBA. Elles montrent, d'une part, la fiabilité et la robustesse des ORB en soulignant, d'autre part, un manque certain des services normalisés et de grandes réalisations concrètes.

Il est aujourd'hui encore peu raisonnable de presser l'entreprise d'appuyer son Système d'Information sur une quelconque solution « standardisée » [Gartner_97, Hart_97]. Les architectures de *Microsoft* et de *Sun* sont encore dans un état instable et celle de l'OMG, qui se construit pourtant depuis plus de cinq ans, commence juste à offrir une interopérabilité acceptable et le minimum de services vitaux [Keuffel_97].

Tout laisse croire cependant que la lente maturation de CORBA approche sa masse critique, d'autant plus qu'elle bénéficie de l'énorme impact de Java [Chauvet_97], utilisé indifféremment depuis quelques mois, tant pour développer des ORB que pour appuyer les applications clientes.

Si, en effet, la couche « transport » utilisée par Java peut très bien être DCOM ou un mécanisme propre (tel *Java Remote Method Invocation* développé par *JavaSoft*), l'alliance objective de l'OMG et de *Sun* (contre *Microsoft*) tend à créer une symbiose qui profite indiscutablement aux deux « alliés » [Curtis_97].

3. RICERCAR : de l'Internet à un intranet

She left the web, she left the loom,
She made three paces through the room,
She saw the water-flower bloom,
She saw the helmet and the plume,
She look'd down to Camelot.
Out flew the web and floated wide;
The mirror crack'd from side to side;
"The curse is come upon me," cried
The Lady of Shalott.

A. Tennyson
(*The Lady of Shalott*)

3.1 L'intranet : une « application » d'entreprise normale *63*

3.2 « L'intranet : un chantier social » *66*

3.3 Réaliser un intranet : construction d'un ensemble réparti et cohérent autour d'une architecture réflexive *68*

3.3.1 Panorama des besoins 68

3.3.2 Cohérence des liens hypertextes 69

3.3.2.1 Cohérence intra-serveur 72

3.3.2.2 Cohérence inter-serveurs 73

3.3.3 Exploitation du dictionnaire 75

3.3.3.1 Interface des « pages » 75

3.3.3.2 Interface des « index » 75

3.3.3.3 Charte graphique et cohérence de la navigation 77

3.3.4 Cohérence de l'exploitation d'un intranet 78

3.3.4.1 Authentification des utilisateurs 78

3.3.4.2 Métrologie des accès 79

3.4 Administration d'un intranet *81*

3.1 L'intranet : une « application » d'entreprise normale

L'informatique traditionnelle ne traite qu'une faible partie de l'ensemble des informations qui circulent dans l'entreprise. Il s'agit des informations dites structurées telles que la paie et la comptabilité. Il reste donc un gisement considérable d'informations à stocker (si cela est utile), à traiter et diffuser. Par ailleurs, force est de constater que les technologies restent mal exploitées jusqu'à présent : elles représentent des investissements considérables pour l'entreprise sans impact significatif sur sa croissance. En effet, elles sont utilisées à structure égale ou sans changement significatif, pour réduire les coûts de traitement de l'information et visent à obtenir des gains de productivité. Elles ne sont donc pas mises en œuvre dans une logique de développement ou dans le cadre d'une organisation en rupture avec les méthodes existantes.

La mise en place de solutions du type intranet (réseau d'entreprise s'appuyant sur les mêmes technologies que celles de l'Internet) présente une opportunité technologique pour gérer cette communication autour du savoir-faire de l'entreprise. En permettant aux collaborateurs de mettre en ligne facilement leurs documents (rapports, notes, études), d'informer des projets en cours, de décrire les activités et l'organigramme des services, l'intranet favorise la constitution d'une base de connaissance aussi large que possible.

Cette base de connaissances ne sera efficace que si elle est véritablement gérée. L'information n'a de valeur que si elle est structurée, accessible et utile. C'est sans doute la tâche la plus délicate à mettre en œuvre car elle suppose que l'entreprise soit en mesure de détecter l'information utile et qu'elle ait la volonté et sache encourager ses collaborateurs à participer à la mise en commun de son savoir et de ses connaissances.

L'intranet est, en plus d'un *medium* rêvé pour gérer une communication interne dynamique, un moyen efficace de travailler en groupe, de partager de connaissances, d'accéder à des bases de données locales ou d'entreprise, *etc.* Il permet de définir l'architecture fonctionnelle du réseau interne et un langage et des représentations utilisés par les applications. Il offre *accessoirement* l'opportunité de doter à moindre coût l'ensemble des postes de travail d'un navigateur vu comme un *client universel* [Lefebvre_97, Sauteur_97].

Cependant, la diversité et la divergence des intérêts qui s'expriment sur l'Internet s'épanouissent dans une organisation où chaque serveur représente un groupe d'expression différent. Une transposition de l'organisation de l'Internet à l'entreprise induirait rapidement des problèmes d'incohérence et d'administration d'ensemble.

Un intranet s'adresse *a priori* à une population **très** différente de celle de l'Internet. Il s'agit de milliers de collaborateurs, *peu familiers des outils*

informatiques, disposant d'un poste de travail connecté à un réseau local et qui subissent l'outil de navigation comme un outil Bureautique supplémentaire venant se greffer au traitement de texte, au tableur ou à la messagerie (la dernière version du *Pack Office* de *Microsoft* suffit pour s'en convaincre).

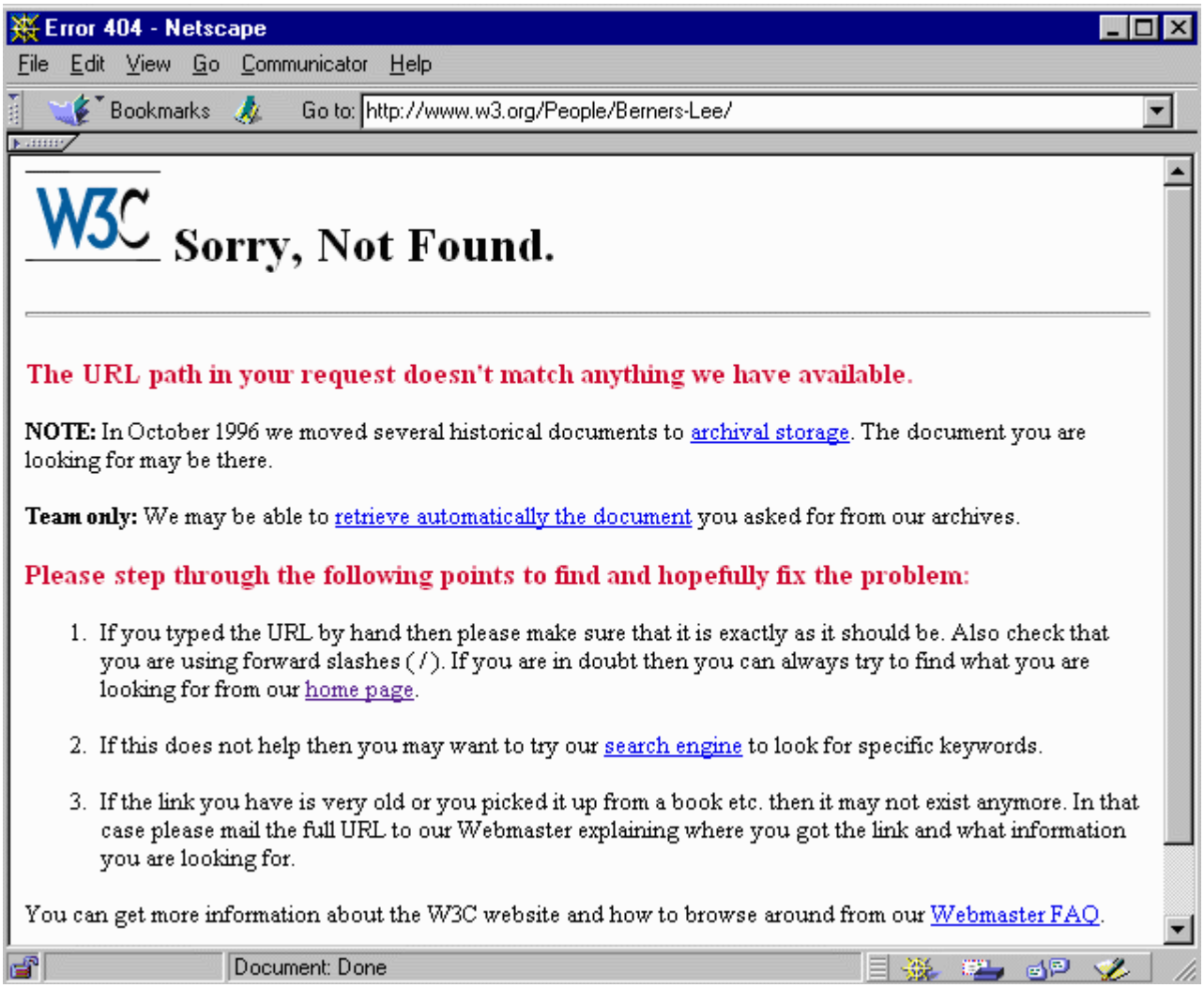


Figure 3. 1 Un message d'erreur HTTP est parfois difficilement exploitable !..

Il est alors délicat d'expliquer à l'utilisateur - non informaticien, auquel la notion d'hypertexte, de liens, *etc.* n'est pas familière - que la disponibilité des données auxquelles il accède n'est pas garantie et que celles-ci peuvent selon le cas, changer d'adresse, qu'il est nécessaire de procéder à des recherches pour y accéder ou qu'il faut régulièrement remettre à jour les références stockées dans son calepin (signets ou *bookmark* selon le navigateur).

La communication transversale entre les différents départements est - naturellement - une communication de proximité ou inspirée par des intérêts

ponctuels. Un des attendus de l'intranet est de contribuer à créer une synergie de communication et d'échanges en offrant un mode de communication « rayonnant » (*broadcast*) qui ne peut que bénéficier à l'ensemble des acteurs. Toute perturbation (freins structurels, surcharge du réseau, mauvaise structuration des données, problèmes d'accès aux documents, lourdeur d'administration, *etc.*) se fera au détriment de l'adhésion et de la fidélisation du plus grand nombre.



Figure 3. 2. … ou n'est pas très parlant pour l'utilisateur.

Si l'architecture technique et le choix des outils informatiques restent à l'évidence de la responsabilité d'un département informatique, l'importance et l'impact d'un intranet se mesureront à la qualité des informations disponibles, à leur structuration, au degré d'implication (donc de leur liberté d'expression) des différents services.

La disparité de la motivation des différents départements est un facteur qui ne doit pas être négligé lors de la mise en place d'un quelconque *medium* de communication. Autant des services « communicants » tels l'Informatique ou la Communication se sentent-ils immédiatement impliqués dans ces démarches, autant des services opérationnels dont la problématique est d'assurer le métier premier de l'entreprise, sont-ils *a priori* hostiles à toute dispersion de leurs efforts vers des tâches considérées comme superflues. Ces difficultés soulignent essentiellement la nécessité de confier la maîtrise d'ouvrage d'un intranet à des fonctionnels (un département de la communication) plus à même d'assurer l'animation constante de ces services et de les diversifier à l'image de l'entreprise.

Il est vraisemblable que la réussite de l'intranet ne se fera que dans la mesure où l'utilisateur considérera qu'il n'a **qu'une seule « application »** qui lui est proposée, le navigateur, et qu'il s'y référera naturellement pour accéder à un ensemble croissant des données de l'entreprise. Cette simplicité d'usage nécessite cependant que l'intranet soit considéré avec la même rigueur que toute autre application d'entreprise.

L'appréhension de l'intranet comme un projet global d'entreprise dans ses facettes techniques et fonctionnelles devrait par ailleurs offrir des garanties de cohérence à une Direction Générale, par définition méfiante à l'égard des nouvelles technologies - *a fortiori* de l'intranet perçu comme une déclinaison dans l'entreprise de l'organisation chaotique de l'Internet.

3.2 « L'intranet : un chantier social »

Aux multiples questions techniques et d'architecture posées par la réflexion autour d'un intranet s'ajoutent des problèmes sociaux, organisationnels et humains, absents de l'Internet [Dufour_96, Augendre_97].

Il est important d'en recenser quelques-uns pouvant séparément freiner le déploiement effectif d'un intranet et qui, combinés, s'avèrent quasi insurmontables :

1. faible technicité des cadres dirigeants et mauvaise perception des concepts en jeu (notamment le client-serveur) ;
2. équipes d'étude et de développement réduites à la portion congrue après plusieurs années de politique de sous-traitance des projets informatiques ;
3. consultation systématique d'intervenants externes qui, d'une part, ne sont pas mandatés pour assurer une cohérence entre leurs différents sujets d'intervention et, d'autre part, n'ont souvent pas d'expérience de solutions intranet et tentent de plaquer telles quelles leurs études basées sur l'Internet ;
4. inadéquation de la réactivité des circuits décisionnels de l'entreprise par rapport à l'évolution des produits ;
5. contraste entre tenants d'un intranet à l'image de l'Internet (libertés de publication et d'accès totales) et tenants d'un intranet rigidifié avec circulation des flux déterminée et contrôlée. Une proposition qui tend à laisser les utilisateurs s'exprimer avec souplesse dans un cadre institutionnel a toutes les chances de rallier contre elle les partisans de ces deux camps ;
6. répugnance de l'entreprise à adopter un modèle de développement en *spirale* [Boehm_88] et, par conséquent, extrême méfiance vis-à-vis de la rapidité des réalisations (quelques heures pour un premier prototype) ;
7. sous-estimation de l'impact organisationnel de l'intranet [Alin_97, Adams_97, Berger_97, Bossard_97] qui remet en cause des circuits séculaires de communication d'entreprise (les niveaux hiérarchiques intermédiaires notamment « qui jouent le rôle de tampon et ralentissent la circulation de l'information » [Bitouzet_97, p.30]) et structure de manière forte les systèmes d'information de l'entreprise.

« L'intranet encourage la fin de l'organisation pyramidale de l'entreprise. Il s'adapte parfaitement au management plus participatif des structures qui fonctionnent davantage par groupes de projets. C'est la fin de la diffusion automatique des informations du sommet vers la base. Il favorise l'émergence des groupes de travail virtuels. » [Théry_96].

Il semble inéluctable que le déploiement de l'intranet se fera à brève échéance. Aussi insistons-nous ici sur l'opportunité apportée au Système d'Information de l'entreprise de mettre rapidement au point une palette technique qui assure aux

divers interlocuteurs (utilisateurs finals, auteurs à la source de l'information, administrateurs, informaticiens, responsables et Direction Générale) des mécanismes **simples** et **robustes** de consultation et de publication d'informations **cohérentes** et **fiables**. Il s'agit notamment d'éviter une prolifération anarchique de serveurs, emballée par la médiatisation à outrance actuelle de l'Internet qui assure le tout-venant de l'extrême simplicité de mise en œuvre, de publication et d'administration d'ensemble.

Un parallèle avec l'I.A. des années quatre-vingts, les énormes espoirs qu'elle a suscités, le battage médiatique, suivis quelques mois plus tard par un désintéressement quasi général en dépit des réalisations effectuées peut sembler exagéré devant la vague actuelle de l'intranet. Il devrait néanmoins inciter les partisans de l'Internet et de l'intranet à ne pas répéter les erreurs du passé.

Il ne fait pas de doute que le principe du client-serveur abordé par le biais d'un couple navigateur/serveur HTTP devrait consacrer cette technologie mais aussi induire de nouvelles formes de communication dans l'entreprise. Il n'est cependant pas à écarter que des problèmes organisationnels, structurels et humains concourent à une dispersion et une dilution des informations, à des freins entraînant leur manque de disponibilité ou de fraîcheur et provoquent à terme un manque de confiance et un rejet partiel des utilisateurs finals.

Les mécanismes dont nous entamons la description dans ce chapitre ne trouvent cependant leur justification que dans un ensemble de serveurs répartis. Il est relativement aisé de gérer une cohérence locale et une qualité de service locales à un serveur (de taille moyenne - quelques centaines de documents) sans pour autant devoir s'embarrasser d'une quelconque architecture (*FrontPage* de *Microsoft* propose une solution séduisante mais cantonnée à un seul serveur).

Il est cependant illusoire de penser assurer des services équivalents dans un réseau englobant des dizaines de serveurs et de milliers de postes de travail où une cohérence locale n'implique pas de cohérence globale. Seule une mise en commun d'informations, fédérées dans des « dictionnaires » permettra de gérer la création, la modification ou le déplacement d'un objet d'un quelconque serveur à un autre en toute transparente pour l'utilisateur final [Robertson_97, Séraphin_97a]. Cette fédération assurera une cohérence et une homogénéisation d'ensemble et offrira des services à valeur ajoutée.

3.3 Réaliser un intranet : construction d'un ensemble réparti et cohérent autour d'une architecture réflexive

3.3.1 Panorama des besoins

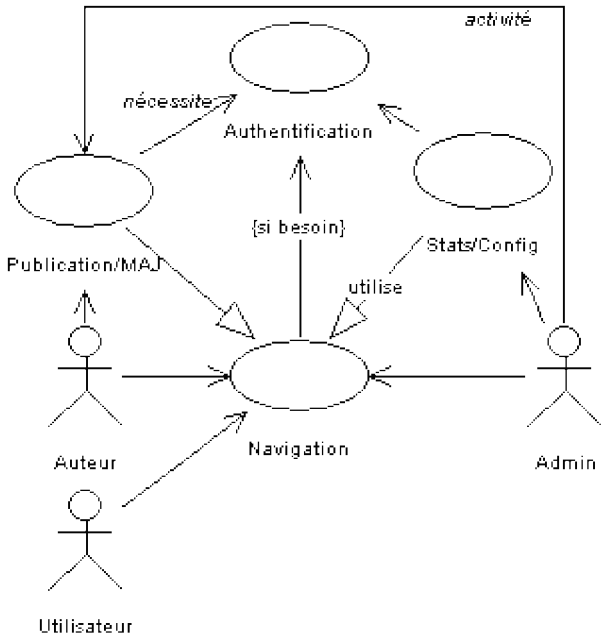


Figure 3. 3. Diagramme des cas d'utilisation d'un intranet

« En général, le manque de procédure génère des serveurs web monstres, de véritables "usines à gaz", totalement ingérables. On rencontre des sites de 10 000 pages HTML sans conception ni architecture précise ! » [Bitouzet_97, p.132].

En l'absence d'outils de gestion des documents hébergés sur un serveur, il est de plus en plus ardu de déterminer où doivent être placées de nouvelles pages, de quels thèmes elles relèvent, de modifier les différentes listes de liens pour en tenir compte, *etc.*

Cette complexité devient rapidement combinatoire dans le cadre d'un ensemble de serveurs, où une multitude de documents peuvent être créés rapidement, où les menus peuvent se référer à des éléments d'origines diverses, les documents migrer d'un serveur à un autre.

Ces différentes manipulations, inévitables dans une entreprise riche de milliers de collaborateurs, sont à l'origine de nombreuses incohérences et freinent l'expansion de l'intranet tant leur répercussion sur les différents intervenants est importante.

L'*utilisateur* est concerné au premier chef par la fraîcheur des liens et la complexité de la navigation :

- fraîcheur des liens : des erreurs d'accès peuvent survenir dès le moment où les liens proposés dans les pages consultées ne sont pas toujours à jour ou ne reflètent pas les modifications intervenues ;
- complexité de la navigation : il n'y a pas de vision claire de « l'arborescence » du serveur.

L'*administrateur* est concerné par la gestion des utilisateurs et leurs droits, la modification des liens (internes ou externes) et la publication de nouveaux liens :

- gestion des utilisateurs : il lui faut déterminer quels utilisateurs sont autorisés à publier, par quelles procédures, quels principes de mise à jour des arborescences, *etc.*
- modification des liens internes : toute modification d'une page (déplacement, changement de nom ou d'extension) a des répercussions sur l'ensemble des pages qui s'y référençaient et qui doivent être modifiées une par une ;
- modification des liens externes : il n'y a aucun moyen de prévenir les autres administrateurs et utilisateurs qu'une page donnée a été modifiée ;
- publication des liens internes ou externes : lors de l'ajout d'une page, il est complexe de déterminer l'ensemble des thèmes concernés et de modifier les liens des menus correspondants.

L'*auteur* est concerné par la fraîcheur des liens et la mise à jour de nouvelles versions de ses documents :

- fraîcheur des liens : l'auteur, voulant se référer à des pages existant par ailleurs, n'a aucune garantie que ses références seront cohérentes lors de mises à jour des pages auxquelles il se réfère ;
- mise à jour de nouvelles versions (comment, où, conversion en HTML, enregistrement dans d'autres menus).

3.3.2 Cohérence des liens hypertextes

La publication d'un document sur un Web traditionnel équivaut à la création d'une nouvelle URL « physique », correspondant au fichier HTML créé. Cette URL est ensuite référencée « manuellement » dans un fichier particulier du répertoire, qui fait office d'index (généralement un fichier nommé *index.html*).

La création de ce document a deux conséquences naturelles :

1. être consulté par un nombre indéterminé de personnes en fonction de son intérêt, s'il est répertorié dans des listes thématiques, indexé par un moteur de recherche, *etc.* Ces personnes s'approprient l'information en mémorisant son URL dans leur calepin (*bookmark*) pour y accéder ultérieurement ;
2. être cité par un lien hypertexte dans un ensemble indéterminé d'autres documents tant internes (physiquement sur le même serveur) qu'externes (localisés sur d'autres serveurs).

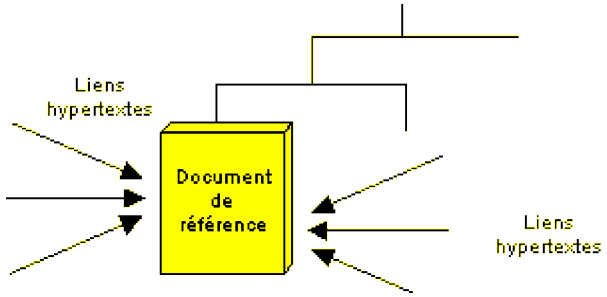


Figure 3. 4. Un document publié est référencé par un ensemble indéterminé d'objets.

Ainsi que nous l'avons indiqué en § 1.2.1.4, p.24, le schéma général d'une URL du protocole HTTP est de la forme :

http://serveur:port/chemin?param1=val1&param2=val2...

que nous réduirons, dans cette section, à :

http://serveur/chemin

Ce schéma montre bien que toute modification d'un des deux termes variables de l'URL rend caduques l'ensemble des références établies ;

1. ***serveur*** : si le document change de serveur ou si celui-ci change de nom ou même éventuellement d'adresse IP (ou de port TCP) ;
2. ***chemin*** : si le document est déplacé dans l'arborescence du serveur, si un des éléments de cette arborescence est renommé ou supprimé, si le nom ou l'extension du fichier sont modifiés ou s'il est supprimé.

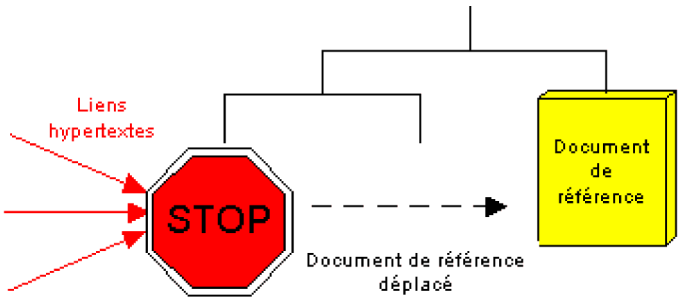


Figure 3. 5. Toute modification de structure (ici un déplacement) provoque une erreur lors de l'accès aux documents.

Il est fréquent de constater, sur un serveur HTTP (qu'il soit de type Internet ou intranet), que des liens ne sont plus à jour ou que l'interrogation de moteurs de recherche pointe sur des documents qui n'existent plus ou ont été déplacées. Ces dysfonctionnements, corrélés à la probabilité élevée qu'un, au moins, des éléments du nommage soit modifié à brève échéance, soulignent sa fragilité et expliquent la brève durée de vie des liens hypertextes.

La complexité croissante de la gestion d'une arborescence physique de fichiers de plus en plus touffue (combinée à la difficulté de maintenir les différents fichiers d'index) justifie par ailleurs un manque éventuel de fraîcheur des informations proposées.

Peu marqué aux débuts du Web, ce manque de robustesse du nommage (tout au moins de son interprétation) et ses effets secondaires, préoccupent une frange croissante de la communauté de l'Internet [12]. Ainsi que nous l'avons déjà évoqué en § 1.2.1, celle-ci a émis en quelques années une série de recommandations [13] où elle précise, dans un premier temps, que « la similarité avec une arborescence physique telle celle d'UNIX ou d'un autre système d'exploitation doit [14] être considérée comme une coïncidence et **ne doit pas être exploitée** pour interpréter des URI comme des noms de fichiers » [Berners-Lee_94a, p.7].

Dans le même temps, une URL est définie comme « une forme d'URI qui exprime une adresse définie par un **algorithme d'accès** calculé par des protocoles réseau » [Sollins_94] [Berners-Lee_94b, p.1].

Elle insiste ensuite en affirmant « qu'un nom de ressource suppose la définition d'un **pointeur stable** qui continue à référencer cette ressource longtemps après qu'elle a changé de localisation ou même cessé d'exister » [Kunze_95, p.1].

La presse et la littérature spécialisées [Bitouzet_97, Hower_97] se font aussi l'écho des problèmes induits par l'accroissement et la dynamique des serveurs

[12] Le problème de la cohérence des liens est un des thèmes majeurs de la conférence annuelle du W3C en 1998.

[13] rfc1630, rfc1736, rfc1737, rfc1738 et rfc1808.

[14] « *Should* » dans la rfc1630 : incitatif mais non comminatoire.

HTTP : « Dès que le volume des données augmente - et surtout le nombre de fichiers - la gestion et la mise à jour deviennent délicates » [Dhénin_97].

3.3.2.1 Cohérence intra-serveur

Il est impossible de déterminer l'origine des liens qui référencent une URL donnée. Ceci élimine toute possibilité de prévenir d'éventuels correspondants au fur et à mesure des modifications. Une seule solution garantit donc qu'une URL permettra toujours d'accéder au *document* correspondant, s'il existe : figer l'URL de l'*objet* [15].

Cette cohérence peut se traduire concrètement de deux manières :

1. ne jamais modifier un lien publié (ce qui entraîne de grandes lourdeurs d'administration et n'est pas vraisemblable à moyen terme) ;
2. modifier l'*interprétation* d'une URL en ne la considérant plus comme l'adresse physique d'un fichier HTML mais comme un *identifiant* de référentiel global d'un *objet* [Abiteboul_89, OSF_93a, OMG_96a].

Cet identifiant peut, dans un premier temps, servir à en calculer la localisation exacte (un pointeur vers le fichier *ad hoc*).

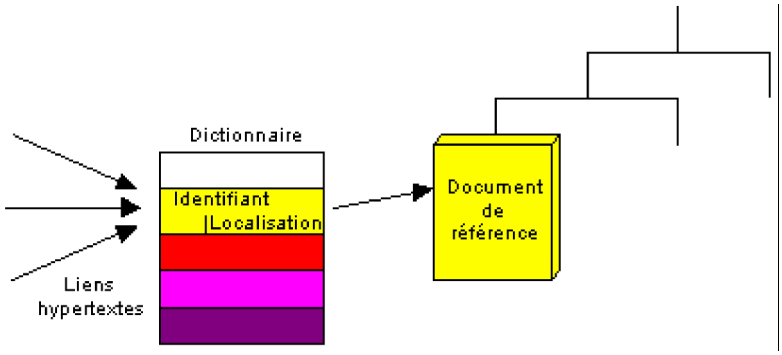


Figure 3. 6 La mise en place d'un dictionnaire fait que les liens ne pointent plus sur le document (variable) en référence mais vers l'entrée (fixe) du dictionnaire

L'enregistrement dans un dictionnaire d'un ensemble de méta-données décrivant les documents peut, dans un deuxième temps, associer à cet identifiant une méthode de construction qui générera dynamiquement le code HTML correspondant.

[15] Le changement de terminologie (de *document* à *objet*) indique ici le changement de paradigme en cours.

Les modifications des documents (déplacement, changement de nom, d'extension, de version, *etc.*) se font alors par le biais de formulaires de publication et de mise à jour et sont automatiquement enregistrées dans le dictionnaire.

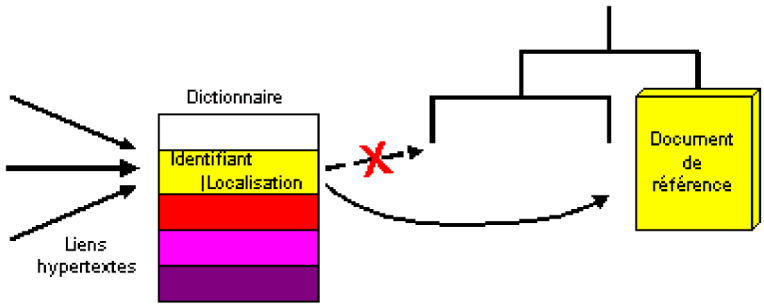


Figure 3. 7. Toute modification du document modifie son entrée dans le dictionnaire de façon transparente pour l'utilisateur final.

La cohérence du dictionnaire, hybride des Référentiels d'Implémentations et d'Interfaces, garantit la pérennité et la transparence des requêtes des utilisateurs.

3.3.2.2 Cohérence inter-serveurs

Cette première approximation de la représentation des objets dans un dictionnaire n'assure toutefois qu'une cohérence locale à un serveur donné (seule la partie `chemin` de l'URL a été traitée jusqu'ici).

Une fédération des référentiels d'interfaces de l'ensemble des serveurs, au travers d'une *méta*-base (un index des dictionnaires [Gardarin_90, Zhang_94, Orfali_95]), étend cette cohérence à l'intranet entier, vu alors comme un seul serveur virtuel.

L'identification et la création d'un noyau minimal de méta-données, nécessaires pour assurer l'autonomie de chacun des sites, permet de répliquer et de synchroniser ces données sur chacun des serveurs.

La redondance de ces informations présente deux avantages notables :

1. réduire le trafic induit sur le réseau pour localiser les objets. Les requêtes relatives à des objets locaux sont traitées localement ;
2. pallier les défaillances ponctuelles des serveurs (pannes des serveurs ou perturbation du réseau) et éviter des goulets d'étranglement. Une requête locale peut ainsi être traitée même si l'ensemble des autres serveurs est inaccessible.

L'hystérésis d'un tel ensemble peut néanmoins générer des incohérences globales induites par la création d'objets ayant le même identifiant, sur deux serveurs différents. Cette duplication ayant lieu grâce à la « simultanéité » des transactions locales ou alors dans le cadre d'un fonctionnement en « mode dégradé » du système à la suite de différentes perturbations.

L'affectation d'un identifiant au serveur apporte une solution satisfaisante à ce problème. Cet identifiant est préfixé à celui des référentiels d'objets, les rendant globalement uniques.

L'usage du préfixe pourra être réservé aux cas où il est impossible de réunir les conditions de la transaction répartie qui correspond à la création d'un nouvel objet. Un système ne gérant pas les transactions réparties devra systématiquement concaténer le préfixe du serveur à celui des objets pour garantir la cohérence d'ensemble.

Cette unicité des interfaces, garantie par-delà des serveurs, permet ainsi à un utilisateur de requérir **n'importe quel objet** de **n'importe quel serveur** du réseau. Celui-ci extrait les méta-données correspondantes de son dictionnaire qui lui indiquent, le cas échéant, l'identifiant du serveur distant auquel il retransmet la requête de l'utilisateur. Celui-ci dispose alors localement de l'ensemble des informations nécessaires à la construction du code HTML correspondant à l'objet.

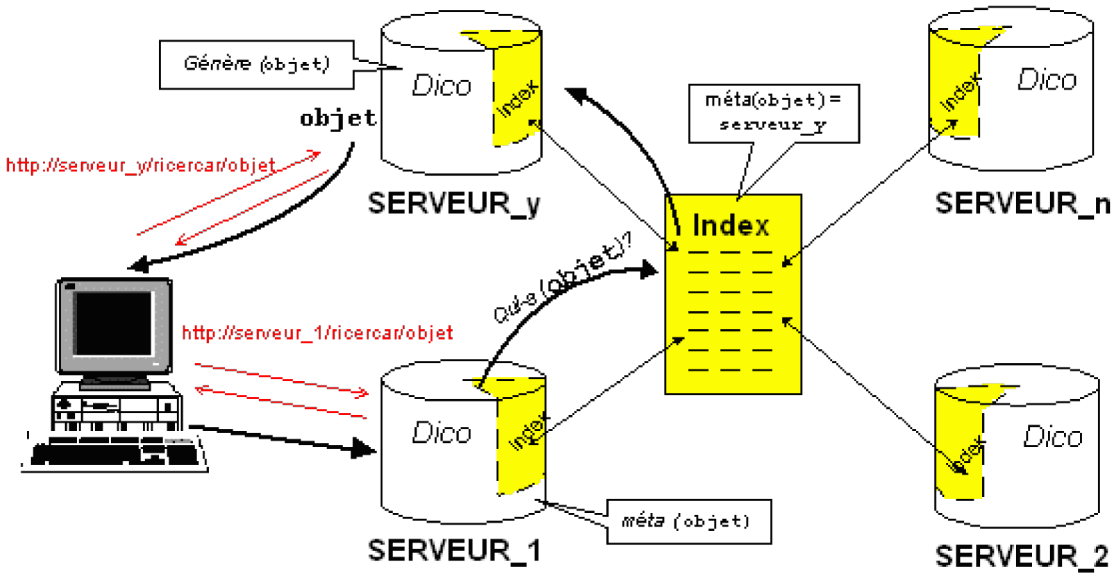


Figure 3. 8. La fédération des dictionnaires assure la cohérence globale de la navigation. La requête d'un objet est automatiquement acheminée vers le serveur ad hoc.

À l'issue de cette étape, la cohérence des liens hypertextes est réalisée par l'indépendance totale entre les objets et les URL qui les référencent. L'indépendance par rapport au `chemin` (la localisation physique) d'un fichier est réalisée grâce à la mise en place d'identifiants d'objets. L'indépendance par rapport au `serveur` est, quant à elle, obtenue par la création d'un index des dictionnaires.

Cette double indépendance réduit le nombre de degrés de liberté du nommage au seul identifiant des objets. Il devient alors possible, dans la limite de l'intranet ainsi construit, de référencer ces objets à partir d'URL relatives [Fielding 95],

<RURL:identifiant>. La requête, adressée à un quelconque serveur, sera traitée localement ou alors automatiquement transmise vers le serveur *ad hoc.*

3.3.3 Exploitation du dictionnaire

3.3.3.1 Interface des « pages »

La mise en place d'un ensemble de dictionnaires d'interfaces n'affecte pas la conception des pages d'information (les documents HTML notamment) qui continuent à employer les outils traditionnels d'édition.

L'auteur, désirant se référer à des objets locaux ou distants, les associe par des liens hypertextes, représentés sous forme d'URL absolues ou relatives, qui référencent les identifiants de ces objets de façon formellement identique à des URL standards.

Ce n'est qu'à son activation qu'un lien engendre une interrogation de ce référentiel de méta-données, qui récupère les informations relatives au document requis, identifie dynamiquement sa classe et invoque alors la méthode *ad hoc* pour en construire une représentation finale en HTML, renvoyée à l'utilisateur.

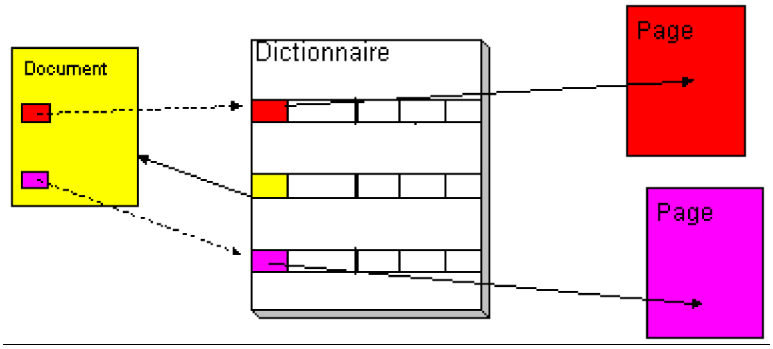


Figure 3. 9. L'auteur d'un document insère des liens vers des IDentifiants de Référentiel globaux, représentés par des URL.

3.3.3.2 Interface des « index »

La modélisation d'un intranet consiste à identifier les propriétés et relations jugées les plus importantes ou les plus significatives pour une entreprise.

Elle présente de nombreux avantages dans la mesure où elle isole d'une part les régularités et les réifie en classes et sous-classes de documents. Elle implémente, d'autre part, une injection de la structure arborescente normale d'un serveur HTTP (racine, index, feuilles) sur ce modèle.

L'identification d'une classe particulière d'objets, les « menus », associe à chacune de ses instances, l'ensemble des documents qui lui sont liés par une relation de type « *a-pour-père* ».

Cette représentation d'un « menu » remplace alors avantageusement le fichier *index.html* fonctionnellement équivalent. L'activation du « menu » extrait du dictionnaire la liste des objets dont il est le « père » en plus de ses méta-informations propres. Elle invoque aussi une méthode *ad hoc* de la classe qui formate (grâce aux méta-données de la classe) et génère à la volée la représentation en HTML de ce menu.

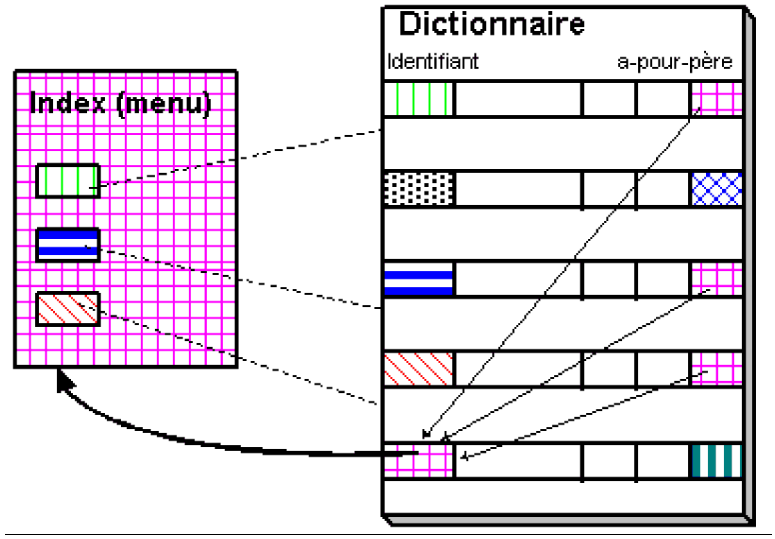


Figure 3. 10 Les menus (l'arborescence virtuelle du serveur) peuvent être générés dynamiquement à partir des méta-données du dictionnaire.

Cette automatisation affranchit les administrateurs du serveur Web de la contrainte de gérer manuellement les modifications dans un fichier *index.html.* Elle garantit aussi que toute mise à jour du serveur, répercutée dans le dictionnaire, est automatiquement rendue disponible aux utilisateurs finals.

Par ailleurs, ces dictionnaires modélisent un ensemble de relations entre objets. Il est alors aisé de concevoir des méthodes génériques qui extraient de dictionnaires l'ensemble des objets qui vérifient une relation, passée en paramètre.

C'est ainsi que la relation « *a-pour-père* » représentée ici, peut être remplacée par « *contient-l'expression-régulière-xxxx* », « *a-été-modifié-récemment* », « *a-été-publié-par* », *etc*. pour générer dynamiquement des menus qui correspondent, respectivement, à l'interrogation d'un moteur de recherche, affichent les dernières mises à jour ou listent les documents d'un auteur donné.

La représentation relationnelle de l'arborescence facilite aussi l'implémentation de méthodes de parcours de graphe (qui peuvent être valués). Ces méthodes, paramétrées, construisent dynamiquement une cartographie d'un serveur ou d'un ensemble de serveurs (qui tienne éventuellement compte de ces valuations).

3.3.3.3 Charte graphique et cohérence de la navigation

Une grande entreprise, soucieuse de formaliser et d'homogénéiser la diffusion des informations à ses utilisateurs, « suggère » généralement à ses collaborateurs l'usage de modèles de documents et d'une charte graphique. Il est cependant courant, au regard des contraintes imposées par des feuilles de style ou des règles complexes, que ces modèles restent l'apanage de quelques puristes ou spécialistes des outils de publication et soient ignorés de la grande majorité de l'entreprise (il suffit de compter le nombre de secrétaires, professionnelles du traitement de texte, qui ont un usage limité des feuilles de style).

La généralisation d'un intranet renforce ce besoin de banalisation de la présentation des données. L'utilisateur final va, en effet, découvrir ou avoir accès à un ensemble croissant de documents et d'applications, au fur et à mesure de leur publication ou de ses besoins. Le succès inévitable de l'intranet entraînera la multiplication d'applications transversales mais aussi de « petites » applications moins stratégiques mais beaucoup plus nombreuses.

Il est alors évident qu'en l'absence de règles de présentation et de navigation, les documents et applications respecteront - au mieux - une logique orientée par département et par projet.

Ces logiques différentes implémenteront *de facto* des règles visuelles et des repères de navigation différents, au prix d'une dégradation de l'ergonomie générale et d'une désorientation de l'utilisateur.

Il est important de souligner que la combinatoire entre le nombre de collaborateurs et le nombre d'applications potentielles sur un intranet rend illusoire toute velléité de former les utilisateurs, préalablement à la publication des nouvelles applications. Seule une initiation succincte à l'usage du navigateur lui-même peut être envisagée (par le biais de documents d'aide publiés sur l'intranet ?).

Il est donc essentiel de renforcer une charte graphique de l'intranet, qui comporterait des règles de navigation intuitives et uniformes : positionnement et sémantique stables des éléments graphiques tels les retours aux sommaires, les logos d'application, les coordonnées des auteurs et gestionnaires des données, *etc.*

Le respect de ces règles de présentation communes (éventuellement déclinées dans l'entreprise dans le respect d'une même logique) permettra alors de capitaliser les efforts de formation des utilisateurs qui, au fur et à mesure de leur familiarisation avec les applications déjà en service, sauront rapidement s'adapter à la logique - identique - des suivantes.

La modélisation de l'intranet par une méta-base implémente, par « effet de bord », une charte graphique, utilisable par les « auteurs » avec un minimum de contraintes.

La spécialisation des objets en classes et sous-classes de documents abstrait, en effet, leurs propriétés communes, référencées dans des variables de classe.

L'auteur dispose d'une feuille de style minimale qui assure la conversion en HTML de documents issus d'un traitement de texte (niveaux de titres, listes à puces ou numérotées).

La réification de ces documents, par le biais d'automates ou de formulaires de publication, se ramène alors à instancier une de ces classes.

De même, les scripts correspondant aux applications référencent une classe *ad hoc* qui décrit le schéma général de navigation.

L'invocation d'un objet extrait du dictionnaire la description de sa classe. Une méthode de génération associée est ensuite utilisée pour composer dynamiquement le document HTML final. Sa présentation (fond, logos, en-tête, texte signataire, dates de création/modification, style [Lie_96], éléments de navigation, *etc.*) réalise l'implémentation de la charte de l'entreprise en réduisant les contraintes issues de l'apprentissage d'outils d'édition rébarbatifs (*cf.* exemples en Annexe 6.6, p.209).

3.3.4 Cohérence de l'exploitation d'un intranet

3.3.4.1 Authentification des utilisateurs

Les différentes versions du protocole HTTP [Berners-Lee_94a, Franks_97] ne couvrent que marginalement les considérations de sécurité et d'authentification des utilisateurs. Seul un algorithme mono-serveur général et de fiabilité limitée [16] y est décrit. Son implémentation sur un intranet implique de gérer par des mécanismes externes la réplication des comptes d'utilisateurs entre les différents serveurs.

Cette tâche d'administration, non triviale, est la source principale de l'incohérence des identifiants ou des mots de passe des utilisateurs. Ceux-ci doivent généralement se souvenir d'une pléthore d'identifications différentes qui leur donnent accès à autant d'applications de l'entreprise avec des droits variés.

Cette complexité pour l'utilisateur est généralement résolue par l'adoption systématique d'un même mot de passe, simple à retenir, ou encore par

[16] *Basic Authentication Scheme :* les identifications des utilisateurs (mots de passe compris) sont écrits en clair dans le fichier de *logs* (d'où des problèmes de sécurité en cas d'utilisation de comptes NT natifs).

l'établissement de « pense-bête » listant la totalité des couples « identifiant - mot de passe », fragilisant ainsi le Système d'Information de l'entreprise.

La tâche d'administration n'est pas plus simple puisqu'il faut définir, pour chaque nouvelle application, la liste des personnes disposant de droits particuliers, générer les identifiants et les mots de passe, *etc*. Il est fréquent que des bases de comptes d'utilisateurs reflètent, d'une part, la présence de collaborateurs ayant changé de fonction ou ayant quitté l'entreprise et, d'autre part, n'intègrent pas des utilisateurs pourtant présents depuis de longues semaines.

La réification de l'utilisateur (sa description dans le modèle de l'intranet) l'intègre dans un cadre global d'authentification (*SSO*, [Zhang_92, Seret_94, OMG_96b]). Un *IDentifiant authentifié* unique (créé par un serveur disposant de méthodes *ad hoc*) lui est associé, qui est alors propagé automatiquement par le *contexte* d'invocation des objets.

Chaque serveur dispose en outre de *Listes de Contrôle d'Accès* (*ACL*) qui maintiennent les droits de création, de modification ou d'invocation des différents objets, par utilisateur ou par groupe d'utilisateurs.

RICERCAR implémente une méthode d'authentification de ses utilisateurs qui enregistre sous la forme de *cookies* [Kristol_97], sur le poste de travail, leur *IDentifiant authentifié* (et encrypté) pour la durée de leur session. Cette méthode assez simple d'authentification peut néanmoins être surchargée, si nécessaire, par une authentification à tierce partie sûre telle *Kerberos* [Steiner_88, Kohl_93].

3.3.4.2 Métrologie des accès

La mesure du trafic sur un intranet doit permettre de répondre à différents types de questions :

- nombre d'utilisateurs différents ;
- nombre d'accès par document ;
- fréquence d'accès ;
- assiduité des utilisateurs ;
- volume de données transférées ;
- répartition horaire des accès ;
- charge d'un serveur, *etc.*

Ces statistiques permettent aux équipes d'exploitation de s'assurer que le serveur est correctement dimensionné pour supporter le nombre et l'intensité des accès. Elles offrent aussi, aux différents auteurs, une mesure de l'intérêt porté aux informations et applications qu'il ont publiées et les aident à déterminer plus finement une fréquence optimale de mise à jour.

En fonction du profil de l'entreprise, des mesures de fréquentation pourraient aussi calculer des statistiques nominatives ou de groupes (d'utilisateurs ou de documents regroupés par type) dans un but de facturation par l'usage, *etc.*

Il est donc important que la métrologie mise en place soit modulaire pour s'adapter à des indicateurs divers, généralement non prévus à l'origine.

Le fichier d'événements (fichier *log*) écrit au fil de l'eau par un serveur HTTP, regroupe de façon atomique l'ensemble des éléments consultés du serveur. L'atomicité de ces informations, stockées « à plat » dans un fichier texte, rend leur exploitation et leur interprétation très malaisées.

En effet, la granularité des enregistrements reflète la complexité des documents consultés (les images, boutons, lignes et autres éléments graphiques de présentation créent chacun un enregistrement dans les *log*) alors qu'il serait beaucoup plus informatif de disposer des données relatives aux différents *objets* consultés. L'interpolation des *hits* annoncés avec un nombre de pages effectivement lues est ainsi très aléatoire.

De plus, un navigateur dispose d'un cache qui mémorise les pages consultées afin d'optimiser les accès au réseau (des serveurs *proxy* peuvent aussi être intercalés, qui mémorisent à leur tour les objets requis). La gestion de ces caches et de leur cohérence peut, à son tour, poser des problèmes [Makpangou_97]. Les fichiers de *log* ne reflètent alors plus qu'une partie des accès réels (des accès successifs d'un utilisateur à une même page n'engendrent qu'un seul enregistrement dans les *log*), rendant leur interprétation encore plus hasardeuse.

Il n'est, enfin, guère envisageable de disposer de mesures relatives à un ensemble de serveurs, de suivre un utilisateur et de comptabiliser ses accès, *etc.*

La mise en place d'un dictionnaire permet de lier aux objets un ensemble de « macro-événements » (activés par des démons) desquels sont extraites des statistiques relatives aux objets consultés (qui, quand, combien de fois, à partir d'où, sous quel nom, *etc.*). Ces données sont ensuite agrégées pour offrir une métrologie ou des éléments de facturation « à la carte ».

La construction dynamique, du code HTML correspondant aux objets, à partir de méthodes, garantit que l'ensemble des méta-informations relatives notamment aux dates de création, de modification, de validité, *etc*. vont bien être générées.

Ces informations règlent, d'une part, le problème de la cohérence des caches intermédiaires qui peuvent traiter correctement les documents qu'ils font transiter. Elles offrent, d'autre part, aux administrateurs la possibilité de décider de l'opportunité de campagnes de mesures précises.

Il suffit alors de modifier les méta-données des classes pour forcer l'expiration des caches (du navigateur et des éventuels serveurs *proxy*) et disposer de chiffres de consultation fiables (au prix, il est vrai, d'une dégradation provisoire des performances d'ensemble [17]).

[17] Par un effet du principe d'Heisenberg.

3.4 Administration d'un intranet

La question de l'administration d'un intranet est d'une telle complexité que nous avons longtemps songé à supprimer ce paragraphe en espérant passer inaperçu.

Il apparaît en effet que le terme même d'administration (dans le cadre d'un intranet) recouvre des réalités très différentes selon que l'interlocuteur-informaticien soit spécialiste des réseaux, gestionnaire de bases de données, documentaliste, *Webmaster*, *etc*. L'utilisateur « normal » de l'entreprise en a, lui aussi, une définition particulière qui diffère par ailleurs de celle du « management » de l'entreprise !

Ce sujet, qui reprend le problème général de l'administration d'applications distribuées, mériterait donc une étude détaillée qui ne relève pas de notre compétence (il nécessiterait notamment de pouvoir [ré]organiser les circuits d'exploitation et de maintenance de l'entreprise).

Aussi, nous sommes-nous contenté de compiler dans une liste les questions que se posent les différents acteurs et qui relèvent de tâches d'administration ou d'exploitation d'un intranet :

- Qui contacter en cas de problème d'accès à un page ?
- Qui contacter si une donnée est fausse ?
- Comment distinguer une information fiable d'une autre qui l'est moins ?
- Qui est responsable de l'information publiée ?
- Comment publier de nouveaux documents ?
- Comment savoir qu'une information a été modifiée ?
- Comment trouver les nouvelles informations ou applications ?
- Comment limiter l'accès en lecture de façon souple ?
- Comment sont gérés les droits d'accès ?
- Y a-t-il autant de bases de comptes que d'applications ?
- Qui consulte les documents (localisation, répartition horaire, *etc.*) ?
- *etc.*

Il nous semble que ces quelques questions soulignent à quel point les tâches d'animation et d'exploitation sont essentielles pour la réussite de l'intranet.

S'il n'est sans doute pas possible ni souhaitable d'exploiter ou de contrôler finement le *contenu* des publications (mais cela relève d'une politique de communication d'entreprise), il est néanmoins indispensable d'offrir aux utilisateurs un certain nombre de guides et de facilités de publication (documents d'explication, modes d'emploi, formulaires automatiques de publication et de mise

à jour, modèles, feuilles de style, *etc.*) qui, en les affranchissant des contraintes techniques, permettront d'assurer automatiquement des tâches d'administration d'ensemble (authentification, droits de publication, type d'information publiée).

Il est aujourd'hui communément admis que l'ensemble des matériels physiques ainsi que les couches basses du modèle OSI doivent être exploités et nécessitent un minimum d'administration et de supervision : le réseau physique, les *hubs*, les routeurs, les serveurs, les systèmes d'exploitation, les logiciels installés.

Une approche traditionnelle de l'exploitation et de l'administration s'arrêterait à ces éléments clairement identifiés. Aussi est-il important de sensibiliser l'entreprise de l'importance de créer et de prendre en charge la nouvelle infrastructure de communication, un noyau intranet, gérant l'authentification des utilisateurs, les mécanismes d'accès aux documents, les éléments de présentation ainsi qu'un ensemble de règles communes de développement et des interfaces, imposés aux différents intervenants.

Nous avons toutefois abordé dans ce chapitre une grande partie de ces questions, où nous avons montré que la cohérence des liens hypertextes ainsi que la génération dynamique des différents éléments de structure d'un ou de plusieurs serveurs intranet réduisent de façon importante les causes d'incohérence et amènent une grande simplification de la maintenance d'ensemble.

Une première étape vers un système d'authentification unique a aussi été décrite, de même que différents éléments de métrologie qui peuvent être modulés en fonctions des interlocuteurs.

Ces éléments, techniques, ne répondent cependant pas, seuls, aux soucis d'exploitation et d'administration mais forment une boîte à outils, évolutive, soumise à une volonté d'organisation et de gestion globale d'un intranet.

Le chapitre suivant précise l'implémentation effective et les choix techniques de RICERCAR, qui se sont largement inspirés et adaptés des contraintes issues du contexte de la RATP. Sa finalité est, dans un premier temps, de concrétiser les fonctionnalités décrites ici et de justifier les choix effectués. Il devrait permettre au lecteur, dans un deuxième temps, de s'assurer de la cohérence de la démarche poursuivie, notamment en tant qu'étape acceptable vers une architecture répartie, s'appuyant sur des solutions techniques normalisées.

4. RICERCAR par le Menu

...Enfin, me plaçant sur un plateau de fer,
Prendre un morceau d'aimant et le lancer en l'air !
Ça, c'est un bon moyen : le fer se précipite,
Aussitôt que l'aimant s'envole, à sa poursuite ;
On relance l'aimant bien vite, et cadédis !
On peut monter ainsi indéfiniment.

E. Rostand
(Cyrano de Bergerac, Acte III, scène 11)

4.1 Réflexions et méta-choix 85
4.2 Intranet : modélisation..... 91
4.3 Choix des outils..... 94
4.4 Équivalence aux Objets dans un SGBD..... 102
4.5 Réalisation du modèle présenté 107
4.6 Caractéristiques..... 131
4.7 Application : génération de Documents..... 138
4.8 RICERCAR, un ORB 150

4.1 Réflexions et méta-choix

4.1.1 Amorçages

Lorsque nous nous sommes attaqués en 1995 au problème de l'intranet, nous étions loin de penser que ces premières lignes de code amorçaient ce qui allait devenir une Thèse d'Université ! Il s'agissait, en effet, simplement de résoudre un problème certes complexe mais bien identifié : comment permettre à quelques milliers d'utilisateurs de la RATP d'accéder de manière fiable à une pléiade future de documents mobiles et de durée de vie aléatoire ?

Le modèle du Web alors suggéré par l'Internet nous semblait inadapté avec ses URL physiques, ses innombrables « Error 403 » ou « Document not Found ».

Notre familiarisation successive avec plusieurs interpréteurs et langages de programmation, capables d'interfacer un serveur Web, nous a d'abord convaincu de l'importance stratégique de cette technologie, moyen efficace de banaliser le client-serveur, en connectant les bases de données d'origines diverses de l'entreprise à un client universel.

Une fois réalisées les interfaces initiales avec des SGBD, un premier amorçage nous a amené à considérer qu'une solution au problème posé (celui de la persistance des liens) consistait à mettre en place une application client-serveur particulière : une base de données qui référencerait l'ensemble des documents publiés sur l'intranet. Cette base, administrée et mise à jour, permettrait aux utilisateurs finals d'accéder à leurs documents à partir d'un identifiant invariant qui masquerait la mobilité ou les modifications réelles.

Nous nous sommes rendu compte, après que cette méta-application eut été développée, que la *méta*-base de données offrait la possibilité de nombreux enrichissements à moindre coût : enregistrer des informations textuelles pour identifier les documents (titres, description, *etc.*) ; générer dynamiquement l'arborescence des serveurs en associant aux documents les « menus » auxquels ils contribuent ; factoriser les éléments de présentation des différents documents dans des scripts afin de permettre une homogénéisation et l'implémentation d'une charte graphique d'entreprise. En bref, il était possible d'y modéliser un intranet [Séraphin_97b].

De légères améliorations de la base ont alors rapidement permis d'y référencer d'abord les figures, dont les URL présentent la même fragilité que celle des documents où elles sont incluses, puis de créer une bibliothèque d'images (puces, fonds, flèches, *etc*.), mises à la disposition de l'ensemble des serveurs (et des auteurs), afin de servir d'éléments de présentation communs.

Une série de modifications « cosmétiques » ont ensuite mis en évidence la nécessité de regrouper dans un ensemble de classes les différents types de

documents publiés, implémentant dans des scripts les méthodes de génération associées.

À la fin de cette étape, le système élaboré implémente : la persistance des liens et la transparence à la localisation des documents ; la génération dynamique de l'arborescence des serveurs ; l'usage automatique d'une charte graphique d'entreprise, par intégration des documents des utilisateurs dans des modèles prédéfinis ; *etc*.

Cependant, le développement de cette application a introduit à son tour un problème de même nature que celui qu'il cherchait à résoudre : il a en effet fallu écrire un ensemble de scripts, répartis ou dupliqués sur plusieurs serveurs et eux aussi souvent modifiés.

Réalisant que ce problème n'est pas aussi aigu que celui d'origine, dans la mesure où il concerne les administrateurs et non plus les utilisateurs finals (qui sont, somme toute, beaucoup plus nombreux que les administrateurs de serveurs Web, et sont présumés techniquement moins compétents), il était néanmoins nécessaire de disposer d'un outillage minimal qui permît d'assurer la maintenance et la documentation de l'ensemble des serveurs et de leurs programmes.

Il nous a fallu répertorier dans un tableur les différents programmes écrits, en déterminer une typologie et réaliser la documentation. Ce n'est que dans un second temps qu'il nous est apparu que la solution à ce (méta) problème était exactement de même nature que le problème initial. Nos programmes sont eux aussi des fichiers de texte, regroupés par thèmes : il suffit donc de les décrire à leur tour dans la méta-base de données.

Ce deuxième amorçage a donc consisté à écrire des scripts provisoires [Pitrat_86], pour enregistrer au fur et à mesure de leur exécution l'ensemble des scripts utilisés dans la méta-base.

En fin de ce processus, RICERCAR référence indifféremment ses propres données ainsi que les documents de l'intranet. Cette uniformisation de la représentation permet d'implémenter simplement une gestion des utilisateurs et des listes de droits associés ainsi que des statistiques de consultation.

Restait à traiter un dernier hiatus qui persistait dans la dichotomie de la représentation, où les classes de documents étaient décrites par des relations entre tables de la méta-base. Cette séparation limitait les possibilités d'introspection du système qui ne pouvait pas accéder à ses propres structures (pour d'éventuelles modifications, rendues par ailleurs complexes à cause de l'éparpillement des informations dans ces relations).

Ce troisième amorçage, structurel (inspiré du modèle *ObjVlisp* [Cointe_85]), nous a alors amené à considérer une approche « tout-document » en décrivant à leur tour les classes de documents par des documents, enregistrées donc dans la méta-base et instanciant une métaclasse de modèles, elle-même vue comme instance de document.

Après cette modification qui a entraîné une profonde révision de l'ensemble des méthodes écrites jusqu'alors (promues au rang de méta-scripts), le système ne gère

plus que des classes de documents, au nombre desquelles se trouve une sous-classe particulière, les scripts. Ceux-ci peuvent désormais, selon qu'ils sont considérés comme des programmes, générer l'ensemble des documents des serveurs, ou alors, vus comme des documents, s'appliquer à eux-mêmes et, au hasard de leur exécution, modifier leur propre code ou plus classiquement générer une auto-documentation [Ferber 89].

Reste alors un dernier amorçage, duquel le lecteur est témoin, se résumant à décrire et à défendre la conception de RICERCAR par un ensemble structuré de fichiers qui, formant cette Thèse, iront naturellement trouver leur place parmi les documents, classes et scripts déjà référencés par RICERCAR.

4.1.2 Réflexions autour d'un intranet (convaincre l'entreprise de se risquer dans la conception d'architectures réparties)

Il nous semble important, après avoir brossé, dans les chapitres précédents, un panorama général ainsi que les attendus de notre Thèse, d'ouvrir une parenthèse pour associer le lecteur à la démarche itérative de conception et aux réflexions menées, qui ont abouti à un ensemble de choix dont la justification serait impossible en dehors de toutes considérations liées au contexte de la RATP.

L'informatique de la RATP a connu dans les années soixante-dix une période faste où des projets audacieux ont été menés avec des succès divers. Les restrictions budgétaires, héritées de la crise économique, ont amené le département [18] (ainsi que l'ensemble de la RATP) à recentrer ses interventions autour du métier premier de l'entreprise. C'est ainsi que la fonction de chef de projet informatique a été progressivement redéfinie comme étant celle d'un coordinateur chargé, d'une part, d'aider la maîtrise d'ouvrage à exprimer ses besoins et, d'autre part, de réaliser le suivi du développement des applications par des sociétés tierces.

La promotion des gestionnaires par rapport aux « techniciens » a ainsi, au fil des ans, naturellement déplacé le barycentre du département et amenuisé les compétences techniques de ses agents. Cette « sédentarisation » des informaticiens a par conséquent nourri un certain nombre d'habitudes qui tendent à privilégier des solutions « sûres et éprouvées ».

C'est ainsi que la politique ambitieuse qui consistait à promouvoir les systèmes ouverts pour permettre à la RATP de mettre ses fournisseurs en concurrence s'est, après des débuts encourageants, résumée à deux grands axes : des serveurs applicatifs HP-UX/ORACLE et, paradoxalement, des postes de travail monocolores, tout *Microsoft*.

[18] **S**ystèmes d'**I**nformation et de **T**élécommunications, (*cf.* Annexe 6.3, p.185)

Cette absence d'hétérogénéité n'est pas neutre notamment dans les études d'architectures, structurantes par définition. Il s'est ainsi avéré que la tendance naturelle, qui est souvent de choisir de ne pas choisir, équivaut à une prime pour la solution *ad hoc* du fournisseur déjà en place. Cette solution attirante, car bien intégrée avec l'existant, se concrétise par des investissements qui réalisent un ensemble de *minima* locaux dont la somme, rarement calculée *a posteriori*, n'a aucune raison de coïncider avec un *optimum* global, en raison des coûts d'administration et de maintien d'une cohérence d'ensemble.

Il est donc indispensable, dans ce contexte de technicité moindre, d'ancrer solidement toute nouvelle proposition d'architecture sur des bases largement adoptées dans l'entreprise et, s'appuyant sur leurs éventuelles lacunes, en dévoyer l'usage pour amorcer une maturation de technologies nouvelles.

C'est dans cet état d'esprit que nous avons abordé le projet de l'intranet avant même la consécration de ce terme. Nous avons en effet été immédiatement convaincus de l'adoption rapide des technologies issues du Web dans l'entreprise, plus en raison de la banalisation de l'accès au client-serveur que par l'adoption d'un modèle de libre diffusion d'information, peu compatible avec les tendances centralisatrices de la communication d'entreprise.

Il nous a fallu par ailleurs nous résoudre à l'impossibilité de conduire une expérimentation à la RATP, basée sur des composants applicatifs réutilisables. L'entreprise a peu d'expérience de projets ayant mis en œuvre des objets ; n'est pas suffisamment sensibilisée à l'importance de définir un dictionnaire commun de données [19] et, ne disposant plus des compétences internes suffisantes, n'est de surcroît pas convaincue de l'intérêt même de ces technologies.

Il était également irréaliste (mais ô combien tentant) d'appuyer un développement autour d'un serveur Web écrit en Lisp (*CL-HTTP* [MIT_97]) ou en Java (*Jigsaw* [W3C_97]), que d'interfacer des serveurs traditionnels avec des bus objets tel CORBA [20], en dépit de l'intérêt, de la portabilité et de l'extensibilité de ces approches [Welsh_97].

La complexité de la « qualification [21] » et du déploiement des applications clientes sur un parc comprenant plus de 10.000 postes de travail en réseau est d'une telle acuité, que plusieurs mois s'écoulent entre la mise en service d'une base de données et la fin du déploiement de son application cliente. Il nous est apparu bien plus important de sensibiliser la RATP à la nécessité d'adopter de nouveaux concepts (généralisation du client-serveur par le biais d'un navigateur, réutilisation des *données*, mise en place progressive d'un référentiel commun, construction *itérative* d'architectures sophistiquées à partir d'assemblage d'outils hétérogènes,

[19] Nous ne comptons plus le nombre de déclinaisons et de réplications d'informations de base, tels le matricule d'un agent, la définition du nombre de voyageurs au km, *etc.*

[20] CORBA 2 n'était alors qu'un nouveau-né.

[21] La qualification consiste à s'assurer, récursivement, à partir de la qualification initiale d'un poste de travail Windows 3.11 sous TCP/IP, de la compatibilité d'une application avec l'ensemble des applications déjà qualifiées.

etc.) que d'essayer un *forcing* en exhibant une solution basée sur des technologies de pointe mais inadaptée à la culture actuelle de l'entreprise.

La diffusion d'un navigateur sur un tel parc (une fois réglée la question picrocholine de son choix) est une clef importante qui offre un terrain de validation et un vecteur de propagation irréversible pour les technologies de l'intranet. Celui-ci induit, par définition, une architecture répartie dont la mise en œuvre réalise une application distribuée. Cette architecture doit, pour réussir, adresser des considérations relatives au nommage, à la communication entre « composants », aux interfaces entre « objets », à l'optimisation des performances et à la cohérence d'ensemble, afin de garantir à ses utilisateurs une qualité de service et des fonctionnalités incrémentales [Coulouris_95].

« Les applications ayant le plus de succès ne sont jamais terminées - Elles évoluent avec l'entreprise qu'elles servent » [Corkill_97].

Nous avons donc défini une stratégie d'approche basée sur un modèle de programmation réflexive qui appuie l'architecture par un ensemble d'applications pilotes qui illustrent les fonctionnalités décrites. Ce modèle [Paepcke_90, Robertson_92], par sa définition d'un référentiel commun pour la description des données de l'entreprise, lentes à évoluer, et de celles des utilisateurs, plus fugaces mais en forte croissance, assure une cohabitation hétérogène mais harmonieuse.

La démarche Orientée Objet, implicite dans un tel paradigme, n'impose toutefois pas l'usage d'un langage ni même d'un système de stockage Objets. Elle implique d'adopter un modèle et une implémentation qui favorisent une réutilisation maximale et la définition d'une architecture applicative, organisée en couches fonctionnelles. Ce découpage instaure une forme d'encapsulation et de polymorphisme qui rendent possible le remplacement d'un « objet » par un autre ayant les mêmes comportements mais implémenté différemment. Il permet ainsi de construire, par incréments successifs, un système de plus en plus élaboré [Boehm_88, Booch_95a, Kruchten_96] où les facteurs de risque les plus importants ont été repérés et éliminés dans ses premières itérations.

RICERCAR est, dans une première acception (celle de l'utilisateur final, la seule qui compte), une architecture extensible qui offre un accès banalisé et fiable aux données de l'entreprise en s'affranchissant de la contrainte « une application cliente pour une application serveur » [Séraphin_97b].

Cette architecture s'appuie, délibérément, sur un assemblage hétérogène et interchangeable de composants « standards » : système d'exploitation, serveur Web, bases de données et métalangage d'interface. Comment en effet vanter les mérites des objets, des composants et de leur réutilisabilité sans avoir la cohérence minimale de réutiliser les composants déjà déployés dans l'entreprise (en construisant un intranet *ex nihilo* à partir de langages de bas niveau ou en définissant une nouvelle architecture basée sur CORBA qui impose une évolution brutale du Système d'Information) ?

Notre moindre compétence sous UNIX et l'absence initiale d'outils (de haut niveau) dans cet environnement nous ont fait préférer une instance de cet assemblage sous Windows NT.

La passivité des rares techniciens influents du département, opposés à tout développement qui s'appuie sur des technologies issues de *Microsoft*, ainsi que le manque de relais internes pour soutenir une quelconque architecture technique, justifient que la première application « officielle » de l'intranet ait seulement vu le jour en septembre de cette année.

Il est à souhaiter que le portage des environnements de développement dans le monde UNIX, la maturité récente des offres basées sur les architectures de l'OMG et la crainte, légitime, de voir le modèle DCOM de *Microsoft* s'imposer « naturellement » dans l'entreprise, contribuent à aplanir les différents et permettent à RICERCAR d'être vu comme une étape dont le polymorphisme peut augurer l'adoption progressive de modèles normalisés.

4.2 Intranet : modélisation

Les différentes normes ou standards qui décrivent des documents (SGML, ODA, *etc*.) s'attachent principalement à en distinguer les représentations physique et logique afin de les rendre indépendantes. Ces normes ne s'intéressent pas, en revanche, à la détermination de méta-données nécessaires (ou suffisantes) pour caractériser ces documents.

La modélisation d'un intranet consiste cependant à abstraire, en fonction des besoins de l'entreprise, un sous-ensemble de caractéristiques remarquables. Celles-ci permettront de décliner la charte graphique et d'implémenter les règles de navigation ainsi que les différents attributs jugés importants pour le confort des utilisateurs.

Il est important que ce modèle puisse être simplement enrichi pour accompagner l'évolution des besoins.

Les paragraphes suivants décrivent une sélection particulière de propriétés que nous avons jugé pertinentes. Les différents étapes de ce choix, qui reflètent des critères subjectifs, ne présentent pas un grand intérêt pour cet exposé.

Il nous paraît, en revanche, important de préciser ici la représentation finalement choisie afin de préciser les *méta-données* implémentées. Celles-ci forment, en effet, les *données* manipulées tout au long de ce chapitre, pour relater la construction du système.

Aussi encourageons-nous vivement le lecteur qui n'aurait pas eu l'occasion d'accéder à la version navigable de ces pages à consulter les exemples, p.93 ou en Annexe 6.6, p.209 et suivantes.

Ce modèle, qui est une réification du « Panorama des besoins » § 3.3.1, p.68, nous a conduit à identifier trois familles de données connexes, de plus en plus spécialisées. Elles visent à concilier respectivement les points du vue de l'*utilisateur* final, de l'*auteur* d'un document et d'un *administrateur* :

- vu de l'*utilisateur,* un document comporte (en sus de son contenu !) les renseignements suivants :
  - ◊ disponibilité (le lien est-il actif ?) ;
  - ◊ description textuelle (titre, description du document, *etc*.) ;
  - ◊ fiabilité (l'accès est-il garanti ?) ;
  - ◊ droits (ai-je le droit d'accéder à ce document ?) ;
  - ◊ traduction (existe-t-il une version dans une autre langue ?) ;
  - ◊ importance relative ;
  - ◊ dates importantes (création, modification) ;

- ◊ renseignements sur l'auteur ;
- ◊ renseignements sur l'administrateur (à qui s'adresser en cas de problème ?) ;
- ◊ éléments de navigation (dans quel ensemble s'inscrit le document ?).

- vues de l'*auteur*, les informations précédentes doivent être précisées par :
  - ◊ la catégorie du document (sa classe) ;
  - ◊ la mesure de l'intérêt de sa contribution (estimation grossière du nombre d'accès) ;
  - ◊ les moyens d'accès (thèmes et rubriques auxquels ce document contribue. S'il est accessible par un moteur de recherche) ;
  - ◊ les éléments graphiques qui lui sont associés (logotype, icône, *etc*.)
- l'*administrateur* détaille, à son tour, l'ensemble des informations décrites :
  - ◊ renseignements sur les utilisateurs (nom, prénom, matricule -RATP-, *etc*.) ;
  - ◊ regroupement des utilisateurs (différents profils à définir) ;
  - ◊ affectation des droits d'accès éventuels (de consultation mais aussi de modification et de création) par profils d'utilisateurs ;
  - ◊ « tracabilité » des documents (qui y a accédé, quand, comment ?) ;
  - ◊ dates (indexation, remise à zéro des compteurs) ;
  - ◊ compteurs (nombre d'accès global, depuis une date fixée) ;
  - ◊ localisation physique des documents (disque, base de données, mémoire) ;
  - ◊ nombre, localisation, reconnaissance de caractères et taille des éventuelles images incluses dans le document ;
  - ◊ ordonnancement des menus et rubriques ;
  - ◊ type des documents (terminaux ou nœuds de l'arborescence ?) ;
  - ◊ description détaillée des classes de documents (composition des en-têtes, textes signataires, choix des fonds d'écran, description des feuilles de style, *etc*.) ;
  - ◊ « dynamicité » des informations (le document peut-il être mémorisé par le cache de l'utilisateur ou doit-il être systématiquement recomposé ?) ;
  - ◊ liste des différents types de navigateurs utilisés (pour pouvoir identifier des problèmes spécifiques à une configuration donnée ou optimiser la génération) ;
  - ◊ informations sur les serveurs (nombre, adresses, localisation des méta-données, administrateurs locaux, *etc.*)

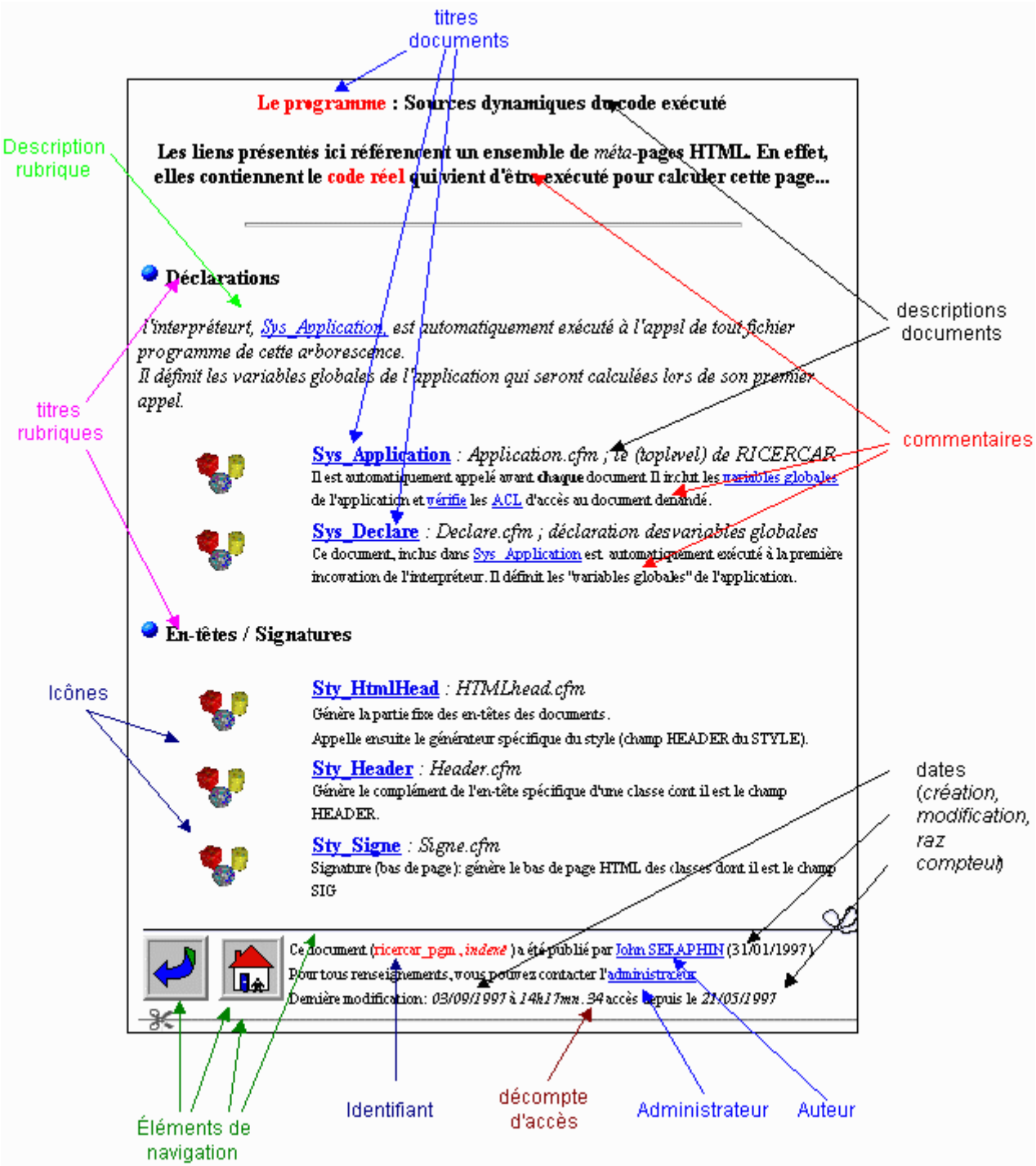


Figure 4. 1. Un exemple de la représentation concrète de différentes méta-données dans un document (ici un menu).

4.3 Choix des outils

Ainsi que nous l'avons signalé précédemment, l'architecture finalement réalisée comprend quatre sous-éléments largement autonomes : un serveur HTTP, un métalangage d'interfaçage, un SGBD, l'ensemble inter-opérant sur un système d'exploitation donné.

Nous avons déterminé ces éléments à la fin de l'année 1995, alors que nous étions les premiers à nous interroger sur le type d'outil qu'il convenait d'utiliser dans l'entreprise. Aussi les choix effectués représentent-ils un environnement de développement et de programmation qui nous est familier, que nous pensons néanmoins compatible avec les choix habituels de la RATP.

Nous avons eu l'opportunité de vérifier à plusieurs reprise, pendant ces deux années, la robustesse de l'architecture par rapport au changement de différents éléments. Ainsi avons-nous déjà remplacé le serveur HTTP (dans le cadre d'essais de *Netscape* ou de *IIS*). Nous avons aussi échangé ou utilisé de façon mixte plusieurs sources ODBC (Excel, MS-Access, SQL-Server, ORACLE/Unix).

Nous avons par ailleurs étudié le portage de RICERCAR dans un autre métalangage que *Cold Fusion* (*iHTML* [ihtml_97] en l'occurrence) puis y avons renoncé par manque de motivation et devant la charge induite par la gestion simultanée de deux versions cohérentes du système ainsi que des éventuels problèmes de communication entre ces implémentations.

Nous envisageons cependant sérieusement de refondre le système dans une architecture CORBA/Java en 1998 si nous obtenons un minimum de soutien et de collaboration de la RATP ou de l'Université René Descartes. Cette refonte ne devrait pas influer sur ses fonctionnalités, mais l'asseoir sur des bases normalisées.

Quant au système d'exploitation, le choix de *Windows NT* [Microsoft_93b] a été établi en raison de la disponibilité initiale des différents métalangages dans cet environnement. Leur portage sous UNIX, en cours, permet donc d'envisager des instances de RICERCAR implantées indifféremment dans chacun de ces deux environnements.

4.3.1 Le serveur HTTP : WebSite© de O'Reilly

Le choix du serveur HTTP a été aussi effectué à une époque où seuls quelques protagonistes disposaient de solutions sous Windows (*Microsoft* ignorait encore superbement l'Internet et envisageait son expansion *via* MSN !).

Bob Denny, cité parmi les contributeurs à la définition du protocole HTTP [Berners-Lee_94a], avait alors mis gracieusement à la disposition de la communauté de l'Internet un serveur sous Windows 3.11 (WinHTTPD 1.4c [Denny_95]).

Le grand succès rencontré par ce serveur a entraîné son portage dans un système d'exploitation 32 bits et sa commercialisation, sous le nom de WebSite, par la maison d'édition *O'Reilly*, connue notamment pour la qualité de ses ouvrages techniques dans l'environnement UNIX. Ce portage a été effectué longtemps avant que *Netscape* ou *Microsoft* aient développé des serveurs HTTP sous NT.

C'est ainsi que WebSite :

- s'exécute indifféremment sous Windows 95 ou NT ;
- gère différents protocoles de cryptage (*SSL* 2, SSL3) ;
- utilise, au choix, une base de comptes d'utilisateurs spécifique ou alors s'appuie sur les comptes du domaine NT [22] ;
- supporte plusieurs serveurs virtuels à partir d'une seule ou de plusieurs adresses IP (*virtual multihoming*) ;
- comprend des API propriétaires (WSAPI) mais aussi les API de *Netscape* et de *Microsoft* (NSAPI et ISAPI) ;
- est le seul à intégrer indifféremment tous les langages de scripts (*ASP*, *Cold Fusion*, *Frontpage*, *iHTML*, *etc*.) ;
- est un des seuls serveurs à supporter intégralement la RFC-1867 [Nebel_95] qui permet la publication (*upload*) de documents HTML par les utilisateurs, sans interface propriétaire (telles les extensions *FrontPage* qui ouvrent des brèches béantes dans la sécurité d'un serveur) ;
- est livré, accessoirement, avec plusieurs tomes de documentation [Peck_96, Peck_97a, Peck_97b], conçus avec la pédagogie habituelle de *O'Reilly*.

La qualité de son interface unique d'administration, dont la simplicité et l'ergonomie contrastent avec celles des concurrents directs, *Microsoft* et *Netscape*, nous a renforcé tout au longs de ces mois dans notre choix pour ce serveur.

[22] Il faut cependant insister sur le manque de sécurité de cette méthode (indépendamment du choix du serveur), puisque cette authentification écrit les mots de passe en clair dans le fichier de *logs* du serveur.

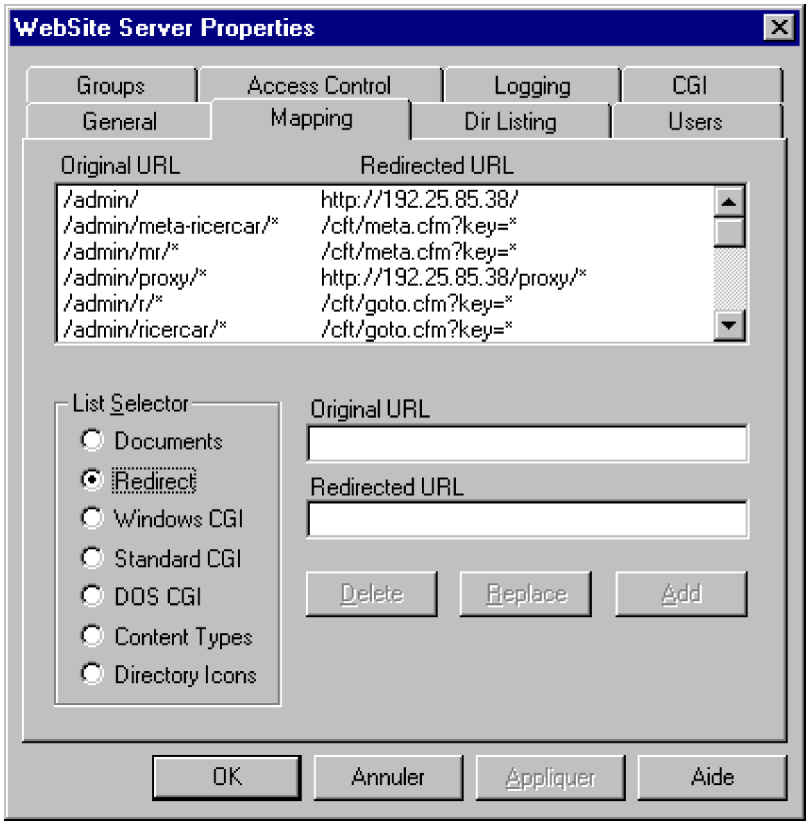


Figure 4. 2. Interface d'administration unique du serveur qui permet une gestion fine au travers des différents onglet.

4.3.2 Bases de Données

Il est vraisemblable que l'extension de l'usage de RICERCAR justifierait un déploiement sur des bases de données du type ORACLE ou SQL-Server, qui offrent des performances bien plus élevées tout en gérant de manière plus satisfaisante les problèmes de robustesse, de concurrence d'accès et de réplication partielle des données.

Il s'avère cependant à l'usage que l'adoption de MS-Access 97 pour les méta-bases, qui restent de dimensions raisonnables (quelques milliers d'enregistrements), est suffisamment simple et fiable pour que son adoption n'ait pas soulevé jusqu'ici de questions majeures (un problème de concurrence d'accès a pu être rapidement résolu par l'affinage des paramètres de l'interface Web-ODBC).

Nous avons mis en œuvre les mécanismes natifs (de MS-Acces) de réplication et de synchronisation des réplicats qui peuvent être automatisés et contrôlés par des scripts Visual Basic. Le changement de SGBD éventuel, nécessiterait l'adaptation

de ces réplications en fonctions des nouveaux mécanismes fournis (qui sont de toutes façons propriétaires).

4.3.3 Le métalangage Cold Fusion

Les CGI ont permis l'extensibilité du protocole HTTP en permettant de connecter un serveur sur des sources de données [McCool_95, Gundavaram_96]. De nombreux *méta*langages structurés ont ainsi été développés pour construire des applications modulaires qui rendent aisées l'interrogation et la mise à jour, sur l'Internet ou en intranet, de bases de données issues d'applications client-serveur.

Nous étant rapidement intéressé aux différents Produit Orienté Web (POW), nous avons opté pour le choix de *Cold Fusion Markup Language* (*CFML*) [Allaire_97a] qui nous a permis, dans un premier temps, de réaliser les premiers prototypes d'interfaces client-serveur avec des bases ORACLE d'entreprise puis, dans un second temps, de servir de support de développement à notre Thèse.

Nous avons ainsi pu concentrer nos efforts sur l'implémentation effective des multiples itérations de l'architecture proposée plutôt que de nous disperser dans la conception d'un langage structuré supplémentaire dont la RATP ne saurait par ailleurs assurer ni la maintenance ni l'évolution.

Après deux années d'utilisation intensive, nous ne pouvons que nous féliciter du choix de ce langage qui a intégré et adapté, dans ses versions successives, le support des différentes technologies émergentes [Biggs_97, Ho_97] :

- enrichissement du langage par la liaison à des classes **C++** externes (écrites par l'utilisateur);
- **Java** (extension du langage par des *applets* système et utilisateurs, paramétrables) ;
- intégration des API d'un des **moteurs d'indexation et de recherche** les plus performants du marché [Verity_96] ;
- gestion (accès, modification) d'annuaires normalisés **LDAP** [Yeong_95, Howes_97] ;
- possibilité d'instancier des classes réparties **COM/DCOM** ;
- un **ORB** (VisiBroker [Visigenic_97]) sera intégré dans la prochaine version (prévue pour le premier semestre 1998).

4.3.3.1 Visite rapide des fonctionnalités du langage

Le langage CFML étend HTML par un ensemble de balises qui permettent d'interroger en SQL n'importe quelle source de données pourvue d'un connecteur ODBC [Microsoft_92] (la plupart des SGBD sous NT ou UNIX) selon un schéma général, indiqué ici :

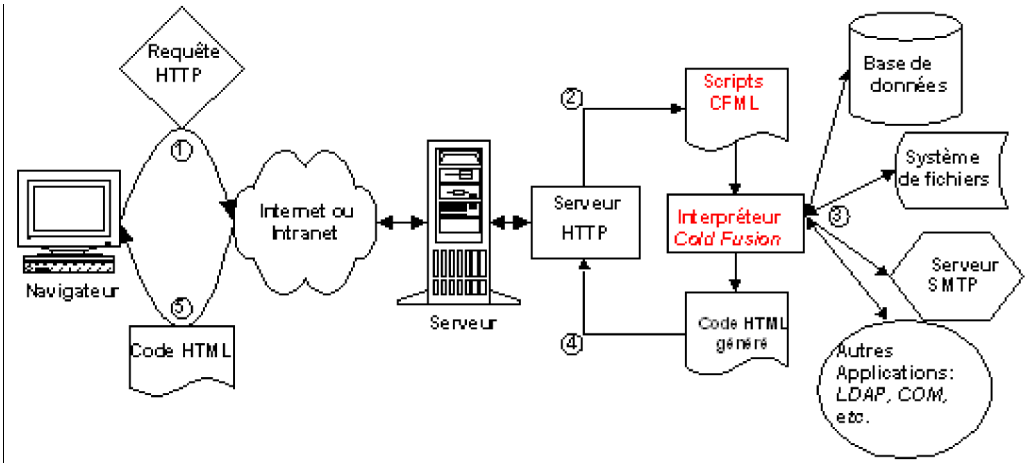


Figure 4. 3. Cheminement d'une requête générant une page HTML à partir d'un interpréteur.

Nous avons jugé pertinent d'expliciter, en quelques paragraphes, les principales balises et fonctions CFML afin de permettre au lecteur (que nous espérons passablement familier de HTML) de suivre le sens général des extraits de code qui émailleront la suite de notre exposé :

- l'ensemble des balises CFML est préfixé par CF (CFQUERY, CFSET, CFOUTPUT, CFLOOP, *etc.*) ;
- l'affectation de variables se fait par la balise CFSET:

```
<!--- un commentaire CFML ---><CFSET var = "une variable">
<!--- les deux termes peuvent être évalués --->
<CFSET "var" & "iable" = "ceci est la valeur">
<!--- Affectation au résultat d'une fonction --->
<CFSET angle = "3.141593">
<CFSET aux = Cos(angle)>
```

- la fonctionnalité centrale du langage consiste à exécuter une requête SQL et de lui associer un nom logique (à l'instar d'une vue) :

```
<CFQUERY Name="ma_req" DataSource="Source_ODBC">
  SELECT      NOM, PRENOM, ADRESSE, TEL
  FROM        PERSONNE
  WHERE ...
</CFQUERY>
```

  Il est ensuite possible de parcourir sélectivement les tuples résultants par une notation pointée, proche de l'objet : *nom_requête.nom_colonne* qui, par une boucle implicite ou explicite, renverra les informations correspondantes ;

- le système fournit classiquement une série de préfixes (en sus des noms de requêtes) :
  - *CGI* pour les variables renvoyées par le serveur Web ;
  - *URL* pour les variables concaténées à l'URL reçue ;
  - *FORM* pour les variables postées dans un formulaire HTML ;
  - *COOKIE* pour les *cookies* instanciés ;
  - *SERVER* préfixe des variables globales, **disponibles pendant tout le fonctionnement du serveur** (sert, avec *CLIENT*, à gérer des sessions);
  - *CLIENT* préfixe des variables globales identifiant un navigateur donné et donc disponibles à chaque accès de ce navigateur (sert, avec *SERVER*, à gérer des sessions);
  - *CALLER* est utilisé dans une procédure locale pour accéder aux variables « externes » à la procédure (sert, avec *ATTRIBUTE*, à gérer l'encapsulation) ;
  - *ATTRIBUTE* permet à la procédure locale de reconnaître les variables locales (sert, avec *CALLER*, à gérer l'encapsulation) ;

- les variables CFML doivent être encadrées de '#' dans tout contexte qui ne permet pas à l'interpréteur de les identifier:

```
<!--- var*i* n'ont pas besoin d'être formellement signalées --->
<CFSET var1 = var2 & var3>

<!--- Seuls les '#' permettent ici à l'interpréteur de faire la distinction --->
<CFOUTPUT> Une variable var1 : #var1#</CFOUTPUT>
```

- l'affichage des résultats utilise la balise <CFOUTPUT> qui, si elle comprend un paramètre "QUERY=...", procède à une boucle implicite sur la requête nommée:

```
<!--- Affichage de la valeur d'une simple variable --->
<CFOUTPUT> Ceci est une #variable#</CFOUTPUT>

<!--- boucle implicite d'affichage sur la requête "ma_req" précédente --->
<CFOUTPUT Query="ma_req">
Nom : #nom#, Prénom : #prenom#,...
</CFOUTPUT>
```

- les fonctions de test et de branchement sont classiquement implémentées (les opérateurs de comparaison étant EQ*ual*, NEQ, GT, LT, LTE, etc.) :

```
<CFIF ma_req.NOM EQ "toto">
...
<CFELSEIF ...>
...
<CFELSE>
...
</CFIF>
```

- le développeur peut inclure des procédures CFML (avec des variables encapsulées) en écrivant un script qui est alors appelé en préfixant le nom du fichier par "CF_" (*CF_mon_script* correspondra par exemple au fichier *mon_script.cfm*) ;

- il est possible d'inclure un script paramétrable dans un autre, grâce à la balise CFINCLUDE, ce qui réalise effectivement la composition de fonctions (c'est ainsi que l'interpréteur invoque les méta-scripts) ;

- il est aussi possible de provoquer une redirection vers un autre script par le biais du marqueur CFLOCATION (qui génère une redirection HTTP 302).

  C'est notamment cette fonction qui nous permettra d'invoquer des envois de message en cascade ;

- l'invocation de classes COM/DCOM se fait par le biais de la balise CFOBJECT qui reçoit en paramètre la classe à instancier ou l'objet à réutiliser.

  C'est cette fonction qui sera étendue pour interfacer le DII de l'ORB prévu ;

- un compilateur incrémental associé à un cache, gérés par le serveur, assurent des performances stables à charge élevée et stockent en mémoire le résultat des requêtes fréquemment exécutées ;

- l'interpréteur *Cold Fusion* s'installe sur tout serveur *Windows 95* , *Windows NT* ou *UNIX* (actuellement *Solaris*) qui dispose d'un serveur HTTP supportant les *CGI* ou une des API les plus courantes du marché (*ISAPI , NSAPI* ou *WSAPI*). Sa conception réflexive est matérialisée par une fonction *evaluate* qui permet de construire - et d'interpréter - non seulement du code HTML mais aussi des expressions CFML ;
- l'administration de l'interpréteur, écrite en CFML, est de ce fait dynamique et peut être modifiée en cours d'exécution ;
- *etc*.

Ce langage, qui dispose par ailleurs de plus d'une centaine de fonctions, ne peut être résumé par cette brève introduction. Nous enjoignons donc le lecteur, désireux d'obtenir des précisions supplémentaires, à consulter la documentation en ligne (complète et indexée, quoiqu'en anglais) de CFML [Allaire 97b].

En résumé, la conception d'une application correspond à l'écriture d'un ensemble de scripts (mélange de CFML et HTML). Ceux-ci vont, en fonction des besoins, quérir des bases de données, appeler d'autres scripts puis générer le code HTML correspondant à la page finale renvoyée à l'utilisateur, enrichissant éventuellement ces bases de données en fonction des différentes interactions avec les utilisateurs.

4.4 Équivalence aux Objets dans un SGBD

4.4.1 Modèles finals

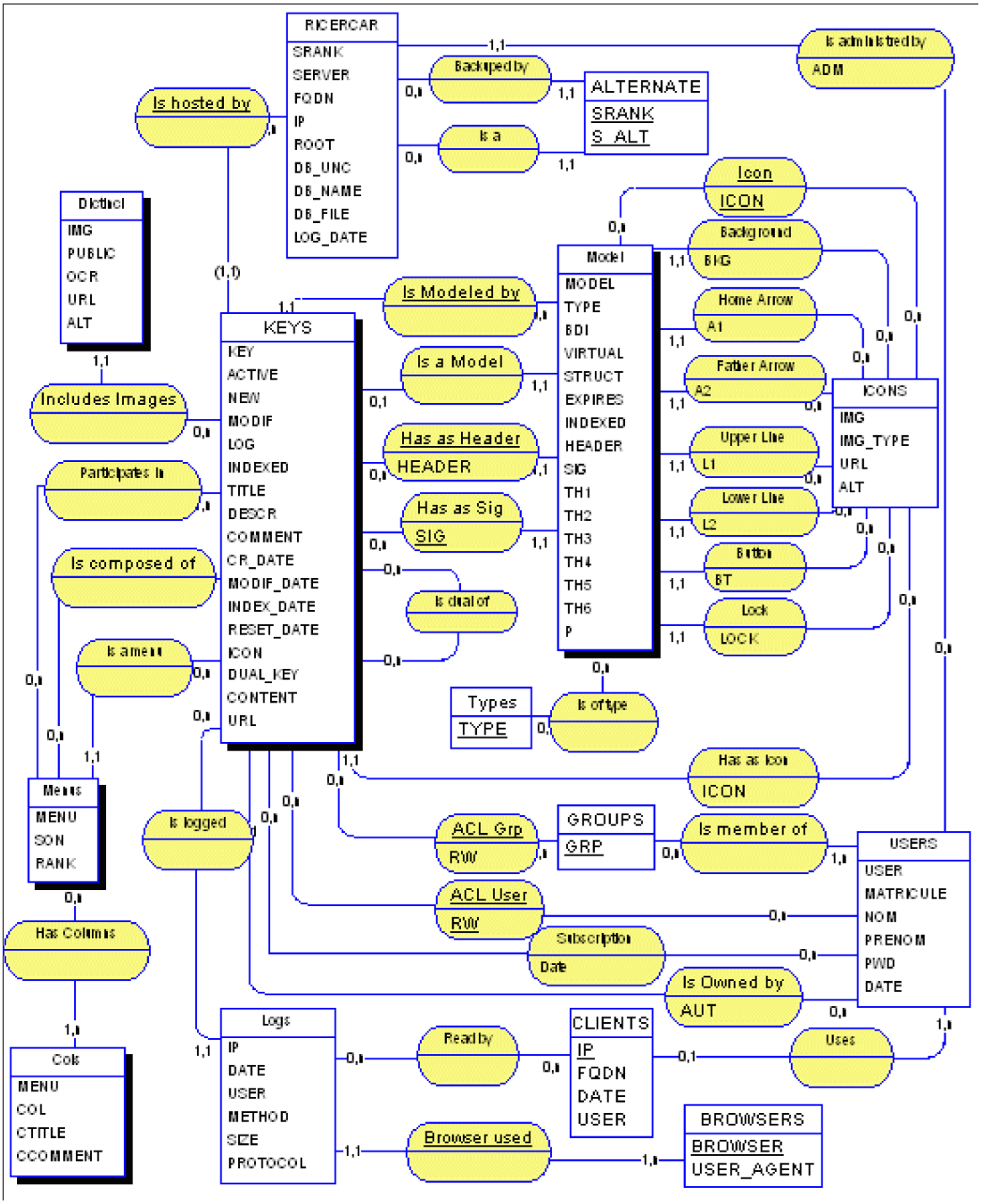


Figure 4. 4. Schéma du MCD complet, correspondant au schéma de classes final.

Après avoir réalisé le modèle objet, dont la construction fait l'objet d'une partie importante de ce chapitre (§ 4.5.3.4, p.124), nous nous sommes inspiré de la méthode Merise [Collongues_89, Tardieu_91], pour réaliser le *MCD* du système, dont le rapport figure en Annexe 6.4, p.195.

Deux heuristiques assez simples nous ont ensuite guidé dans la répartition des méta-données dans deux ensembles (le modèle Merise ne s'intéresse pas beaucoup aux problèmes de distribution et de réplication des données) :

1. isoler les données nécessaires à l'ensemble des serveurs de celles qui peuvent rester locales ;
2. assurer un minimum de redondance qui minimise les effets d'une panne locale.

L'application de ces heuristiques nous a ainsi conduit à concevoir la méta-base correspondante comme l'union de deux bases attachées (selon la terminologie MS-Access) :

1. une première base, répliquée sur l'ensemble des serveurs, grâce aux mécanismes de synchronisation intrinsèques, et qui implémente la première heuristique ;
2. une seconde base qui comporte les données spécifiquement locales au serveur.

Cette répartition remplit les objectifs classiques des bases fédérées [Gardarin_90] :

- **indépendance à la localisation** : permettre aux utilisateurs de consulter n'importe quel document sans qu'ils n'aient besoin de savoir s'il est local ou distant ;
- **indépendance à la fragmentation** : favoriser les accès locaux en divisant les relations en un ensemble de sous-relations (fragmentation verticale) et en stockant sur chaque site l'ensemble des données nécessaires à son fonctionnement (fragmentation horizontale) ;
- **indépendance à la duplication** : dupliquer sur chaque serveur les informations nécessaires à son bon fonctionnement (classes, méthodes, *etc.*) ;
- **autonomie des bases** : la duplication et la fragmentation, permettant de disposer localement de toutes les informations relatives à un serveur, lui assurent son autonomie (il n'est que très faiblement affecté par la panne des autres serveurs) ;
- **extensibilité** : pouvoir enrichir le réseau par la mise en place incrémentale de nouveaux serveurs ;
- **performances** : la réplication et la duplication des données favorisant les accès locaux aux données assurent des performances comparables à celles d'une base locale.

Le schéma physique des bases suivant, indique la répartition et le placement des différentes propriétés et relations :

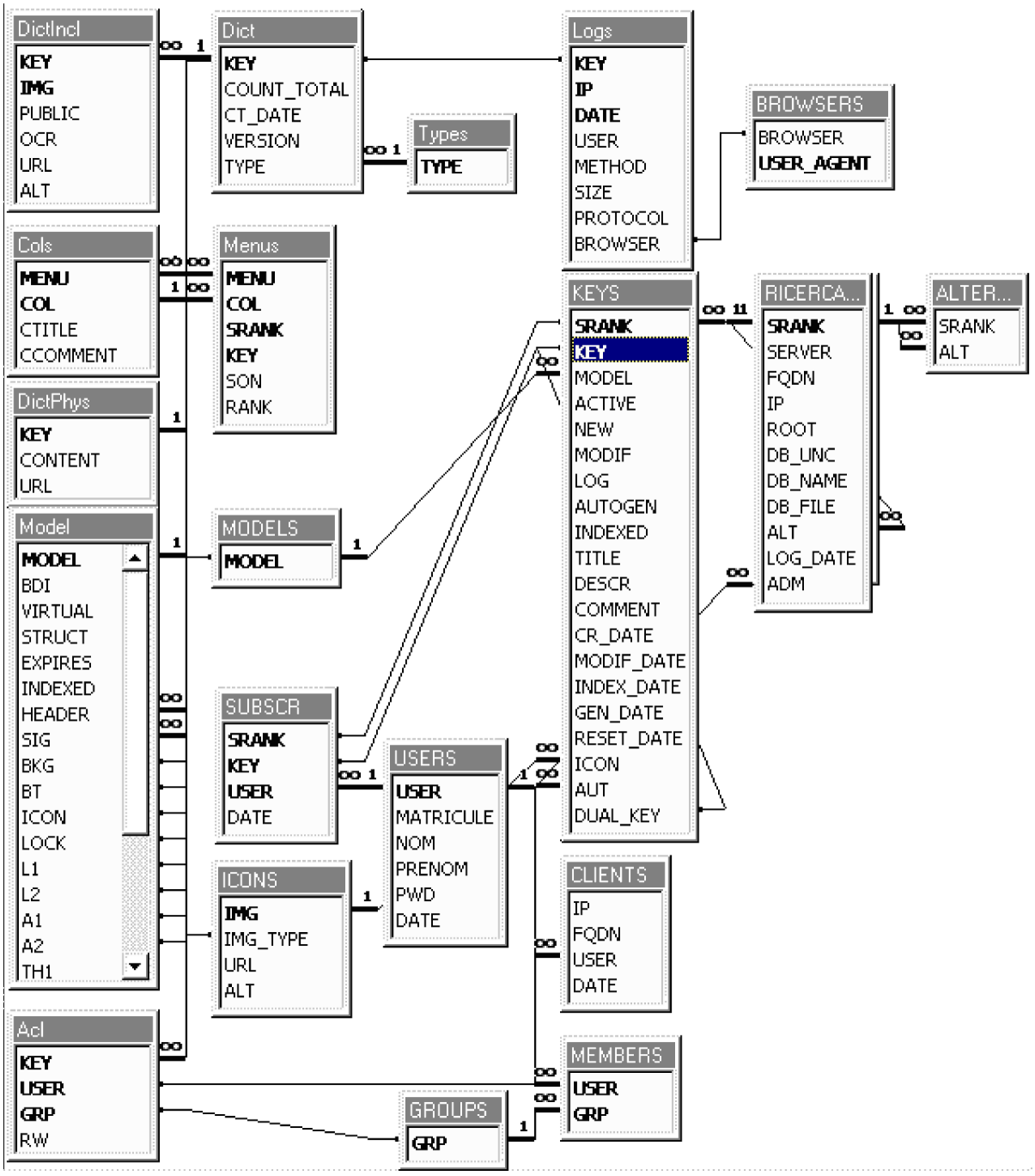


Figure 4. 5. Schéma physique de la base fédérée de RICERCAR. Rappel (cf. § 0.4.2, p.9) : les tables en MAJUSCULES sont répliquées sur l'ensemble des serveurs.

L'implémentation d'un concept objet, à partir d'un langage procédural qui interface des bases relationnelles, se révèle néanmoins délicate [Rumbaugh_96, Bouzeghoub_97]. Les représentations des objets et de leurs relations sont matérialisées par des tables et une classe est parfois réalisée à partir de relations entre plusieurs tables.

La nécessité de gérer un changement constant de paradigme peut, d'une part, s'avérer très coûteuse en raison des multiples jointures (si les bases sont correctement normalisées, *cf.* § 2.3 p.43) et , d'autre part, nécessiter de maintenir la sémantique de l'application dans des représentations différentes.

De nombreuses études ont néanmoins été menées pour accommoder des structures d'objets à des langages procéduraux ou à des formats de stockage relationnels d'autant que [Gardarin_89, Coad_91, Coulouris_95, Linthicum_97b] ont souligné l'intérêt d'une approche objet au niveau conceptuel, indépendamment du fait que l'implémentation soit ou non objet (X-Window est l'exemple d'un tel environnement, totalement écrit en C [Young_90]).

Cet enchevêtrement de niveaux d'interprétations simultanés entretient une confusion dans l'explication d'ensemble (ce chapitre notamment) et impose un surcroît de clarté et de structuration dans le style de programmation pour décrire l'implémentation des fonctionnalités naturelles des objets à partir d'une base des méta-données.

En pratique, le nombre restreint des classes ainsi que la relative simplicité du *modèle* d'un intranet nous ont cependant permis d'optimiser le placement des objets (notamment la possibilité offerte par *Cold Fusion* de stocker le résultat des jointures dans le cache du serveur, à la première exécution d'une requête) et de réduire les inconvénients d'une implémentation relationnelle de RICERCAR.

La reconstitution d'une classe d'objets se ramène effectivement à une vue de la base de données et les méthodes sont, selon le cas, des colonnes calculées ou des procédures écrites en CFML, qui procèdent à des calculs *ad hoc* (éventuellement d'autres requêtes SQL) en fonction de ces vues.

Ainsi, par exemple, une classe des *Parisiens* pourrait-elle être vue par :

```
<CFQUERY Name="Parisiens" DataSource="Source_ODBC">
  SELECT      NOM, PRENOM, ADRESSE, DATE_NAISSANCE
  FROM        PERSONNE
  WHERE       VILLE = 'Paris'
</CFQUERY>
```

Il serait ensuite possible de se référer à *Parisiens.nom, Parisiens.prenom* pour référencer les attributs correspondants, alors que la méthode *âge* devrait être calculée en fonction de l'attribut *date_naissance.*

L'envoi de messages (*cf.* paragraphe 4.5.2.1, p.108) est, quant à lui, réalisé par des redirections HTTP paramétrées :

```
<CFLOCATION URL="http://serveur/méthode?obj=objet?param=arg..>
```

L'héritage y est simulé « à la main » par agrégation et par délégation et l'encapsulation des données privées n'est assuré que par des conventions de nommage (nous verrons plus loin que CFML permet de gérer une réelle encapsulation).

Il est certain que la fragilité des classes représente la principale faiblesse d'une telle représentation, dans la mesure où le schéma est complètement dynamique, soumis à l'interprétation de requêtes SQL.

L'insertion d'un niveau d'interprétation supplémentaire (*cf.* § 4.5.3.1) permet cependant de rigidifier l'ensemble par l'introduction d'une couche fonctionnelle qui implémente le modèle et réalise une encapsulation qui masque la faiblesse d'expression du modèle objet originel.

4.5 Réalisation du modèle présenté

4.5.1 Conventions de représentation et de lecture

Nous supposons, pour la commodité de la notation et pour simplifier les descriptions à venir, qu'un ensemble de fonctions primitives sont connues des différents objets. Ces fonctions sont représentées ici dans un pseudo-code, inspiré du Lisp [23] :

- *(getobj 'un_string)* renvoie l'**objet** persistant dont l'identifiant est « un_string ».
  Cette fonction est par la suite indiquée sous la forme d'une macro *&*.
  Ainsi la notation *&un_string* est-elle équivalente à la fonction *(getobj 'un_string)* ;
- l'envoi de message est représenté par la notation suivante :
  (send <objet> <sélecteur> < arg1>...< argn>)
  où *<objet>* indique l'objet receveur, *<sélecteur>* est le sélecteur du message, et *< arg1>...< argn>* les différents arguments de cet envoi de message.

Concrètement, *<sélecteur>* est soit le nom d'une méthode de la classe d'*<objet>* (ou déléguée à la classe) soit le nom d'un attribut ;

- nous n'expliciterons pas ici le détail de certaines méthodes ou fonctions jugées "évidentes" et dont le nom, explicite, permet d'en saisir l'essence ;
- la fonction Lisp *defmethod* est utilisée pour décrire la définition d'une méthode.

4.5.2 Premiers tours de manivelle

Comme nous l'évoquons en préambule, le paradigme central autour duquel nous articulons la construction de RICERCAR est celui de l'intranet vu en tant qu'un système d'exploitation distribué orienté objet [Black_94], constitué ici des classes *Clients, Serveurs* et *Documents* ainsi que leurs sous-classes éventuelles.

Après amorçage, la représentation actuelle de RICERCAR confond dans une même représentation « Orientée-*Documents* » les documents des utilisateurs finals ainsi que des *expertises*, définies comme des documents capable d'opérer sur d'autres documents passés en argument, et de retourner une valeur (du code HTML ou encore, un script CFML, interprété à son tour en HTML), à la manière d'une procédure.

---

[23] Nous faisons là l'hypothèse que l'expression en Lisp des méthodes sera plus familière au lecteur que le script CFML correspondant.

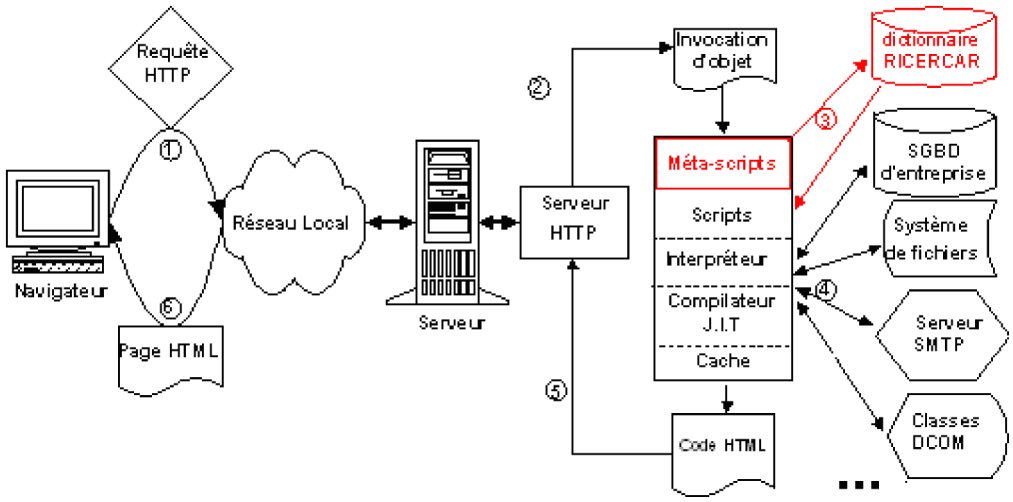


Figure 4. 6. L'introduction d'un niveau « méta » dans l'interprétation permet de représenter les méta-données de RICERCAR dans un « Référentiel d'Interfaces ».

Cependant, à la différence d'une procédure classique, une expertise est ici décrite en CFML, c'est-à-dire par un ensemble de règles SI... ALORS (les requêtes SQL n'ayant d'autre but que de réaliser l'instanciation des variables persistantes du méta-modèle).

Il nous a semblé important d'inviter le lecteur à nous accompagner pendant les premières itérations de la construction du système afin de lui fournir un fil conducteur qui aboutisse au modèle final.

4.5.2.1 Correspondance entre URL et envoi de message

l'URL d'un objet accessible par un serveur HTTP, qui est de la forme générale,

http://serveur/chemin?param0=arg0&param1=arg1&...argn

(*cf.* § 1.2.1.4, p.24), est trivialement unifiée à un envoi de message :

http://serveur/méthode?objet=obj&param1=arg1&...argn

que nous notons dans la suite *(send serveur 'méthode obj arg1... argn).*

4.5.2.2 Cohérence des liens d'un serveur

Il s'agit, dans un premier temps, de poser les premiers éléments d'amorçage du système :

Trois classes d'objets sont présentes :

1. les postes utilisateurs, *Clients*, spécifiés par leur adresse *IP* (et éventuellement le nom DNS de leur poste, *FQDN*) et disposant d'un navigateur capable d'interpréter le langage HTML (méthode *html*) ;

2. les *Serveurs* HTTP, spécifiés par leur adresse *IP* ou leur nom DNS (*fqdn*) ainsi que l'identifiant de la base de données du dictionnaire, *db_name.*

   L'interprétation du *goto* de *Serveurs*, ne fait qu'invoquer la méthode *goto* de la classe de l'argument reçu, en l'occurrence une instance de *Documents.*
   Elle est simplement définie par:

   (defmethod goto &Serveurs (obj)
   (send obj 'goto))

3. les *Documents* auxquels sont associées les *url* correspondantes. La méthode *goto* se contente de retourner la valeur de l'attribut *url*:

(defmethod goto &Documents ( )
(output (valof url)))

Ce schéma rudimentaire correspond trivialement au diagramme de classes suivant :

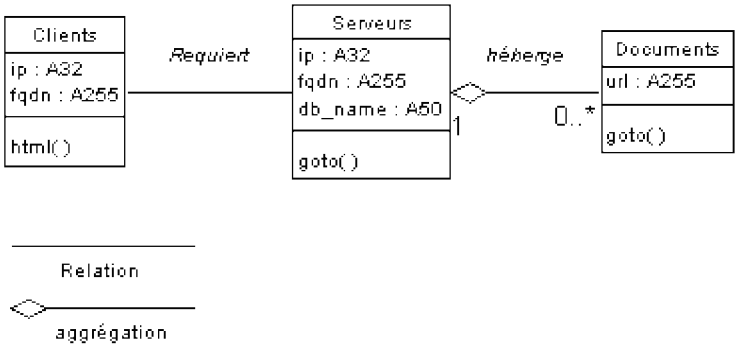


Figure 4. 7. Diagramme de classes initial : Clients, Serveurs et Documents.

La requête d'un document *doc* par un *client* peut ainsi être représentée par le diagramme de collaboration d'objets :

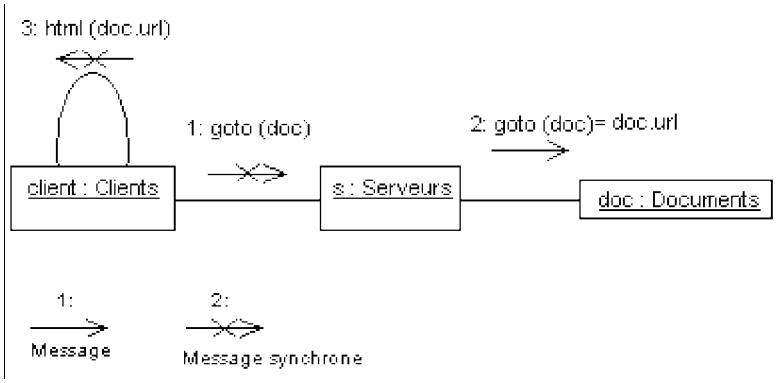


Figure 4. 8. diagramme de collaboration correspondant au traitement de l'appel d'un document doc.

L'implémentation concrète de ce schéma est assurée par la mise en place d'une table *Documents* comprenant deux colonnes, *key* (l'identifiant du document) et *url*.

La méthode *goto* de la classe *Documents* est alors décrite par le script CFML, *goto.cfm* (où *bd* est le nom de la base de données) :

```
<CFQUERY Name=  "goto" DataSource= "bd">
  SELECT       URL
  FROM         DOCUMENTS
  WHERE        KEY = 'doc'
</CFQUERY>

<CFLOCATION URL = "goto.url">
```

C'est ainsi que l'accès à une instance *doc* de *Documents*, à laquelle correspond le fichier *doc.html*, et jusqu'alors consultée par le biais de l'URL

<URL:http://s/chemin/doc.html>,

est désormais accessible par

<URL:http://s/goto.cfm?key=doc>.

L'association, dans la table *Documents*, d'un identifiant à toutes les URL (relatives ou absolues) référencées par le serveur HTTP assure l'accès à ces objets, indépendamment de leur localisation.

4.5.2.3 Génération automatique des index

La mise en place d'une classe de *Documents* inspire aussitôt la définition de méthodes qui procèdent au regroupement de ses instances selon des critères particuliers.

En l'occurrence, le fonctionnement d'un serveur HTTP « classique » qui regroupe thématiquement des documents HTML dans un fichier *index.html* (dont la maintenance est le cauchemar de tout administrateur normalement constitué) peut être totalement automatisé. Il suffit d'ajouter un attribut *thème* à *Documents* (éventuellement multi-valué) et de créer une méthode *goto-thème* qui permettra de récupérer l'ensemble des instances concernées :

```
(defmethod goto-thème &Serveurs (th)
      (letn boucle ((inst (send &Documents 'list-instances) ))
        (and  inst
              (if (memq  th (send (car inst) 'thème))
                  (output (send (car inst) 'url)) )
              (boucle (cdr inst)) ) ) )
```

De façon concrète, une colonne *theme* est ajoutée à la table *Documents* qui référence le thème auquel chaque document contribue.

La méthode *goto-thème* est alors implémentée par le script CFML qui suit :

```
<HTML>
<HEAD>
<TITLE>Liste des documents de thème <CFOUTPUT>"#theme#"
</TITLE>
</HEAD>
<BODY>
<CFQUERY Name=  "goto-theme" DataSource= "db">
  SELECT  KEY, URL
  FROM    DOCUMENTS
  WHERE  THEME = '#theme#'
</CFQUERY>

<H1>Liste des documents de thème
<CFOUTPUT>"#theme#"</CFOUTPUT></H1>
<UL>
  <CFOUTPUT Query="goto-theme">
     <LI><A HREF="#url#">#key#</A> (#url#)
  </CFOUTPUT>
</UL>
</BODY></HTML>
```

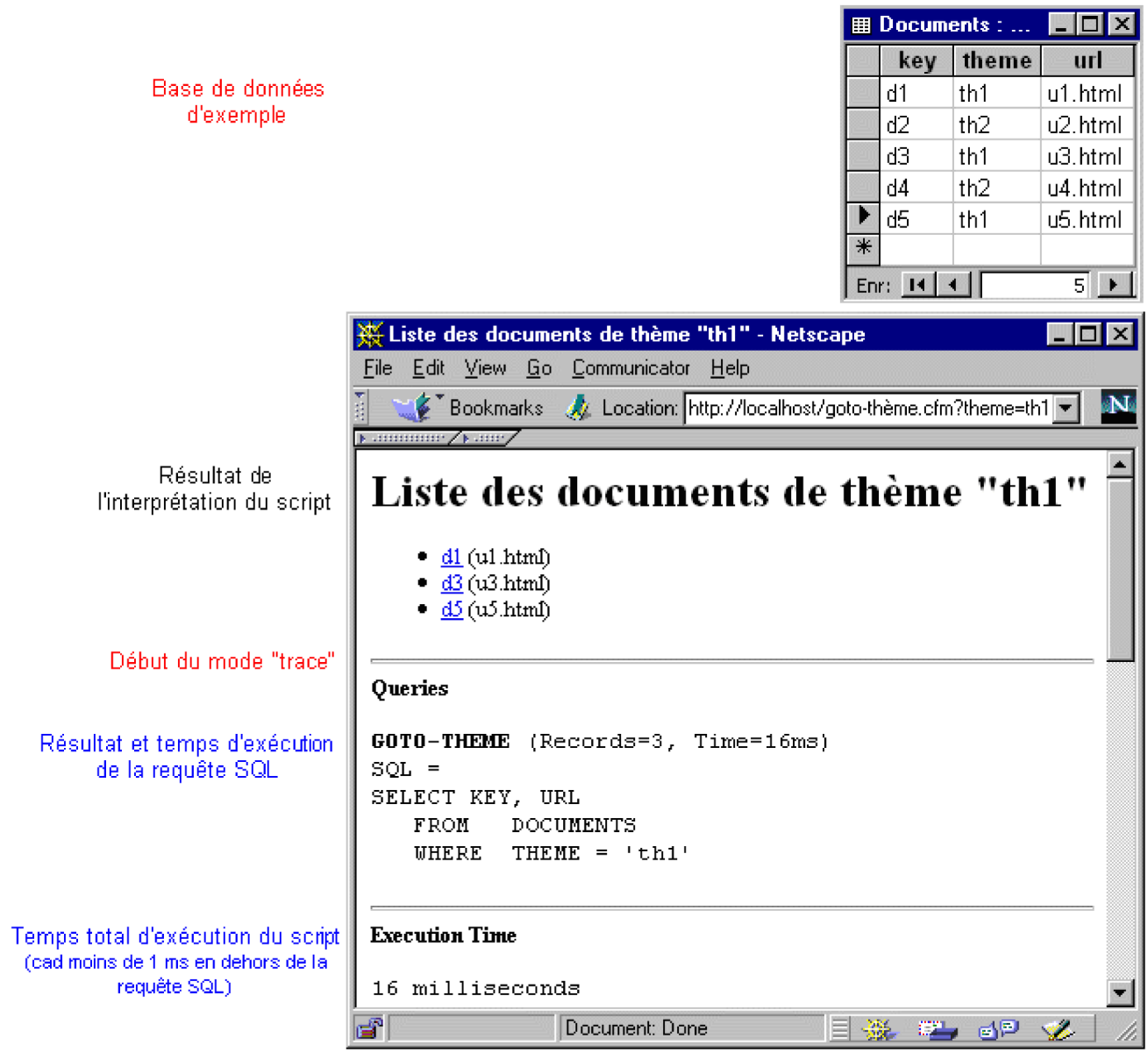


Figure 4. 9. Exécution de "(send localhost 'goto-thème 'th1)" en mode trace, à partir d'une base d'exemple.

4.5.2.4 Généralisation de la démarche

Un examen de la solution esquissée laisse cependant entrevoir une série de sources d'incohérence et de complexité ultérieures. L'implémentation décrite nécessite en effet de modifier le schéma des classes et d'ajouter de nouvelles méthodes à la classe *Serveurs* à chaque nouvelle fonctionnalité. Ces modifications, qui entraînent celle de la structure de la méta-base impliquent une maintenance importante du code des requêtes ainsi que la fragilisation de la structure d'ensemble.

La généralisation de la démarche de réification des documents HTML finals (les feuilles de l'arborescence), met cependant en évidence la possibilité de créer une classe supplémentaire. Cette classe, *Menus*, correspond aux nœuds de l'arbre d'un serveur ou à tout autre fichier (*index.html*) qui regroupe thématiquement des éléments du sous-arbre.

Le diagramme des classes doit alors être adapté :

- la classe *Documents* est promue au rang de super-classe ;
- elle se spécialise en *Pages* (qui récupère l'ancienne définition de la méthode *goto* de *Documents*) et *Menus*;
- l'association entre *Menus* et *Pages* indique qu'une instance de *Menus* référence un ensemble d'instances de *Pages*.

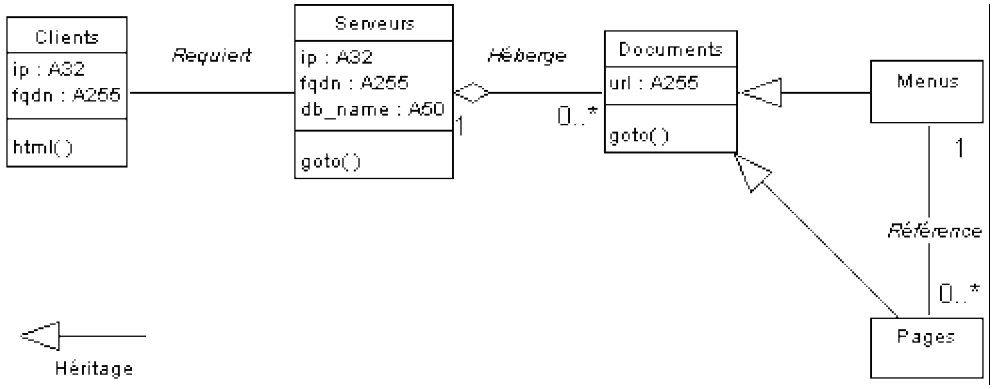


Figure 4. 10. Documents se spécialise en Menus et Pages.

Le diagramme de collaboration qui schématise l'appel d'une instance de *Menus* reste, quant à lui, semblable à celui exprimé précédemment.

De façon concrète, l'implémentation de la classe *Menus* se traduit par la définition d'une deuxième table dans la base de données, *Menus*, qui comporte deux colonnes, *menu* et *key*. Cette table référence autant de tuples qu'il y a d'éléments dans un *menu* donné.

La méthode *goto* de *Menus* est redéfinie pour extraire la liste des éléments d'un *menu* :

```
(defmethod goto &Menus ()
  (letn gen-list (( x  (send self 'liste-références) ))
            (and x
                    (output (send (car x) 'url))
                    (gen-list (cdr x)) )) )
```

Elle est effectivement implémenté en CFML par le code suivant :

```
<HTML>
<HEAD>
<TITLE>Liste des items du menu <CFOUTPUT>"#key#"</CFOUTPUT>
</TITLE>
</HEAD>
<BODY>

<CFQUERY Name=  "goto" DataSource= "db">
  SELECT       PAGES.KEY AS KEY, URL
  FROM         MENUS, PAGES
  WHERE        MENUS.MENU = PAGES.KEY
  AND          MENUS.KEY = '#key#'
</CFQUERY>

<H1>Liste des items du menu
<CFOUTPUT>"#key#"</CFOUTPUT></H1>

<UL>
  <CFOUTPUT Query="goto">
     <LI><A HREF="#url#">#key#</A> (#url#)
  </CFOUTPUT>
</UL>

</BODY>
</HTML>
```

Une nouvelle incohérence apparaît alors dans cette implémentation qui ne représente qu'un cas particulier d'un *menu* qui ne peut référencer que des *Pages*. Il n'est ainsi pas possible, de référencer un menu dans un autre, ce qui exclut de construire dynamiquement l'arborescence entière du serveur

Par ailleurs, l'ajout de toute sous-classe de *Documents* nécessiterait de modifier le code de la méthode *goto* de *Menus* en y incluant un sélecteur de classes.

La généralisation du diagramme des classes suffit cependant pour indiquer que les *Menus* peuvent référencer l'ensemble des *Documents*, donc toutes leurs sous-classes.

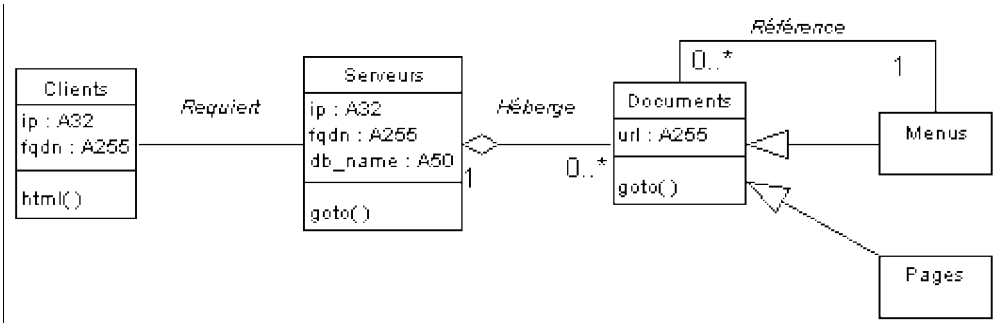


Figure 4. 11. Généralisation de la composition des Menus.

La méthode *goto* est elle aussi adaptée pour traiter le cas général en appelant récursivement la méthode *goto* pour tous les *Documents* référencés par un menu :

```
(defmethod goto &Menus ()
  (letn gen-list (( x  (send self 'liste-références) ))
              (and x
                        (output (send (car x) 'goto))
                        (gen-list (cdr x)) )) )
```

Cette modification apparaît dans le code CFML par :

```
...(même code que précédemment)


<CFQUERY Name=  "goto" DataSource= "db">
  SELECT    DOCUMENTS.KEY AS KEY, URL
  FROM      MENUS, DOCUMENTS
  WHERE     MENUS.MENU = DOCUMENTS.KEY
  AND       MENUS.KEY = '#key#'
</CFQUERY>

<H1>Liste des items du menu
<CFOUTPUT>"#key#"</CFOUTPUT></H1>

<UL>
  <CFOUTPUT Query="goto">
     <LI><A HREF="goto.cfm?key=#key#">#key#</A> (send #key# 'goto)
  </CFOUTPUT>
</UL>...
```

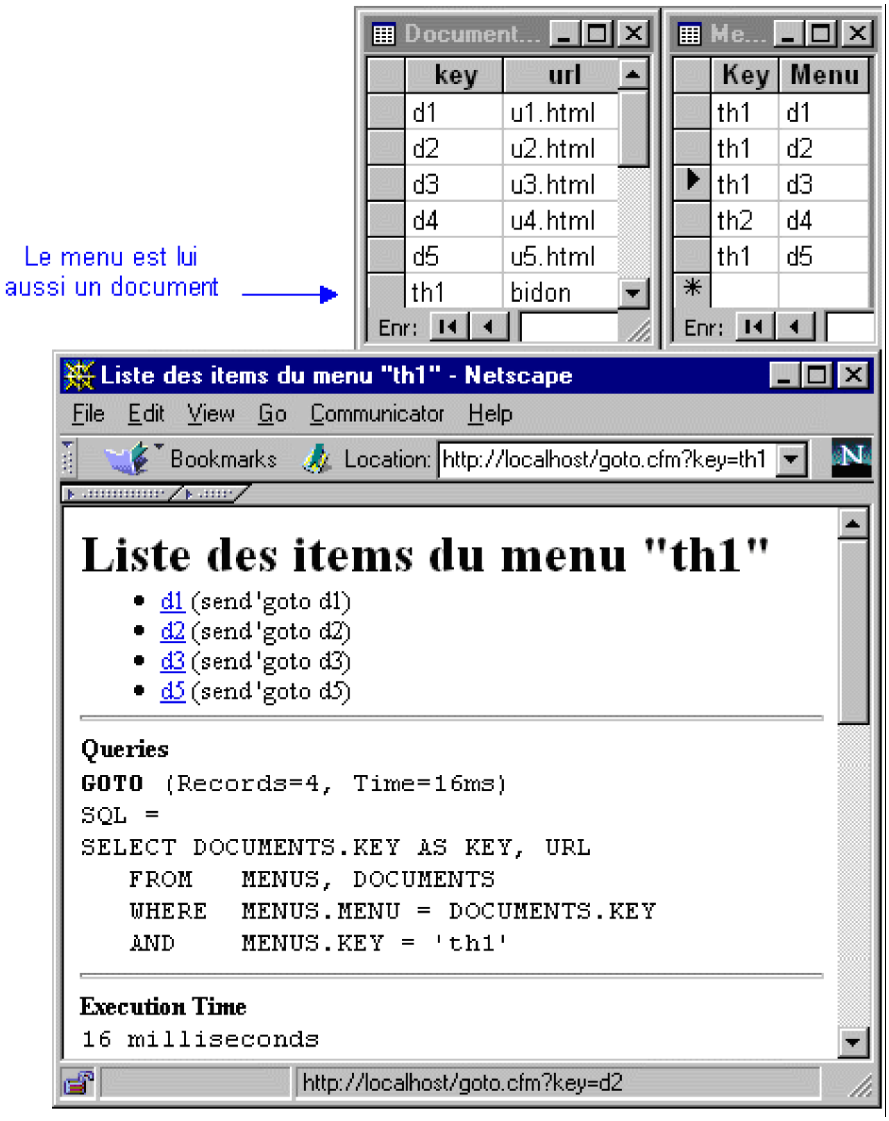


Figure 4. 12. Exécution en mode trace de "(send m1 'goto)" qui génère désormais un appel récursif pour chaque élément du menu.

4.5.3 Le modèle

4.5.3.1 Méta-interprétation

Les itérations précédentes montrent cependant que la classe *Serveurs*, qui est obligatoirement réceptrice des messages des utilisateurs (puisqu'une requête HTTP est émise par un client à un serveur), n'en est **jamais** le destinataire final.

Aussi, plutôt que de définir au niveau de *Serveurs* une pléthore de méthodes (à mettre à jour incessamment) pour relayer les messages vers *Documents* et ses sous-classes, avons-nous remplacé ces méthodes par une seule méta-méthode (une fonction directement programmée dans la classe) qui analyse le message reçu, l'interprète et, par défaut, le réémet vers la classe de l'objet passé en paramètre :

```
(de Sys_Application (m o . arg)
  (send o m arg) )
```

Cette démarche, classique dans l'écriture d'un interpréteur dans son propre langage[24], offre ainsi la possibilité de différencier les messages en implémentant, si besoin, un système de filtrage. Elle revient notamment à considérer cette méthode comme le *toplevel* d'un méta-interpréteur, qui intercepte et traite l'ensemble des interactions entre les *Clients* et les *Documents*.

L'implémentation de cet interpréteur au niveau de la classe *Serveurs* est d'autant plus intéressante que sa répartition effective, sur l'ensemble des instances de *Serveurs*, permet de réaliser simplement les procédures de routage entre serveurs :

```
(de Sys_Application (m o . arg)
   (let ((s (send o 'hébergé-par) ))      ; ; retrouve le serveur hébergeant 'o'
      (if (eq self s)                     ; ; si 'o' est local (même serveur)
         (send o m arg)                   ; ; alors exécuter la méthode
         (send s m (cons o  arg)) )) )    ; ; sinon réémettre au bon serveur
```

La version actuelle (un peu plus sophistiquée) de cette fonction *Sys_Application* est concrètement définie en CFML par le script *application.cfm*.

Son invocation à chaque requête d'utilisateur explique qu'elle ait fait l'objet d'optimisations particulières dans la mesure où l'interpréteur qu'elle réalise est effectivement chargé de servir ou de relayer dans les plus brefs délais un nombre indéterminé d'accès simultanés.

[24] *Cf.* l'implémentation minimaliste d'un méta-interpréteur Lisp « *(dfe Meta-eval (exp) exp)* » ou encore les *vtables* en C++ qui sont des tables de pointeurs vers les implémentations des méthodes.

4.5.3.2 Enrichissement des classes

L'étape qui suit consiste simplement à spécifier plus finement les propriétés des différents éléments, à partir du modèle déjà décrit. Une série de nouveaux attributs [25] ainsi que des classes supplémentaires sont définis, dont la combinaison va permettre d'offrir une meilleure ergonomie à l'utilisateur final ainsi qu'une simplification dans la conception des documents publiés :

- La classe *Serveurs* permet de répertorier l'ensemble des informations nécessaires pour référencer un nouveau serveur sur l'intranet.

  Serveurs est implémenté par les tables *RICERCAR* et *ALTERNATE*.

Attributs	**Type**	**Description**	**Contrainte/ Défaut**
srank	I *(Integer)*	Identifie de façon unique le serveur. Automatiquement préfixé à l'identifiant des objets crées par les utilisateurs (ce qui permet de garantir l'unicité des identifiants sur l'ensemble de l'intranet).	
server	A50 *(chaîne de 50 caractères)*	le nom logique du serveur.	
s_alt	*Servers*	serveur de substitution	non utilisé
fqdn	A255	le nom DNS du serveur.	
ip	A32	L'adresse IP du serveur	
root	A255	Le chemin physique correspondant à la racine du serveur.	
db_unc	A50	Le chemin complet vers la base de données du serveur	
db_name	A50	Le nom de la base de données du serveur.	
db_file	A50	Le nom (éventuel) du fichier de la base de données du serveur.	
log_date	DT *(DaTe)*	Date à laquelle les *log* du serveur ont été centralisées.	
adm	*Users*	L'administrateur du serveur	

[25] Nous avons choisi de nommer les attributs en anglais pour des raison pratiques (de traduction en vue de publications internationales) mais aussi pour rester homogène avec les nommages HTTP et HTML habituels.

- La classe *Documents* est ainsi enrichie de différents attributs qui décrivent, par des textes brefs, les objets référencés. Une série d'indicateurs booléens et différentes dates (création, modification, *etc.*) fournissent de plus les éléments nécessaires pour que la modification des méthodes *goto* permettent de générer des *Documents* plus sophistiqués :

Attributs	Type	Description	Contrainte/Défaut
key	A50	Identifiant global de l'objet.	Unique, sauf si *(eq (send (send self 'isa) 'type) 'SYSTEM)* [26]
active	BL *(BooLéen)*	Indique si le document correspondant est disponible ou pas (permet d'invalider provisoirement l'accès).	Vrai
new	BL	Indique que le document mérite d'être signalé dans les nouveautés du serveur	Faux
modif	BL	Indique que chaque modification du document mérite d'être signalée	Faux (n'a de sens que si (*send self 'new*).
Indexed	BL	Indique si le contenu du document correspondant doit être indexé.	*(send (send self 'isa) 'indexed)*
log	BL	Indique si chaque accès à l'objet doit être enregistré dans les *logs* du serveur	Vrai
title	A255	Titre du document (utilisé par la méthode ... comme ancre au lien correspondant au document).	Obligatoire.
descr	A255	Description brève du contenu du document	
comment	Avar *(chaîne de caractères de longueur VARiable)*	commentaire supplémentaire pour décrire le contenu du document.	
cr_date	DT	date de création de l'objet	*(Now)*

[26] Les méthodes sont de type SYSTEM et sont donc répliquées sur chacun des serveurs.

Attributs	Type	Description	Contrainte/Défaut
modif_date	DT	date de la dernière modification	
index_date	DT	date d'indexation (éventuelle).	n'a de sens que si *(send self 'indexed)*
reset_date	DT	date de remise à zéro du compteur d'accès affichable du document	
icon	*Icons*	Identifiant de l'icône associée au document.	*(send (send self 'isa) 'icon)*
dual_key	*Documents*	Identifiant (éventuel) d'un document dual associé au document. Il référence notamment une traduction.	
count_total	I	Nombre d'accès total au document à la date *ct_date*	Ne peut être remis à zéro
ct_date	DT	Date à laquelle *count_total* a été calculé	
version	I	Numéro de version du document. Exploité pour gérer les accès à des versions successives d'un même document	Utilisé en affichage uniquement.
content	HTML	code HTML du document. Stocké directement dans le champ de l'objet.	N'a de sens que si *(eq (send (send self 'isa) 'type) 'VIRTUAL)*
url	A255	chemin relatif pour accéder au code HTML du corps du document	Si *(eq (send (send self 'isa) 'type) 'WWW)* alors *self.url* est une URL, sinon c'est le chemin relatif d'un fichier
aut	*Users*	Le propriétaire de l'objet	

Documents, ainsi que ses différentes sous-classes, sont implémentées par les tables *KEYS*, *Dict*, *Dictphys* et *Dictincl* ;

- il a par ailleurs semblé intéressant de subdiviser un menu en rubriques (*columns* en anglais), ce qui est rendu possible par la création de la classe *Rubriques* qui regroupe un ensemble ordonné de *Documents*. Elle est ensuite agrégée dans *Menus* :

Attributs	Type	Description
ctitle	A255	Titre de la rubrique
ccomment	Avar	Texte de commentaire descriptif
rank	I	Ordre de la rubrique dans son *Menus*

  La classe *Rubriques* est alors implémentée par la table *Cols* ;

- à partir de la classe *Images*, un ensemble de sous-classes permettent de réifier les différents éléments graphiques globaux aux différents serveurs, les rendant ainsi accessibles par l'ensemble des *Documents.*

  Elles représentent les icônes ou puces associées aux *Documents,* les différents fonds de page, les lignes de séparation stylisées et les flèches utilisées pour indiquer les retours au sommaire, précédent, suivant, *etc*.

Attributs	Type	Description
img	A50	Identifiant de l'image
url	A255	URL (ou chemin relatif) de l'image
alt	Avar	Texte ALTernatif décrivant l'image
tsize	I	Taille de l'image en octets (fournie par l'OS *via* CF)
tdate	DT	Date de création de l'image (fournie par l'OS *via* CF)

  Ces différentes images sont créées et chargées dans le cache du serveur lors de l'exécution de la première requête d'utilisateur, et sont ensuite directement accessibles dans l'ensemble des méta-scripts.

  Les classes *Icons*, *Bkg*, *Lines*, *Arrows* et *Images* sont implémentées par la table *ICONS* ;

- il s'avère enfin nécessaire de représenter plusieurs types de *Documents* qui justifient la création de nouvelles sous-classes de *Documents* (décrites en § 4.5.3.5, p.127) : *Sty_Post*, *Sty_Pages-en*, *Sty_Img*, *Sty_www*, *etc*.

4.5.3.3 Modèles : identification entre classe et méta-objet

Le souci d'homogénéiser la présentation des documents finals dans le respect d'une charte graphique d'entreprise, induit une régularité des documents.

C'est ainsi qu'une structure commune apparaît, au fur et à mesure de la création des nouvelles sous-classes de *Documents* (présence et placement d'informations semblables aux mêmes endroits : en-têtes, signatures, éléments de navigation - flèches, logos, puces, *etc*.) à laquelle viennent se rajouter des marqueurs imposés par la norme HTML (balises de début et fin, méta-informations, *etc*.).

Le modèle décrit jusqu'ici ne propose aucun mécanisme pour assurer une factorisation de ces éléments qui sont ainsi répliqués dans l'ensemble des méthodes.

Leur régularité nous a cependant permis de procéder au regroupement des éléments de structure par la définition d'une métaclasse, *Modèles*, qui va répertorier les différentes propriétés de ces *Documents* et permettre ainsi de procéder à une typologie de ces méta-informations.

Attributs	**Type**	**Description**	**Contrainte/ Défaut**
model	A50	Nom de la sous-classe de *Documents*	
type	A50	Type du document	
bdi	BL	Indique si le document est un objet RICERCAR ou un document HTML standard	Vrai
virtual	BL	Indique si le code HTML du corps du document est stocké dans l'attribut *content*.	Faux
struct	BL	Indique si l'objet correspond à un index du serveur.	Faux. N'a de sens que si *(eq (send (send self 'isa) 'type) 'META)*
expires	BL	Indique si l'en-tête HTTP du document généré doit empêcher le stockage dans un cache (pragma : no-cache)	Faux
indexed	BL	Indique si les méthodes d'indexation s'appliquent à cet objet.	Vrai
header	*Pages*	Instance du *Documents* décrivant l'en-tête de la classe.	
sig	*Pages*	Instance du *Documents* décrivant sa signature	
bkg	*Bkg*	Instance de l'image de fond	

Attributs	Type	Description	Contrainte/ Défaut
icon	*Icons*	Instance de l'image d'icône	
bt	*Icons*	Instance de l'image de bouton standard	
lock	*Icons*	Instance de l'image de « verrou »	
a1	*Arrows*	Instance de l'image de « flèche retour »	
a2	*Arrows*	Instance de l'image de « flèche Home »	
l1	*Lines*	Instance de l'image de « ligne1 »	
l2	*Lines*	Instance de l'image de « ligne2 »	
th1	A255	Style utilisé pour un titre HTML de niveau 1	non utilisé
th2	A255	Style utilisé pour un titre HTML de niveau 2	non utilisé
th3	A255	Style utilisé pour un titre HTML de niveau 3	non utilisé
th4	A255	Style utilisé pour un titre HTML de niveau 4	non utilisé
th5	A255	Style utilisé pour un titre HTML de niveau 5	non utilisé
th6	A255	Style utilisé pour un titre HTML de niveau 6	non utilisé
p	A255	Style utilisé pour un paragraphe HTML	non utilisé

Modèles est alors implémentée par les tables *MODELS, Model* et *Types.*

Le schéma de classe résultant est ainsi le suivant (la description des différentes sous-classes de *Documents* est fournie au paragraphe suivant. Le préfixe *Sty_* est omis pour simplifier les schémas) :

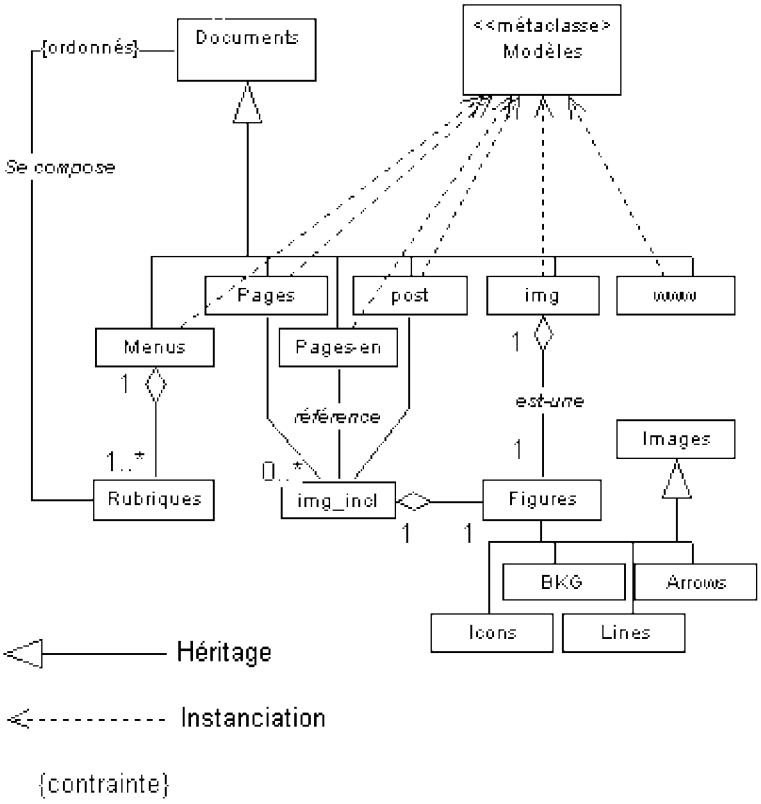


Figure 4. 13. Le paramétrage des variables de classe de Modèles, permet d'instancier les différentes sous-classes de Documents

4.5.3.4 Uniformisation de la représentation des données

Une dernière dichotomie persiste cependant après l'instanciation de la métaclasse *Modèles*. Deux ensembles sont côte à côte : d'une part les *Documents* (des utilisateurs) qui servent à animer le serveur HTTP et, d'autre part, les classes et les scripts CFML qui, somme toute, ont la même structure que ces documents.

Il est en effet aisé de se représenter qu'un script CFML a un nom, peut être décrit par un texte bref, est regroupé avec des scripts similaires, possède un auteur, connaît des modifications, peut être d'accès protégé, *etc*. Il est de surcroît considéré par une catégorie particulière d'utilisateurs, que sont les programmeurs et concepteurs du système, exactement de la même façon qu'un texte de réglementation ou qu'une image sont consultés par un utilisateur « normal ».

Un amorçage structurel permet toutefois d'uniformiser cette représentation. Il s'inspire du modèle général *ObjVlisp* [Cointe 85], où la classe *Documents* joue le rôle de la classe *Object* et *Modèles* celles de la classe *Class.*

Cet amorçage consiste à représenter les différentes classes par des instances de *Documents*, en leur associant comme « contenu » **le code des méthodes** *goto* correspondantes, renommées alors, par commodité, du nom de la classe qu'elles représentent.

Finalement, les instances de *Modèles* jouent trois rôles :

1. ce sont des classes de *Documents* qui vont pouvoir être instanciées par des *Documents* finals ;
2. ce sont elles-mêmes des *Documents* qui peuvent être consultés et générés comme n'importe quelle instance de *Documents* ;
3. elles pointent dans leur champ *url* vers le code des méthodes (les anciennes méthodes *goto*) exécutées pour générer les balises HTML renvoyées au navigateur de l'utilisateur.

C'est ainsi que, par exemple, la classe *Sty_Page*, est sous-classe de *Documents*. Elle est aussi instance de *Modèles* (une autre sous-classe de *Documents*). Sa méthode *goto* (le code qu'elle exécute) est alors identifié au document *Sty_Page* lui-même (c'est à dire concrètement, que ce code exécutable se trouve référencé dans le champ *url* du document *Sty_Pages*).

Le schéma de classes qui suit indique bien l'uniformisation de la représentation. Il est complété des classes d'utilisateurs ainsi que de leurs relations :

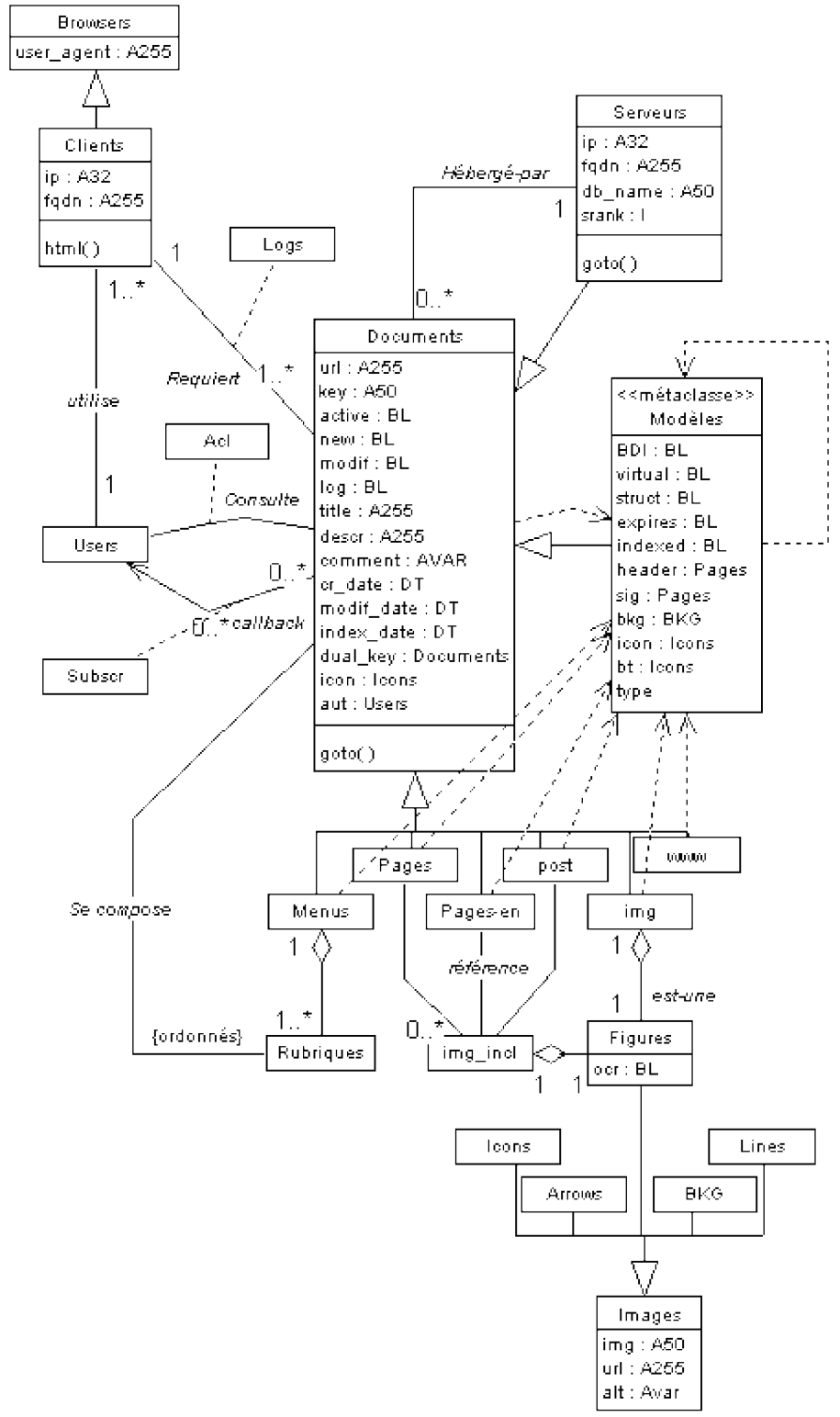


Figure 4. 14. Diagramme de Classes de RICERCAR.

4.5.3.5 Les sous-classes de Documents

Différentes sous-classes de *Documents* sont adaptées pour instancier la classe *Modèles* (*cf.* <*RURL:ricercar_pgm*>, pour le code des méthodes correspondantes) :

Model : Table

MODEL	BDI	VIRTUAL	STRUCT	EXPIRES	INDEXED	HEADER	SIG	BKG	BT	ICON	LOCK	L1	L2	A1	A
sty_goto	☑	☐	☐	☐	☐			ricercar_bkg		ico_ricerc	lock				
sty_header	☑	☐	☐	☐	☐			nuages	bt_bleu	ico_ricerc	lock				
sty_img	☑	☐	☐	☐	☑	Sty_Header	Sty_Signe	ricercar_bkg	bt_bleu	bt_ratp	lock	ligne	ligne	anho	ap
sty_menu	☑	☐	☑	☐	☑	Sty_Header	Sty_Signe	ricercar_bkg	bt_bleu	ico_ricerc	lock	ligne	ligne	anho	ap
sty_meta	☑	☐	☐	☐	☑	Sty_Header	Sty_Signe	formule	bt_bleu	ico_ricerc	lock	ligne	ligne	anho	ap
sty_page	☑	☐	☐	☐	☑	Sty_Header	Sty_Signe	ricercar_bkg	bt_bleu	ico_ricerc	lock	ligne	ligne	anho	ap
sty_page_en	☑	☐	☐	☐	☑	Sty_Header	Sty_signe_en	ricercar_bkg	bt_bleu	ico_ricerc	lock	ligne	ligne	anho	ap
sty_pagenph	☑	☐	☐	☑	☑	Sty_Header	Sty_Signe	ricercar_bkg	bt_bleu	ico_ricerc	lock	ligne	ligne	anho	ap
sty_post	☑	☐	☐	☑	☐	Sty_Header	Sty_Signe	ricercar_bkg	bt_bleu	ico_ricerc	lock	ligne	ligne	anho	ap
sty_raw	☑	☐	☐	☐	☐	Sty_Header	Sty_Signe	ricercar_bkg	bt_bleu	bt_ratp	lock	ligne	ligne	anho	ap
sty_signe	☑	☐	☐	☐	☐			nuages	bt_bleu	ico_ricerc	lock				
Sty_tree	☑	☐	☑	☐	☑	Sty_Header	Sty_Signe	ricercar_bkg	bt_bleu	relations	lock	ligne	ligne	anho	ap
sty_vpage	☑	☑	☐	☐	☑	Sty_Header	Sty_Signe	ricercar_bkg	bt_bleu	ico_ricerc	lock	ligne	ligne	anho	ap
sty_www	☐	☐	☐	☐	☑	Sty_Header	Sty_Signe	ricercar_bkg		ico_ricerc	lock	ligne	ligne	anho	ap
	☑	☐	☐	☐	☐				bt_bleu	ico_ricerc	lock				

Enr: 14 sur 14

Figure 4. 15. Les différentes classes de Documents dans le détail (ici la table Model).

- *Sty_goto* est une classe particulière. Elle ne possède aucune instance et sa méthode générique (le fichier indiqué dans le champ *url* du *Documents* correspondant) ne présente qu'un seul intérêt : invoquer le méta-interpréteur *Sys_Application*.
- *Sty_Page* est la classe par défaut des documents. Ses instances se composent d'une en-tête, d'un texte signataire, sont mémorisées par le cache (n'expirent pas), sont indexées et utilisent les éléments graphiques standards ;
- *Sty_Page_en* est identique à *Sty_Pages* mais est en anglais ;
- *Sty_Post*, identique à *Sty_Pages*, sert à instancier les formulaires qui, à ce titre, ne doivent pas être mémorisés par les différents caches ;
- les *Sty_Pagenph* (pour *n*on *p*arsed *h*eader) sont des *Sty_Post* qui ne sont pas indexés ;
- les *Sty_Vpage* sont des *Sty_Page* virtuelles, dont le contenu est lui-même stocké dans la méta-base ;
- les *Sty_Img* sont des *Documents* dégénérés, réduits à des images ;
- les *Sty_Menu* décrivent un nœud de l'arborescence du serveur ;

- *Sty_Tree* est de même nature que *Sty_Menu*. Seule la méthode générique de construction diffère, qui permet de construire récursivement l'arborescence du serveur à partir du nœud indiqué (méthode interfacée avec une *applet* Java de construction d'arbre) ;
- *Sty_www* permet de regrouper un ensemble de documents disponibles sur l'Internet (le champ URL référence une URL HTTP complète). Il est ainsi possible d'utiliser cette classe en conjonction avec un pare-feu pour limiter les accès des utilisateurs à l'Internet aux seuls *Documents* dont la classe est *Sty_www* ;
- *Sty_header* est la classe des différents *Documents* d'en-tête de *Documents* (une instance de *Documents* se composant d'un *Document* d'en-tête, d'un corps et d'un *Documents* signataire) ;
- *Sty_Signe* est la classe des *Documents* signataires ;
- *Sty_raw* référence des documents dans des formats non reconnus par un navigateur (des documents Word, Excel, *etc*.). La dénomination *raw* indique que ces *Documents* sont envoyés sous forme *brute* pour être traités par l'utilisateur en fonction des applications installées sur son poste.

4.5.3.6 Les classes non « Documentées [27] »

- *Figures* est une sous-classe d'*Images* et correspond aux différentes images (figures) entrant dans la composition des *Documents.*

  La présence de l'attribut *ocr* est en prévision de l'association d'un outil d'*OCR* pour procéder à la reconnaissance de caractères éventuels d'une *Figures* et permettre ainsi, lors de l'indexation du *Documents*, d'indexer aussi le texte reconnu.

Attributs	Type	Description	Contrainte
img	A50	Identifiant de l'image	
public	BL	Indique si la figure est "publique"	non utilisé
ocr	BL	Indique si la figure a été OCRisée	
url	A255	URL (ou chemin relatif) de l'image	
alt	Avar	Texte ALTernatif (L'OCR de l'image s'il existe)	

[27] Il faut comprendre *Documenter* comme étant l'action de « transformer en (instance) sous-classe de *Documents* ».

Toute recherche ultérieure par mots-clefs pourra ainsi retrouver les *documents* à partir d'informations figurant dans le texte du *Documents* ainsi que dans les *Figures.*

La classe *Figures* est concrètement implémentée par la table *Dictincl*

- La classe *Users* décrit les utilisateurs identifiés, disposant d'un compte sur l'intranet.

  Elle est implémentée par la table *USERS*.

Attributs	Type	Description	Contrainte / Défaut
user	A10	Identifiant normalisé de l'utilisateur	pour la RATP
matricule	I	matricule de l'utilisateur	pour la RATP
nom	A50	nom de l'utilisateur	
prenom	A50	prénoms de l'utilisateur	
pwd	I	*Check code* du mot de passe de l'utilisateur	
date	DT	date de création	

- La classe *Groups* définit les groupes d'utilisateurs de l'intranet.

  Elle est implémentée par les tables *GROUPS* et *MEMBERS.*

- La classe *Clients* instancie les différents postes de travail se connectant à l'intranet (ainsi que les *Users* correspondants).

  Elle est implémentée par la table *CLIENTS.*

Attributs	Type	Description	Contrainte/ Défaut
ip	A32	adresse IP du poste connecté	
fqdn	A255	nom DNS du poste	si applicable
user	*Users*	identifiant de l'utilisateur	si identifié
date	DT	date de première connexion	

- *Acl* est une classe-association qui regroupe les droits éventuels (consultation, modification) des *Users* ou *Groups* associés aux *Documents*.

  Elle est implémentée par la table *Acl*.

- *Subscr* est une classe-association qui permet à une instance de *Users* de s'abonner à une instance de *Documents.* Chaque mise à jour de l'instance de *Documents* déclenche alors une méthode qui va émettre un message pour informer *Users* des modifications intervenues.

  La classe *Subscr* est implémentée par la table *SUBSCR*.

- *Logs* est aussi une classe-association dont chaque instance décrit un accès à une instance de *Documents*.

  Elle est implémentée par la table *Logs.*

Attributs	Type	Description
key	*Documents*	Identifiant du *Documents* accédé.
ip	A32	adresse IP du poste connecté
date	DT	date d'accès
user	*Users*	identifiant de l'utilisateur
method	A50	Méthode HTTP utilisée (GET, POST, HEAD...)
size	I	Taille du fichier transmis
protocol	A50	Le protocole de communication employé (i.e. HTTP/1.0)
browser	*Browsers*	navigateur utilisé

- Enfin, la classe *Browsers* répertorie les différentes instances de navigateurs s'étant manifestés sur l'intranet.

Elle est implémentée par la tables *BROWSERS.*

Attributs	Type	Description
browser	I	Identifiant du navigateur.
User_agent	A100	Texte d'identification renvoyé par la variable normalisée HTTP_USER_AGENT

4.6 Caractéristiques

4.6.1 Les principales méthodes

4.6.1.1 L'interpréteur

Chaque serveur dispose de **trois** règles de réécriture :

1. *http://serveur/* → *http://serveur/cft/goto.cfm?key=serveur*
2. */ricercar/** → */cft/goto.cfm?key=**
3. ***/meta-ricercar/** → **/cft/meta.cfm?key=*** [28]

Une requête d'utilisateur est émise sous la forme d'une URL relative (elle permet d'accéder à l'objet quel que soit le serveur requis) dont la forme générale est *<RURL:objet>*.

À sa réception, cette URL est automatiquement transformée par le serveur, en */cft/goto.cfm?key=objet*

L'appel de *goto.cfm* lance la boucle de méta-interprétation *Sys_Application* (qui invoque à son tour *Sys_Declare*) :

```
(de Sys_Application (server method doc user)
  (and (send server 'Sys_Declare)
       (ifn doc
        (error-need-key)
        (ifn (exist doc)
             (error-not-exist doc)
             (ifn (eq server (send doc 'hébergé-par))
                  (Sys_Application (send doc 'hébergé-par) doc)
                  (ifn (send doc 'active)
                       (error-not-active doc)
                       (ifn (Sys_Acl doc user)
                            (error-forbidden doc user)
                            (let ((c (or (neq 'goto method)
                                         (send doc 'isa) ) ))
                              (progn (send server 'Sys_logs)
```

[28] Il est donc **toujours** possible d'accéder à la méthode de classe d'un objet (ou au méta-objet) en invoquant la méthode *Sty_Meta* (*meta.cfm*) à la place de n'importe quelle méthode.

```
                                    (send doc 'Sys_Maj)
                                    (apply (send (send c 'header) 'url) doc)
                                    (apply (send c 'url) doc)
                                    (apply (send (send c 'sig) 'url) doc) )
                )) )) ))) )
```

L'exécution du méta-interpréteur engendre un code CFML, interprété dans un deuxième temps en HTML par l'interpréteur CFML. Et c'est ce code HTML, résultant d'une double interprétation qui est finalement renvoyé au navigateur (qui interprète le code HTML pour en générer une représentation physique, affichée à l'utilisateur).

Lors de sa première exécution, l'interpréteur construit et stocke dans le cache du serveur (grâce à *Sys_Declare*) les objets correspondants aux images, icônes, fonds... tous les éléments graphiques globaux. Ceux-ci pourront être ensuite utilisés directement à partir de leur identifiant.

À l'invocation d'un *Documents*, la récupération de ses méta-données instancie éventuellement un ensemble d'images provisoires qui correspondent au images incluses dans ce *Documents.* Elles sont identifiées par IMG*x*, où *x* est le rang de l'image ($0<x<100$).

L'interpréteur, puis le code CFML/HTML instanciant le *Documents,* peuvent donc utiliser et faire afficher leurs différents attributs (nom, champ *alt*, date de création, taille).

4.6.1.2 Les méthodes « internes » au système

Les méthodes décrites ici sont invoquées par l'interpréteur *Sys_Application*, pour déterminer si le *Documents* requis est d'accès contrôlé et, le cas échéant, si l'utilisateur dispose des autorisations d'accès nécessaires.

Elles invoquent automatiquement la méthode d'identification, si l'utilisateur n'est pas authentifié et vérifient ses droits à mettre à jour le document (droit matérialisé par une icône MAJ, générée dans la signature de la page HTML) :

- *Sys_Acl* : cette méthode forme le pivot du système de sécurité. C'est elle qui va donner l'autorisation de requérir le fichier physique correspondant au *corps* du *Documents* et, éventuellement, le droit de le mettre à jour.

Elle vérifie d'abord si le *Documents* requis est d'accès contrôlé (s'il possède au moins une ACL de lecture).

S'il n'y a pas d'ACL, ce qui est le cas le plus général, la méthode se termine en donnant les autorisations d'accès (elle est un peu plus compliquée à cause des droits de mise à jour).

Par ailleurs, son invocation par l'interpréteur se fait dans trois contextes :

1. l'utilisateur est identifié et authentifié : la méthode invoque *Sys_VerifAcl* qui compare les autorisations de l'utilisateur avec les ACL (de lecture et d'écriture) avant de donner les autorisations *ad hoc* ;
2. l'utilisateur n'est pas encore identifié : *Sys_Acl* invoque alors le *Documents*-formulaire d'identification, *passe*, auquel elle transmet, en continuation, le *Documents* requis par l'utilisateur ;
3. l'utilisateur vient de s'identifier (à partir du *Documents*-formulaire, *passe*, à la suite de l'étape précédente) : la requête en cours est donc celle de la construction d'un *Documents*, d'accès protégé, par un utilisateur dont l'identification n'a pas encore été authentifiée (*cf.* exemple en § 4.7.1.3, p.142).

Sys_Acl invoque *Sys_Crypt* pour récupérer le *check code* du mot de passe qu'elle compare à celui enregistré et, en cas de succès, enregistre les coordonnées de l'utilisateur sur le domaine entier de l'intranet dans des *cookies* (*riruser* et *rirpwd*).

En cas d'échec de l'identification, une nouvelle invocation de *Passe* est effectuée.

- *Sys_VerifAcl* : cette méthode vérifie que l'utilisateur dispose d'un droit de lecture, d'un droit de mise à jour (qui implique un droit de lecture), qu'il fait partie d'un groupe qui dispose de ces droits ou, enfin, qu'il est un administrateur (ce qui lui confère tous les droits).
- *Sys_Crypt* : implémente un algorithme simple de cryptage du mot de passe de l'utilisateur qui retourne un nombre (*check code*). En fonction du contexte de l'invocation de la méthode, ce nombre est soit comparé avec celui qui est enregistré dans l'instance de *Users,* soit enregistré à la création de l'utilisateur. Il est enfin utilisé pour stocker dans un *cookie*, le temps de la session, le couple (*nom, pwd*). Cette méthode de stockage n'est actuellement pas très fiable puisque la législation française interdit encore l'utilisation de *SSL* ou toute autre forme de cryptage des communications. En conséquent, le *check code* est actuellement émis et reçu en clair, ce qui en réduit l'efficacité.
- *Sys_Maj* : cette méthode se charge de la vérification et de la mise à jour des méta-données d'un *Documents* avant la génération du code HTML correspondant. Elle gère trois événements :
  1. elle compare la date de modification du fichier physique à celle stockée et met cette dernière à jour ;
  2. en cas de modification, elle invoque, le cas échéant, la méthode de réindexation du *Documents* ;
  3. toujours en cas de modification, elle gère l'envoi des messages de notification active - mode *push* - aux utilisateurs abonnés (méthode *Ami*). C'est aussi grâce à elle, qui garantit que les deux méthodes de notification passive *Nourir* et *Rime* - mode *pull* - seront toujours pertinentes (*cf.* § .4.8.1.1, p.153).

En l'occurrence, la méthode *Sys_Maj* parasite la requête du premier utilisateur [29] qui accède à un *Documents* après sa modification, pour mettre à jour ses méta-données, le réindexer et notifier les utilisateurs des changements intervenus.

Elle réalise ainsi une réindexation incrémentale des données de RICERCAR qui évite de devoir gérer des processus lourds de réindexation ou de mise à jour globales.

- *Sys_Logs* : cette méthode effectue les différentes journalisations juste avant que l'interpréteur invoque la méthode de la classe du *Documents*. Elle enregistre les différents paramètres de connexions (utilisateur, date, serveur, objet, taille, navigateur utilisé, méthode HTTP, *etc.*).

C'est la compilation des tables issues de cette méthode qui permettront ensuite de réaliser les statistiques modulaires d'ensemble.

4.6.1.3 Les macro-méthodes

La double interprétation initiée par *Sys_Application* permet de mettre un ensemble de macro-méthodes à la disposition des utilisateurs finals.

La principale finalité de ces « macros » est d'offrir aux utilisateurs (système compris) la possibilité de référencer les *identifiants* des différents objets connus plutôt que leur localisation, dans un formalisme proche du HTML (sous forme de balises CFML) :

- **CF_AHREF** (*Sys_AHref*) : attend un paramètre KEY qui est l'identifiant du *Documents* (l'URL relative, en l'occurrence) vers lequel l'utilisateur souhaite faire un lien. Son comportement et ses autres paramètres sont ceux de la balise HTML, HREF.

  L'interprétation de cette macro-commande convertit l'URL relative en URL absolue et positionne éventuellement des marqueurs graphiques  et  qui indiquent respectivement que le lien demandé est un lien externe à RICERCAR (dont la validité ne peut donc être garantie) ou que ce lien est erroné (en cas d'erreur dans le nom de l'objet demandé ou si celui-ci a été détruit ou n'est pas encore créé).

  Exemple : <CF_AHREF KEY= "toto" >lien hypertexte</A> générera

  ```
  <A HREF= "http://sit-sio.intra.ratp/cft/page.cfm?key=toto">
  lien hypertexte</A>
  ```

- **CF_IMG** (*Sys_Img*) : attend un paramètre SRC qui est le nom d'une instance de sous-classes d'*Images*. C'est soit une des *Images* globales du serveur (appelée par son identifiant) soit une des images incluses dans le document en cours de construction (appelée par IMG*x*).

[29] Elle est cause d'un surcoût de 2 à 3 secondes dans la génération du code HTML.

Cette macro-commande accepte par ailleurs tous les paramètres connus de la balise HTML, IMG, qu'elle génère. Elle récupère automatiquement, de la méta-base, les champs *alt* correspondant aux images traitées.

Exemple : <CF_IMG SRC="IMG3"> où IMG3 est l'identifiant local de la troisième image incluse dans le *Documents* appelant, sera transformé en,

```
<IMG SRC = "http://sit-sio.intra.ratp/sio/rep/montoto.gif"
ALT = "Ceci est un texte alternatif">
```

- **CF_FIG** (*Sys_Fig*) : est très comparable à CF_IMG. Elle en diffère en considérant l'image comme une figure centrée dont elle affiche, en légende, le contenu du champ *alt* correspondant (c'est cette macro qui est utilisée pour générer l'ensemble des figures de la version « navigable » de cette thèse).

  Exemple: <CF_FIG SRC="IMG3"> sera transformé en,

```
<CENTER>
<IMG   SRC = "http://dgc.intra.ratp/sio/rep/montoto.gif"
       ALT = "Ceci est la légende de la figure">
<em>Ceci est la légende de la figure</em>
</CENTER>
```

- **CF_FORM** (*Sys_Form*) : accepte les mêmes paramètres que la balise HTML FORM qu'elle émule. Son paramètre ACTION est cependant l'identifiant du *Documents* qui va traiter le formulaire.

  Exemple : <CF_FORM ACTION="traiter" METHOD=POST> qui génère,

```
<FORM ACTION = "http://sit-sio.intra.ratp/cft/page.cfm?key=traiter"
METHOD = POST>
```

- **CF_INCLUDE** (*Sys_Include*) : inclut le code de l'objet passé dans le paramètre KEY, s'il existe.

Cette méthode correspond à une composition de fonctions puisque l'interprétation du code de KEY est effectuée dans le contexte appelant.

C'est cette macro qui est utilisée, notamment, pour invoquer les *Documents* d'en-tête et signataires et qui va permettre d'interpréter le code du *corps* d'un *Documents*.

Exemple : <CF_INCLUDE KEY= "Sty_Header"> qui génère,

```
<CFINCLUDE TEMPLATE = "/cft/local/header.cfm" >
```

4.6.1.4 Les Documents

Ainsi que le lecteur-internaute a déjà pu s'en rendre compte, une instance de (sous-classe de) *Documents* se compose, par construction, d'un en-tête et d'un document signataire (tous deux des *Documents*) encadrant une instance de *Documents* (que nous avons appelée *corps*).

Cet emboîtement, qui pourrait potentiellement conduire à une récursivité infinie, est effectivement terminé par la méthode de classe de ces trois *Documents*, qui considère que leurs en-têtes et textes signataires sont nuls.

Les sous-classes de *Documents* implémentent chacune leur méthode de construction dont l'invocation conduit à la génération d'un script CFML.

Ainsi, l'appel d'une URL, sous la forme *http://serveur/ricercar/document* est-il d'abord redirigé (grâce à la règle de réécriture du serveur HTTP) pour invoquer la méthode *goto* du serveur (qui est, accessoirement, instance de *Documents* !).

C'est alors cette méthode *goto* de *Serveurs* qui, initialise le méta-interpréteur, retrouve la classe du *Documents* et invoque sa méthode de construction (les diagrammes de séquence en § 4.7.1.1, § 4.7.1.3, p.138 et suivantes montreront concrètement les différentes étapes de génération du code HTML).

Le script CFML, issu de cette méta-interprétation, se compose *in fine* de la concaténation (dans le sens CF_INCLUDE) des différents scripts intermédiaires générés.

C'est à l'issue de l'interprétation de ce script, qu'est construit le code HTML, finalement renvoyé au navigateur de l'utilisateur pour (interprétation et) affichage.

4.6.2 L'architecture

En résumé, l'architecture proposée à l'issue de cette construction itérative, est résolument orientée « objets-documents » : les serveurs, les modèles de documents et les images incluses sont sous-classes de la classe *Documents*.

Il aurait sans doute pu être possible de pousser le raisonnement à son terme en tentant de *Documenter* les classes d'utilisateurs ainsi que celles gérant les navigateurs, *etc*., mais cela eût relevé de l'exercice de style et ne permettait pas *a priori* d'accroître la synergie d'ensemble.

L'uniformisation de la représentation permet cependant d'adresser indifféremment les documents et les scripts des utilisateurs finals ainsi que leurs différentes **classes,** confondues avec les **méthodes** utilisées.

Ce dualisme « programmes = données » évite la duplication des méthodes nécessaires pour générer les *Documents* d'un intranet.

Une même instance peut donc remplir deux rôles :

1. comme un **document** à générer, si elle est passée en paramètre d'une méthode ;
2. ou alors **programme** (ou modèle de document), si elle est elle-même invoquée en tant que méthode (*resp*. classe).

Cette ambivalence leur (les classes et méthodes) permet aussi de générer leur propre documentation à partir des méta-données de la base. Elles sont regroupées par menus, sont documentées, gèrent des droits et des statistiques d'accès, *etc*.

C'est ainsi que la *Documentation* des classes et des méthodes les fait bénéficier de la même indépendance à la localisation non seulement en tant que *Documents* mais aussi en tant que procédures exécutables. Elles peuvent être modifiées ou déplacées en toute transparence pour leur utilisateur final, en l'occurrence le système lui-même.

Cette homogénéisation de la structure permet enfin au système d'avoir accès à la totalité de son propre catalogue en accédant à la métaclasse *Modèles*, ce qui pourrait lui permettre d'ajuster dynamiquement ses paramètres, le cas échéant.

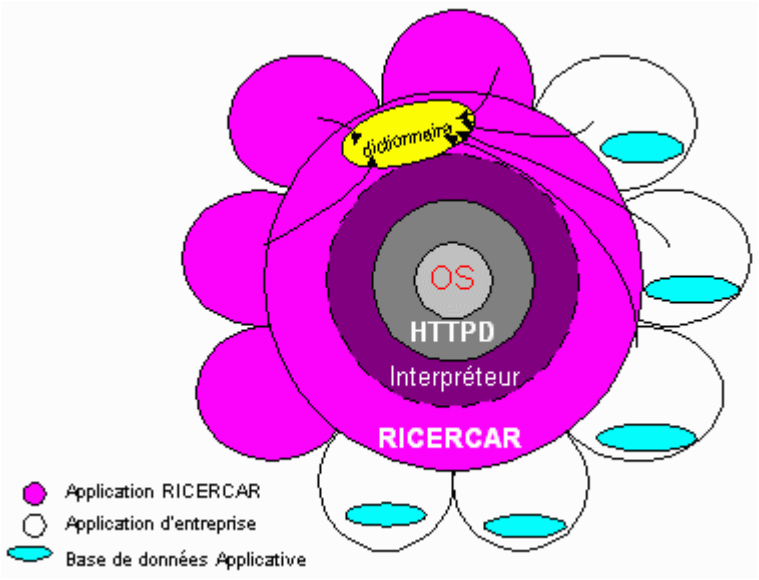


Figure 4. 16. La réflexivité de RICERCAR assure l'homogénéité entre les applications d'entreprise et le noyau du système, qui utilisent les mêmes représentations

Enfin, les méthodes peuvent être vues comme des expertises (quoique totalement procédurales) liées à une sous-classes de *Documents*. Elles peuvent être déléguées entre sous-classes compatibles, aux risques de l'appelant (c'est ainsi qu'une instance de *Sty_Menu* peut exécuter la méthode *Sty_Tree* (*Arbre.cfm*), c'est aussi la raison pour laquelle la classe *Sty_Post*, qui ne diffère de *Sty_Page* que par la valeur de ses variables de classe, utilise la même méthode, *Page.cfm*).

Les fonctionnalités des méthodes exposées ici devraient permettre la compréhension des diagrammes de séquences de séquences que nous présentons dans la section suivante.

4.7 Application : génération de Documents

Les paragraphes suivants détaillent donc deux séries de diagrammes de séquences qui illustrent le cheminement complet des requêtes d'utilisateur :

1. le premier cas (simple) correspond à une requête « normale » ;
2. le second cas montre la configuration d'accès la plus complexe (le pire des cas).

D'autre part, nous utilisons la notation suivante où :

- le *client* est le navigateur de l'utilisateur (un interpréteur HTML) ;
- les D*i*R et D*i*L sont respectivement les **D**ictionnaires **R**éparti et **L**ocal du serveur *i*, noté S*i*.

Il est enfin important de préciser que le nombre d'allers-retours matérialisés dans ces diagrammes ne représentent pas autant de requêtes aux méta-bases mais indiquent, fonctionnellement, les opérations effectuées.

4.7.1.1 Cas simple (le plus fréquent)

Cette première série de diagrammes regroupe en fait les deux cas d'accès les plus fréquents, pour un utilisateur quelconque (non identifié) qui :

1. « clique » sur un lien pour accéder à un *Documents*, *doc*, en libre accès, se trouvant sur le même serveur *s1* à partir duquel il émet sa requête **(1)**;
2. veut accéder à un document à partir d'une URL sauvegardée dans son calepin **(3)**.

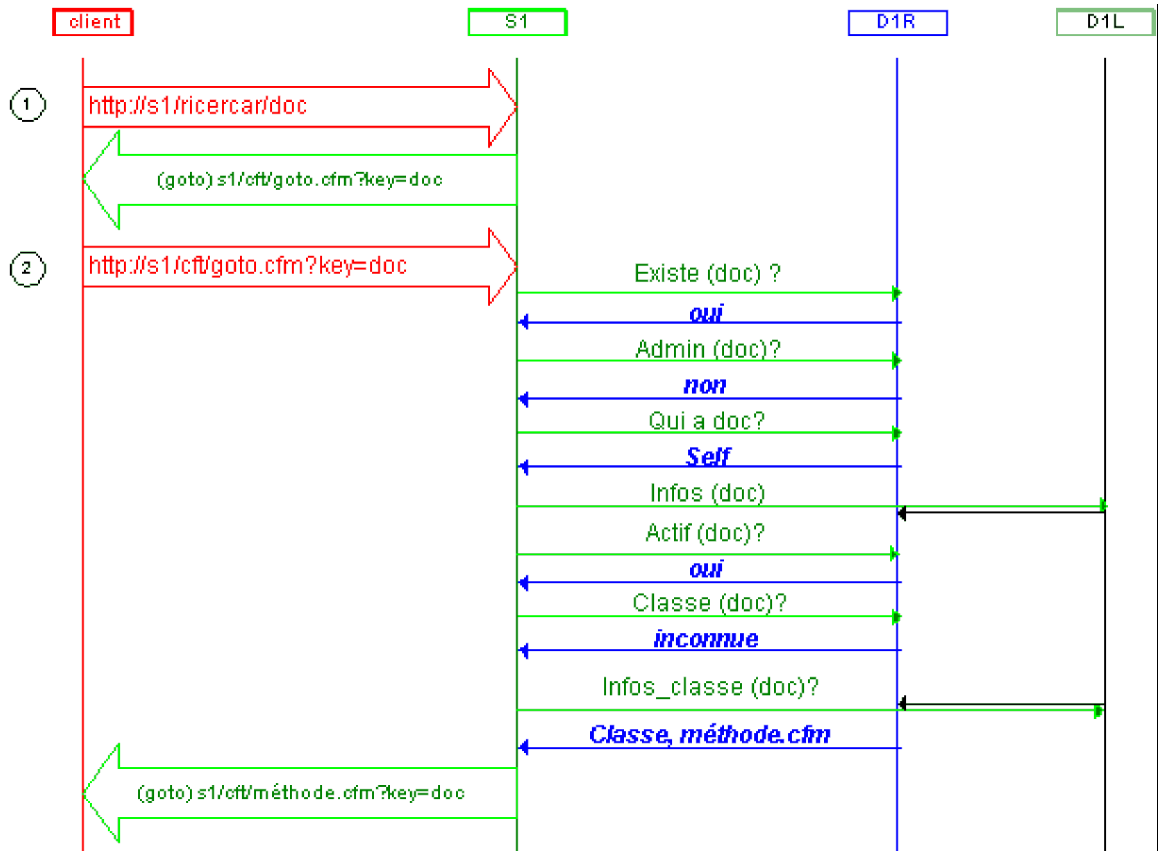


Figure 4. 17. Interrogation de la méta-base pour retrouver la classe de doc.

(1) *http://s1/ricercar/doc*
Le serveur HTTP applique une de ses règles de réécriture :
*/ricercar/** → */cft/goto.cfm?key=**

(1) est donc transformé en

(2) *http://s1/cft/goto.cfm?key=doc*
La requête **(2)** est transmise à l'interpréteur CFML.
goto.cfm, qui correspond au code exécutable de *Sty_goto*, génère un appel au *méta*-interpréteur, *Sys_application*.
Celui-ci initialise les variables de l'application puis exécute une batterie de tests :

- interrogation de l'index : « *doc* existe-t-il ? » (*oui*)
- « *doc* est-il une méthode ou un document d'administration ? » (*non*. Si *oui* alors il est local)

→ Extraire les références du serveur hébergeant *doc* .

- « *s2* est-il le serveur hébergeant *doc* ? » (*oui* - *Self*)

→ Récupérer les informations relatives à *doc*

- « *doc* est-il actif ? » (*oui*)
- « Sait-on quel est la classe de *doc* ? » (*non*)

→ Récupérer les informations relatives à la (méthode de la) classe de *doc* (*Classe, méthode.cfm*).

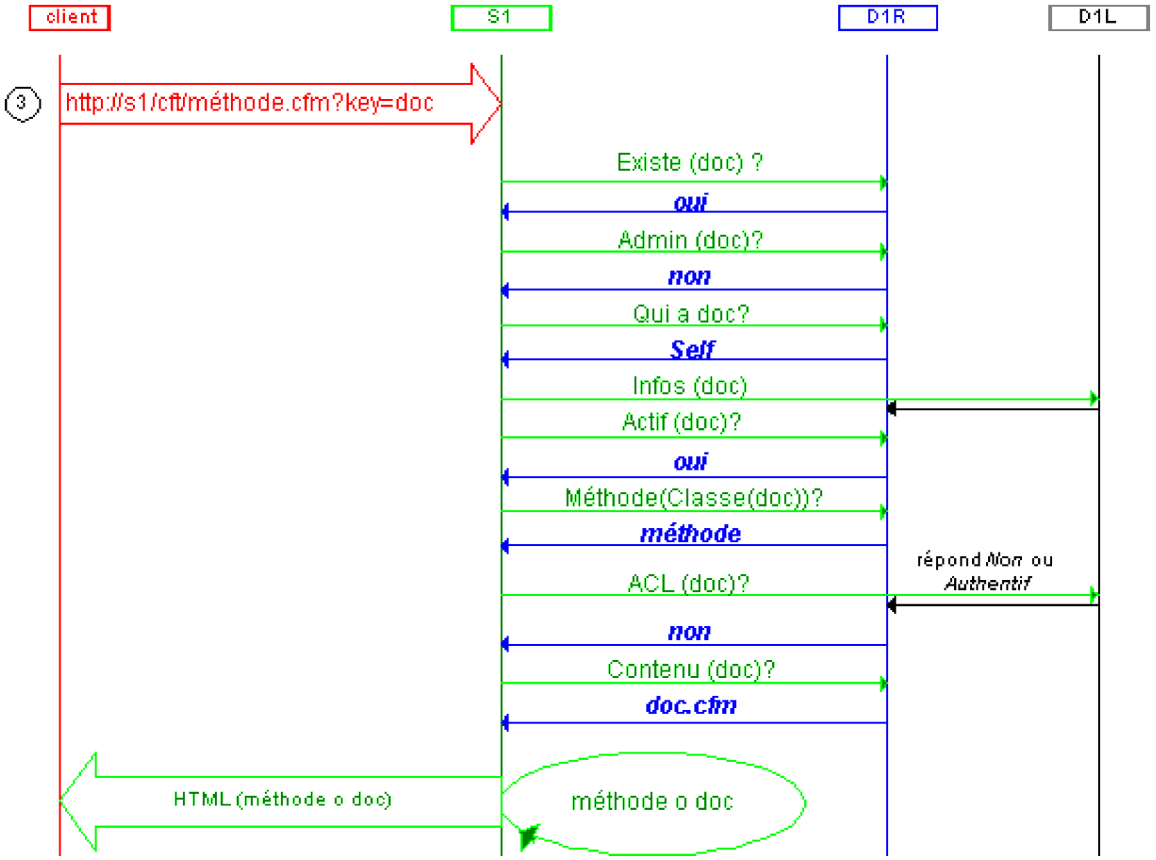


Figure 4. 18. Cette étape (3), qui est la séquence la plus courte possible, correspond à la récupération d'une adresse mémorisée dans un calepin.

(3) *http://s1/cft/modele.cfm?key=document*

Cette requête est acheminée jusqu'à *Sys_Application*, génère les tests suivants :

- interrogation de l'index : « *document* existe-t-il ? » (*oui*)
- « *document* est-il un document d'administration ? » (*non*)
  → Extraire les références du serveur hébergeant *document* .
- « *s1* est-il le serveur hébergeant *document* ? » (*oui*)
  → Récupérer les informations relatives à *document*
- « Est-ce que *modele.cfm* est le code du modèle (la classe) associé à *document* ? » (*oui*)
- « *document* a-t-il des droits d'accès particuliers ? » (*non*)
  → Récupérer la localisation physique du fichier associé à *document*.
  → Générer le document HTML **en exécutant le code** de *Modele* (*modele.cfm*) avec *document* comme paramètre.

➔ **Renvoyer le document HTML au navigateur**.

4.7.1.2 Exemple de cas simple

Le tableau suivant est un extrait du fichier de *log* du serveur auquel nous avons concrètement soumis une requête simple, identique à celle décrite ici [30] :

- la première colonne du tableau, ajoutée par nos soins, renvoie à la numérotation dans le diagramme.
- le **Client** est le poste de travail qui a émis la requête (nous avons omis ici le nom du domaine DNS qui surchargeait inutilement le tableau) ;
- le **Serveur** *S1* est ici représenté par *sit-sio.intra.ratp* ;
- les **Date**s sont celles enregistrées dans les *logs* et n'ont qu'une précision limitée à la seconde ;
- les **Commande**s **HTTP** correspondent aux méthodes HTTP exécutées par le serveur : GET ou POST, suivies de l'URL relative de la requête ;
- **Code HTTP** est le code-résultat normalisé [Berners-Lee_96], renvoyé par le serveur HTTP à l'issue de la requête.
  « 302 » correspond à une redirection (un envoi de message).
  « 200 » veut dire « document correctement transmis à l'utilisateur »
  « 404 » voudrait dire « document non trouvé » (!) ;
- **Nb Octets** représente la taille en octets du corps du document transmis.

	Client	Serveur	Date	Commande HTTP	Code HTTP	Nb Octets
1	fafner	sit-sio.intra.ratp	13/Nov/1997 :14:48:19	GET /ricercar/flash	302	0
2	fafner	sit-sio.intra.ratp	13/Nov/1997 :14:48:19	GET /cft/goto.cfm ?key=flash	302	190
3	fafner	sit-sio.intra.ratp	13/Nov/1997 :14:48:19	GET /cft/page.cfm ?key=flash	200	8445
	fafner	sit-sio.intra.ratp	13/Nov/1997 :14:48:19	GET /Logos/flash.gif	200	4236

Noter que l'ensemble du traitement s'est effectué en moins d'une seconde (qui est la précision minimale disponible).

Le document HTML final, qui fait 8445 octets (*cf.* Annexe 6.6.1, p.209) a bien été généré par la méthode *Sty_page* (*page.cfm*, en **3**).

Le logotype du *Documents*, non chargé dans le cache, a été requis du serveur alors que les autres éléments graphiques (lignes , fonds, *etc*.) ont été récupérés à partir du cache mémoire.

[30] Pour vous, qui lisez la version électronique de ces pages, cet exemple est désormais trivial puisqu'il correspond à la majorité des accès que vous avez effectués.

4.7.1.3 Le pire des cas

Le deuxième diagramme représente l'accès d'un utilisateur, *user*, à un objet, *doc*, situé sur le serveur *s2* ; *user* émet sa requête sur le serveur *s1* ; *doc* est protégé en consultation et *user*, qui a les droits d'accès, ne s'est pas encore identifié sur l'intranet.

Ce diagramme montre ainsi le « pire des cas », d'une requête qui nécessite la collaboration de trois serveurs : *s2*, *s1* mais aussi *s0* qui gère l'authentification (SSO) de *user* pour accéder finalement au document souhaité.

Cette configuration n'est toutefois possible qu'**une seule fois au maximum** par session d'utilisateur, l'identification de *user* étant par la suite stockée dans des variables de session.

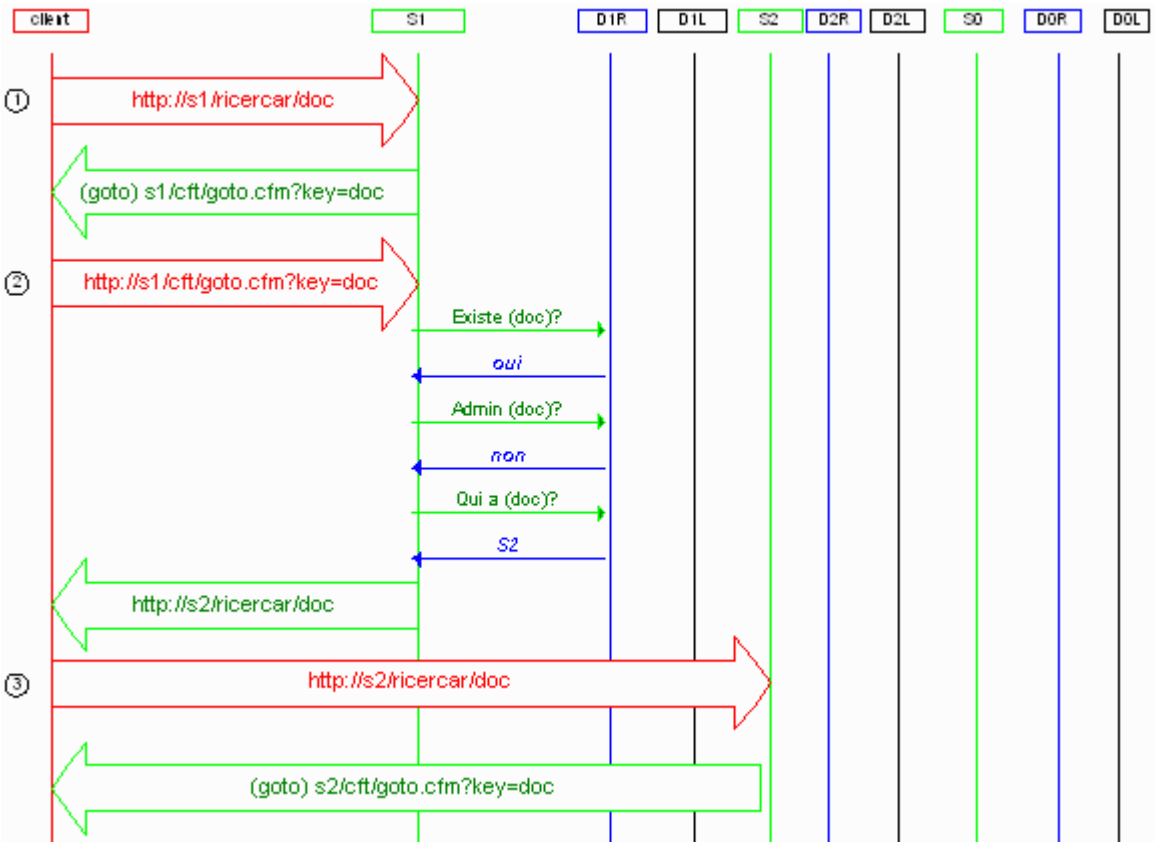


Figure 4. 19. Cas le plus complexe (rare) de calcul de page, effectué par RICERCAR. Première phase : renvoyer la requête au « bon » serveur.

La requête (manuelle) saisie est donc :

(1) *http://s1/ricercar/doc*
Le serveur HTTP applique une de ses règles de réécriture :
*/ricercar/** → */cft/goto.cfm?key=**
(1) est donc transformé en

(2) *http://s1/cft/goto.cfm?key=doc*
La requête **(2)** est transmise à l'interpréteur CFML.
goto.cfm, qui correspond au code exécutable de *Sty_goto*, génère un appel au *méta*-interpréteur, *Sys_application*.
Celui-ci initialise les variables de l'application puis exécute une batterie de tests :

- interrogation de l'index : « *doc* existe-t-il ? » (*oui*)
- « *doc* est-il une méthode ou un document d'administration ? » (*non*. Si *oui* alors il est local)

→ Extraire les références du serveur hébergeant *doc* .

- « *s1* est-il le serveur hébergeant *doc* ? » (*non*, *c'est s2*)

  → Transmettre la requête à *s2.*

(**3**) *http://s2/ricercar/doc*
Le serveur HTTP applique sa règle de réécriture :
*/ricercar/** → */cft/goto.cfm?key=**
(3) est donc transformé en

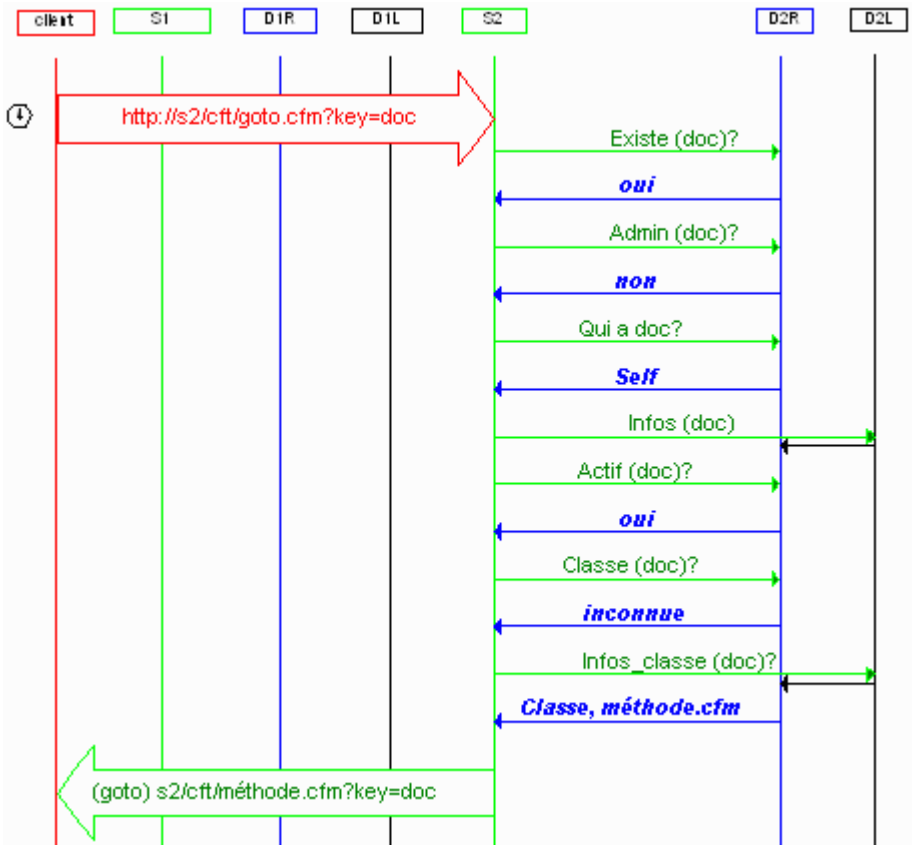


Figure 4. 20. Récupération de la méthode de la classe du document

(4) *http://s2/cft/goto.cfm?key=doc*

- *... reprise des tests depuis le début...*
- « *s2* est-il le serveur hébergeant *doc* ? » (*oui* - *Self*)

→ Récupérer les informations relatives à *doc*

- « *doc* est-il actif ? » (*oui*)
- « Sait-on quelle est la classe de *doc* ? » (*non*)

→ Récupérer les informations relatives à la (méthode de la) classe de *doc* (*Classe, méthode.cfm*).

→ Regénérer la requête avec les informations complètes :

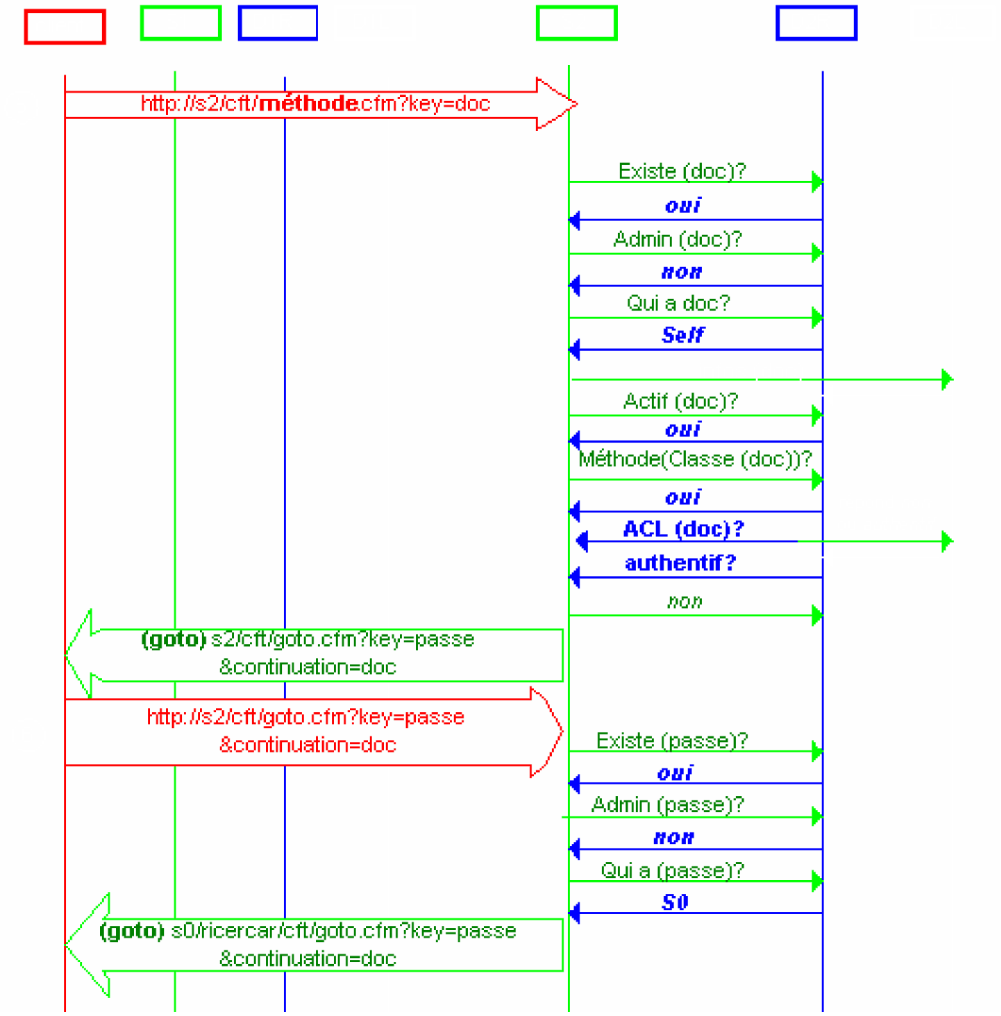


Figure 4. 21. Le document est protégé et l'utilisateur n'est pas identifié. Invocation du document d'identification (passe).

(5) *http://s2/cft/méthode.cfm?key=doc*

- *... reprise des tests depuis le début....*
- « Sait-on quelle est la méthode associée à la classe de *doc* ? » (*oui*)
- « *doc* est-il protégé en lecture ? » (*oui*)
- « L'utilisateur est-il authentifié ? » (*non*)

→ Renvoyer le document d'authentification (*passe*) avec *doc* comme continuation locale.

(6) *http://s2/cft/goto.cfm?key=**passe**&continuation=**doc***

- *... reprise des tests depuis le début ...*
- « *s2* est-il le serveur hébergeant *passe* ? » (*non*, c'est *s0*)

→ Transmettre la requête à *serveur0*.

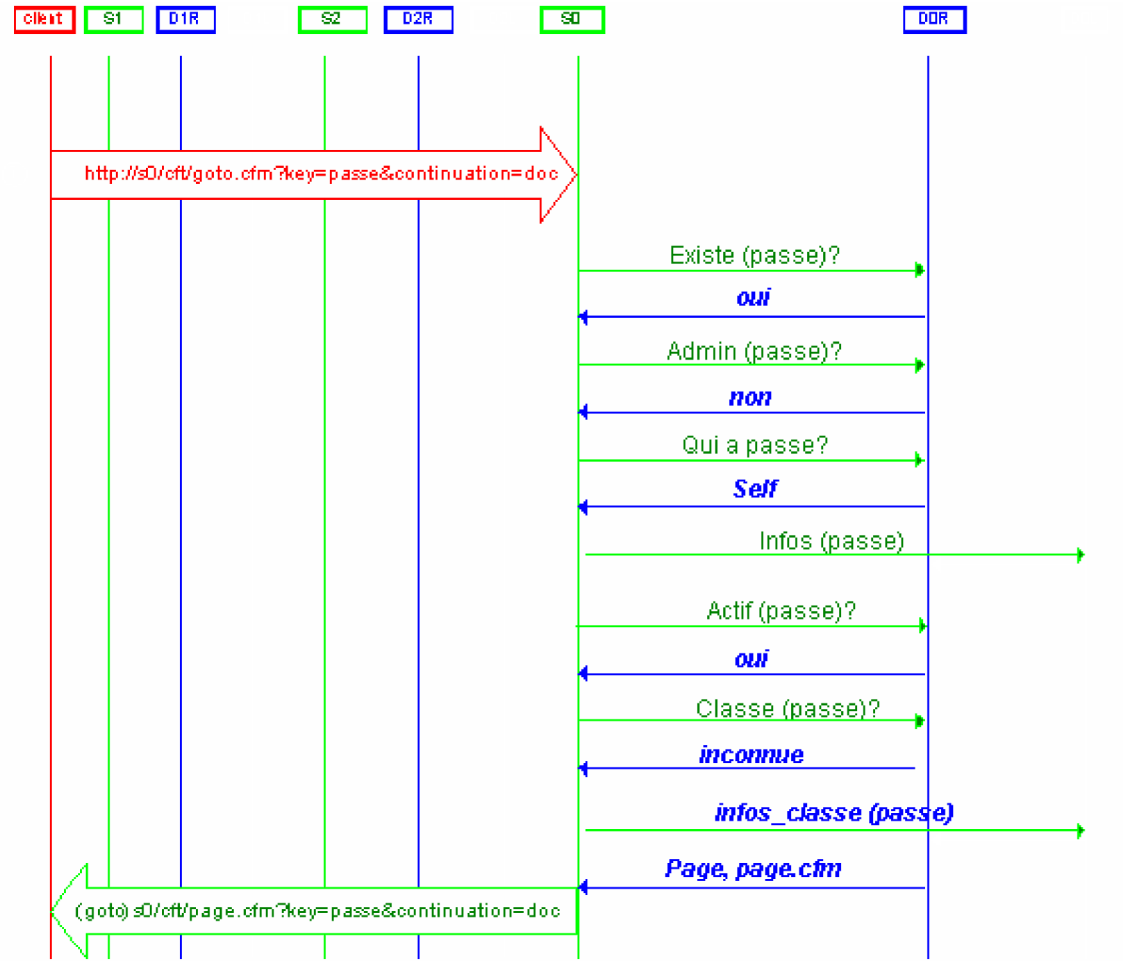


Figure 4. 22. Récupération des informations relatives au document (formulaire) d'identification

(7) *http://s0/cft/goto.cfm?key=**passe**&continuation=**doc***

- *... reprise des tests depuis le début...*
- « Sait-on quelle est la classe de *passe* ? » (*non*)

→ Récupérer les informations relatives à la (méthode de la) classe de passe (*Pages, page.cfm*).
→ Regénérer la requête avec les informations complètes (ne pas oublier la continuation) :

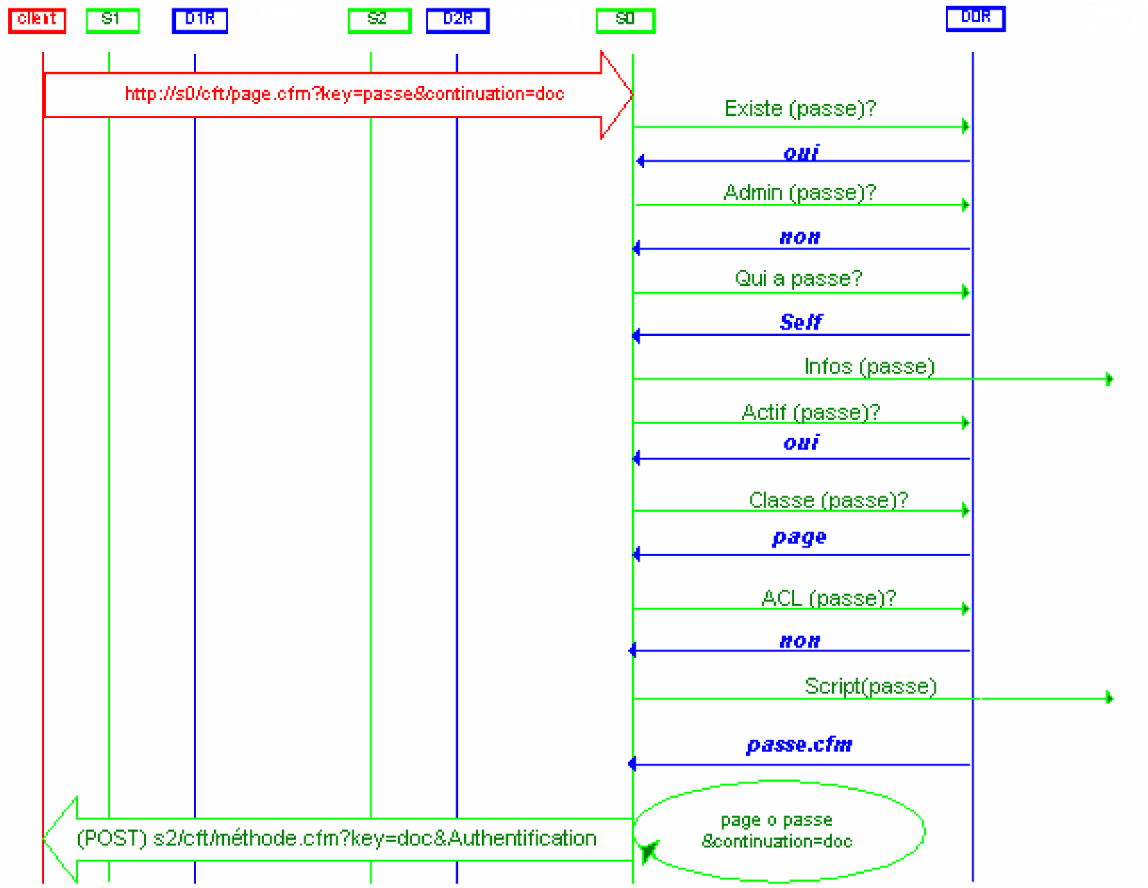


Figure 4. 23. L'utilisateur s'identifie. Le formulaire POSTe sa requête originelle, accompagnée de son identification.

(8) *http://s0/cft/page.cfm?key=**passe**&continuation=**doc***

- *... reprise des tests depuis le début...*
- « passe est-il protégé en lecture ? » (*non, sinon bouclage*)

→ Récupérer les informations relatives à la classe de *passe* (*Pages, page.cfm*).
→ Récupérer le code HTML/CFML du fichier associé à *passe*.
→ Générer le document HTML **en exécutant le code** de *Page* avec *passe* comme paramètre.

➔ Renvoyer le document HTML au navigateur (le formulaire d'identification).

L'utilisateur saisit son identifiant et mot de passe puis valide le formulaire.

➔ Exécution de *passe* : POSTer (au sens HTTP) l'identifiant et le mot de passe saisis par l'utilisateur avec la nouvelle requête à *doc*.

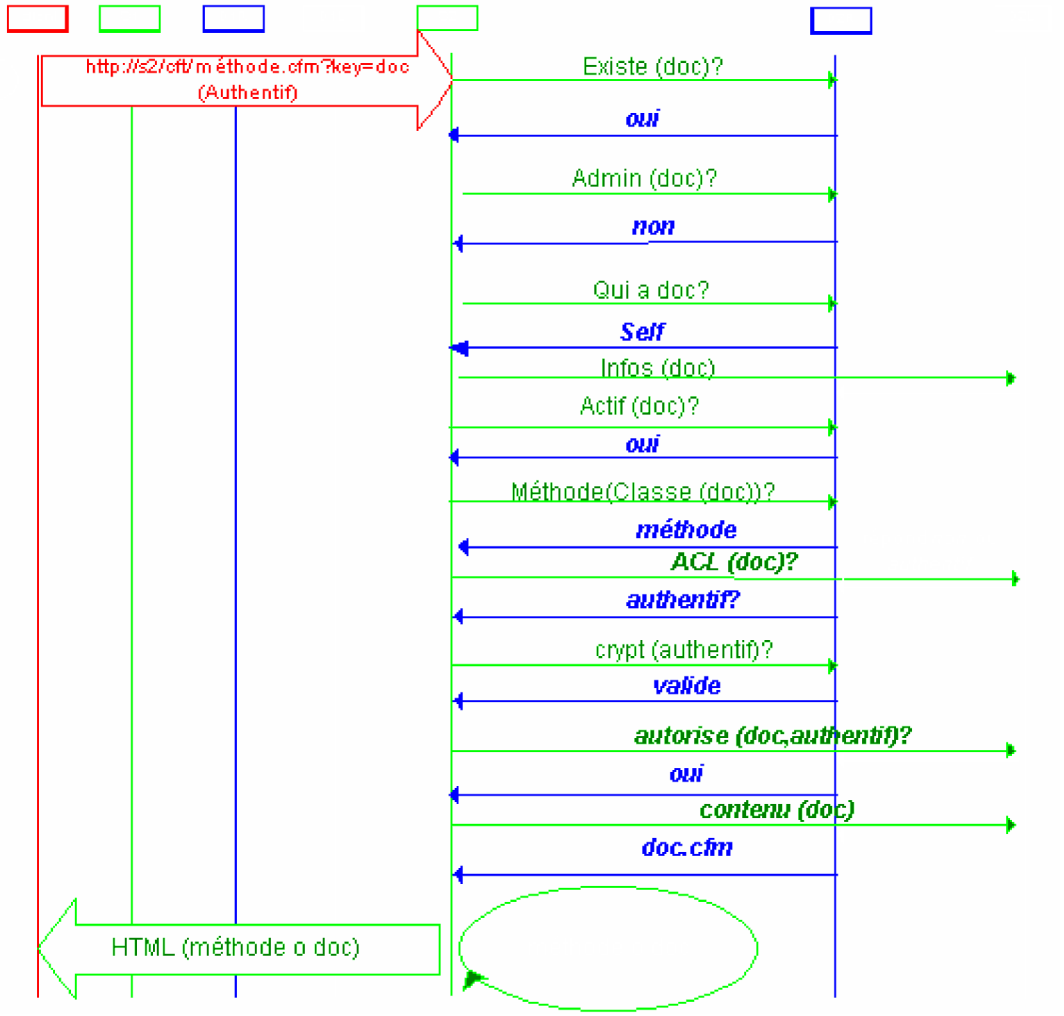


Figure 4. 24. Récupération des informations relatives au document (formulaire) d'identification

(9) *http://s2/cft/méthode.cfm?key=**doc***
(Authentification stockées dans les variables POSTées)

- ... reprise des tests depuis le début...
- « *doc* est-il protégé en lecture ? » (*oui*)
- « l'utilisateur est-il authentifié ? » (*non*)
- « l'utilisateur vient-il de s'identifier ? » (*oui*)
  → Cryptage du mot de passe et comparaison de l'identifiant et du *Check code* généré dans la table *USERS*.
- « La vérification est-elle positive ? » (*oui*)
  ➔ **Authentifier l'utilisateur** (enregistrer - par le biais de *cookies* -l'identifiant et le *Check code* de l'utilisateur)
- « l'utilisateur a-t-il le droit de consulter doc ? » (*oui*)
  → Récupérer la localisation physique du fichier associé à *doc*.
  → Générer le document HTML **en exécutant le code** de *Méthode* avec *doc* comme paramètre.
  ➔ **Renvoyer le document HTML au navigateur**.

4.7.1.4 Exemple du pire des cas

Le tableau suivant est un extrait rapproché des fichiers de *log* des serveurs auxquels nous avons concrètement soumis une requête identique à celle décrite dans les pages précédentes (exemple en Annexe).

Les colonnes sont identiques que dans l'exemple précédent :

- Le **Serveur** *S1* est ici représenté par *sit-cis.intra.ratp*,
  S2 est *sit-sio.intra.ratp*,
  S0 est *ricercar.intra.ratp* ;

(**1** à **7** totalisent ainsi le trafic induit sur le réseau par les différentes redirections générées par RICERCAR). Le document *Passe*, généré en **8**, comporte 6053 octets de HTML et le document *doc* final (en **9**) « pèse » 27685 octets.

	Client	Serveur	Date	Commande HTTP	Code HTTP	Nb Octets
1*	fafner	sit-cis.intra.ratp	27/Jul/1997 : 17:22:29	GET /RICERCAR/doc	302	0
2	fafner	sit-cis.intra.ratp	27/Jul/1997 : 17:22:29	GET /cft/goto.cfm ?key=doc	302	187
3*	fafner	sit-sio.intra.ratp	27/Jul/1997 : 17:22:29	GET /RICERCAR/doc	302	0
4	fafner	sit-sio.intra.ratp	27/Jul/1997 : 17:22:29	GET /cft/goto.cfm ?key=doc	302	195
5	fafner	sit-sio.intra.ratp	27/Jul/1997 : 17:22:29	GET /cft/page.cfm ?key=doc	302	191
6	fafner	sit-sio.intra.ratp	27/Jul/1997 : 17:22:30	GET /cft/goto.cfm ?key=passe &continuation=doc	302	217
7	fafner	ricercar.intra.ratp	27/Jul/1997 : 17:22:30	GET /cft/goto.cfm?key=passe &continuation=doc	302	0
8	fafner	ricercar.intra.ratp	27/Jul/1997 : 17:22:30	GET /cft/page.cfm ?key=passe &continuation=doc &MSG=IDENTIFICATION	200	6053

*Le temps écoulé entre **8** et **9** (21 secondes) est celui qu'a pris l'utilisateur pour valider le formulaire d'identification (passe).*

9	fafner	sit-sio.intra.ratp	27/Jul/1997: 17:25:52	POST /cft/page.cfm ?KEY=doc	200	27685

Les étapes **1** et **3**, (marquées d'une astérisque) se résument à réécrire */ricercar/objet.* Elles pourraient être considérées comme une perte de ressources (temps et réseau). C'est cependant le prix à payer pour satisfaire les *desiderata* des utilisateurs qui souhaitent pouvoir échanger des URL sous une forme ergonomique <*RURL:mon_url*>, plutôt que
<*RURL:/cft/goto.cfm?key=mon_url*> ou encore
<*RURL:/cft/méthode.cfm?key=mon_url*>.

4.8 RICERCAR, un ORB

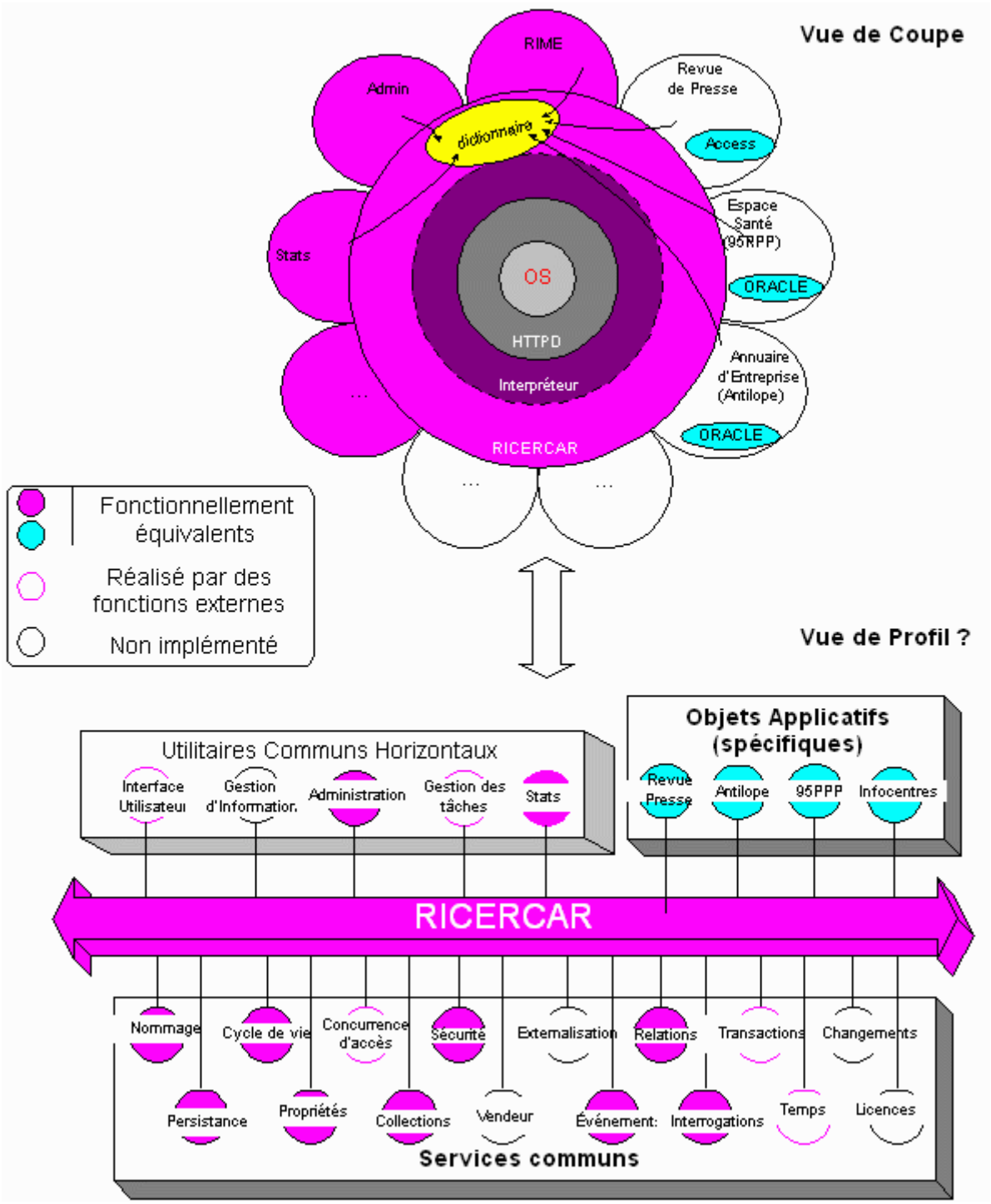


Figure 4. 25. Correspondance fonctionnelle entre un ORB et RICERCAR.

Il est encore trop tôt pour espérer réaliser une implémentation aussi performante de RICERCAR qui s'appuierait sur l'architecture normalisée de l'OMG [31]. Il est cependant aujourd'hui important, à défaut de se poser la question « Et pourquoi pas CORBA ? », de montrer comment sont traités et déclinés les différents points d'architecture et de services spécifiés dans l'OMA.

Le consortium de près de 800 entreprises qui forment l'OMG a, effectivement, eu l'occasion en plus de huit ans, de poser l'ensemble des questions relatives à la répartition d'objets, qui répondent aux interrogations de ses nombreux adhérents. Il n'est donc pas étonnant que le modèle de CORBA soit systématiquement repris comme point de repère pour tout système réparti.

Si les questions sont clairement posées, les réponses sont, en revanche, loin d'être toutes apportées ou même d'être satisfaisantes. La démarche consensuelle de l'OMG, ainsi que la complexité des problèmes évoqués, sont telles que de longs mois s'écoulent avant qu'un point particulier d'architecture soit complètement spécifié ou que les mérites respectifs de plusieurs implémentations soient éclaircis.

[Orfali_97b, p.246] montre ainsi que, début février, les grands fournisseurs d'ORB (*Digital*, *Visigenic*, *IBM*, *Iona*, *Sun*, *etc*.) ne s'accordaient que sur la fourniture du service de nommage et proposaient, pour les plus avancés d'entre eux, cinq ou six services sur les seize spécifiés (les autres étant planifiés pour 1997 ou 1998). Cette étude recoupe celle, publiée par [Frey_97] en juin, qui indique qu'il n'est pas rare de surcroît, que de grands fournisseurs d'ORB n'aient même pas encore intégré d'IR (*Orb Plus 2.5* de *Hewlett Packard*, par exemple).

Nous sommes cependant convaincu de la synergie de CORBA et de Java, qui forment un dernier rempart contre l'hégémonie de *Microsoft*. Cette solution promet en effet, de façon réaliste, une réelle interopérabilité dans des entreprises ayant déployé de façon mixte leurs applications sous UNIX et sous NT et qui souhaitent pouvoir continuer à bénéficier du meilleur de ces deux technologies.

Il est donc probable que les prochains mois vont connaître une explosion d'ORB, mieux intégrés au monde Bureautique et offrant de plus en plus de services, à des coûts acceptables (le prix moyen d'un ORB se situe aujourd'hui autour de 30 kF).

Il est aussi inévitable que les différentes solutions réparties élaborées soient opposées et comparées, telles DCE [Brando_95b], CORBA [Brando_95a, Orfali_95, Schmidt_95, Chauvet_97, Curtis_97, Dreyfus_97] ou DCOM [Davis_97, Flynn_97, OMG_97] et que ces études s'interrogent déjà sur la pertinence de passerelles entre ces différentes solutions (une nouvelle *Tour de Babel* en perspective ?).

En tout état de cause, l'architecture de CORBA marquera autant l'informatique distribuée que le modèle en sept couches de l'ISO et ce, quel que soit son succès commercial.

---

[31] Cette phrase ne nous engage pas au delà de novembre 1997 !

Il est désormais courant que le terme ORB soit employé pour qualifier tout « bus logiciel » qui assure la communication entre objets répartis (il nous arrive ainsi fréquemment de lire « l'ORB OLE de *Microsoft* » [Orfali 96, p.70] [Linthicum 97a]).

Aussi, la figure que nous avons plaquée comme première illustration de l'assertion émise en titre, est-elle volontiers provocatrice, puisque nous y montrons que RICERCAR implémenterait à lui seul bien plus de services que toute solution commerciale actuellement disponible autour de CORBA !

Cette comparaison ne peut cependant pas s'exprimer en ces termes : CORBA est une norme qui spécifie le détail des différents services qui doivent être rendus et indique précisément les méthodes que reconnaît un ORB ainsi que leurs signatures. De plus, l'OMG s'efforce d'apporter une interopérabilité globale entre clients et serveurs hétérogènes et précise les interfaces qui doivent être disponibles au niveau des clients comme des serveurs.

Il est donc normal que toute architecture répartie puisse être représentée de façon macroscopique selon le même schéma que celui de l'OMA (même l'architecture de *Microsoft* pourrait être déclinée selon un tel modèle !).

De même, l'utilité de notre comparaison avec CORBA ne se pose pas en termes de compatibilité mais revient à *réutiliser* cette architecture comme *méta-modèle* pour simplifier l'explication des différents niveaux de fonctionnalités implémentés.

En l'occurrence, le problème que nous avons traité comporte un ensemble d'hypothèses, formulées plus ou moins implicitement :

- le client est un navigateur connecté à un réseau local (*vs.* tous types de clients qui vont s'interfacer, statiquement, par le biais de souches ou, dynamiquement, grâce à l'IR et au DII) ;
- HTML est le seul langage connu du client (*vs.* Le nombre de projections de langages, supportées par l'OMG : C, C++, SmallTalk, Java, COBOL, *etc*.) ;
- HTTP est le protocole établi entre un client et un serveur (*vs.* IIOP ou toute autre instance du GIOP) ;
- l'ensemble des données sont référencées dans des SGBD [32] (le format des méta-données de CORBA est, lui, quelconque).

L'ensemble de ces hypothèses simplifie ainsi le processus de modélisation d'une application qui, dans CORBA, doit être spécifiée en IDL, compilée en souche et squelette, respectivement liés au client et au serveur, puis implémentée par les méthodes ainsi décrites.

C'est, de façon générale, un des atouts majeurs d'une architecture intranet qui utilise un « client léger » et s'affranchit totalement des problèmes de compatibilité ou d'interopérabilité des applications par rapport au client : c'est le protocole

[32] Ceci est un cas particulier : nous aurions pu modéliser ces données dans un annuaire LDAP par exemple.

HTTP qui nivelle la communication entre clients et serveur (nivellement par le bas en attendant que IIOP ou qu'un autre protocole se généralise).

Le processus complexe de l'OMG, incontournable dans un souci d'interopérabilité générale (qui permet d'utiliser plusieurs langages de programmation différents et, dans une mesure croissante, des ORB de fournisseurs différents), se réduit dans le cas de RICERCAR à devoir définir des interfaces en SQL entre les méta-bases et les nouvelles bases de données applicatives.

C'est ainsi que la méta-base implémentée par RICERCAR joue le double rôle des Référentiels d'Interface et d'Implémentations, en spécifiant la structure des données ainsi que les objets disponibles à tout moment.

Le méta-interpréteur, *Sys_Application* + *Sys_Declare*, remplace, quant à lui, l'ORB, le BOA [33] et le DSI (*cf.* § 2.5.4, p.54) : il a en charge la localisation des objets, l'identification de leur classe, la vérification des signatures des méthodes, l'acheminement des requêtes entre utilisateurs et serveurs puis le renvoi des résultats (ce que réalise bien, *in fine*, un ***b**rocanteur* [34] de **r**equêtes **o**bjet).

4.8.1 Les « Services » et « Objets Applicatifs » en bref

4.8.1.1 « Services Communs »

La mise en place d'un méta-interpréteur distribué qui interface un ensemble de méta-bases fédérées nous a permis de décrire, dans les sections précédentes, un ensemble de services génériques communs, couverts fonctionnellement par RICERCAR.

Un service de **nommage** fiable a ainsi été mis en place. Il offre un accès **persistant** aux objets à partir de leur identifiant, utilisé comme paramètre de méthodes ou alors sous la forme d'URL relatives, pour l'interface externe. Cette persistance est par ailleurs renforcée par la gestion de caches qui gardent en mémoire les propriétés des objets les plus fréquemment requis.

La **sécurité** de l'ensemble est assurée par les méthodes *Sys_Acl*, *Sys_VerifAcl* et *Sys_Crypt* qui gèrent le cryptage, l'identification, l'authentification, les privilèges et les autorisations d'accès. La fiabilité de ces méthodes pourra être simplement renforcée par l'usage de protocoles de communication sécurisés (SSL). Elle pose notamment les bases d'un mécanisme d'authentification unique.

---

[33] Le BOA est de toute façon dépendant de l'ORB (d'où la spécification en cours d'un P*ortable* OA indépendant, dans CORBA).

[34] *Broker* veut dire courtier ou brocanteur en Anglais. La traduction « négociateur » usuelle est un peu abusive.

Un ensemble de *Documents* de publication (*Adm_Keys*), de mise à jour ou de destruction (*Adm_Maj*) gèrent le **cycle de vie** des *Documents* et effectuent l'administration de l'ensemble en fonction des autorisations établies (*cf.* Annexe 6.6.3.2, p.229).

Il est important d'offrir aux utilisateurs des moyens d'être informés de la réalisation de certains **événements**, dans une sélection effectuée par les administrateurs. Nous avons implémenté en l'occurrence plusieurs mécanismes de notification (*cf.* Annexe 6.6.3.1, p.225) qui s'appuient sur un schéma classique *push-pull* :

1. la notification active (*push*) propose à l'utilisateur de souscrire un abonnement et l'informe, par messagerie, de la réalisation des événements relatifs à l'abonnement en cours.

   AMI (**A**bonnement aux **M**odifications de l'**I**ntranet) permet à l'utilisateur identifié de demander à être notifié par messagerie lors de la modification d'un ensemble de *Documents* (ceux ayant les propriétés *New* et *Modif*).

2. la notification passive (*pull*) offre à l'utilisateur un point d'entrée qui lui permet de suivre l'évolution d'une situation donnée.

NOURIR (**NOU**veautés sur le **R**éseau **I**ntranet de la **R**ATP [35]) propose à l'utilisateur la liste, ordonnée par date, des nouveautés ou modifications intervenues sur une sélection de serveurs.

RIME (**R**echerche sur l'**I**ntranet par **M**ots-cl**E**fs) exploite l'indexation incrémentale réalisée par *Sys_Maj* pour mettre à sa disposition la liste, triée par pertinence décroissante [36], de *Documents* correspondants à des critères de recherche saisis.

Si les services décrits ici sont implémentés par des méthodes et scripts CFML, l'assemblage des différents composants logiciels induit à son tour des propriétés réalisées alternativement par la base de données, par le métalangage ou alors par le système d'exploitation sous-jacent.

La **concurrence** des accès aux serveurs et aux bases de données est intégralement prise en charge par l'interface de *Cold Fusion* qui gère les verrous sur les données ainsi qu'une file d'attente de requêtes.

Des **transactions** simples (pas de transactions imbriquées ou sous-transactions) sont intrinsèquement gérées par le SGBD entre ses différentes tables.

La synchronisation des horloges des serveurs (gestion du **temps**) peut, quant à elle s'appuyer sur une quelconque méthode standardisée telle *NTS* ou alors utiliser un processus spécifique (synchronisation sur une horloge radio, CGI, *etc.*). Il est entendu que la répartition des serveurs sur un même fuseau horaire simplifie les traitements.

---

[35] Respectivement, **NOU**veautés sur le **R**éseau **I**ntranet de l'université **R**ené Descartes.

[36] C'est le moteur de recherche (*Search'97* [Verity_96]) qui gère les critères de pertinence selon des algorithmes spécifiques.

4.8.1.2 « Objets Applicatifs » à la RATP

La mise en place de versions « intranet » de quelques applications d'entreprise s'est faite jusqu'ici avec une simplicité déconcertante. Il s'est en effet agi, dans une majorité de cas, d'applications client-serveur qui ont été développées par ailleurs autour d'un SGBDR (généralement ORACLE).

Il a alors suffi de s'inspirer des requêtes SQL déjà écrites pour les intégrer, en tant que *Documents*, dans le schéma de navigation défini pour l'intranet.

En l'occurrence, notre première intégration (mai 1996) a concerné l'annuaire d'entreprise **ANTILOPE** (**AN**nuaires **T**éléphoniques **I**nformatisés des **L**ieux, **O**rganisations et **P**ersonnes de l'**E**ntreprise), une application basée autour d'ORACLE 7.2/HP-UX (client développé en Visual Basic). Le développement de cet interface a été effectué en quelques heures et une version intranet d'Antilope était disponible sur l'intranet quelques semaines avant le déploiement du premier poste client (*cf.* Annexe 6.6.4.1, p.234).

Comme, par ailleurs, la RATP a décidé de limiter les accès de ses agents à l'Internet et l'a soumis a une autorisation hiérarchique préalable, nous nous sommes basé sur une corrélation entre cette table d'autorisations et Antilope pour permettre aux utilisateurs de créer leurs comptes intranet. Cette méthode de création automatique de compte n'offre bien entendu pas de grandes garanties de sécurité, dans la mesure où l'identité d'un utilisateur peut être usurpée par quiconque ayant accès à son poste de travail. Elle s'avère cependant *a priori* suffisante actuellement.

Un ensemble d'autres applications ont été intégrées aussi simplement à l'intranet de la RATP. Nous nous contentons ici d'en citer quelques unes, pour illustrer la diversité des sujets traités :

- **95 PPP** : la RATP relève d'un régime particulier de la sécurité sociale et dispose de plusieurs centres de soins regroupant les différentes spécialités. L'offre médicale peut notamment être consultée en interne à partir d'un Minitel, en composant le 95 (PPP étant le département des Prestations sociales).

  La réalisation d'une interface intranet pour interroger la base *ORACLE 7.2/SV-R4* a été effectuée par un stagiaire en quelques semaines (*cf.* Annexe 6.6.4.3, p.238);

- **MERCURE** est une application client-serveur sous *ORACLE 7.2/HP-UX*, qui assure le suivi des courriers « minute » des départements. Un prototype intranet a été réalisé par un stagiaire en juillet 1996. Ce prototype n'a pas donné lieu à une extension dans l'entreprise, à la suite de l'absence, à cette époque, du déploiement d'un navigateur dans l'ensemble des départements ;

- Ce même stagiaire a, par ailleurs procédé à deux prototypages d'une sélection d'**IG** (**I**nstructions **G**énérales, des textes réglementaires) pour comparer les mérites respectifs d'une passerelle propriétaire vers l'intranet, interfacée à RICERCAR et d'une solution directement intégrée au système.

- **Revue de Presse** : mise à la disposition des agents d'une revue de presse quotidienne (en remplacement progressif de la version « papier » - 1800 exemplaires quotidiens, 25 pages en moyenne, début 1997) ;
- Différents **infocentres** (*cf.* Annexe 6.6.4.2, p.237) sont actuellement en cours d'interfaçage par leurs chefs de projets respectifs.

  La simplicité de développement de ces interfaces a, d'une part, inspiré la mise en place d'une documentation automatique de ces infocentres (générée à partir de bases Access). Elle a, d'autre part, permis de générer des scripts d'accès Unix ou SQL et de les présenter sous forme de pages HTML dynamiques ;
- *etc.*

4.8.1.3 « Objets Applicatifs » à l'Université René Descartes

Nous avions sans nul doute préjugé de nos moyens en pensant être capable de consacrer l'été 97 à poursuivre nos activités à la RATP **et** à compléter la rédaction de cette Thèse **et** à préparer simultanément, avec les ingénieurs de l'UFR, les méthodes nécessaires pour que les étudiants de deuxième cycle disposent d'un espace de publication Web, à partir d'un serveur RICERCAR.

En l'occurrence, notre ambition initiale s'est pour le moment limitée à exploiter la base d'annuaire de l'UFR de Mathématiques et Informatique (MS-Access 97) que nous avons aidé à publier selon une procédure semblable à Antilope (*cf.* Annexe 6.6.4.4, p.240).

De même, nous nous sommes appuyés sur cette base pour mettre au point des méthodes d'authentification qui, dans ce cas précis, créent automatiquement les comptes sur le domaine Windows NT, en cours de déploiement.

Par ailleurs ces procédures devraient être reprises à l'identique pour instancier un autre serveur RICERCAR, au laboratoire *Cognicom* (créé au 1/1/1998, à l'UFR de Psychologie, sous la direction d'Olivier Houdé).

En tout état de cause, ces développements, qui commenceront par l'enrichissement de l'annuaire, n'interviendront pas avant le second semestre universitaire.

5. Conclusion

Je vois bien que vous attendez une péroraison ; mais en vérité vous vous trompez bien fort, si vous croyez que j'ai gardé dans ma mémoire tout le verbiage que je viens de vous débiter. Les Grecs disaient autrefois : Je hais un convive qui a trop bonne mémoire *; et moi je vous dis à présent :* Je hais un auditeur qui se souvient de tout*. Adieu donc, illustres et chers amis de la Folie, applaudissez-moi, portez-vous bien et divertissez-vous.*

Erasme
(Éloge de la folie)

5.1 Quo Vadis ?

Le fonctionnement et les performances actuels de RICERCAR, après deux ans d'expérimentation et d'amorçage [37], sont encourageants. De nombreux modules et applications d'entreprise ont été intégrés à moindre coût sous forme de méthodes qui interfacent le noyau fonctionnel commun (modularité et réutilisation du code) et sans impact mesurable sur les temps de calcul.

Il est sans doute remarquable de noter que notre démarche a recueilli des réactions très diverses en fonction des profils des interlocuteurs [Welsh_97]:

- Les « non professionnels » ont rapidement réalisé que les questions techniques avaient été appréhendées et qu'il leur était possible de focaliser leur attention sur les questions essentielles : comment publier un document, comment le mettre à jour, comment en limiter les accès, quel sens a un réseau d'entreprise, quelle nécessité pour une charte graphique ? *etc.* ;
- Les « professionnels » (informaticiens ou responsables du département informatique) se sont, quant à eux, répartis en deux groupes :
  1. une partie de l'encadrement supérieur et des techniciens « pointus » se sont retrouvés dans leur méfiance vis-à-vis de l'architecture orthodoxe (pour la RATP) proposée :
     - les premiers (à quelques exceptions notables près), imprégnés par les discours sur la simplicité de l'Internet et de ses solutions, n'ont d'abord pas admis que l'intranet allait structurer l'informatique de l'entreprise, donnant un coup de fouet au client-serveur, et qu'il fallait, pour en éviter l'éclatement, mettre en place une **infrastructure de services**, administrée et exploitable ;
     - les seconds, ayant suivi l'évolution des technologies issues de l'Internet et ayant le réflexe d'associer une arborescence physique aux protocoles FTP et HTTP, ont eu tôt fait de souligner les contraintes inhérentes à la mise en **cohérence du système**, comme un frein à l'expansion de l'intranet ; cohérence jugée par ailleurs inutile puisqu'inexistante sur l'Internet ;

[37] Le noyau du système est passé en près de deux ans - de la V.1 à la V.5.4 - par trois amorçages, effectués en une vingtaine de versions intermédiaires pour préserver les performances d'ensemble. Parallèlement, le code est passé de 80 Ko à 40 Ko.

2. les informaticiens issus de disciplines « non TCP/IP » ont généralement accueilli le système avec réserve dans un premier temps, déroutés par le hiatus entre la simplicité de mise en place d'un serveur Web, tant déclamée dans la presse spécialisée, et la complexité apparente de RICERCAR.

   Ce n'est que dans un second temps que les avantages d'une solution distribuée et cohérente leur est apparue dans le cadre de l'intranet. L'intégration des modèles et des feuilles de style a notamment été appréciée par les développeurs, satisfaits de pouvoir publier rapidement et dynamiquement le contenu de leurs bases de données, sans devoir gérer individuellement toutes les options de présentation de ces données.

Des nombreuses améliorations ont d'ores et déjà été réalisées par plusieurs stagiaires pendant ces deux années [38], notamment en termes d'outils d'administration et de publication. Il reste néanmoins à en améliorer l'ergonomie et aussi à élaborer des statistiques plus sophistiquées que celles actuellement disponibles.

Des évolutions plus fondamentales sont bien entendu souhaitables dans un domaine où pratiquement tout reste à faire. Le lecteur, familier des conclusions traditionnelles de thèses s'attend, à ce moment de notre discours, à l'inévitable liste de modifications et de « développements que l'auteur n'a pas faits et qu'il est bien sûr de ne jamais faire » [Pitrat_90, p.12].

Aussi, ne pouvant nous soustraire à cette tradition (la conclusion) mais ne voulant pas nous engager dans des projets dont nous ne maîtrisons pas la réalisation, nous nous limiterons à citer quelques pistes - qui nous intéressent et auxquelles nous avons déjà réfléchi - que nous ne pourrons pas mener de front seul. Leur concrétisation reste, de toutes façons, subordonnée à l'accord de la RATP ou pourra éventuellement rentrer dans le cadre d'expérimentations communes avec l'Université René Descartes :

- l'évolution vers un des grands standards que sont CORBA (ou même DCOM) devient envisageable techniquement et ne devrait pas poser de difficultés majeures (*cf.* § 4.8, p.150).
  Nous avons dû la différer jusqu'ici à la suite de l'absence d'outils de développement de haut niveau mais aussi par manque de vision stratégique claire de l'entreprise (si un ORB est désormais inclus dans le navigateur de *Netscape*, *Microsoft* fait quant à lui le forcing pour imposer COM/DCOM sur l'ensemble des serveurs et des postes de travail de l'entreprise, souvent acquise à ses solutions).

  Il semblerait toutefois que notre contribution autour de bus logiciels commence à porter des fruits. Un groupe de travail a été récemment créé, qui évoque CORBA dans ses réflexions préliminaires ;

[38] Des automates ont ainsi été écrits pour convertir une arborescence traditionnelle HTML en *Documents* RICERCAR et, respectivement, pour convertir en HTML et publier - ou mettre à jour - des documents Word, envoyés par messagerie à une BAL d'administration.

- la détermination d'un modèle de réplication des *données* de RICERCAR (visant à dupliquer les documents sur un ensemble pertinents de serveurs) figure parmi nos préoccupations. Il s'agit d'assurer :

  1. la minimisation du trafic sur le réseau : servir à l'utilisateur le document demandé à partir du serveur *topologiquement* le plus proche (la *distance* étant à déterminer) ;

  2. une tolérance aux pannes et perturbations du réseau : être capable de pallier la défaillance d'un serveur en répartissant les accès qui lui sont destinés vers un ensemble d'autres serveurs.

Des problèmes complexes de synchronisation des données doivent cependant être résolus. Il faut notamment pouvoir déterminer quel exemplaire du document peut être mis à jour par l'utilisateur, quels mécanismes de réplication utiliser en cas de modification, quelles *valuations* pour déterminer le serveur optimal (*distance* minimale) par rapport au couple (utilisateur, document demandé), *etc.*

Par ailleurs, la redondance des informations contenues dans la *méta*-base et dans les réplications locales est théoriquement suffisante pour permettre de reconstituer un des serveurs du réseau ou, au besoin, de se substituer à chacun des serveurs le constituant ;

- faire évoluer les bases de données actuelles vers un ensemble d'annuaires X.500 [CCITT_X.500] ou LDAP [Yeong_95, Howes_97] qui permettront d'utiliser les mécanismes intrinsèques de réplication des données (en résolvant une partie des problèmes énoncés au point précédent) ainsi que les différents protocoles de requête prévus entre DSA/Serveurs LDAP ;
- « Un programme d'Intelligence Artificielle qui a de bons résultats n'est pas intéressant... » [Pitrat_84] [39].

Le développement incrémental de RICERCAR s'est imposé dans le cadre d'une recherche essentiellement effectuée en entreprise où il n'est pas acceptable de proposer un système opérationnel - fût-il *intéressant* - dont les performances soient médiocres. Nous avons donc finalement dû limiter notre ambition à un système *intéressant* **et** ayant de *bons résultats* : ce n'est donc pas - encore - un programme d'*Intelligence Artificielle.*

Les amorçages effectués ont cependant permis de construire RICERCAR comme un système *multi-agents* [Tisseau 96] auquel il manque la *réflexivité opératoire* [Ferber_89].

Les serveurs sont intrinsèquement des *objets distribués*, dont le dictionnaire remplit déjà le rôle d'un *tableau noir* [Ermann_80] embryonnaire.

L'index fédéré - lui-même un *tableau noir* - en est le *dispositif de contrôle* [Hayes-Roth_85].

[39] J.Pitrat explique qu'un programme d'IA performant est généralement procédural sans que ces procédures soient issues d'une compilation de métaconnaissances (faite par le système lui-même), d'où l'intérêt relatif des « bonnes » performances.

Les *Documents* ainsi que les applications (des *expertises*, quoique totalement procédurales) qui y sont hébergés sont des *agents* qui peuvent *collaborer* ou entrer en *concurrence* dans la *résolution* de leurs buts en échangeant *des messages asynchrones* [40] *avec continuation* [Briot 89] (les redirections d'URL - en-têtes HTTP 302 - paramétrées) : génération, sélection, publication, déplacement, destruction, optimisation, *etc.*

S'il ne fait pas de doute que cette dernière piste semble la plus prometteuse (et de plus longue haleine, ce qui obère ses chances d'être réalisée à la RATP), il est aussi probable que la résolution des problèmes de réplication et d'optimisation que nous nous sommes posés relèvera d'un *amorçage* plus général qui mettra en perspective, en termes de solutions et d'outils informatiques, des technologies théoriquement germaines - Bases de Données Réparties, Annuaires, Intelligence Artificielle Distribuée - mais actuellement disparates.

Par delà des technologies, un intranet cimente une architecture stratégique et structurante. Il apporte, dans une première approche, une palette de solutions à un ensemble de questions posées depuis de nombreuses années dans une grande organisation décentralisée. Il ne fait pas de doute que l'explosion des sites Web fait entrevoir ce que pourrait être un *Système de l'Information*. Il est cependant important que nous réalisions rapidement, après l'effervescence naturelle liée à la découverte de nouveaux *media* de communication, une séparation entre le monde libre et heureusement anarchique de l'Internet où chaque îlot représente une communauté indépendante et celui, plus strict, de la communication d'entreprise qui doit assurer un minimum d'homogénéité.

Or, après de nombreuses années de technicité moindre, la capacité à construire ou même à valider des architectures s'est amenuisée et l'entreprise, comme tant d'autres, a lentement fini par déléguer à ses fournisseurs et à ses consultants la tâche essentielle de bâtir son Système d'Information.

Cette thèse se place aussi dans le cadre d'un amorçage autour d'une stratégie non plus basée sur des choix d'outils mais aboutissant à un *Système de l'Information*, réparti et cohérent. Notre contribution à ce schéma, focalisée autour de l'intranet, aura montré qu'une réflexion systématique peut adresser les différentes facettes de sa gestion et que l'ensemble des modules mis en place dans RICERCAR crée une synergie qui permet de disposer finalement d'une solution « complète », modulaire et facilement adaptable à un ensemble de situations non déterminées à l'avance.

[40] Les messages IIOP peuvent être asynchrones, à la différence des requêtes HTTP.

6. Annexes

« Encore une fois, nous avons besoin de cohérence et de contraintes adaptées pour créer une industrie de composants florissante. Seul le consommateur a la liberté du choix ; les développeurs doivent en être exemptés »

[Orfali_96, p.525]

6.1 Bibliographie *165*

6.2 Glossaire *179*

6.3 La RATP *185*

6.4 Rapport du MCD *195*

6.5 Le code des méta-scripts de RICERCAR *208*

6.6 Exemples de Documents générés par RICERCAR *209*

6.7 Table des figures *243*

6.8 Récapitulatif et index des citations in extenso de Ricercar, constituant des auto-références *245*

6.1 Bibliographie

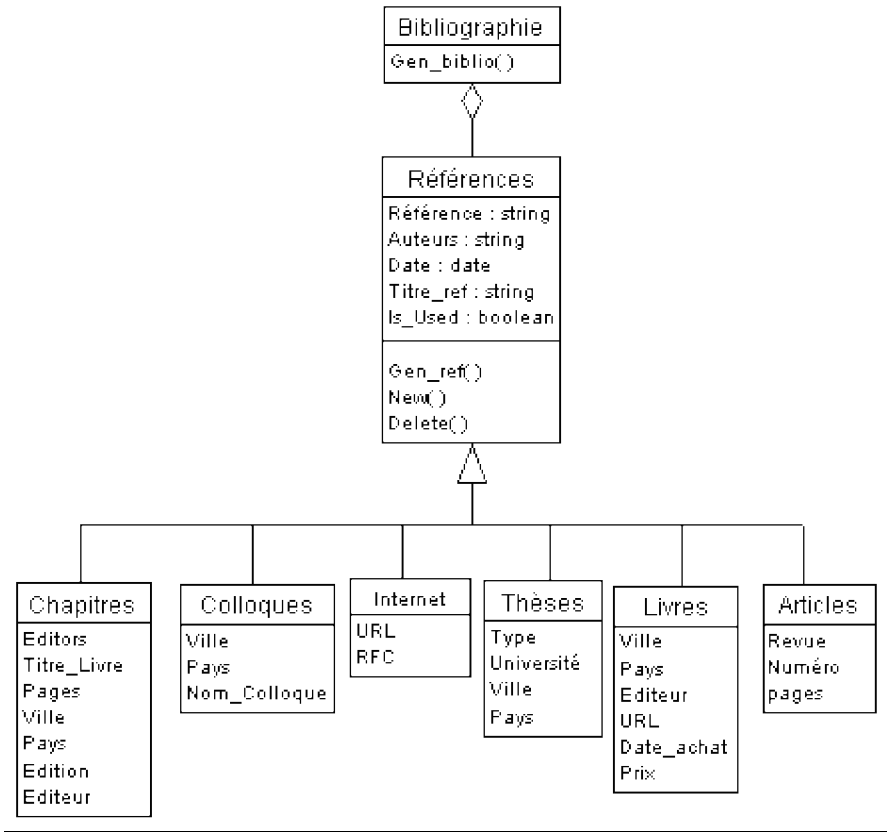


Figure 5. 1. Diagramme des Classes de la Bibliographie.

[Abiteboul_89] Abiteboul S., Kanellakis P., *Object Identity as a Query Language Primitive* , Rapport de Recherche de l'INRIA, n° 1022, Avril 1989.

[Adams_97] Adams S., *Le principe de Dilbert* , First Éditions, Lonrai, France, Mars 1997.

[Alin_97] Alin F., Lafont D., Macary J.-F., *Le projet intranet: de l'analyse des besoins de l'entreprise à la mise en œuvre des solutions* , Eyrolles, Paris, France, 1997.

[Allaire_97a] Allaire J., *Introducing COLD FUSION 3.0* , Juillet 1997 , <*RURL:allaire*> .

[Allaire_97b] Allaire J., *COLD FUSION 3.1 User Guide* , 1997 , *<RURL:allaire_guide>* .

[Archimbaud_95] Archimbaud J.-L., *L'Internet professionnel* , CNRS Editions, Paris, France, 1995.

[Augendre_97] Augendre M., *Un enjeu pour les organisations* , Sciences Humaines, n° 16, Mars 1997, pp.42-45.

[Bach] Bach J.-S., *Musikalisches Opfer*, Potsdam, Allemagne, 7 mai 1747.

[Barthes_90] Barthes J.-P., Ferber J., Gloess P.Y., Nicolle A., Volle P., *Objets et Intelligence Artificielle* , Troisièmes journées du PRC_IA, Mars 1990, Paris, France.

[Bengrid_96] Bengrid Y., Seret D., *Interopérabilité des applications dans un environnement transactionnel réparti environnement transactionnel réparti* , CNRS RenPar8, Juin 1996, Bordeaux, France.

[Berger_97] Berger P., *L'intranet, un chantier social* , Le Monde Informatique, n° 728, 27 Juin 1997, p.16.

[Berners-Lee_89] Berners-Lee T., *Information Management: A Proposal* , CERN, Suisse, Genève, Mars 1989.

[Berners-Lee_90] Berners-Lee T., Cailliau R., *World Wide Web for a HyperText project.* , CERN, Suisse, Genève, 1990.

[Berners-Lee_94a] Berners-Lee T., *Universal Resource Identifiers in WWW* , Juin 1994 , Internet RFC-1630, *<URL:http://www.internic.net/rfc/rfc1630.txt>* .

[Berners-Lee_94b] Berners-Lee T., Masinter L., McCahill M., *Universal Resource Locators (URL)* , Décembre 1994 , Internet RFC-1738, *<URL:http://www.internic.net/rfc/rfc1738.txt>*.

[Berners-Lee_95] Berners-Lee T., Connolly D., *Hypertext Markup Language - 2.0.* , Novembre 1995 , Internet RFC-1866, *<URL:http://www.internic.net/rfc/rfc1866.txt>*.

[Berners-Lee_96] Berners-Lee T., Fielding R., Frystyk H., *Hypertext Transfer Protocol -- HTTP/1.0* , Mai 1996 , Internet RFC-1945, *<URL:http://www.internic.net/rfc/rfc1945.txt>*.

[Biggs_97] Biggs M., *Cold Fusion keeps getting better* , InfoWorld, n° 30, Vol.19, 28 Juillet 1997.

[Bitouzet 97] Bitouzet C., Fournier P., et Montcel B.T, *Management et intranet* , Hermès, Paris, France, Février 1997.

[Black_94] Black A., Walpole J., *Objects to the rescue! Or httpd: the next generation operating system* , ACM-SIGOPS 94, Sixth SIGOPS European Workshop, (Editeur), Septembre 1994, ACM Press, Wadern, Allemagne.

[Boehm_88] Boehm B., *A spiral Model of Software Development and Enhencement* , IEEE Computer 21, n° 5, Mai 1988, pp.61-72.

[Booch_91] Booch G., *Object Oriented Analysis and Design with Applications* , Benjamin/Cummings, Redwood City, U.S.A, 1991.

[Booch_95a] Booch G., *Object Solutions* , Addison Wesley, Redwood City, U.S.A, 1995.

[Booch_95b] Booch G., Rumbaugh J., *Unified Method for Object Oriented Development* , Documentation Set Version 0.8, Octobre 1995, Santa-Clara, U.S.A.

[Borenstein_93] Borenstein N., Freed N., *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies* , Septembre 1993 , Internet RFC-1521, <*URL:* http://www.internic.net/rfc/rfc1521.txt>.

[Bossard_97] Bossard Consultants, *Intranet et l'entreprise : Tout ce qui va vous empêcher de faire de l'INTRANET* , Communication Privée, Mars 1997, Paris, France.

[Bouzeghoub_94] Bouzeghoub M., Gardarin G., Valduriez P., *Objets: Du C++ à Merise Objet* , Eyrolles, Paris, France, 1994.

[Bouzeghoub_97] Bouzeghoub M., Gardarin G., Valduriez P., *Les Objets : Édition revue et augmentée* , Eyrolles, Paris, France, 1997, (1ère Ed. 1994).

[Brando_95a] Brando T., *Interoperability and the CORBA Specification* , The MITRE Corporation, Février 1995, Bedford, U.S.A.

[Brando_95b] Brando T., *Comparing DCE and CORBA* , The MITRE Corporation, Mars 1995, Bedford, U.S.A.

[Bray_97] Bray T., Paoli J., Sperberg-McQueen C., *Extensible Markup Language (XML)* , 7 Août 1997 , W3C Working Draft, <*URL:* http://www.w3.org/pub/WWW/TR/WD-xml.html>.

[Briot_87] Briot J.-P., Cointe P., *A Uniform Model for Object-Oriented Languages Using the Class Abstraction* , IJCAI, 1987, Milano, Italy.

[Briot_89] Briot J.-P., *Actalk : a Testbed for Classifying and Designing Actor Languages in the Smalltalk-80 Environment* , Proceedings of ECOOP'89, 1989, Notthingham, UK.

[Brown_96] Brown N., Kindel C., *Distributed Component Object Model Protocol – DCOM/1.0* , Novembre 1996 , draft-brown-dcom-v1-spec-01 (expiré), <*URL:*: *http://www.internic.net/internet-drafts/draft-brown-dcom-v1-spec-01.txt*>.

[Bryan_88] Bryan M., *SGML -- An Author's Guide to the Standard Generalized Markup Language* , Addison Wesley, Massachussetts, U.S.A, 1988.

[Bush_45] Bush V., *As We May Think* , The Atlantic Monthly, Boston, U.S.A, Juillet 1945.

[Cattel_94] Cattel R., Kaufmann M., *The Object Database Standard: OMDG-93* , OMG, San Francisco, U.S.A, 1994.

[CCITT_X.500] CCITT, *Réseaux de communications de données : Annuaire* , Novembre 1988 , CCITT Recommandations X.500 à X.521.

[Chauvet_97] Chauvet J.-M., *Corba, ActiveX et Java Beans* , Eyrolles, Paris, France, Mars 1997.

[Chen_76] Chen P., *The Entity-Relationship Model - Towards a Unified View of Data*, ACM Transactions of Database Systems, n°1, Vol. 1, Mars 1976.

[Coad_91] Coad P., Yourdon E., *Object Oriented Analysis* , Prentice Hall, Englewood Cliffs, U.S.A, 1991.

[Codd_70] Codd E., *A Relational Model of Data for Large Shared Data Banks* , Communications ACM, n° 6, Vol.13, 1970, pp.377-387.

[Cointe_85] Cointe P., *Implémentation et interprétation des langages objet - Application aux langages SmallTalk, ObjVlisp et Formes* , Thèse de Doctorat d'État, Université Paris 6, 1985, Paris, France.

[Colan_97] Colan M., *InfoBus Specification* , 21 Août 1997 , Draft Version 0.04A, <*URL:* *http://java.sun.com/beans/infobus/ibspec.html*>.

[Collongues_89] Collongues A., Hugues J., Laroche B., *MERISE : méthode de conception* , Dunod, Paris, France, 1989.

[Comer_96] Comer D., *TCP/IP : Architecture, protocoles, applications* , InterÉditions, Paris, France, 1996, (1ère Ed. 1990, Prentice Hall).

[Corkill_97] Corkill D., *Countdown to Success : Dynamic Objects, GBB, and RADARSAT-1*, Communications ACM, n° 5, Vol. 40, Mai 1997, pp.49-58.

[Coulouris_95] Coulouris G., Dollimore J., Kindberg T., *Distributed Systems : Concepts and Design (Second Edition)* , Addison-Wesley, Londres, Angleterre, 1995, (1ère Ed. 1990).

[Curtis_97] Curtis D., *Java, RMI and CORBA* , 1997 , <*URL:* http://www.omg.org/news/wpjava.htm>.

[Custer_93] Custer H., *Inside Windows NT* , Microsoft Press, Redmond, U.S.A, 1993.

[Davis_97] Davis D., *Time to choose your objects* , Datamation, Mars 1997.

[Denny_95] Denny B., *Windows httpd* , Mai 1995 , <*URL:* http://tech.west.ora.com/win-httpd/>.

[Dhénin_97] Dhénin C., *De l'approche statique à l'accès dynamique : Comment faire cohabiter Web et base de données* , Le Monde Informatique, n° 719, 25 avril 1997, pp.22-23.

[Dilley_95] Dilley J., *OODCE : A C++ Framework for the OSF Distributed Computing Environment* , Proceedings of the Winter Usenix Conference, Usenix Association, Janvier 1995, U.S.A.

[Dreyfus_97] Dreyfus P., *CORBA : Theory and Practice* , Juin 1997 , <*URL:* http://developer.netscape.com/news/viewsource/dreyfus_corba.html>.

[DSA_91] Danish Standards Association, *SGML - ODA : Présentation des concepts et comparaison fonctionnelle* , AFNOR, Paris, France, 1991, (1ère Ed. 1989, Dansk Standardiseringsrad).

[Dufour_96] Dufour A., *Internet* , PUF, Paris, France, Février 1996, (1ère Ed. 1995).

[Edwards_94] Edwards N., Rees O., *Object Wrapping (for WWW) - The key to Integrated Services*, 1994, <*URL:* http://www.ansa.co.uk/ANSA/ISF/1464/1464prt1.html>.

[Ermann_80] Ermann L.D., Hayes-Roth F., Lesser V.R. and Reddy D.R., *The Hearsay-II speach understanding system : Integrating knowlegde to resolve uncertainty* , Computing Surveys, n°12, 1980, pp.213-253.

[Ferber_89] Ferber J., *Objets et agents : une étude des structures de représentation et de communications en Intelligence Artificielle*, Thèse de Doctorat d'État, Université Paris 6, 8 Juin 1989, Paris, France.

[Ferber_95] Ferber J., *Les Systèmes Multi-Agents : vers une intelligence collective* , InterÉditions, Paris, France, 1995.

[Fielding_95] Fielding R., *Relative Uniform Resource Locators* , Juin 1995 , Internet RFC-1808, <*URL:* http://www.internic.net/rfc/rfc1808.txt>.

[Fielding_97] Fielding R., Gettys J., Mogul J., Frystyk H., Berners-Lee T., *Hypertext Transfer Protocol -- HTTP/1.1* , Janvier 1997 , Internet RFC-2068, <*RURL :*rfc2068>.

[Flanagan_96] Flanagan D., *Java in a Nutshell* , O'Reilly & Associates, Inc., Sebastopol, California, 1996.

[Flynn_97] Flynn J., Clarke B., *ActiveX unmasked* , Datamation, Janvier 1997.

[Franks_97] Franks J., Hallam-Baker P., Hostetler J., Leach P., Luotonen A., Sink E. and Stewart L., *An Extension to HTTP : Digest Access Authentication* , Janvier 1997 , Internet RFC-2069, <*RURL:*rfc2069>.

[Frey_97] Frey A., Bel B., *Les ORB font dialoguer les objets* , Informatiques Magazine, n° 30, 1 Juin 1997, pp.71-77.

[Gardarin_89] Gardarin G., Cheiney J.-P., Kiernan G., Pastre D., Stora H., *Managing complex objects in an extensible relational DBMS* , Rapport de Recherche de l'INRIA, n° 981, Mars 1989.

[Gardarin_90] Gardarin G., Valduriez P., *SGBD Avancés : Bases de données objets, déductives, réparties* , Eyrolles, Paris, France, 1990.

[Gartner_97] Gartner Group, *Object References and Relational Data* , Research Note, n° T-401-1512, 16 Juillet 1997.

[Goldberg_83] Goldberg A., Robson D., *SMALLTALK-80 : the language and its implementation* , Addison Wesley, Massachussetts, U.S.A, 1983.

[Goldfarb_90] Goldfarb C., *The SGML Handbook* , Oxford University Press, Oxfors, U.K, 1990.

[Greiner_80] Greiner R., Lenat D., *RLL : A Representation Language Language* , Proceedings of AAAI-80, 1980, Stanford, U.S.A.

[Grévisse_88] Grévisse M., *Le bon usage : grammaire française* , Duculot, Paris, France, 1988, (12ème Ed. Refondue par Goosse A.).

[Gulesian_97] Gulesian M., Gulesian P., *Uniting Object-Oriented and Distributed Systems* , DBMS, n° 8, Vol.10, Juillet 1997, pp.84-91.

[Gundavaram_96] Gundavaram S., *CGI Programming on the World Wide Web* , O'Reilly & Associates, Inc., Sebastopol, California, 1996.

[Hart_97] Hart T.J., *Questioning CORBA* , DBMS, n° 3, Vol.10, Mars 1997, pp.52-53.

[Hayes-Roth_85] Hayes-Roth F., *A Blackboard Architecture for Control* , Readings in Distributed Artificial Intelligence, A.H. Bond and L. Gasser (Editeur), 1985, Morgan Kaufmann Publishers, Inc., San Mateo, California.

[Ho_97] Ho. B, *Cold Fusion 3.0 : empower your Web apps* , 22 Juillet 1997 , CNET, <URL: http://www.cnet.com/Content/Reviews/JustIn/Items/0,118,189,00.html?builder>.

[Hofstadter_85] Hofstadter D., *Gödel, Escher, Bach : Les Brins d'une Guirlande Éternelle* , InterÉditions, Paris, France, 1985.

[Hower_97] Hower R., *Beyond Broken Links* , DBMS, n° 8, Vol.10, Juillet 1997, pp.S9-S11.

[Howes_97] Howes T., Smith M., *LDAP : Programming Directory-Enabled Applications with Lightweight Directory Access Protocol* , Macmillan Technical Publishing, Indianapolis, Indiana, 1997.

[IBM_81] IBM Corporation, *SQL/Data System Concepts and Facilities* , 1981 , GH 24-5013D.

[ihtml_97] Inline HTML, *Making Dynamic Web Sites Affordable* , 1997 , <URL: http://www.ihtml.com/>.

[Ion_97] Ion P., Miner R., *Mathematical Markup Language* , 10 Juillet 1997 , W3C Working Draft, <URL: http://www.w3.org/TR/WD-math/>.

[Kay_76] Kay A., Goldberg A., *SMALLTALK-72 Instruction Manual* , Technical Report SSL-76-6, 1976, Palo Alto, U.S.A.

[Kayser_98] Kayser D., *Complexité* , Vocabulaire de sciences cognitives (à paraître), Houdé O., Kayser D., Koenig O., Proust J., Rastier F. (Editeur), 1998, PUF, Paris, France.

[Keuffel_97] Keuffel W., *CORBA : Masterminds Object Management* , DBMS, n° 3, Vol.10, Mars 1997, pp.42-50;71.

[Kohl_93] Kohl J., Neuman C., *The Kerberos Network Authentication Service (V5)* , Septembre 1993 , Internet RFC-1510, <URL: http://www.internic.net/rfc/rfc1510.txt>.

[Kristol_97] Kristol D. and Montulli L., *HTTP State Management Mechanism* , Février 1997 , Internet RFC-2109, <RURL:rfc2109>.

[Kruchten_96] Kruchten P., *A Rational Development Process* , CrossTalk, n° 9, Vol. 7, Juillet 1996, pp.11-16.

[Kunze_95] Kunze J., *Functionnal Recommendations for Internet Resource Locators* , Février 1995 , Internet RFC-1736, <URL: http://www.internic.net/rfc/rfc1736.txt>.

[Laurière_76] Laurière J.-L., *Un langage et un programme pour énoncer et résoudre des problèmes combinatoires* , Thèse de Doctorat d'État, Université Paris 6, 21 Mai 1976, Paris, France.

[Leddy_93] Leddy W., Khanna A., *DCE++ : A C++ API for DCE"* , HaL document #030-00209, 31 Mars 1993, Austin, U.S.A.

[Lefebvre_97] Lefebvre A., *Intranet : client-serveur universel* , Eyrolles, Paris, France, 1997.

[Lie_96] Lie H.W., Bos B., *Cascading Style Sheets, level 1* , 17 Décembre 1996 , W3C Recommandation, <*URL:* http://www.w3.org/pub/WWW/TR/REC-CSS1>.

[Linthicum_97a] Linthicum D., *Distributed Objects Get New Plumbing* , DBMS, n° 1, Vol.10, Janvier 1997.

[Linthicum_97b] Linthicum D., *Application Architect : Mixing Tuples and Objects* , DBMS, n°12, Vol.10, Décembre 1997.

[Makpangou_97] Makpangou M., *Relais : Un protocole de maintien de cohérence de caches Web coopérants*, NOTERE97, Pau, France, Novembre 1997.

[Martin_97] Martin D., *La criminalité informatique* , PUF, Paris, France, Avril 97.

[McCool_94] McCool R., *The Common Gateway Interface* , 1994 , <*URL:* http://hoohoo.ncsa.uiuc.edu/cgi/overview.html>.

[Merle_97] Merle P., *CorbaScript - CorbaWeb : Propositions pour l'accès à des objets et services distribués* , Thèse d'Université, Université Lille 1, Janvier 1997, Lille, France.

[Microsoft_92] Microsoft, *Microsoft ODBC SDK Preliminary Release* , Microsoft, Redmond, U.S.A, Mars 1992.

[Microsoft_93a] Microsoft, *Object Linking and Embedding Version 2 (OLE2) Programmer's Reference, Volumes 1 and 2* , Microsoft Press, Redmond, U.S.A, 1993.

[Microsoft_93b] Microsoft, *Microsoft Win32 Programmer's Reference* , Microsoft Press, Redmond, U.S.A, 1993.

[Miller_96] Miller J., Krauskopf T., Resnick P., Treese W., *PICS Label Distribution Label Syntax and Communication Protocols* , 31 Octobre 1996 , W3C Recommendation, <*URL:* http://www.w3.org/pub/WWW/TR/REC-PICS-labels-961031.html>.

[Minsky_74] Minsky M., *A Framework for representing Knowledge* , The Psychology of Computer Vision, Winston P. (Editeur), 1974, Mc Graw Hill, New York, U.S.A.

[MIT_97] M.I.T, *Common Lisp Hypermedia Server* , 1997 , <URL: http://www.ai.mit.edu/projects/iiip/doc/cl-http/home-page.html>.

[Mowbray_95] Mowbray T., Zahavi R., *The Essential CORBA: Systems Integration using Distributed Objects* , John Wiley and Sons, New York, U.S.A, 1995.

[Muller_97] Muller P.-A., *Modélisation objet avec UML* , Eyrolles, Paris, France, 1997.

[Nebel_95] Nebel E., Masinter L., *Form-based File Upload in HTML* , Novembre 1995 , Internet RFC-1867, <URL: http://www.internic.net/rfc/rfc1867.txt>.

[Nelson_67] Nelson T., *Getting it out of our system* , Information Retrieval: A Critical Review, Schechter G. (Editeur), 1967, Thomson Books, Washington, U.S.A..

[Neumann_48] Neumann J., *La théorie générale et logique des automates* , Sciences Cognitives ! Textes Fondateurs (1943-1950), Pélissier A., Tête A. (Editeur), 1996, PUF, Paris, France.

[Newell_57] Newell A., Shaw J., Simon H., *Empirical Explorations of the Logic Theory Machine* , Proceedings of the Western Joint Computer Conference, 1957, Cambridge, U.S.A.

[Newell_61] Newell A., Simon H., *GPS, a program that simulates human thought* , Lernende Automaten, Billing H. (Editeur), 1961, Oldenbourg, Munich, RFA.

[Nicolas_95] Nicolas J., *Connexion de la RATP sur l'Internet* , Document interne à la RATP, 1995, Noisy-le-Grand, France.

[Nielsen_97] Nielsen H., Connolly D., Khare R., *PEP - an Extension Mechanism for HTTP* , 14 Juillet 1997 , W3C Working Draft, <URL: http://www.w3.org/TR/WD-http-pep.html>.

[NSCA_95] NCSA HTTPd Development Team, *Server Side Includes (SSI)* , 28 Septembre 1995, <URL: http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html>.

[OMG_91] Object Management Group, *The Common Object Request Broker: Architecture and Specification* , OMG, Massachussetts, U.S.A, Décembre 1991.

[OMG_92] Object Management Group, X/Open, *Object Management Architecture Guide, Revision 2.0* , OMG, Massachussetts, U.S.A, 1 Septembre 1992.

[OMG_93a] Object Management Group, *Object Services Architecture* , OMG, Massachussetts, U.S.A, Février 1993.

[OMG_93b] Object Management Group, X/Open, *The Common Object request Broker : Architecture and Specification* , OMG, Massachussetts, U.S.A, Décembre 1993.

[OMG_94a] Object Management Group, *IDL C++ Language Mapping Specification* , OMG, Massachussetts, U.S.A, Septembre 1994.

[OMG_94b] Object Management Group, *Common Object Services Specifications, Volume 1, Revision 1.0* , John Wiley and Sons, New York, U.S.A, Mars 1994.

[OMG_94c] Object Management Group, *Common Object Services Specifications, Volume 2* , John Wiley and Sons, New York, U.S.A, 1994.

[OMG_94d] Object Management Group, *Common Object Facilities Architecture* , OMG, Massachussetts, U.S.A, 1994.

[OMG_95a] Object Management Group, *IDL - Ada Language Mapping Specification (Version 1.0)* , OMG, Massachussetts, U.S.A, Mai 1995.

[OMG_95b] Object Management Group, *Common Object Services Specifications* , OMG, Massachussetts, U.S.A, 31 Mars 1995.

[OMG_96a] Object Management Group, *The Common Object Request Broker: Architecture and Specification. Revision 2.0* , OMG, Massachussetts, U.S.A, Juillet 1996, (1ère Ed. Juillet 1995).

[OMG_96b] Object Management Group, *CORBAservices* , OMG, Massachussetts, U.S.A, 1996.

[OMG_96c] Object Management Group, *CORBAfacilities* , OMG, Massachussetts, U.S.A, 1996.

[OMG_97] OMG, *Comparing ActiveX and CORBA/IIOP* , 1997 , *<URL:* http://www.omg.org/news/activex.htm>.

[Orfali_95] Orfali R., Harkey D., Edwards J., *Essential Distributed Objects Survival Guide* , John Wiley and Sons, New York, U.S.A, 1995.

[Orfali_96] Orfali R., Harkey D., Edwards J., *Objets Répartis : Guide de Survie* , International Thomson Publising, Paris, France, 1996, (Traduction de [Orfali_95]).

[Orfali_97a] Orfali R., Harkey D., *Client/server Programming with JAVA and CORBA* , John Wiley and Sons, New York, U.S.A, 1997.

[Orfali_97b] Orfali R., Harkey D., Edwards J., *Instant CORBA* , John Wiley and Sons, New York, U.S.A, 1997.

[OSF_93a] OSF, *OSF DCE Application Development Guide, Revision 1.0* , Prentice Hall, Englewood Cliffs, U.S.A, 1993.

[OSF_93b] OSF, *OSF DCE Application Development Reference, Revision 1.0* , Prentice Hall, Englewood Cliffs, U.S.A, 1993.

[Paepcke_90] Paepcke A., *PCLOS: Stress testing CLOS experiencing the metaobject protocol* , Proceedings of ECOOP/OOPSLA '90, 1990, New York, U.S.A.

[Peck_96] Peck S.B., Arrants S., *Building your own WebSite* , O'Reilly & Associates, Inc., Sebastopol, California, 1996.

[Peck_97a] Peck S., Helfand J., Walsh M.-J., *WebSite : Mastering The Elements* , O'Reilly & Associates, Inc., Sebastopol, Californie, Septembre 1997.

[Peck_97b] Peck S., Helfand J., Walsh M.-J., *WebSite : Creating Dynamic Content* , O'Reilly & Associates, Inc., Sebastopol, Californie, Septembre 1997.

[Pitrat_66] Pitrat J., *Réalisation de programes de démonstration de théorèmes utilisant des méthodes heuristiques* , Thèse de Doctorat, Faculté des Sciences, 1966, Paris, France.

[Pitrat_84] Pitrat J., *Quelques Remarques sur ' Intelligence- artificielle, mythes et limites '* , Intelligence -Artificielle, Mythes et Limites, Dreyfus H.L. (Editeur), 1984, Flamarion, Paris, France.

[Pitrat_85] Pitrat J., *MACISTE ou comment utiliser un ordinateur sans écrire de programme* , Colloque Intelligence Artificielle, 1985, Toulouse, France.

[Pitrat_86] Pitrat J., *Le problème du Bootstrap* , Colloque Intelligence Artificielle, 1986, Strasbourg, France.

[Pitrat_90] Pitrat J., *Métaconnaissance, Futur de l'Intelligence Artificielle* , Hermès, Paris, France, 1990.

[Pitrat_95] Pitrat J., *De la machine à l'intelligence* , Hermès, Paris, France, 1995.

[Pitrat_96] Pitrat J., *Ce titre contient quatre 'a', un 'b', cinq 'c', cinq 'd', dix-neuf 'e', deux 'f', un 'g', deux 'h', treize 'i', un 'j'...* , Publications du LAFORIA, n° 26, Septembre 1996.

[Pitrat_98] Pitrat J., *Métacognition* , Vocabulaire de sciences cognitives (à paraître), Houdé O., Kayser D., Koenig O., Proust J., Rastier F. (Editeur), 1998, PUF, Paris, France.

[Pujolle_88] Pujolle G., Schwartz M., *Réseaux Locaux Informatiques* , Eyrolles, Paris, France, 1998.

[Quillian_68] Quillian R., *Semantic Memory* , Semantic Information Processing, Minsky M. (Editeur), 1968, MIT Press, Cambridge, U.S.A.

[Ragget_97] Ragget D., Le Hors A., Jacobs I., *HTML 4.0 Specification* , 8 Juillet 1997 , W3C Working Draft, <*URL:* http://www.w3.org/TR/WD-html40-970708>.

[Robertson_92] Robertson P., *On reflection and refraction* , Proceedings of the 1992 International Workshop on New Models for Software Architecture, 1992.

[Robertson_97] Robertson P., *Integrating Legacy Systems with Modern Corporate Applications*, ACM Communications, n° 5, Vol. 40, Mai 1997, pp.39-46.

[Rosenberry_92] Rosenberry W., Kenney D., Fisher G., *Understanding DCE* , O'Reilly & Associates, Inc., Sebastopol, California, 1992.

[Rosenblueth_43] Rosenblueth A., Wiener N., Bigelow J., *Comportement, but et téléologie* , Sciences Cognitives ! Textes Fondateurs (1943-1950), Pélissier A., Tête A. (Editeur), 1996, PUF, Paris, France.

[Rumbaugh_96] Rumbaugh J., Blaha M. Eddy F., Premerlani W., Lorensen W., *OMT : Modélisation et conception orientées objet* , Masson, Paris, France, 1996, (1ère Ed. 1991, Prentice Hall).

[Sauteur_97] Sauteur B., *Intranet : le monde client-serveur enfin disponible !*, Informatiques Magazine, n° 33, 1 Septembre 1997, p.24.

[Schmidt_95] Schmidt D., Vinoski S., *Object Interconnexions : Modeling Distributed Object Applications* , C++ Report, Février 1995.

[Schmidt_97] Schmidt D., Vinoski S., *Object Adapters : Concepts and Terminology* , C++ Report, Octobre 1997.

[Schneider_97] Scheider B., Rosensohn N., *Télétravail : réalité ou espérance ?*, PUF, Paris, France, Février 97.

[Séraphin_97a] Séraphin J., *Réseau Intranet : Communication dans un Ensemble Réparti, Cohérent et Administré, à la RATP* , Expo Intranet, 30 Mai 1997, Paris, France.

[Séraphin_97b] Séraphin J., *Réaliser un Intranet : Construction d'un Ensemble Réparti et Cohérent autour d'une Architecture Réflexive* , NOTERE'97, Novembre 1997, Pau, France, <*RURL:* notere97>.

[Seret_94] Seret D., Zhang X.-X., *Network management with X500 Distributed services* , International Conference IFIP/ULPAA-94, (Editeur), Juin 1994, North-Holland, Barcelone, Espagne.

[Simon_91] Simon H., *Sciences des systèmes : Sciences de l'artificiel* , Dunod, Paris, France, 1991, (1ère Ed. 1969, MIT Press).

[Sollins_94] Sollins K., Masinter L., *Functionnal Requirements for Uniform Resource Names* , Décembre 1994 , Internet RFC-1737, <URL: http://www.internic.net/rfc/rfc1737.txt>.

[Steier_96] Steier D., Mitchell T., *Mind Matters : A Tribute to Allen Newell*, Lawrence Erlbaum Associates, Mahwah, U.S.A, 1996.

[Steiner_88] Steiner J., Neumann C., Schiller J., *Kerberos : an authentication service for open network systems*, Proceedings of the Winter Usenix Conference, 1988, Berkeley, U.S.A.

[Tardieu_91] Tardieu H. Rochfeld A., Colletti R., *La méthode Merise : Principes et Outils*, 1991.

[Tello_89] Tello E, *Object-Oriented Programming for Artificial Intelligence : a guide to tools and system design* , Addison Wesley, Massachussetts, U.S.A, 1989.

[Théry_96] Théry M., Stratégies, n° 964, 19 Avril 1996.

[Tisseau_96] Tisseau G., *Intelligence Artificielle : Problèmes et méthodes* , PUF, Paris, France, Avril 1996.

[Verity_96] Verity, *Search'97* , 11 Décembre 1996 , <URL: http://www.verity.com/PR/961211p.html>.

[Visigenic_97] Visigenic, *VisiBroker for Java Reference Guide* , Janvier 1997, <URL: http://www.visigenic.com/techpubs/vbjava/ref/startme.htm>.

[Viveney_94] Viveney R., *DCE IDL With C++ Support -- Functional Specification* , Octobre 1994 , OSF DCE SIG RFC 48.2.

[Vogel_97] Vogel A., Duddy K., *Java Programming with CORBA* , John Wiley and Sons, New York, U.S.A, 1997.

[W3C_97] W3 Consortium, *Jigsaw* , 1997, <URL: http://www.w3.org/Jigsaw/>.

[Welsh_97] Welsh T., *A reactionary can only boggle at radical technology* , 1997 , <URL: http://www.omg.org/news/welsh2.htm>.

[Yeong_95] Yeong W., Howes T., Kille S., *Lightweight Directory Access Protocol* , Mars 1995 , Internet RFC-1777, <URL: http://www.internic.net/rfc/rfc1777.txt>.

[Yergeau_97] Yergeau F., Nicol G., Adams G., Duerst M., *Internationalization of the Hypertext Markup Language* , Janvier 1997 , Internet RFC-2070, <URL: http://www.internic.net/rfc/rfc2070.txt>.

[Young_90] Young A., *The X Window System, Programming and Applications with Xt* , Prentice Hall, Englewood Cliffs, U.S.A, 1990.

[Zakaria_95] Zakaria M., Seret D., Traverson B., *Les protocoles de communication client/serveur* , 2ème congrès biennal AFCET 95, Octobre 1995, Toulouse, France.

[Zhang_92] Zhang X.-X., Tesseron J.-L., Remy C., Seret D., *Implementation of access control in an X500 environment* , International Conference IFIP/INDC-92, (Editeur), Mars 1992, North-Holland, Helsinki, Finlande.

[Zhang_94] Zhang X.-X., Seret D., *Applying OSI directory naming architecture to integrated network management* , IEEE Journal on Selected Area, n° 6, Vol. 12, Août 1994, pp.1000-1010.

6.2 Glossaire

ActiveX
Technologie propriétaire *Microsoft* qui s'appuie sur COM pour permettre le téléchargement et l'exécution de procédures distantes.

API *Application Programming Interface*
Ensemble de fonctions qui fait office d'interface.

BOA *Basic Object Adapter*
Adaptateur d'objets élémentaire. *Canif suisse* d'un ORB. Dépend de l'implémentation et est non portable.

CDR *Common Data Representation*
Règles communes de représentation de données, définies dans le GIOP de CORBA 2.

CGI *Common Gateway Interface*
Interface qu'utilise un serveur Web pour communiquer avec les langages de programmation.

COM *Component Object Model*
Modèle objet (COMposants) de *Microsoft*. Ne gère pas l'héritage mais l'agrégation de méthodes et propriétés.

CORBA *Common Object Request Broker Architecture*
Architecture répartie définie par l'OMG

DCE *Distributed Computing Environment*
Architecture répartie définie par l'OSF

DCOM *Distributed Component Object Model*
Architecture répartie définie par *Microsoft*

DDE *Dynamic Data Exchange*
Premier modèle de communication entre applications de *Microsoft*.

DNS *Domain Name Service*
Protocole de la suite TCP/IP, de gestion de noms de domaines.

DSSSL *Document Style Semantic and Specification Language*

DTD *Document Type Definition*
Ensemble des déclarations de balises qui vont définir la structure des documents traités.

FQDN *Fully Qualified Domain Name*
Nom complet d'un élément du réseau, tel qu'enregistré par un DNS

FTP — *File Transfer Protocol*
Protocole de transfert de fichiers standard, consacré sur les machines UNIX. Il a largement inspiré HTTP.

GDA — *Global Directory Agent*
Interface d'interrogation d'annuaires, dans DCE.

GDS — *Global Directory Service*
Service d'annuaires de DCE, basé sur X.500

HTML — *Hyper Text Markup Language*
DTD normalisée spécifique (donc une spécification de SGML) qui est utilisée pour la conception des documents disponibles sur le Web. *<URL:*: *http://www.w3.org/hypertext/WWW/MarkUp/html-spec*>

HTTP — *Hyper Text Transfer Protocol*
Protocole utilisé entre un serveur WWW et le navigateur " client ", qui interprète HTML.
<URL: *http://www.w3.org/pub/WWW/Protocols/HTTP1.0/draft-ietf-http-spec.html*>

IA — *Intelligence Artificielle*

IDL — *Interface Definition Language*
Langage de Définition d'Interface : normalisé par l'OMG. Permet de convertir les définitions de classes d'objets de langages standards dans un langage unique connu d'un ORB.

IETF — *Internet Engineering Task Force*
Groupe de normalisation de l'Internet

IIOP — *Internet Inter-Orb Protocol*
Protocole normalisé (CORBA 2) de communication entre ORB.

JDBC — *Java DataBase Connectivity*
Ensemble d'API qui permettent d'interfacer le langage Java avec les SGBD courants

Java — Langage orienté objet, développé par Sun Microsystems, dont l'usage principal s'illustre dans le cadre de *l'Internet actif* ou *Intranet actif*. Ce langage s'inspire de la syntaxe de C++ en augmentant la sécurité des applications. La compilation d'un programme Java, se fait dans un langage-machine intermédiaire (machine virtuelle), qui est ensuite interprété sur le poste client final. Des interpréteurs sont fournis sur la plupart des plateformes.

LDAP — *Lightweight Directory Access Protocol*
Protocole simplifiant X.500.
<URL: *http://www.internic.net/rfc/rfc1777.txt*>.

LOA — *Library Object Adaptor*
Adaptateur d'Objets en cours de définition qui vise à interfacer des bibliothèques de fonctions

LRPC — *Lightweight Remote Procedure Call*
Procédures simplifiées de RPC

MathML — *Mathematical Markup Language*
Sur-ensemble de HTML pour définir et représenter les formules mathématiques
<URL: http://www.w3.org/TR/WD-math/>

Méta-connaissance
Connaissance portant ou manipulant des connaissances.
cf. [Pitrat_98]

Middleware
Désigne l'ensemble des logiciels nécessaires pour permettre et organiser la communication et l'échange de messages entre client et serveur

MIME — *Multi-purpose Internet Mail Extensions*
C'est une spécification, extensible, de la messagerie sur *Internet* qui offre un moyen d'échanger des messages multi-média (son, image, binaires, multilingue...).
<URL: http://www.internic.net/rfc/rfc1521.txt>.

News
Un ensemble de forums thématiques où l'ensemble des abonnés (potentiellement toute personne accédant à *Usenet*) peut déposer (ou répondre à) un message, accessible par tout les autres abonnés. Certains groupes de News sont d'accès libre (n'importe qui peut y déposer un message) alors que d'autres sont " modérés " (un " modérateur " reçoit les messages proposés et les publie en fonction de critères qui ont été déterminés auparavant et qui sont régulièrement rappelés).

NNTP — *Network News Transfer Protocol*
Protocole mis en oeuvre dans la gestion des News.

NOS — *Network Operating System*
Système d'exploitation Réseau.

NTP — *Network Time Protocol*
Protocole de synchronisation d'horloges

OCR — *Optical Character Recognition*
Reconnaissance Optique de Caractères

ODA — *Office Document Architecture*
Norme européenne, concurrente de SGML, tombée en désuétude.

ODBC *Open DataBase Connectivity*
Ensemble d'API qui permettent d'interfacer et d'interroger des SGBD en SQL

ODL *Object Definition Language*
Langage propriétaire de Microsoft, utilisé pour décrire les classes COM

OLE *Object Linking and Embedding*
Colle de base de l'architecture de *Microsoft.*

OMA *Object Management Architecture*

OMG *Open Management Group*

<URL: http://www.omg.org*>*

OODCE *Object-Oriented Distributed Computing Environment*

ORB *Object Request Broker*
Négociateur (**B**rocanteur) de **R**equêtes d'**O**bjets

OSF *Open Software Foundation*
un consortium international qui s'est regroupé en 1996 avec l'X/Open pour former l'Open Group
<URL: http://www.osf.org*>*

PEP *Protocol Extension Protocol*

<URL: http://www.w3.org/TR/WD-http-pep*>*

POA *Persistant (Portable) Object Adaptator*
Adaptateur d'objets, en cours de définition par l'OMG, sensé remplacer le BOA.

Proxy *Mandataire, gérant*
Se dit d'un objet (ou programme) qui agit en lieu de celui auquel est destiné le message (par procuration).

RFC *Request For Comments*
Les spécifications de l'Internet sont définies dans des RFC.

RFI *Request For Information*

RFP *Request For Proposals*

RPC *Remote Procedure Call*

SGML *Standard Generalized Markup Language*
C'est une norme ISO (ISO 8879) destinée à l'origine au monde de l'édition, permettant de définir logiquement un ensemble de structures de documents, appelée DTD, indépendamment des supports à partir desquels ils seront consultés (écran, imprimante, Télétype...).
<URL: http://www.ex.ac.uk/SGML/whysgml1.html*>*

Skeletton *Squelette*
Utilisé pour indiquer une structure qui contient les informations nécessaires pour interfacer l'objet (côté serveur dans CORBA, côté client chez *Microsoft*).

SMTP *Simple Mail Tranfer Protocol*
C'est la messagerie standard des systèmes sous UNIX.
<RURL:rfc0821>

SSI *Server Side Includes*

SSL *Secure Sockets Layer*

SSO *Single-Sign-On*
Système permettant à un utilisateur de s'identifier une seule fois et qui gère l'ensemble des autres identifiants/mots de passe de l'utilisateur ainsi que ses droits

STL *Sigle à Trois Lettres*
Désolé, je n'ai pas pu résister à la réflexivité ;-)

Stub Souche d'arbre (litt.) ou talon de chèque : utilisé pour indiquer une structure qui contient les informations nécessaires pour interfacer l'objet (côté client dans CORBA, côté serveur chez *Microsoft*).

URI *Uniform Resource Identifier*
C'est une chaîne de caractères formatée qui sert d'identifieur pour une ressource. L'usage typique d'une URI dans HTML est d'identifier les liens hypertextes. Les URIs incluent en pratique les URL.
<URL:http://www.w3.org/hypertext/WWW/MarkUp/html-spec/html-spec_2.html#GLOSS35>

URL *Uniform Resource Locator*
C'est une chaîne de caractères formatée qui permet de " localiser " une ressource, en fournissant une identification abstraite. Cette chaîne précise aussi la méthode qui va être utilisée pour accéder à la ressource (ftp, http, news...).
<RURL:rfc1738>

UTC *Universal Time Coordinates*
Temps absolu qui calcule le nombre de 100 10^{-9} ièmes de secondes écoulées depuis l'établissement du calendrier Grégorien, le 15 octobre 1582. Se cale sur le méridien de Greenwich.

UUID *Universal Unique Identifier*
Identifiant d'objet unique, défini par DCE.

VRML *Virtual Reality Modeling Language*
c'est un langage de description d'objets, permettant des représentations interactives, calculées en trois dimensions.
<*URL:* *http://vrml.wired.com/vrml.tech/vrml10-3.html*>

W3C *World Wide Web Consortium*
Consortium mondial chargé de la définition et de la normalisation du Web.
<*URL:* *http://www.w3.org/*>

Web ou encore *WWW, World Wide Web, W3*
Littéralement " toile d'araignée mondiale ". Représente le réseau mondial constitué par l'ensemble des ordinateurs connectés par *Internet* et offrant un service HTTP.

XML *eXtended Markup Language*
<*URL:* *http://www.w3.org/TR/WD-xml*>

6.3 La RATP

6.3.1 Quelques chiffres

Raison sociale	**R**égie **A**utonome des **T**ransports **P**arisiens
Siège Social	54, quai de la Râpée, 75012 PARIS
Place de l'entreprise sur le marché	1er employeur en Île-de-France
Métro	Monopole (1,09 milliard de voyageurs)
RER	Partage avec la SNCF (350 millions de voyageurs)
BUS	1er transporteur (836 millions de voyageurs)
Principaux concurrents	**APTR** (75 millions de voyageurs)
Pour le réseau bus	**ADATRIF** (105 millions de voyageurs)
Les agents actifs en 1996	39439

6.3.1.1 Le trafic et le parc

Près de 2,3 milliards de voyageurs transportés en 1996, soit 9,2 millions de voyageurs par jour ouvrable, et 80% du trafic des transports publics de l'Île-de-France

Métro	• 15 lignes pour une longueur totale de 201,5 km • 372 stations
RER	• 2 lignes (RER A et B) pour une longueur totale de 115,1 km dont 66 stations
BUS	• 285 lignes dont 67 pour Paris, et 218 pour la banlieue • 2900 km de lignes • 602 terminus, 7231 points d'arrêt
Tramway	• 2 lignes de 20 km de longueur avec 34 stations

6.3.2 Historique

La première forme de transports collectifs utilisant la voie publique se présente avec la société des « Carrosses cinq sols » entre 1761 et 1762, puis il fallut attendre 1828 pour connaître les voitures hippomobiles baptisées « omnibus ». Le système concurrentiel incite la municipalité à créer la notion de service public, en 1854, afin de satisfaire le besoin de transport à bas prix. Ainsi, un siècle avant la création de la RATP, l'idée de monopole sous le contrôle de la puissance publique est évoquée.

1900

Mise en service de la première ligne de Métro à Paris, munie de voitures légères à traction électrique (ligne 1).

1913

Les dix lignes de la CMP (compagnie du métropolitain parisien) et du Nord-Sud forment un réseau de 93 Km. L'accroissement spectaculaire du trafic pousse la CMP à constituer des rames de quatre, cinq puis huit voitures en bois.

1920 à 1938

il y a coexistence de deux entreprises de transport bien distinctes : la **CMP** pour le réseau souterrain et la **STCRP** (Société des Transports en Commun de la Région Parisienne) pour le réseau de surface.

Cette cohabitation pose un problème de cohérence et d'équité. Paris a un double réseau de transport avec une double culture d'exploitation.

1945

La CMP et la STCRP fusionnent et forment la RATP.

1949

Élargissement du secteur public français avec la nationalisation de la RATP. Cette entreprise qui assure la mission de services publics en contrepartie d'obligations, confère à ses agents, la sécurité de l'emploi, une grille de classification et de rémunération ainsi qu'un régime social particulier.

1960 à 1975

La politique d'aménagement et d'urbanisme de la région Île-de-France relance les investissements sur les infrastructures des transports collectifs. En 1969 est créé le RER. Entre 1967 et 1975 la modernisation technique entraîne l'apparition de nouveaux matériels roulants et l'automatisation de divers secteurs (pilotage automatique, escaliers mécaniques). Cette période voit la naissance de la carte orange qui représente une nouvelle dimension commerciale pour la RATP.

1976 à 1988

Ces années symbolisent la fin du développement extensif des réseaux. Seule l'interconnexion entre la RATP et la SNCF à Gare du Nord est réalisée. La subvention accordée par l'état pour les investissements baisse.

1989

Le projet METEOR, premier métro automatique est lancé. Ce sera la quatorzième ligne de métro. Elle traversera Paris sur 7,2 km, depuis la Madeleine jusqu'à Tolbiac, en passant par Châtelet, gare de Lyon et Bercy. Sa mise en service est prévue pour le second semestre 1998.

1990 Nouvelle organisation de la RATP

Pour assurer pleinement sa mission de service public et réaliser ses objectifs la RATP engage une réforme de ses structures. L'entreprise s'engage sur la voie de la décentralisation, de la simplification de la ligne hiérarchique.

6.3.3 La structure de la RATP

6.3.3.1 Organisation Générale

La RATP est créée par une loi du 21 mars 1948. Au 1er janvier 1949, date de mise en application de la loi, la RATP devient un **établissement public à caractère industriel et commercial** (EPIC) doté d'une autonomie financière. Autour du conseil d'administration, la RATP est organisée sur 3 niveaux hiérarchiques.

6.3.3.1.1 Les Unités Décentralisées ou Spécialisées

Ces entités regroupent un ensemble cohérent d'activités. Elles peuvent être:

- **opérationnelles** : lignes de métro ou de RER, centres bus ;
- **techniques** : ateliers, activités de maintenance ;

- **spécialisées** : ces dernières apportent les compétences nécessaires dans les domaines financiers, juridiques, commerciaux pour la réalisation des missions des unités décentralisées.

À leur tête un chef d'unité, disposant de moyens et de marges de manœuvre, évalué sur ses résultats et chargé de mettre lui-même en œuvre la décentralisation dans son unité.

6.3.3.1.2 Les départements

Ils regroupent un certain nombre d'unités décentralisées et disposent de moyens propres d'études et de soutien.

À leur tête, un directeur de département, ayant la responsabilité entière devant la direction générale de la mise en œuvre des orientations décidées, du bon fonctionnement de son secteur et de la meilleure utilisation des moyens alloués.

Dans certains départements, il existe des groupes de soutien ou missions. Ils mettent en cohérence les politiques, les moyens, et les ressources des unités décentralisées ou spécialisées du département.

6.3.3.1.3 La direction générale

La direction générale est organisée autour du Président Directeur Général (nommé par le ministre des transports) et d'une équipe de 4 directeurs généraux adjoints.

Elle dispose des outils de cohérence, de pilotage et de contrôle stratégiques. Elle est responsable en dernier ressort devant l'opinion et les pouvoirs publics du fonctionnement de l'entreprise. La décentralisation a réorganisé la RATP autour de **4 pôles d'activités** qui sont porteurs d'enjeux et d'engagements pour l'entreprise :

- le pôle « **service aux voyageurs** », en ce qui concerne le trafic et des recettes, la qualité de service et les performances économiques des réseaux.
- le pôle « **finances, Gestion et Développement** », pour ce qui concerne l'engagement à stabiliser les concours publics et la capacité à anticiper les évolutions, à innover, à s'adapter et à promouvoir le développement.
- le pôle « **Maintenance, Travaux et Politique industrielle** », pour ce qui concerne la pérennité du patrimoine, le développement des compétences et du savoir faire technique au profit de l'amélioration de la qualité de service et des performances économiques.
- le pôle « **Social et International** », pour ce qui concerne la responsabilité sociale pour assurer le progrès de tous, pour s'ouvrir à l'international et faire de l'entreprise un groupe mondial de transport urbain.

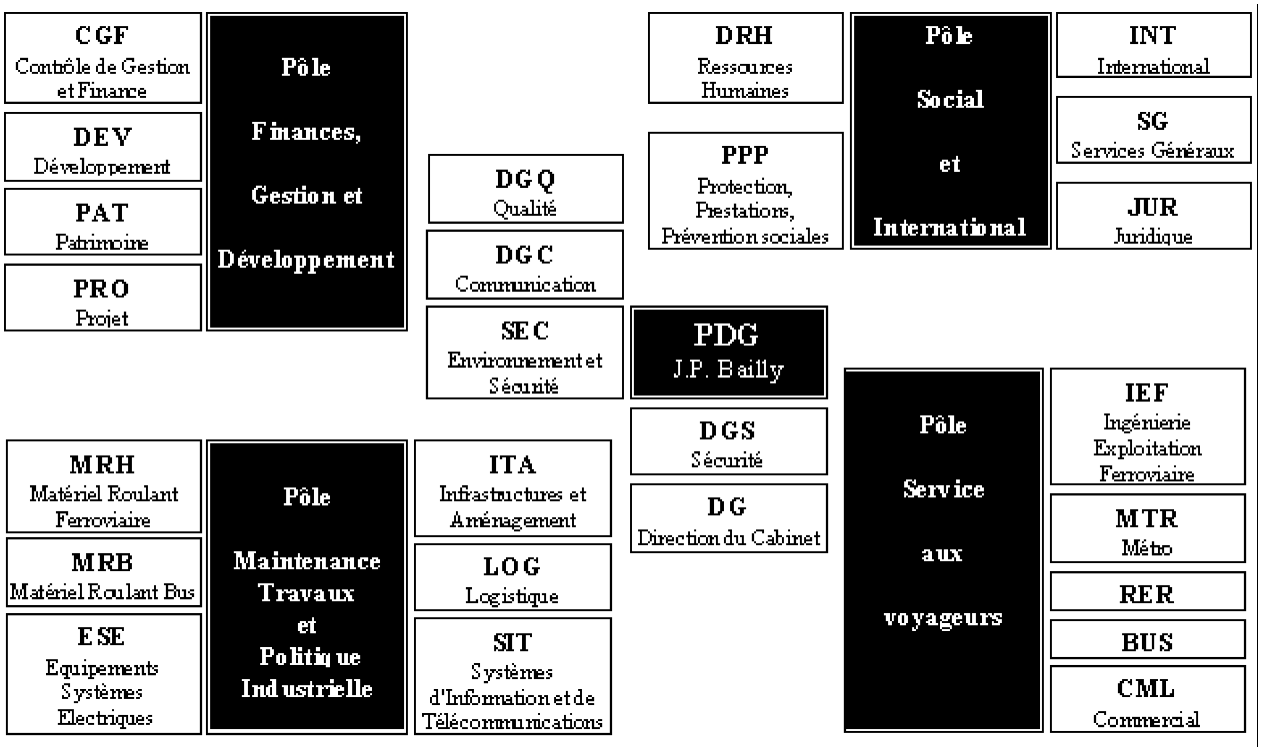


Figure 0. 2. Structure de la RATP, les quatre pôles et leurs départements associés.

6.3.3.1.4 Le département des Systèmes d'Information et de Télécommunications

Le département **SIT** est rattaché au pôle d'activités **M**aintenance **T**ravaux et **P**olitique **I**ndustrielle. Il se compose de 8 unités, 3 groupes de soutien et deux missions transversales, organisés en fonction des différentes missions qui leur sont attribuées.

- Au sein de l'entreprise, SIT a pour but de définir les orientations stratégiques pour les systèmes d'informations et de télécommunications de la RATP. Il doit assurer les ressources informatiques et de télécommunications transversales à l'entreprise. Il peut réaliser à la demande des autres départements des systèmes d'informations locaux.
- Au service du voyageur, il doit fournir les ressources nécessaires à l'information du voyageur en affichant les horaires de trains et de métros et les perturbations qui affectent les lignes. Privilégier l'accès aux moyens de transport pour faciliter l'accès à la station et rendre agréable le voyage. De plus ce département étudie, réalise et maintient l'ensemble des équipements de vente des titres de transport et de contrôle d'accès.

- Au service de l'entreprise, il fournit, exploite et maintient les réseaux et les installations de télécommunications, il les modernise et leur propose de nouvelles fonctions qu'il teste. Il conduit les études de prospectives et accompagne la mise en œuvre des politiques d'entreprise.

En 1997, le département SIT compte 1339 professionnels dont les trois quarts sont à la maintenance et un quart à l'étude et réalisation des projets. Ces personnes sont réparties dans les différentes unités décentralisées techniques.

Les ressources financières sont de deux types : le programme d'investissements de 400 MF (50% au Service voyageurs, 20% aux systèmes d'informations, 30% aux télécommunications), et le budget d'exploitation de 550 MF (40% au service voyageurs, 40% aux systèmes d'informations, et 20% aux télécommunications). SIT a pour objectifs de garantir la qualité de service selon les besoins de l'entreprise, et d'assurer la pérennité du patrimoine.

6.3.4 Les réseaux informatiques de la RATP

6.3.4.1 RAMEAU (Réseau d'Acheminement Multi-protocole inter-Établissement à hAUt débit.)

RENATER
RACINE
F.D.D.I.

6.3.4.2 Internet à la RATP [41]

La RATP n'a pu obtenir de l'organisme de gestion du réseau Internet (le NIC – *Network Information Center*) que quelques réseaux de classe C. De ce fait, la quasi-totalité des adresses IP de la RATP ne sont pas officielles et possèdent potentiellement des doublons à l'extérieur de la RATP. Il était donc impératif de cloisonner l'intérieur de la RATP du reste de l'Internet. Aussi le serveur de noms installé à la RATP est-il indépendant de l'Internet.

Le réseau Internet ne connaît de la RATP que le domaine ratp.fr et la machine arts.ratp.fr. Cette dernière sert de passerelle entre le réseau Internet et le réseau RATP.

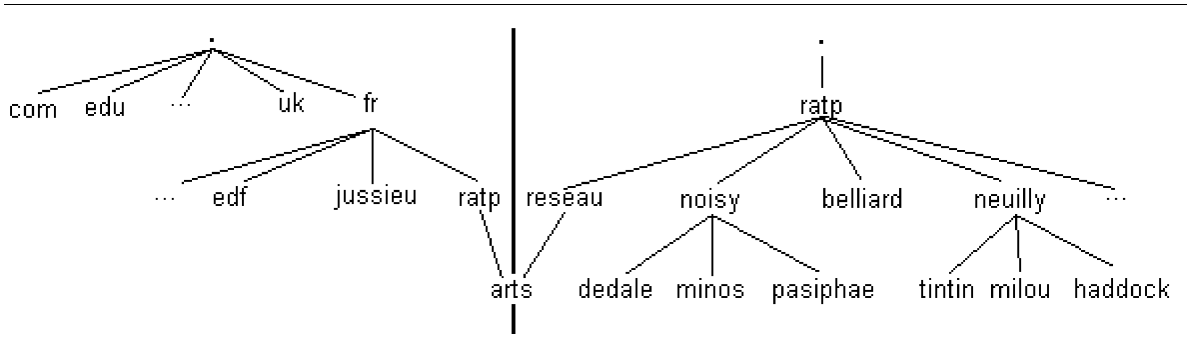


Figure 0. 3. Le DNS « officiel » et celui de la RATP

L'espace de noms de la RATP est structuré autour de l'organisation géographique de la RATP c'est-à-dire que le domaine ratp. – domaine de plus haut niveau de notre DNS – est divisé en autant de sous-domaines qu'il y a de bâtiments. Chaque bâtiment de la RATP constitue donc un domaine indépendant. Un nom d'ordinateur sur le réseau RATP est donc de la forme *ordinateur.bâtiment*.ratp.

6.3.4.3 La passerelle ARTS

Tous les services réseau à l'intérieur du domaine ratp. (terminal virtuel TELNET, transfert de fichiers FTP, *etc.*) sont accessibles à partir de n'importe quelle machine du réseau RATP pour peu qu'ils soient disponibles sur cet ordinateur et configurés correctement. Mais ces mêmes services étaient impossibles entre l'Internet et le réseau RATP du fait de l'étanchéité qui existait entre ces deux réseaux. Cette étanchéité, imposée par l'adressage IP non officiel des réseaux RATP, assurait la sécurité du réseau RATP vis-à-vis de l'extérieur.

[41] Ces informations sont extraites (avec l'autorisation de l'auteur) d'un document [Nicolas_95] interne à la RATP, décrivant les mode de connexion à l'Internet.

La nécessité d'échanges de données entre diverses sociétés et la RATP a imposé d'ouvrir le réseau RATP à l'extérieur. Du fait de l'adressage IP non officiel et dans une optique sécuritaire, l'ouverture a consisté à installer une passerelle –point de passage unique et obligé – entre les deux réseaux.

Cette passerelle (appelée ARTS pour *Accès Réseau Télécom Sécurisé*) a pour vocation d'identifier et d'authentifier les utilisateurs. *Elle ne peut par contre jamais contrôler ce qui est fait sur les serveurs de la RATP. Elle ne résout donc pas les problèmes de sécurité au niveau de chaque serveur.*

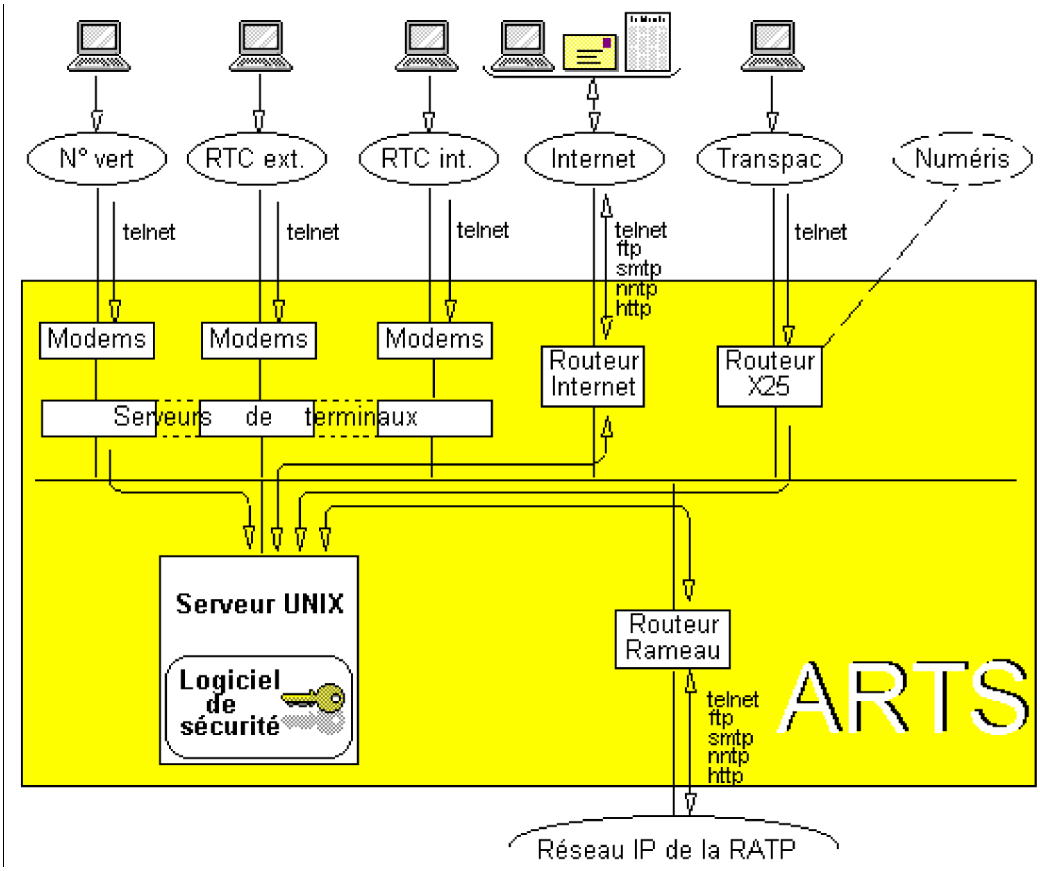


Figure 0. 4. La passerelle ARTS

La passerelle ARTS s'insérant entre le réseau RATP et l'Internet sert de pare-feu vis-à-vis de l'Internet. Elle ne route aucune information au sujet du réseau interne qui est ainsi invisible de l'extérieur (ce qui résout le problème de l'adressage non-officiel des réseaux de la RATP) .

Cela signifie par exemple qu'un utilisateur interne doit se connecter sur le logiciel de sécurité du pare-feu pour ouvrir une connexion TELNET avec le monde extérieur. De la même manière, un utilisateur externe désirant atteindre un ordinateur interne doit se connecter sur le pare-feu.

Cela signifie également que tout le courrier électronique interne à destination de l'extérieur doit être dirigé sur le pare-feu et que celui-ci doit recevoir tout le courrier en provenance de l'extérieur et le redistribuer en interne. Le serveur de la passerelle ARTS est ainsi le *mailhost* de la RATP.

6.4 Rapport du MCD

Listes des objetsListes des objetsListes des objetsListes des objets***Listes des objets***

Liste des informationsListe des informationsListe des informationsListe des informations**Liste des informations**

Nom	Type	Utilisé par
A1	A50	
A2	A50	
ACTIVE	BL	KEYS (KEYS)
ADM	A10	
ALT	TXT	ICONS (ICONS)
ALT	TXT	DictIncl (DictIncl)
AUT	A10	
BDI	BL	Model (Model)
BKG	A50	
BROWSER	I	BROWSERS (BROWSERS)
BT	A50	
CCOMMENT	TXT	Cols (Cols)
COL	I	Cols (Cols)
COMMENT	TXT	KEYS (KEYS)
CONTENT	LVA	KEYS (KEYS)
COUNT_TOTAL	N	KEYS (KEYS)
CR_DATE	DT	KEYS (KEYS)
CT_DATE	D	KEYS (KEYS)
CTITLE	A30	Cols (Cols)
Date	A10	
DATE	D	CLIENTS (CLIENTS)
DATE	DT	USERS (USERS) Logs (Logs)
DB_FILE	A50	RICERCAR (RICERCAR)
DB_NAME	A50	RICERCAR (RICERCAR)
DB_UNC	A50	RICERCAR (RICERCAR)
DESCR	A100	KEYS (KEYS)
DUAL_KEY	A50	KEYS (KEYS)
EXPIRES	BL	Model (Model)
FQDN	A30	CLIENTS (CLIENTS)
FQDN	A50	RICERCAR (RICERCAR)
GRP	A50	GROUPS (GROUPS)
HEADER	A50	Model (Model)
ICON	A50	
ICON	A50	

Nom	Type	Utilisé par
ICON	A50	KEYS (KEYS)
IMG	A50	ICONS (ICONS)
		DictIncl (DictIncl)
IMG_TYPE	A1	ICONS (ICONS)
INDEX_DATE	DT	KEYS (KEYS)
INDEXED	BL	KEYS (KEYS)
		Model (Model)
IP	A50	CLIENTS (CLIENTS)
IP	A30	RICERCAR (RICERCAR)
IP	A50	Logs (Logs)
KEY	A50	KEYS (KEYS)
L1	A50	
L2	A50	
LOCK	A50	
LOG	BL	KEYS (KEYS)
LOG_DATE	DT	RICERCAR (RICERCAR)
MATRICULE	N	USERS (USERS)
MENU	A50	Cols (Cols)
MENU	A10	Menus (Menus)
METHOD	A50	Logs (Logs)
MODEL	A50	Model (Model)
MODIF	BL	KEYS (KEYS)
MODIF_DATE	DT	KEYS (KEYS)
NEW	BL	KEYS (KEYS)
NOM	A30	USERS (USERS)
OCR	BL	DictIncl (DictIncl)
P	A50	Model (Model)
PRENOM	A30	USERS (USERS)
PROTOCOL	A50	Logs (Logs)
PUBLIC	BL	DictIncl (DictIncl)
PWD	A20	USERS (USERS)
RANK	I	Menus (Menus)
RESET_DATE	DT	KEYS (KEYS)
ROOT	A30	RICERCAR (RICERCAR)
RW	I	
S_ALT	I	ALTERNATE (ALTERNATE)
SERVER	A50	RICERCAR (RICERCAR)
SIG	A50	Model (Model)
SIZE	N	Logs (Logs)
SON	BL	Menus (Menus)
SRANK	SI	RICERCAR (RICERCAR)
SRANK	I	ALTERNATE (ALTERNATE)
STRUCT	BL	Model (Model)
TH1	A50	Model (Model)
TH2	A50	Model (Model)
TH3	A50	Model (Model)

Nom	Type	Utilisé par
TH4	A50	Model (Model)
TH5	A50	Model (Model)
TH6	A50	Model (Model)
TITLE	A100	KEYS (KEYS)
TYPE	A50	Model (Model)
		Types (Types)
URL	VA255	KEYS (KEYS)
URL	A30	ICONS (ICONS)
		DictIncl (DictIncl)
USER	A10	Logs (Logs)
USER	A10	USERS (USERS)
USER	A50	
USER	A10	
USER	A10	CLIENTS (CLIENTS)
USER_AGENT	A100	BROWSERS (BROWSERS)
VERSION	SI	KEYS (KEYS)
VIRTUAL	BL	Model (Model)

Liste des entitésListe des entitésListe des entitésListe des entités**Liste des entités**

Nom	Libellé
ALTERNATE	Table de substitution des serveurs (non utilisée)
BROWSERS	Répertorie les différents navigateurs rencontrés
CLIENTS	Postes de travail
Cols	Rubriques ordonnées d'un menu
DictIncl	Tables des images incluses dans les documents
GROUPS	Liste des groupes d'utilisateurs
ICONS	Ensemble des Images connues du système
KEYS	Dictionnaire de TOUS les documents (décrit aussi les classes)
Logs	Enregistrement des documents accédés
Menus	Ensemble des documents-menus du serveur
Model	Description des classes de documents
RICERCAR	Table des serveurs de l'intranet
Types	
USERS	Utilisateurs Identifiés

Liste des associationsListe des associationsListe des associationsListe des associations**Liste des associations**

Nom
ACL Grp
ACL User
Background
Backuped by
Browser used
Button

Nom
Father Arrow
Has as Header
Has as Icon
Has as Sig
Has Columns
Home Arrow
Icon
Includes Images
Is a
Is a menu
Is a Model
Is administred by
Is composed of
Is dual of
Is hosted by
Is logged
Is member of
Is Modeled by
Is of type
Is Owned by
Lock
Lower Line
Participates in
Read by
Subscription
Upper Line
Uses

Informations des entitésInformations des entitésInformations des entitésInformations des entités***Informations des entités***

Entité ALTERNATEEntité ALTERNATEEntité ALTERNATEEntité ALTERNATE**Entité ALTERNATE**

Liste des propriétés

Nom	Libellé	Type	I	O
SRANK	Numéro du serveur "origine"	I	Oui	Oui
S_ALT	Numéro du serveur "émulé"	I	Oui	Oui

Liste des références

Association -> Entité (Lien)	Card.	Dép.
Backuped by	1,1	Non
->RICERCAR	0,n	Non

Association -> Entité (Lien)	Card.	Dép.
Is a	1,1	Non
->RICERCAR	0,n	Non

Entité BROWSERSEntité BROWSERSEntité BROWSERSEntité BROWSERS**Entité BROWSERS**

Liste des propriétés

Nom	Libellé	Type	I	O
BROWSER	Index du Navigateur	I	Oui	Oui
USER_AGENT	Texte d'identification renvoyé par la variable HTTP_USER_AGENT	A100	Non	Oui

Liste des références

Association -> Entité (Lien)	Card.	Dép.
Browser used	1,n	Non
->Logs	1,1	Non

Entité CLIENTSEntité CLIENTSEntité CLIENTSEntité CLIENTS**Entité CLIENTS**

Liste des propriétés

Nom	Libellé	Type	I	O
IP	Adresse IP du poste connecté	A50	Oui	Oui
FQDN	Nom DNS du PC (Fully Qualified Domain Name)	A30	Non	Non
DATE	Date de première connection	D	Non	Oui
USER	IDentifiant de l'utilisateur (si disponible)	A10	Non	Non

Liste des références

Association -> Entité (Lien)	Card.	Dép.
Read by	0,n	Non
->Logs	0,n	Non
Uses	0,1	Non
->USERS	1,n	Non

Entité ColsEntité ColsEntité ColsEntité Cols**Entité Cols**

Liste des propriétés

Nom	Libellé	Type	I	O
MENU	Nom du document-menu décrit	A50	Oui	Oui
COL	Index de la rubrique	I	Oui	Oui
CTITLE	Titre de la rubrique	A30	Non	Oui
CCOMMENT	Texte de comentaire	TXT	Non	Non

Liste des références

Association -> Entité (Lien)	Card.	Dép.

Association -> Entité (Lien)	Card.	Dép.
Has Columns	1,n	Non
->Menus	0,n	Non

Entité DictInclEntité DictInclEntité DictInclEntité DictIncl**Entité DictIncl**

Liste des propriétés

Nom	Libellé	Type	I	O
IMG	Identifiant de l'image	A50	Oui	Oui
PUBLIC	Indique si l'image est "publique" (non utilisé)	BL	Non	Oui
OCR	Indique si l'image a été OCRisée	BL	Non	Oui
URL	URL (ou chemin relatif) de l'image	A30	Non	Oui
ALT	Texte ALTernatif (L'OCR de l'image s'il existe)	TXT	Non	Non

Liste des références

Association -> Entité (Lien)		
Includes Images ->KEYS		

Entité GROUPSEntité GROUPSEntité GROUPSEntité GROUPS**Entité GROUPS**

Liste des propriétés

Nom	Libellé	Type	I	O
GRP	Nom du Groupe	A50	Oui	Oui

Liste des références

Association -> Entité (Lien)	Card.	Dép.
Is member of	0,n	Non
->USERS	1,n	Non
ACL Grp	0,n	Non
->KEYS	0,n	Non

Entité ICONSEntité ICONSEntité ICONSEntité ICONS**Entité ICONS**

Liste des propriétés

Nom	Libellé	Type	I	O
IMG	Identifiant de l'image	A50	Oui	Oui
IMG_TYPE	(L)OGO, LI(N)E, (A)RROW, (I)CON, (B)ACKGROUND	A1	Non	Oui
URL	URL (ou chemin relatif) de l'image	A30	Non	Oui
ALT	Texte ALTernatif (L'OCR de l'image s'il existe)	TXT	Non	Non

Liste des références

Association -> Entité (Lien)	Card.	Dép.
Lock	0,n	Non
->Model	1,1	Non
Button	0,n	Non
->Model	1,1	Non
Lower Line	0,n	Non
->Model	1,1	Non
Upper Line	0,n	Non
->Model	1,1	Non
Father Arrow	0,n	Non
->Model	1,1	Non
Background	0,n	Non
->Model	1,1	Non
Home Arrow	0,n	Non
->Model	1,1	Non
Has as Icon	0,n	Non
->KEYS	1,1	Non
Icon	0,n	Non
->Model	0,n	Non

Entité KEYSEntité KEYSEntité KEYSEntité KEYS**Entité KEYS**

Liste des propriétés

Nom	Libellé	Type	I	O
KEY	ID du document. Unique sur tout l'intranet SAUF pour les Classes et méthodes	A50	Oui	Oui
ACTIVE	Indique si le document est actif (consultable)	BL	Non	Oui
NEW	indique si le document peut être répertorié dans les nouveautés	BL	Non	Oui
MODIF	indique si le document (dont NEW est vrai) peut être répertorié à chaque modification	BL	Non	Oui
LOG	indique si le document peut être enregistré dans les *log*	BL	Non	Oui
INDEXED	Indique si les documents de la classe doivent être indexés	BL	Non	Oui
TITLE	Titre du document	A100	Non	Oui
DESCR	Texte de description du document	A100	Non	Non
COMMENT	un commentaire	TXT	Non	Non

Nom	Libellé	Type	I	O
CR_DATE	Date de création du document	DT	Non	Oui
MODIF_DATE	Date de modification (permet de lancer la méthode d'indexation)	DT	Non	Oui
INDEX_DATE	Date d'indexation (si applicable)	DT	Non	Oui
RESET_DATE	Date de RAZ du compteur	DT	Non	Oui
ICON	la puce (ICONS) du document (celle de la classe par défaut)	A50	Non	Oui
DUAL_KEY	Clef de l'éventuel document dual, en 2nde langue (anglais)	A50	Non	Non
COUNT_TOTAL	Nombre d'accès total au document à la date CT_DATE	N	Non	Oui
CT_DATE	Date à laquelle COUNT_TOTAL à été calculé	D	Non	Oui
VERSION	version du document	SI	Non	Non
CONTENT	Texte contenant le code HTML du document si VIRTUAL est vrai	LVA	Non	Non
URL	URL physique ou relative du corps du document	VA255	Non	Oui

Liste des références

Association -> Entité (Lien)	Card.	Dép.
Is hosted by	1,1	Oui
->RICERCAR	1,n	Non
Is Owned by	1,1	Non
->USERS	0,n	Non
Is logged	0,n	Non
->Logs	1,1	Non
Is of type	1,1	Non
->Types	0,n	Non
->Model	0,n	Non
Is Modeled by	1,1	Non
->Model	0,n	Non
ACL User	0,n	Non
->USERS	0,n	Non
ACL Grp	0,n	Non
->GROUPS	0,n	Non
Participates in	1,n	Non
->Menus	0,n	Non
Includes Images	0,n	Non

Association -> Entité (Lien)	Card.	Dép.
->DictIncl	1,1	Non
Is a menu	0,n	Non
->Menus	1,1	Non
Subscription	0,n	Non
->USERS	0,n	Non
Is dual of	0,n	Non
->KEYS	0,n	Non
Is dual of	0,n	Non
->KEYS	0,n	Non
Has as Icon	1,1	Non
->ICONS	0,n	Non
Is composed of	0,n	Non
->Menus	0,n	Non
Has as Header	0,n	Non
->Model	1,1	Non
Has as Sig	0,n	Non
->Model	1,1	Non
Is a Model	0,1	Non
->Model	1,1	Non

Entité LogsEntité LogsEntité LogsEntité Logs**Entité Logs**

Liste des propriétés

Nom	Libellé	Type	I	O
IP	Adresse IP du PC	A50	Oui	Oui
DATE	Date de connexion	DT	Oui	Oui
USER	Id du user (si connue)	A10	Non	Non
METHOD	Méthode HTTP utilisée (GET, POST, HEAD...)	A50	Non	Oui
SIZE	Taille du fichier transmis	N	Non	Oui
PROTOCOL	Le protocole de communication employé (ie HTTP/1.0)	A50	Non	Oui

Liste des références

Association -> Entité (Lien)	Card.	Dép.

Association -> Entité (Lien)	Card.	Dép.
Browser used	1,1	Non
->BROWSERS	1,n	Non
Read by	0,n	Non
->CLIENTS	0,n	Non

Entité MenusEntité MenusEntité MenusEntité Menus**Entité Menus**

Liste des propriétés

Nom	Libellé	Type	I	O
MENU	Identifiant (Idem que KEY) du menu	A10	Oui	Oui
SON	Indique si le MENU père de KEY est MENU	BL	Non	Oui
RANK	Rang de KEY dans la rubrique COL du menu MENU	I	Non	Oui

Liste des références

Association -> Entité (Lien)	Card.	Dép.
Participates in	0,n	Non
->KEYS	1,n	Non
Has Columns	0,n	Non
->Cols	1,n	Non
Is a menu	1,1	Non
->KEYS	0,n	Non
Is composed of	0,n	Non
->KEYS	0,n	Non

Entité ModelEntité ModelEntité ModelEntité Model**Entité Model**

Liste des propriétés

Nom	Libellé	Type	I	O
MODEL	Nom d'une classe de documents	A50	Oui	Oui
TYPE	Indique si la classe représente des documents, des méthodes, des menus...	A50	Non	Oui
BDI	Indique si c'est une classe de documents gérés par RICERCAR	BL	Non	Oui
VIRTUAL	Indique si c'est une classe de documents virtuels (CONTENT de DICTPHYS)	BL	Non	Oui
STRUCT	Indique si la classe décrit des index	BL	Non	Oui
EXPIRES	Indique s'il faut positionner l'en-tête HTTP pragma à "no-cache"	BL	Non	Oui
INDEXED	Indique si les documents de la classe doivent être indexés	BL	Non	Oui
HEADER	Identifiant du document dont le code permet de générer l'en-	A50	Non	Oui

Nom	Libellé	Type	I	O
	tête des instances			
SIG	Identifiant du document dont le code permet de générer la signature des instances	A50	Non	Oui
TH1	description de la feuille de style (CSS) associée à H1	A50	Non	Non
TH2	description de la feuille de style (CSS) associée à H2	A50	Non	Non
TH3	description de la feuille de style (CSS) associée à H3	A50	Non	Non
TH4	description de la feuille de style (CSS) associée à H4	A50	Non	Non
TH5	description de la feuille de style (CSS) associée à H5	A50	Non	Non
TH6	description de la feuille de style (CSS) associée à H6	A50	Non	Non
P	description de la feuille de style (CSS) associée à P	A50	Non	Non

Liste des références

Association -> Entité (Lien)	Card.	Dép.
Is Modeled by	0,n	Non
->KEYS	1,1	Non
Background	1,1	Non
->ICONS	0,n	Non
Home Arrow	1,1	Non
->ICONS	0,n	Non
Father Arrow	1,1	Non
->ICONS	0,n	Non
Upper Line	1,1	Non
->ICONS	0,n	Non
Lower Line	1,1	Non
->ICONS	0,n	Non
Button	1,1	Non
->ICONS	0,n	Non
Lock	1,1	Non
->ICONS	0,n	Non
Icon	0,n	Non
->ICONS	0,n	Non
Has as Header	1,1	Non
->KEYS	0,n	Non
Has as Sig	1,1	Non
->KEYS	0,n	Non

Association -> Entité (Lien)	Card.	Dép.
Is a Model	1,1	Non
->KEYS	0,1	Non
Is of type	0,n	Non
->Types	0,n	Non
->KEYS	1,1	Non

Entité RICERCAREntité RICERCAREntité RICERCAREntité RICERCAR**Entité RICERCAR**

Liste des propriétés

Nom	Libellé	Type	I	O
SRANK	Identifiant du serveur, automatiquement préfixé à l'identifiant des documents	SI	Oui	Oui
SERVER	Nom logique du serveur.	A50	Non	Oui
FQDN	Nom DNS de la machine hébergeant le serveur (un alias éventuellement)	A50	Non	Oui
IP	Adresse IP du serveur	A30	Non	Oui
ROOT	Chemin absolu de la racine du serveur (c:\www)	A30	Non	Oui
DB_UNC	UNC vers le répertoire où se trouve la BD du serveur	A50	Non	Oui
DB_NAME	Nom ODBC de la BD	A50	Non	Oui
DB_FILE	Nom du fichier de la BD	A50	Non	Oui
LOG_DATE	Date de récupération des *log* du serveur	DT	Non	Oui

Liste des références

Association -> Entité (Lien)	Card.	Dép.
Is hosted by	1,n	Non
->KEYS	1,1	Oui
Backuped by	0,n	Non
->ALTERNATE	1,1	Non
Is a	0,n	Non
->ALTERNATE	1,1	Non
Is administred by	1,1	Non
->USERS	0,n	Non

Entité TypesEntité TypesEntité TypesEntité Types**Entité Types**

Liste des propriétés

Nom	Libellé	Type	I	O
TYPE	Indique si la classe représente des documents, des méthodes, des menus...	A50	Oui	Non

Liste des références

Association -> Entité (Lien)	Card.	Dép.
Is of type	0,n	Non
->KEYS	1,1	Non
->Model	0,n	Non

Entité USERSEntité USERSEntité USERSEntité USERS**Entité USERS**

Liste des propriétés

Nom	Libellé	Type	I	O
USER	Identifiant normalisé de l'utilisateur	A10	Oui	Oui
MATRICULE	Matricule RATP	N	Non	Oui
NOM	Nom complet	A30	Non	Oui
PRENOM	Prénoms	A30	Non	Oui
PWD	Check code du mot de passe	A20	Non	Oui
DATE	Date de connexion	DT	Non	Oui

Liste des références

Association -> Entité (Lien)	Card.	Dép.
Is member of	1,n	Non
->GROUPS	0,n	Non
ACL User	0,n	Non
->KEYS	0,n	Non
Is Owned by	0,n	Non
->KEYS	1,1	Non
Subscription	0,n	Non
->KEYS	0,n	Non
Uses	1,n	Non
->CLIENTS	0,1	Non
Is administred by	0,n	Non
->RICERCAR	1,1	Non

6.5 Le code des méta-scripts de RICERCAR

Il nous a paru inutile d'encombrer ces pages, de *listings* de programmes.

Vous pouvez accéder librement à la totalité du code de RICERCAR soit à partir de <*RURL:ricercar_pgm*>, soit, pour ceux que nous aurions omis de faire figurer dans ce menu, en invoquant directement chacun des *Documents* qui constituent les méthodes du système.

6.6 Exemples de Documents générés par RICERCAR

6.6.1 Un cas simple : requête de l'objet « flash »

6.6.1.1 « Flash.html », tel qu'enregistré par son auteur (la DGC).

<P><B>Le lundi 20 octobre, s'est tenue une réunion avec l'ensemble des organisations

syndicales sur la politique de l'emploi.</B>

<P>La Direction de la RATP a confirmé l'augmentation des emplois statutaires,

de l'ordre de 200 à 300 en 1998. Ils permettront la mise en oeuvre des accroissements

de service (Météor, prolongement de la ligne 13 dans Saint-Denis, Autrement Bus).

<P>Elle a réaffirmé l'engagement de l'entreprise dans le Plan emploi-jeunes.

Dans ce cadre, la RATP contribuera à la création de 1000 emplois jeunes en 3 ans,

en partenariat avec d'autres acteurs urbains.

<P>Afin de donner transparence et lisibilité pour tous à chacune de ces mesures,

la Direction de la RATP a proposé aux organisations syndicales la

<B>création d'un "Observatoire des emplois"</B>.

Il permettra information et dialogue sur les évolutions quantitatives et qualitatives

des métiers, qu'ils soient créés dans l'entreprise ou grâce à sa contribution.

6.6.1.2 Le document résultant de l'exécution de la méthode Sty_page



6.6.1.3 Les traces de l'exécution de l'interpréteur

```
QY_EXISTKEY (Records=1, Time=31ms)                    flash existe-t-il ?
SQL =
SELECT  *
      FROM    KEYS, RICERCAR
      WHERE   KEYS.KEY = 'flash'
      AND     (KEYS.SRANK = RICERCAR.SRANK
        OR  (KEYS.SRANK = 0  AND RICERCAR.SRANK= 0))
QY_KEY (Records=1, Time=78ms)                         récupère ses
SQL =                                                 métadonnées
SELECT  *
      FROM    KEYS, Model, Dict, USERS
      WHERE   KEYS.KEY = 'flash'
      AND     KEYS.SRANK = 3
      AND     KEYS.KEY = Dict.KEY
      AND     KEYS.ACTIVE
      AND     Model.MODEL = KEYS.MODEL
      AND     USERS.USER = KEYS.AUT
QY_GOTO (Records=1, Time=0ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'sty_goto'
QY_PAPA (Records=1, Time=15ms)                        récupère son
SQL =                                                 « père »
SELECT  *
      FROM    Menus
      WHERE   Menus.SON
      AND     Menus.KEY = 'flash'
QY_PAGE (Records=1, Time=16ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   DictPhys.KEY = 'sty_page'
QY_SYS_ACL (Records=1, Time=15ms)                     retrouve la méthode
SQL = SELECT  URL                                     de vérification
      FROM    DictPhys                                des ACL
      WHERE   KEY = 'sys_acl'
QY_PHYS (Records=1, Time=16ms)                        localise le
SQL =                                                 document flash
SELECT  *
      FROM    DictPhys
      WHERE   KEY = 'flash'
```

```
QY_STY_HEAD (Records=1, Time=16ms)                    retrouve la méthode
SQL =                                                 générale de génération
SELECT  URL                                           d'en-tête
      FROM    DictPhys
      WHERE   KEY = 'Sty_HTMLHead'
QY_INCLUDE (Records=1, Time=15ms)                     retrouve la méthode
SQL =                                                 de notification des maj
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'sys_maj'
QY_INCLUDE (Records=1, Time=16ms)                     retrouve la méthode
SQL =                                                 d'enregistrement
SELECT  URL                                           des logs
      FROM    DictPhys
      WHERE   KEY = 'sys_logs'
QY_CLIENT (Records=1, Time=31ms)                      vérifie que le poste
SQL =                                                 appelant est répertorié
SELECT  IP
      FROM    CLIENTS
      WHERE   IP = '192.25.86.2'
QY_BW (Records=1, Time=32ms)                          vérifie que le
SQL =                                                 navigateur utilisé est
SELECT  BROWSER                                       répertorié
      FROM    BROWSERS
      WHERE   USER_AGENT = 'Mozilla/4.02 [en] (WinNT; I)'
INS_LOGS (Records=0, Time=15ms)                       invocation de la
SQL =                                                 méthode des logs
INSERT  INTO Logs
            (KEY, IP, USER, METHOD, SIZE, PROTOCOL, BROWSER)
      VALUES ('flash', '192.25.86.2', 'JS207151',  'GET', 1, 'HTTP/1.0',71)
```

6.6.1.4 Les traces de l'exécution de Sty_page (invoquée par l'interpréteur)

```
QY_SYS_HEADER (Records=1, Time=0ms)                   retrouve la méthode
SQL =                                                 de génération d'en-
SELECT  URL                                           tête de la classe
      FROM    DictPhys
      WHERE   KEY = 'Sty_Header'
QY_INCLUDES (Records=0, Time=16ms)                    cherche les images
SQL =                                                 incluses
SELECT  *
      FROM    DictIncl
      WHERE   KEY = 'flash'
      ORDER BY IMG
```

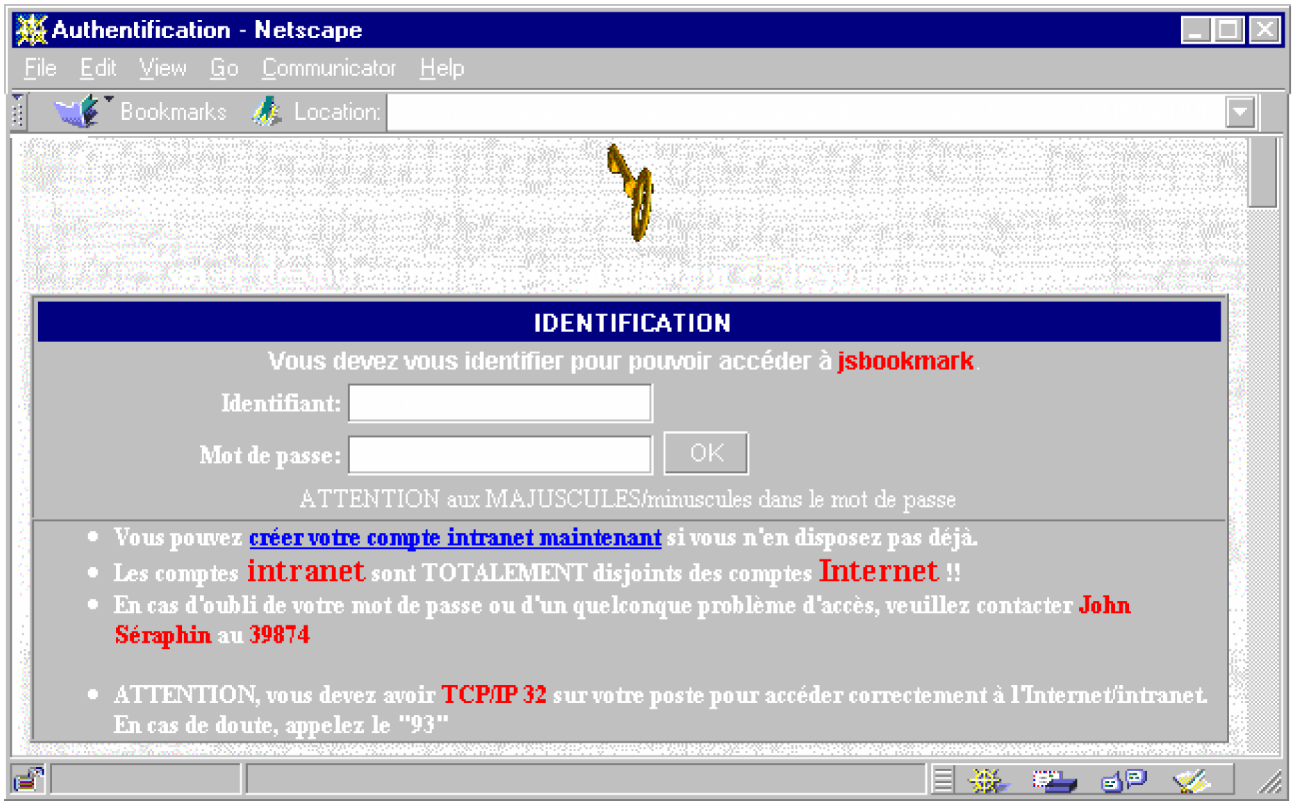
QY_INCLUDE (Records=1, Time=0ms) SQL = SELECT URL FROM DictPhys WHERE KEY = 'Sty_Signe'	***retrouve la méthode de génération de texte signataire de la classe***
QY_COUNT (Records=1, Time=16ms) SQL = SELECT Count(*) AS COUNT FROM Logs WHERE DATE >= {ts '1997-10-22 13:23:37'} AND KEY = 'flash'	***génération et mise en forme des différentes métadonnées***
XIST (Records=1, Time=47ms) SQL = SELECT KEY FROM KEYS WHERE KEY = 'sio_aconsulter'	**...**
XIST (Records=1, Time=47ms) SQL = SELECT KEY FROM KEYS WHERE KEY = 'rir_sio'	**...**
XIST (Records=1, Time=47ms) SQL = SELECT KEY FROM KEYS WHERE KEY = 'ami_1'	**...**
XIST (Records=1, Time=47ms) SQL = SELECT KEY FROM KEYS WHERE KEY = 'adm_maj'	**...**
Execution Time **786 ms**	***(dont 516 ms de SQL)***

6.6.2 Le pire des cas : requête de « jsbookmark », d'accès protégé, à partir d'un autre serveur

La requête saisie est

<URL:http://sit-cis.intra.ratp/ricercar/jsbookmark>

6.6.2.1 Redirection sur le serveur d'authentification



6.6.2.2 Les traces de l'exécution de l'interpréteur (et service de sécurité)

```
QY_EXISTKEY (Records=1, Time=31ms)                    première phase (étapes 1-3):
SQL =                                                 retrouver le bon serveur
SELECT *       FROM    KEYS, RICERCAR
      WHERE   KEYS.KEY = 'jsbookmark'
      AND     (KEYS.SRANK = RICERCAR.SRANK
        OR  (KEYS.SRANK = 0  AND RICERCAR.SRANK= 0))

Execution Time
47 milliseconds

Redirection vers: "http://sit-sio.intra.ratp/cft/goto.cfm?key=jsbookmark"
```

```
QY_EXISTKEY (Records=1, Time=31ms)                    On est à l'étape 5
SQL =                                                 (pas d'intérêt à la 4)
SELECT *
      FROM    KEYS, RICERCAR
      WHERE   KEYS.KEY = 'jsbookmark'
      AND     (KEYS.SRANK = RICERCAR.SRANK
        OR  (KEYS.SRANK = 0  AND RICERCAR.SRANK= 0))

QY_KEY (Records=1, Time=94ms)
SQL =
SELECT *
      FROM    KEYS, Model, Dict, USERS
      WHERE   KEYS.KEY = 'jsbookmark'
      AND     KEYS.SRANK = 3
      AND     KEYS.KEY = Dict.KEY
      AND     KEYS.ACTIVE
      AND     Model.MODEL = KEYS.MODEL
      AND     USERS.USER = KEYS.AUT

QY_GOTO (Records=1, Time=15ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'sty_goto'

QY_PAPA (Records=1, Time=16ms)
SQL =
SELECT *
      FROM    Menus
      WHERE   Menus.SON
      AND     Menus.KEY = 'jsbookmark'
```

```
QY_PAGE (Records=1, Time=0ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   DictPhys.KEY = 'sty_page'

QY_SYS_ACL (Records=1, Time=0ms)                    entrée dans Sys_Acl
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'sys_acl'

QY_ACL (Records=1, Time=31ms)                       "jsbookmark" possède
SQL =                                               des ACL...
SELECT  *
      FROM    Acl
      WHERE   KEY = 'jsbookmark'

QY_ACLR (Records=1, Time=16ms)                      ..dont au moins une de
SQL =                                               lecture
SELECT  *
           FROM    Acl
           WHERE   KEY='jsbookmark'
           AND     RW = 1

QY_ACLW (Records=0, Time=16ms)                      ... et aucune en écriture
SQL =                                               (test inutile ici)
SELECT  *
           FROM    Acl
           WHERE   KEY='jsbookmark'
           AND     RW > 1                           l'utilisateur n'étant pas

Redirection vers le document d'identification:      authentifié, faut le faire

"http://ricercar.intra.ratp/cft.goto.cfm?key=passe&idp=jsbookmark
&MSG=IDENTIFICATION"

Execution Time
282 milliseconds                                    (dont 219ms  de SQL)
```

```
QY_EXISTKEY (Records=1, Time=32ms)                    On est à l'étape 6
SQL =
SELECT  *
      FROM    KEYS, RICERCAR
      WHERE   KEYS.KEY = 'passe'
      AND     (KEYS.SRANK = RICERCAR.SRANK
        OR  (KEYS.SRANK = 0  AND RICERCAR.SRANK= 0))


QY_KEY (Records=1, Time=93ms)
SQL =
SELECT  *
      FROM    KEYS, Model, Dict, USERS
      WHERE   KEYS.KEY = 'passe'
      AND     KEYS.SRANK = 1
      AND     KEYS.KEY = Dict.KEY
      AND     KEYS.ACTIVE
      AND     Model.MODEL = KEYS.MODEL
      AND     USERS.USER = KEYS.AUT

QY_GOTO (Records=1, Time=16ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'sty_goto'

QY_PAPA (Records=1, Time=16ms)
SQL =
SELECT  *
      FROM    Menus
      WHERE   Menus.SON
      AND     Menus.KEY = 'passe'

QY_PAGE (Records=1, Time=15ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   DictPhys.KEY = 'sty_post'

QY_SYS_ACL (Records=1, Time=16ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'sys_acl'

QY_ACL (Records=0, Time=16ms)                         "passe" est en accès libre.
SQL =                                                 Heureusement, sinon
SELECT  *                                             le système boucle
      FROM    Acl
      WHERE   KEY = 'passe'
```

```
QY_PHYS (Records=1, Time=15ms)                    on le localise
SQL =
SELECT  *
      FROM    DictPhys
      WHERE   KEY = 'passe'

QY_STY_HEAD (Records=1, Time=16ms)                et on commence à le
SQL =                                             construire
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'Sty_HTMLHead'

QY_INCLUDE (Records=1, Time=15ms)                 ... en effectuant les
SQL =                                             les opérations
SELECT  URL                                       d'administration ad hoc
      FROM    DictPhys
      WHERE   KEY = 'sys_maj'

QY_INCLUDE (Records=1, Time=0ms)                  ...
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'sys_logs'

QY_CLIENT (Records=1, Time=31ms)                  ...
SQL =
SELECT  IP
      FROM    CLIENTS
      WHERE   IP = '192.25.86.114'

QY_BW (Records=1, Time=47ms)                      ...
SQL =
SELECT  BROWSER
      FROM    BROWSERS
      WHERE   USER_AGENT = 'Mozilla/4.02 [en] (WinNT; I)'

INS_LOGS (Records=0, Time=15ms)                   ...
SQL =
INSERT  INTO Logs
            (KEY, IP, USER, METHOD, SIZE, PROTOCOL, BROWSER)
      VALUES ('passe', '192.25.86.114', 'JS207151', 'GET', 1, 'HTTP/1.0', 71)

QY_SYS_HEADER (Records=1, Time=15ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'Sty_Header'

QY_INCLUDES (Records=0, Time=16ms)                y a-t'il des images
SQL =                                              incluses dans "passe" ?
SELECT  *
      FROM    DictIncl
      WHERE   KEY = 'passe'                       non !
      ORDER BY IMG
```

```
QY_ACTION (Records=1, Time=31ms)                    mais il y a un
SQL =                                               formulaire à générer
SELECT  RICERCAR.SRANK , DB_NAME , FQDN , IP        (on est dans l'évaluation
      FROM    KEYS, RICERCAR                        du paramètre que
      WHERE   KEYS.KEY = 'jsbookmark'               traite Sty_page)
      AND     KEYS.SRANK = RICERCAR.SRANK

QY_ACTION1 (Records=1, Time=47ms)
SQL =
SELECT  URL
      FROM    KEYS, DICTPHYS
      WHERE   KEYS.KEY = 'jsbookmark'
      AND     KEYS.MODEL = DictPhys.KEY

QY_INCLUDE (Records=1, Time=0ms)                    récupération du
SQL =                                               document signataire
SELECT  URL                                         et génération
      FROM    DictPhys
      WHERE   KEY = 'Sty_Signe'

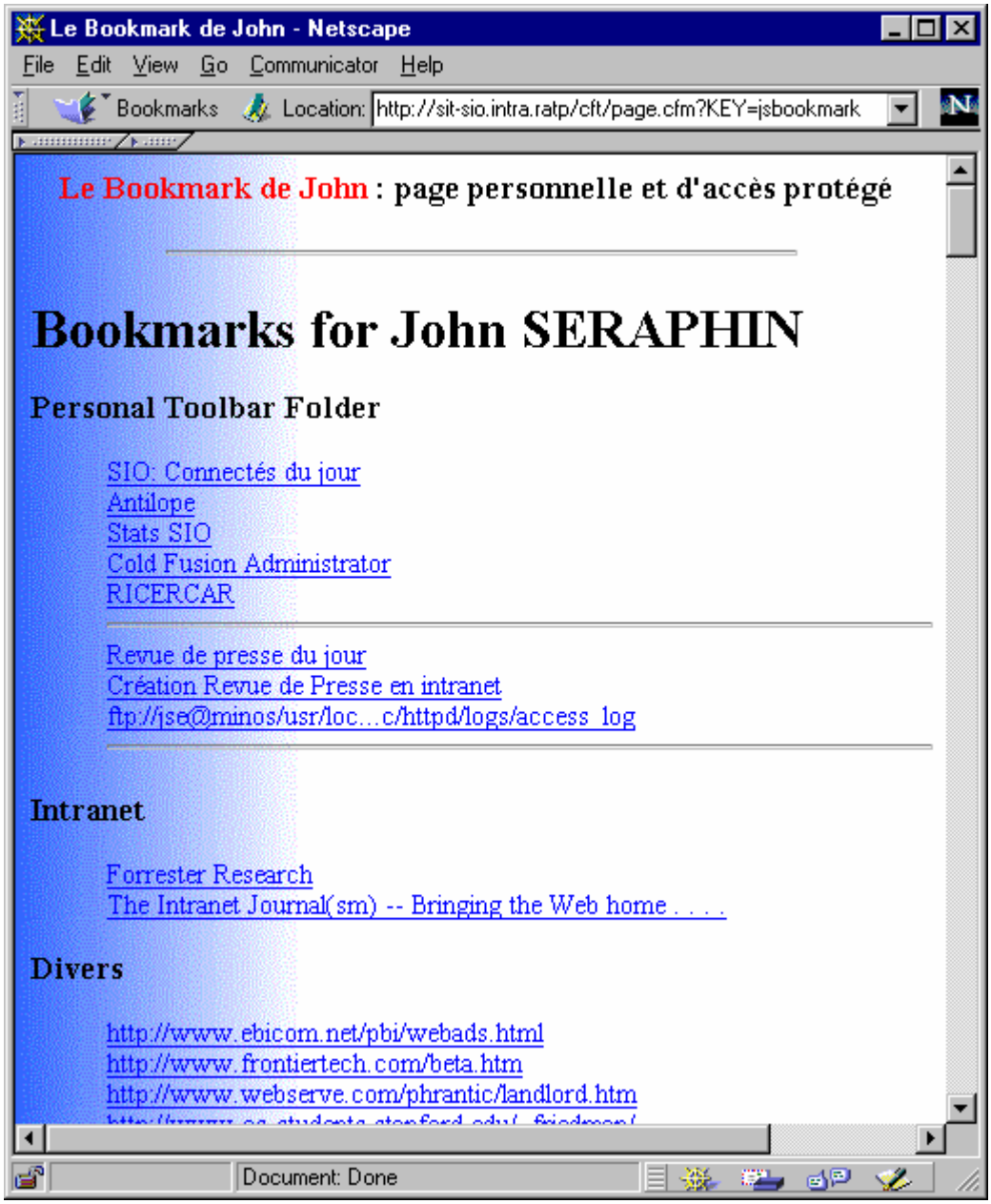
QY_COUNT (Records=1, Time=47ms)
SQL =
SELECT  Count(*) AS COUNT
      FROM    Logs
      WHERE   DATE >= {ts '1997-05-21 15:38:48'}
      AND     KEY = 'passe'

XIST (Records=1, Time=47ms)
SQL =
SELECT  KEY
      FROM    KEYS
      WHERE   KEY = 'usersgest'

XIST (Records=1, Time=47ms)
SQL =
SELECT  KEY
      FROM    KEYS
      WHERE   KEY = 'ricercar'

Execution Time
850 milliseconds                                    (dont 593 ms en SQL)
```

6.6.2.3 Accès authentifié à « jsbookmark »



6.6.2.4 Les traces de l'exécution de l'interpréteur

```
QY_EXISTKEY (Records=1, Time=31ms)                    On est à l'étape 9
SQL =
SELECT  *
      FROM    KEYS, RICERCAR
      WHERE   KEYS.KEY = 'jsbookmark'
      AND     (KEYS.SRANK = RICERCAR.SRANK
        OR  (KEYS.SRANK = 0  AND RICERCAR.SRANK= 0))

QY_KEY (Records=1, Time=79ms)
SQL =
SELECT  *
      FROM    KEYS, Model, Dict, USERS
      WHERE   KEYS.KEY = 'jsbookmark'
      AND     KEYS.SRANK = 3
      AND     KEYS.KEY = Dict.KEY
      AND     KEYS.ACTIVE
      AND     Model.MODEL = KEYS.MODEL
      AND     USERS.USER = KEYS.AUT

QY_GOTO (Records=1, Time=15ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'sty_goto'

QY_PAPA (Records=1, Time=16ms)
SQL =

SELECT  *
      FROM    Menus
      WHERE   Menus.SON
      AND     Menus.KEY = 'jsbookmark'

QY_PAGE (Records=1, Time=15ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   DictPhys.KEY = 'sty_page'

QY_SYS_ACL (Records=1, Time=16ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'sys_acl'

QY_ACL (Records=1, Time=16ms)                         "jsbookmark" possède
SQL =                                                 des ACL
SELECT  *
      FROM    Acl
      WHERE   KEY = 'jsbookmark'

QY_ACLR (Records=1, Time=15ms)                        dont une de lecture
SQL =
```

```
SELECT  *
        FROM    Acl
        WHERE   KEY='jsbookmark'
        AND     RW = 1

QY_ACLW (Records=0, Time=16ms)
SQL =
SELECT  *
        FROM    Acl
        WHERE   KEY='jsbookmark'
        AND     RW > 1

QY_CRYPT (Records=1, Time=0ms)                  l'utilisateur vient de
SQL =                                           s'identifier : on crypte
SELECT  URL                                     son mot de passe
            FROM    DictPhys
            WHERE   KEY = 'sys_crypt'

QY_IDENT (Records=1, Time=31ms)                 qu'on compare au
SQL =                                           check code stocké
SELECT  *                                       dans la base
            FROM    USERS
            WHERE   USER = 'JS207151'
            AND     PWD = '1957700052'

QY_ISADM (Records=1, Time=31ms)                 on apprend que
SQL =                                           l'utilisateur est
SELECT  GRP                                     aussi administrateur
            FROM    MEMBERS
            WHERE   USER = 'JS207151'
            AND     GRP = 'ADMIN'

QY_VERIFACL (Records=1, Time=15ms)              on vérifie que les ACL
SQL =                                           de"jsbookmark"
SELECT  URL                                     permettent bien la
        FROM    DictPhys                        lecture à l'utilisateur
        WHERE   KEY = 'sys_verifacl'

QY_PHYS (Records=1, Time=16ms)                  Oui ! Le reste est
SQL =                                           désormais classique :
SELECT  *                                       composition et
    FROM    DictPhys                            génération du
    WHERE   KEY = 'jsbookmark'                  document final

QY_STY_HEAD (Records=1, Time=15ms)
SQL =
SELECT  URL
    FROM    DictPhys
    WHERE   KEY = 'Sty_HTMLHead'

QY_INCLUDE (Records=1, Time=16ms)
SQL =
SELECT  URL
    FROM    DictPhys
    WHERE   KEY = 'sys_maj'
```

```
QY_INCLUDE (Records=1, Time=0ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'sys_logs'

QY_CLIENT (Records=1, Time=31ms)
SQL =
SELECT  IP
      FROM    CLIENTS
      WHERE   IP = '192.25.86.114'

QY_BW (Records=1, Time=31ms)
SQL =
SELECT  BROWSER
      FROM    BROWSERS
      WHERE   USER_AGENT = 'Mozilla/4.02 [en] (WinNT; I)'

INS_LOGS (Records=0, Time=32ms)
SQL =
INSERT  INTO Logs
            (KEY, IP, USER, METHOD, SIZE, PROTOCOL, BROWSER)
      VALUES ('jsbookmark', '192.25.86.114', 'JS207151', 'POST', 51,'HTTP/1.0', 71)

QY_SYS_HEADER (Records=1, Time=0ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'Sty_Header'

QY_INCLUDES (Records=0, Time=16ms)
SQL =
SELECT  *
      FROM    DictIncl
      WHERE   KEY = 'jsbookmark'
      ORDER BY IMG

QY_INCLUDE (Records=1, Time=0ms)
SQL =
SELECT  URL
      FROM    DictPhys
      WHERE   KEY = 'Sty_Signe'

QY_COUNT (Records=1, Time=16ms)
SQL =
SELECT  Count(*) AS COUNT
      FROM    Logs
      WHERE   DATE >= {ts '1997-10-22 13:23:37'}
      AND     KEY = 'jsbookmark'

XIST (Records=1, Time=47ms)
SQL =
SELECT  KEY
      FROM    KEYS
      WHERE   KEY = 'essai'
```

```
XIST (Records=1, Time=47ms)
SQL =
SELECT  KEY
      FROM    KEYS
      WHERE   KEY = 'rir_sio'
```

Execution Time
894 milliseconds ***(dont 563 ms de SQL)***

6.6.3 Services Communs

6.6.3.1 Événements



*Figure 0. 5 **NOU**veautés sur le **R**éseau **I**ntranet de la **R**ATP (NOURIR)*



*Figure 0. 6 **A**bonnement aux **M**odifications de l'**I**ntranet (AMI)*

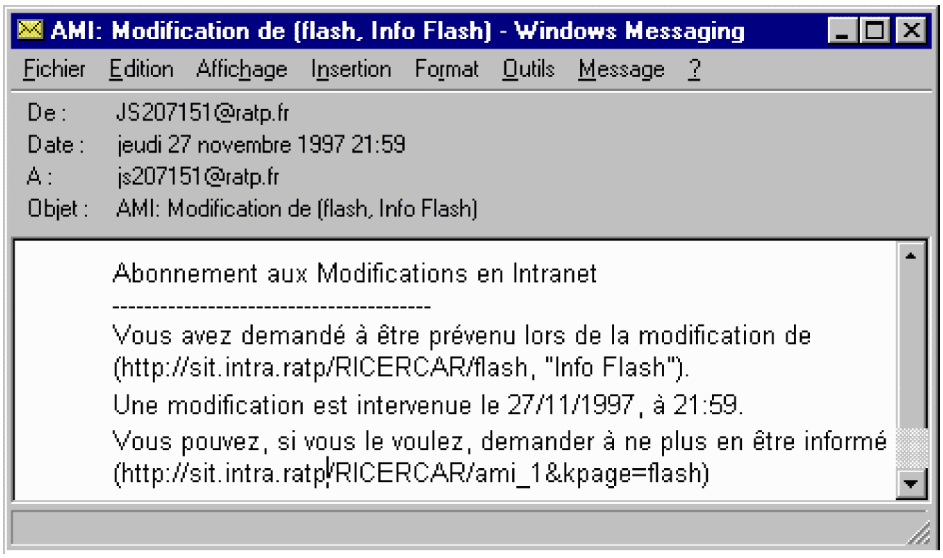


Figure 0. 7 Les abonnés sont prévenus à chaque modification du Documents.



Figure 0. 8 Recherche sur l'Intranet par Mots-clEfs (RIME)



Figure 0. 9 Documents qui RIMEnt avec les critères de recherche

6.6.3.2 Cycle de Vie

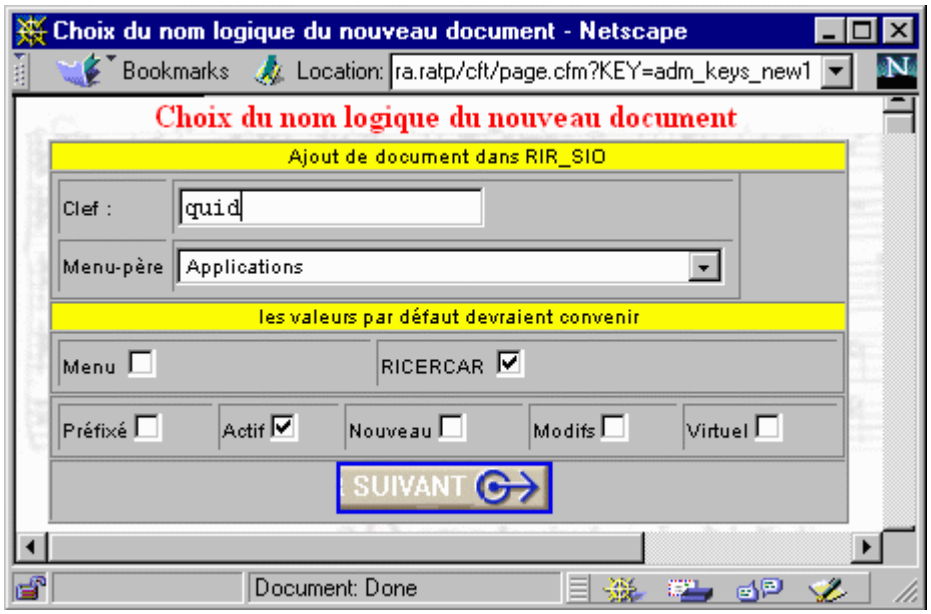
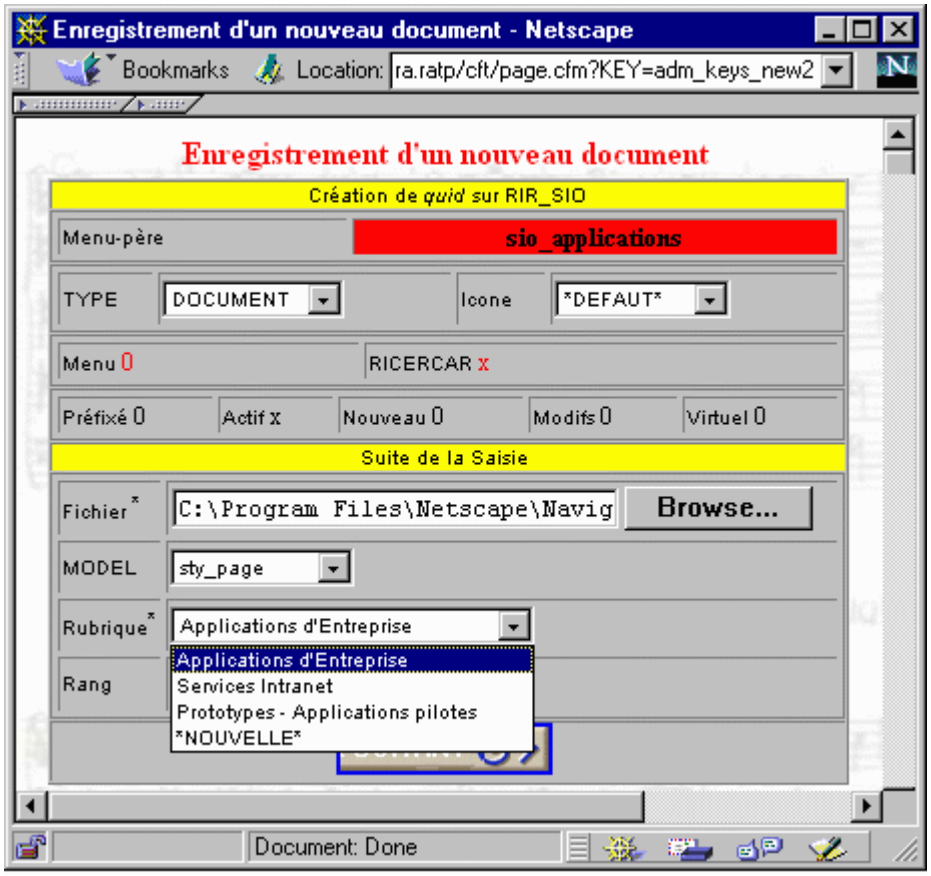


Figure 0. 10 Méthode de création d'un Documents



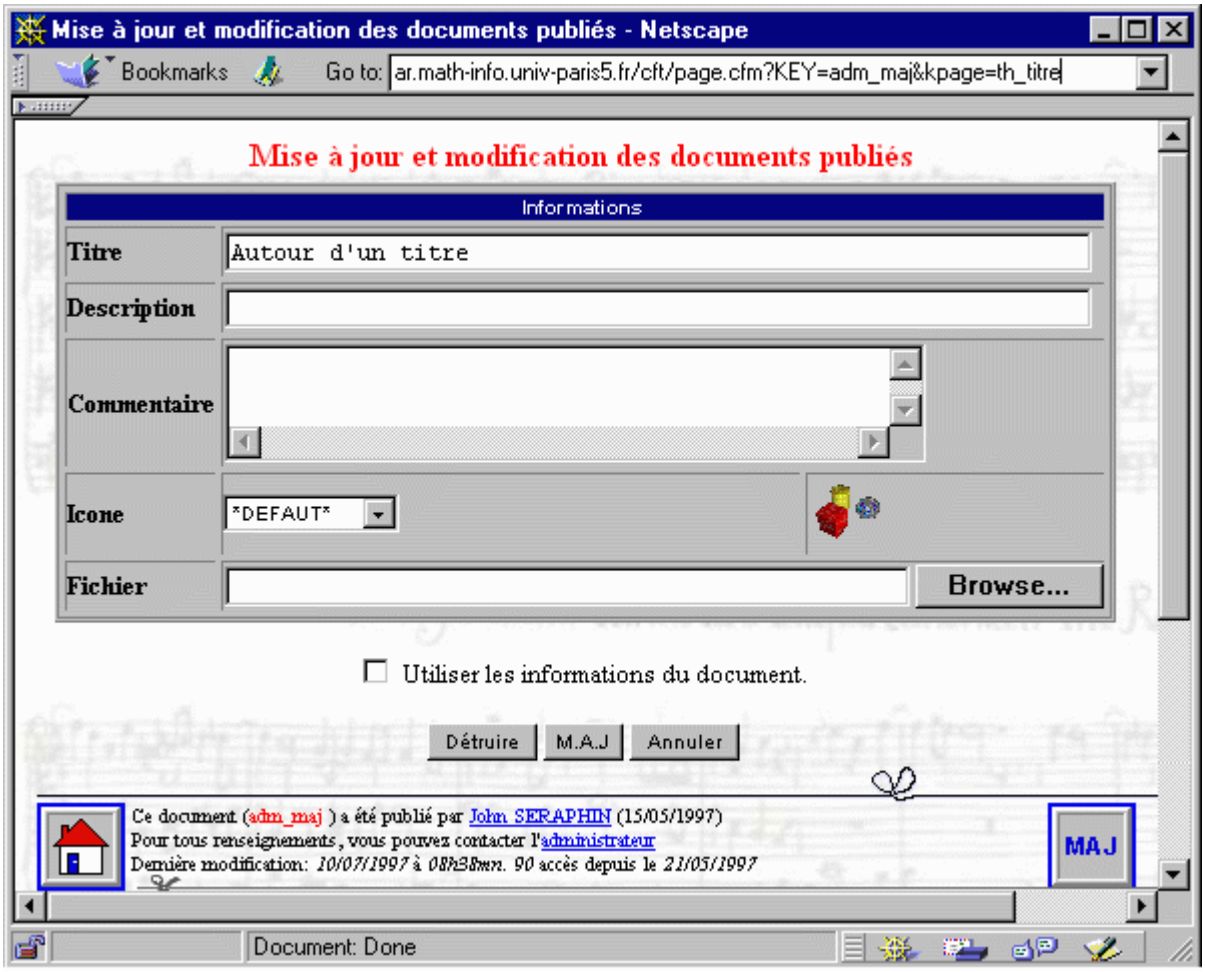


Figure 0. 11 Méthode de mise à jour et de destruction

6.6.3.3 Statistiques



Figure 0. 12 Le menu des statistiques

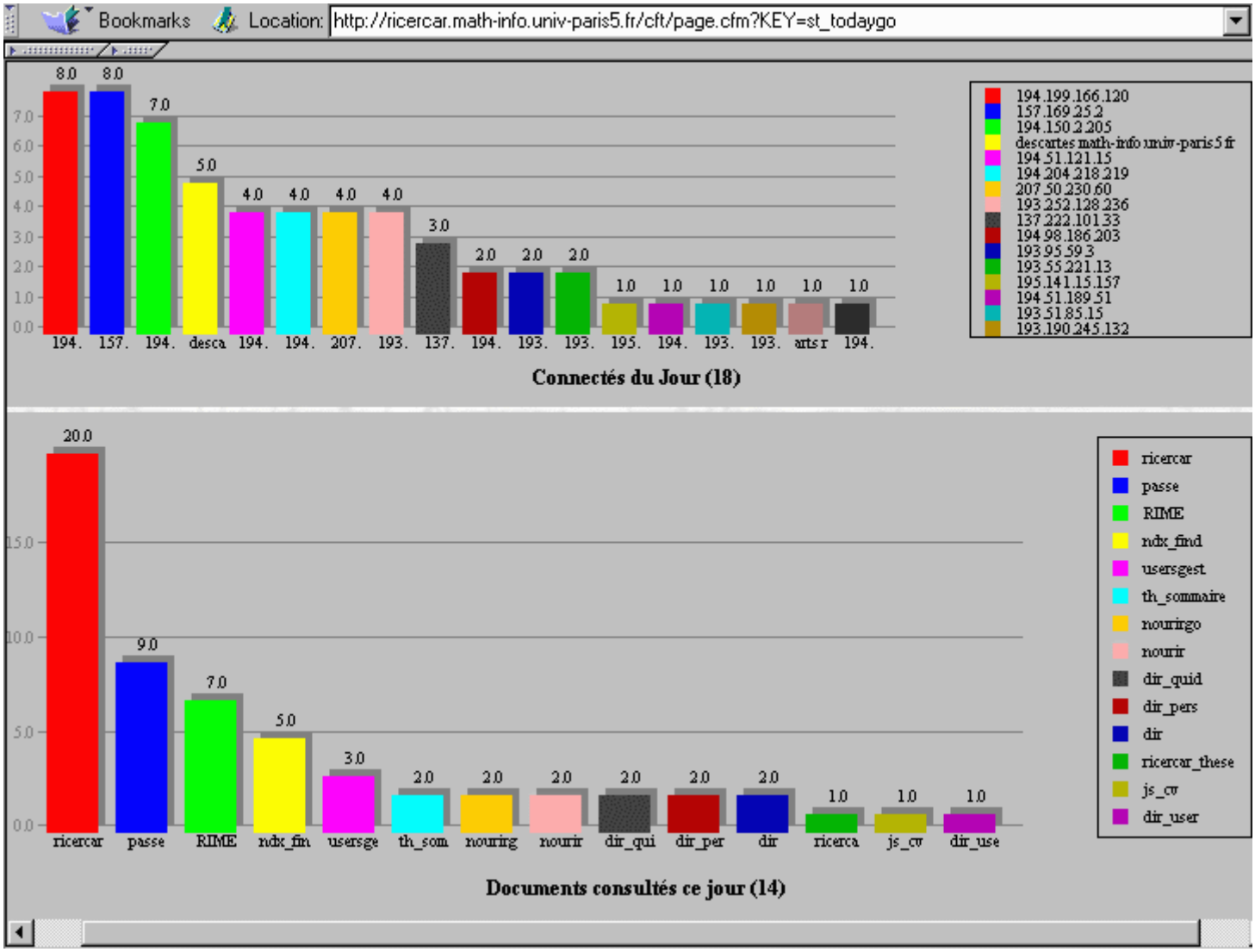


Figure 0. 13 Exemple : connectés d'un jour

6.6.4 Les Objets Applicatifs



Figure 0. 14 Extrait du menu des applications

6.6.4.1 Antilope



Figure 0. 15. Le menu de la version intranet de l'annuaire d'entreprise

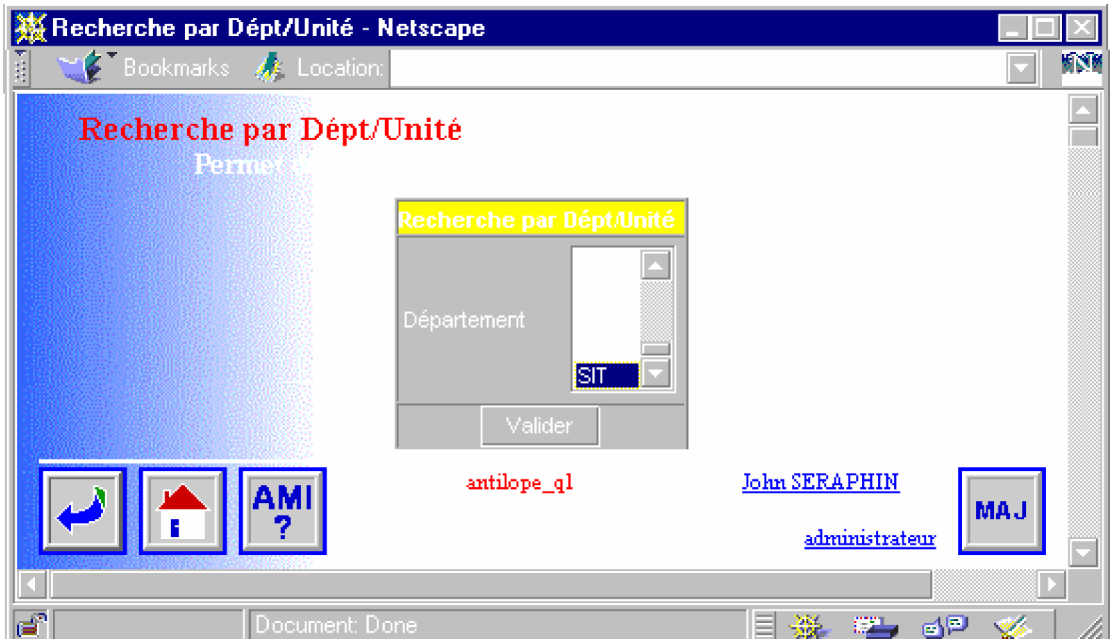


Figure 0. 16 Précisions successives de la recherche à partir de la structure de la RATP

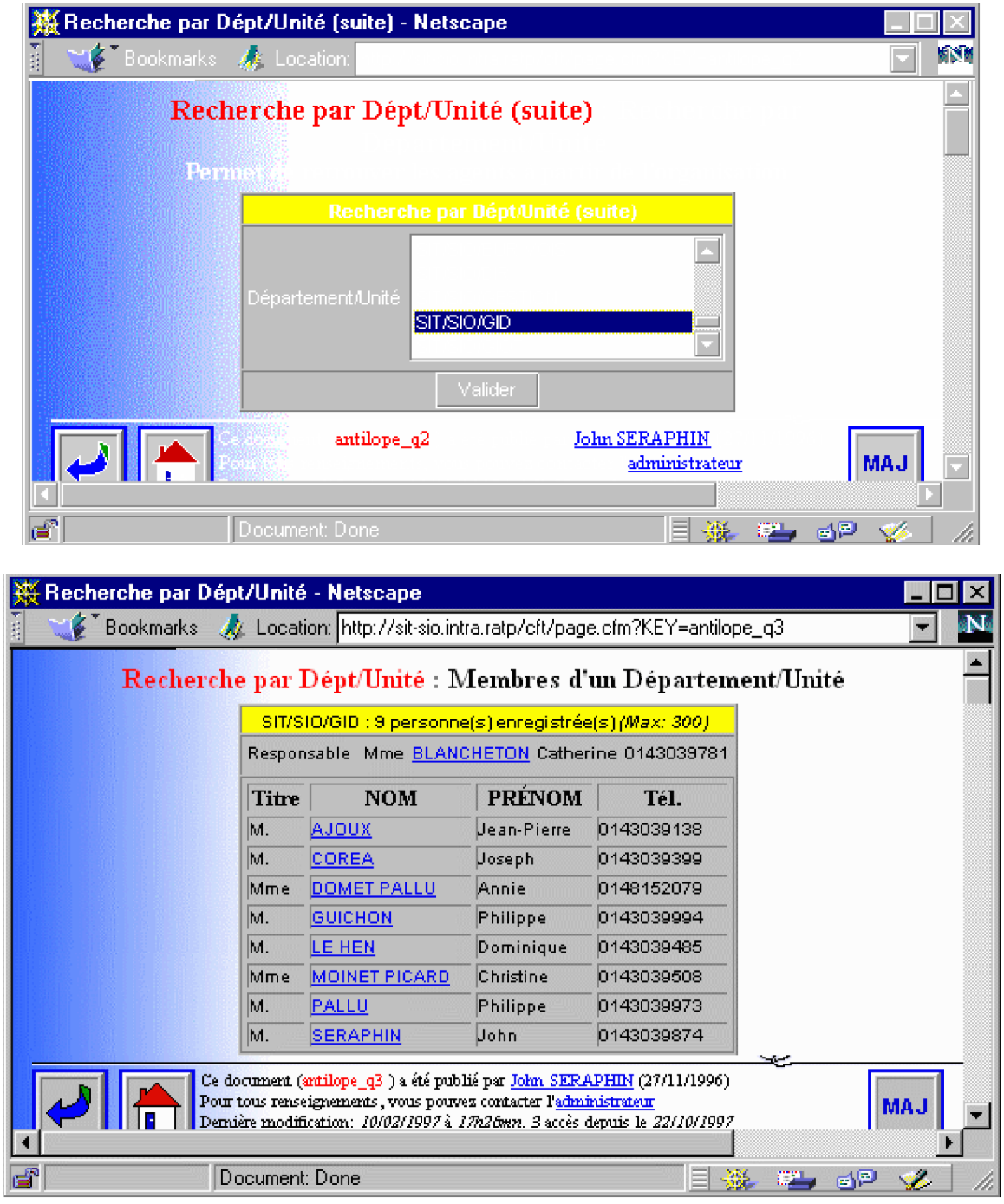


Figure 0. 17 La liste triée des membres de l'Unité Locale



Figure 0. 18 L'extrait de la fiche individuelle d'un agent

6.6.4.2 Infocentres

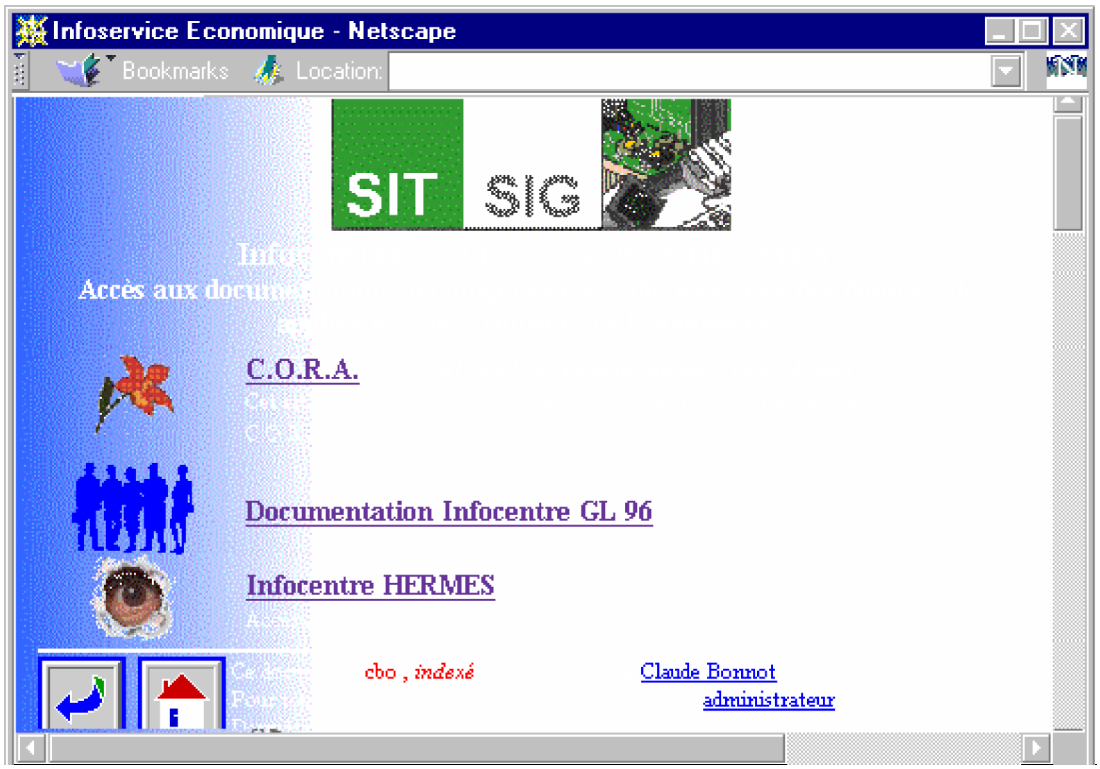


Figure 0. 19 Des interfaces d'accès ou de documentation automatique de plusieurs infocentres sont en cours de conception par les chefs de projet (de façon autonome).



6.6.4.3 95 PPP



Figure 0. 20 La classe « Menus » décline l'identité visuelle commune, selon les goûts des responsables de PPP.

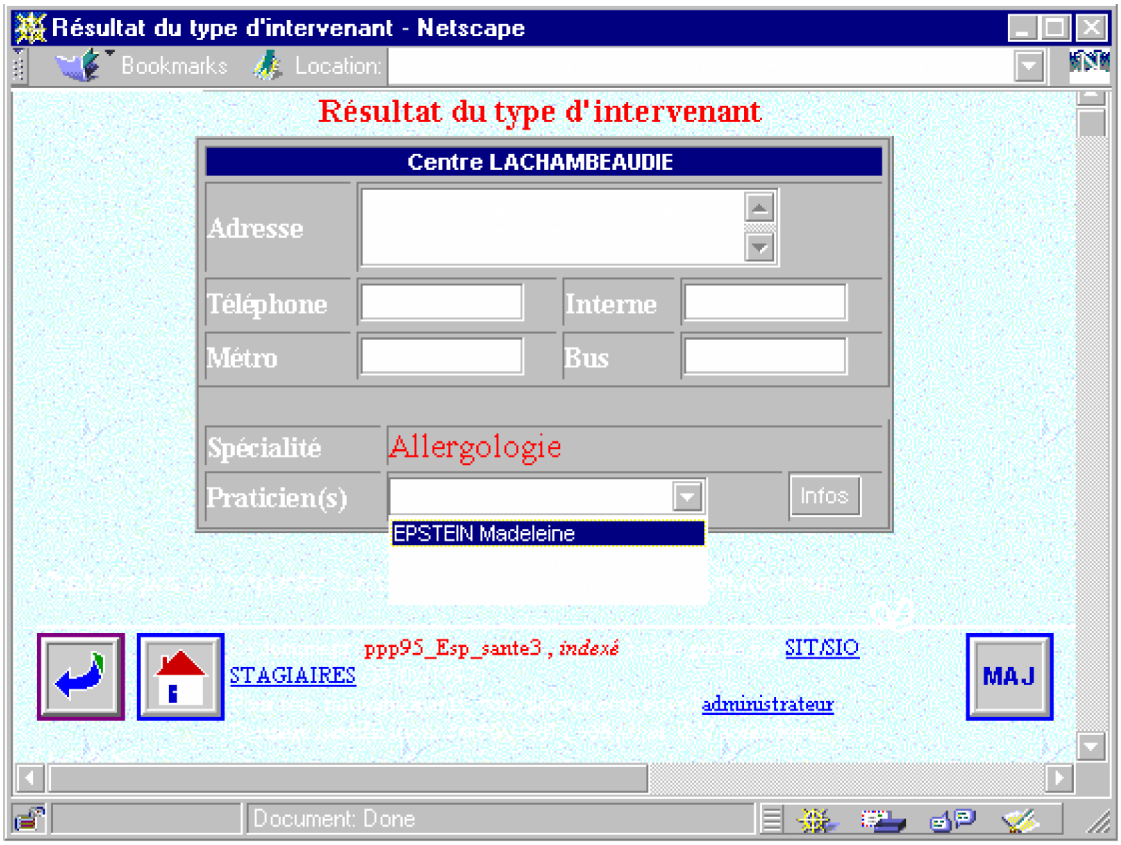
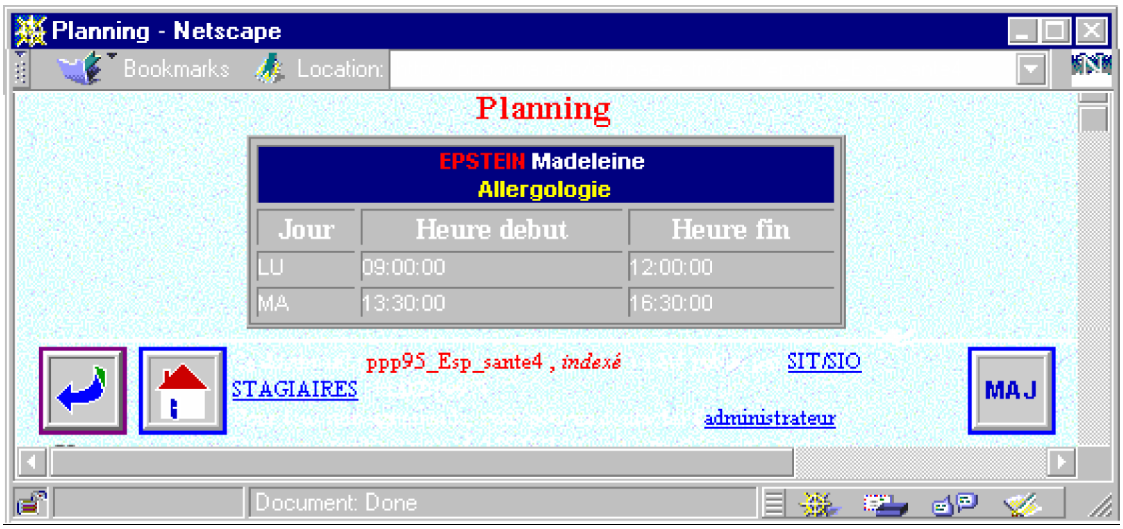


Figure 0. 21 Une fois le centre de soins choisi, il est possible de sélectionner la spécialité voulue ainsi que les horaires de la semaine courante.



6.6.4.4 Annuaire des UFR

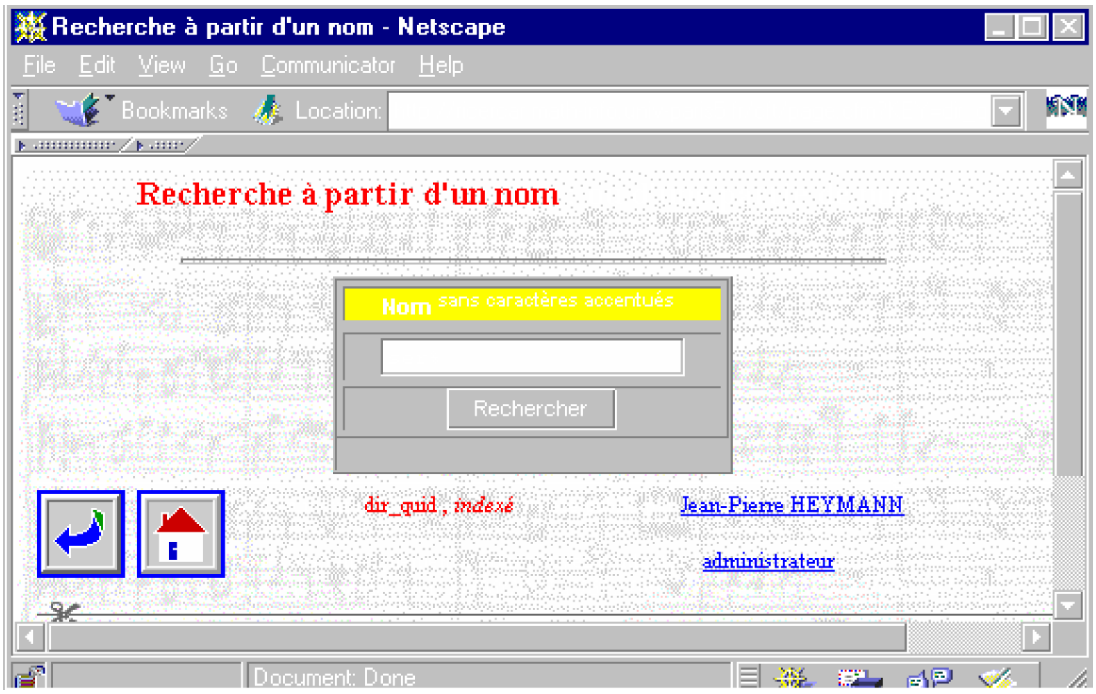


Figure 0. 22 Recherche dans l'annuaire



Figure 0. 23 Liste triée des réponses avec liens pour plus de détail.

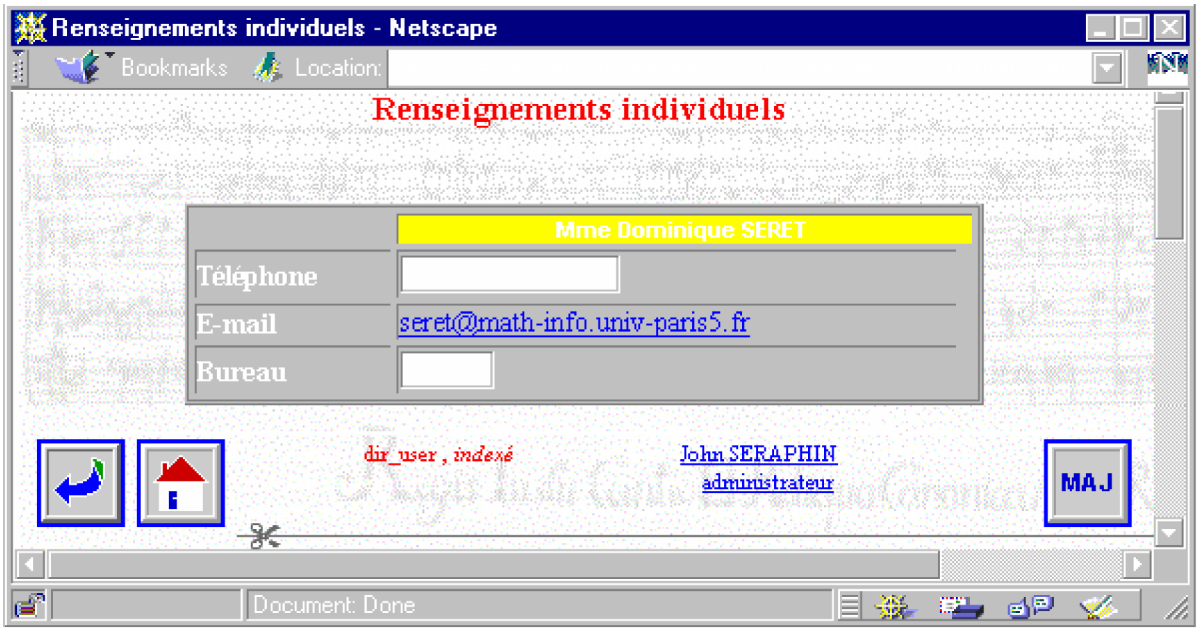


Figure 0. 24 À tout seigneur, tout honneur. En attendant qu'une photo soit disponible.

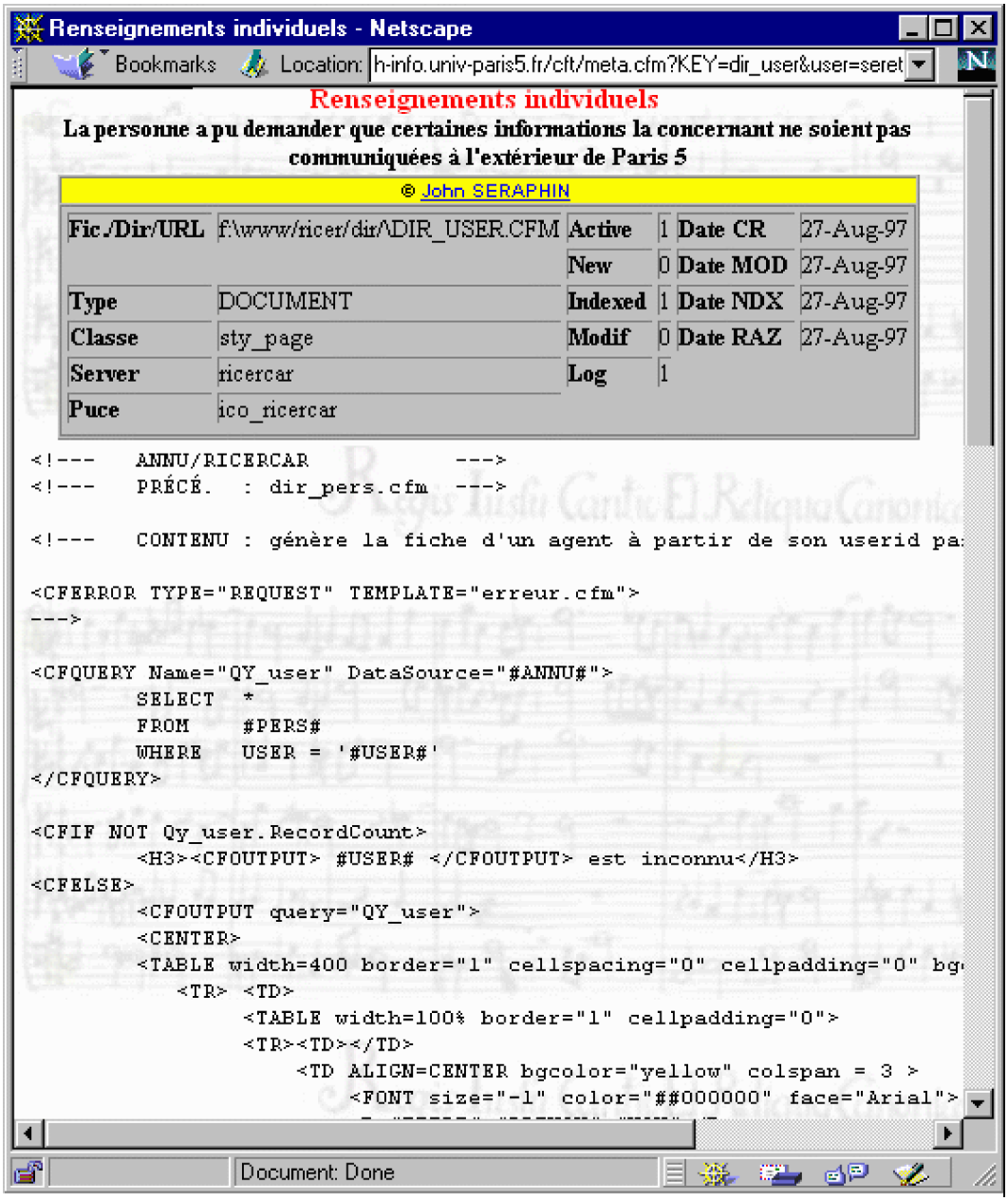


Figure 0. 25 Le code (extrait) de la méthode précédente.

6.7 Table des figures

Figure 2. 1. L'héritage permet de créer une hiérarchie de classes de plus en plus spécialisées. 37
Figure 2. 2. L'héritage multiple peut entraîner des incohérences dans la modélisation. 38
Figure 2.3. Le modèle ObjVlisp permet une représentation finie du « tout objet », où la classe « Class » est la matrice de toutes les métaclasses du système. 39
Figure 2.4. L'encapsulation isole la structure de l'objet des méthodes qu'il implémente 40
Figure 2.5. Le polymorphisme permet d'implémenter une même méthode avec des algorithmes différents 41
Figure 2.6. L'agrégation permet de décrire une « voiture » comme l'agrégation d'un « moteur », d'un « châssis », etc. La classe « pneu » délègue ainsi sa méthode « gonfler » à la classe « voiture ». 42
Figure 2. 7 L'architecture définie par l'OMG. 48
Figure 2. 8 Une requête client-serveur transitant par l'ORB. 49
Figure 2. 9 Architecture schématique d'un ORB. 51
Figure 2. 10 Activation d'une implémentation. 54
Figure 2. 11 L'architecture Inter-ORB de CORBA 2. 55

Figure 3. 1 Un message d'erreur HTTP est parfois difficilement exploitable !.. 64
Figure 3. 2. ... ou n'est pas très parlant pour l'utilisateur. 65
Figure 3. 3. Diagramme des cas d'utilisation d'un intranet 68
Figure 3. 4. Un document publié est référencé par un ensemble indéterminé d'objets. 70
Figure 3. 5. Toute modification de structure (ici un déplacement) provoque une erreur lors de l'accès aux documents. 71
Figure 3. 6 La mise en place d'un dictionnaire fait que les liens ne pointent plus sur le document (variable) en référence mais vers l'entrée (fixe) du dictionnaire 72
Figure 3. 7. Toute modification du document modifie son entrée dans le dictionnaire de façon transparente pour l'utilisateur final. 73
Figure 3. 8. La fédération des dictionnaires assure la cohérence globale de la navigation. La requête d'un objet est automatiquement acheminée vers le serveur ad hoc. 74
Figure 3. 9. L'auteur d'un document insère des liens vers des IDentifiants de Référentiel globaux, représentés par des URL. 75
Figure 3. 10 Les menus (l'arborescence virtuelle du serveur) peuvent être générés dynamiquement à partirdes méta-données du dictionnaire. 76

Figure 4. 1. Un exemple de la représentation concrète de différentes méta-données dans un document (ici un menu). 93
Figure 4. 2. Interface d'administration unique du serveur qui permet une gestion fine au travers des différents onglet. 96
Figure 4. 3. Cheminement d'une requête générant une page HTML à partir d'un interpréteur. 98
Figure 4. 4. Schéma du MCD complet, correspondant au schéma de classes final. 102
Figure 4. 5. Schéma physique de la base fédérée de RICERCAR.Rappel (cf. § 0.4.2, p.9) : les tables en MAJUSCULES sont répliquées sur l'ensemble des serveurs. 104
Figure 4. 6. L'introduction d'un niveau « méta » dans l'interprétation permet de représenter les méta-données de RICERCAR dans un « Référentiel d'Interfaces ». 108
Figure 4. 7. Diagramme de classes initial : Clients, Serveurs et Documents. 109
Figure 4. 8. diagramme de collaboration correspondant au traitement de l'appel d'un document doc. 110
Figure 4. 9. Exécution de "(send localhost 'goto-thème 'th1)" en mode trace, à partir d'une base d'exemple. 112
Figure 4. 10. Documents se spécialise en Menus et Pages. 113
Figure 4. 11. Généralisation de la composition des Menus. 115

Figure 4. 12. Exécution en mode trace de "(send m1 'goto)" qui génère désormais un appel récursif pour chaque élément du menu. 116
Figure 4. 13. Le paramétrage des variables de classe de Modèles, permet d'instancier les différentes sous-classes de Documents 124
Figure 4. 14. Diagramme de Classes de RICERCAR. 126
Figure 4. 15. Les différentes classes de Documents dans le détail (ici la table Model). 127
Figure 4. 16. La réflexivité de RICERCAR assure l'homogénéité entre les applications d'entreprise et le noyau du système, qui utilisent les mêmes représentations 137
Figure 4. 17. Interrogation de la méta-base pour retrouver la classe de doc. 139
Figure 4. 18. Cette étape (3), qui est la séquence la plus courte possible, correspond à la récupération d'une adresse mémorisée dans un calepin. 140
Figure 4. 19. Cas le plus complexe (rare) de calcul de page, effectué par RICERCAR. Première phase : renvoyer la requête au « bon » serveur. 142
Figure 4. 20. Récupération de la méthode de la classe du document 144
Figure 4. 21. Le document est protégé et l'utilisateur n'est pas identifié. Invocation du document d'identification (passe). 145
Figure 4. 22. Récupération des informations relatives au document (formulaire) d'identification 146
Figure 4. 23. L'utilisateur s'identifie. Le formulaire POSTe sa requête originelle, accompagnée de son identification. 147
Figure 4. 24. Récupération des informations relatives au document (formulaire) d'identification 148
Figure 4. 25. Correspondance fonctionnelle entre un ORB et RICERCAR. 150

Figure 6. 1. Diagramme des Classes de la Bibliographie. 165
Figure 6. 2. Structure de la RATP, les quatre pôles et leurs départements associés. 189
Figure 6. 3. Le DNS « officiel » et celui de la RATP 192
Figure 6. 4. La passerelle ARTS 193
Figure 6. 5 ***NOU****veautés sur le* ***R****éseau* ***I****ntranet de la* ***R****ATP (NOURIR)* 225
Figure 6. 6 ***A****bonnement aux* ***M****odifications de l'****I****ntranet (AMI)* 226
Figure 6. 7 Les abonnés sont prévenus à chaque modification du Documents. 226
Figure 6. 8 Recherche sur l'Intranet par Mots-clEfs (RIME) 227
Figure 6. 9 Documents qui RIMEnt avec les critères de recherche 228
Figure 6. 10 Méthode de création d'un Documents 229
Figure 6. 11 Méthode de mise à jour et de destruction 230
Figure 6. 12 Le menu des statistiques 231
Figure 6. 13 Exemple : connectés d'un jour 232
Figure 6. 14 Extrait du menu des applications 233
Figure 6. 15. Le menu de la version intranet de l'annuaire d'entreprise 234
Figure 6. 16 Précisions successives de la recherche à partir de la structure de la RATP 234
Figure 6. 17 La liste triée des membres de l'Unité Locale 235
Figure 6. 18 L'extrait de la fiche individuelle d'un agent 236
Figure 6. 19 Des interfaces d'accès ou de documentation automatique de plusieurs infocentres sont en cours de conception par les chefs de projet (de façon autonome). 237
Figure 6. 20 La classe « Menus » décline l'identité visuelle commune, selon les goûts des responsables de PPP. 238
Figure 6. 21 Une fois le centre de soins choisi, il est possible de sélectionner la spécialité voulue ainsi que les horaires de la semaine courante. 239
Figure 6. 22 Recherche dans l'annuaire 240
Figure 6. 23 Liste triée des réponses avec liens pour plus de détail. 240
Figure 6. 24 À tout seigneur, tout honneur. En attendant qu'une photo soit disponible. 241
Figure 6. 25 Le code (extrait) de la méthode précédente. 242

6.8 Récapitulatif et index des citations in extenso de Ricercar, constituant des auto-références

Ratio Caractères/Page	*1214,7*
Imprimé, le	*29/04/2008 9:02*
Créé, le	*10/03/97 19:49*
Enregistrements (dernier - nombre)	*28/04/2008 5:03 - 421 fois*
Ratio Mots/Page	*214,78*
Caractères	*297094*
Acrostiches	pp. 3 (crabe) ; 84 ; 86 ; 245
Ratio Caractères/Mot	*5,66*
Sigles	pp. 1 ; 3 ; 3 (crabe) ; 32 ; 68 ; 78 ; 87 ; 88 ; 245