Neuf principes fondateurs pour la conception fonctionnelle dans une architecture orientée services

Thierry MOINEAU
Capgemini
Coeur Defense - 110 Esplanade du Gal de Gaulle
92931 La Defense Cedex
thierry.moineau@capgemini.com

Résumé: Les Systèmes d'Information de nombreuses grandes entreprises se sont construits graduellement au cours des vingt dernières années sous forme d'applications indépendantes où les informations sont dupliquées. Il en résulte des incohérences, des saisies multiples et un service peu satisfaisant pour les utilisateurs et pour l'entreprise. Pour résoudre ces ruptures, il est souvent décidé de restructurer le Système d'Information autour de référentiels de données utilisés par l'ensemble des traitements informatiques suivant un modèle d'architecture orientée services. Ce choix, qui pourrait paraître purement technique, a en fait de nombreuses implications au niveau fonctionnel, qui, si elles ne sont pas prises en compte dès la conception, peuvent conduire à l'échec de la restructuration du système d'information : les cloisonnements se recréent d'eux mêmes, les invocations de services se transforment en couplage implicite et le système d'information devient encore plus complexe et moins agile qu'avant. Nous présentons ci-dessous neuf principes de conception, issus de notre expérience sur plusieurs grands programmes de refonte du Système d'Information.

Mots-clés : Architecture Orientée Service, Service Oriented Architecture, SOA, principes de conception.

--oo0oo--

Les Systèmes d'Information de nombreuses grandes entreprises se sont construits graduellement au cours des vingt dernières années sous forme d'applications indépendantes où les informations sont dupliquées.

Ceci se traduit par les cinq fameuses ruptures

- Rupture des applications : les mises à jour des données ne sont pas répercutées entre applications.
- Rupture des identifiants : une même information est accessible via de multiples identifiants (par exemple un même article est identifié différemment par l'application de gestion des commande et par l'application de gestion des stocks), ce qui rend les répercutions des mises à jour difficiles voir impossibles.
- Rupture de la chaîne informatique : les échanges entre applications ne sont pas industrialisés, ce qui entraîne des défauts de traitement et des erreurs dans la répercutions des mises a jour.
- Rupture temporelle : les délais de répercussion des mises à jour d'information entre applications sont longs (plusieurs semaines voire plusieurs mois).
- Rupture géographique : les données sont dispersées dans les applications implantées dans les différentes entités géographiques (pays par exemple).

Il en résulte des incohérences, des saisies multiples et un service peu satisfaisant pour les utilisateurs et pour l'entreprise.

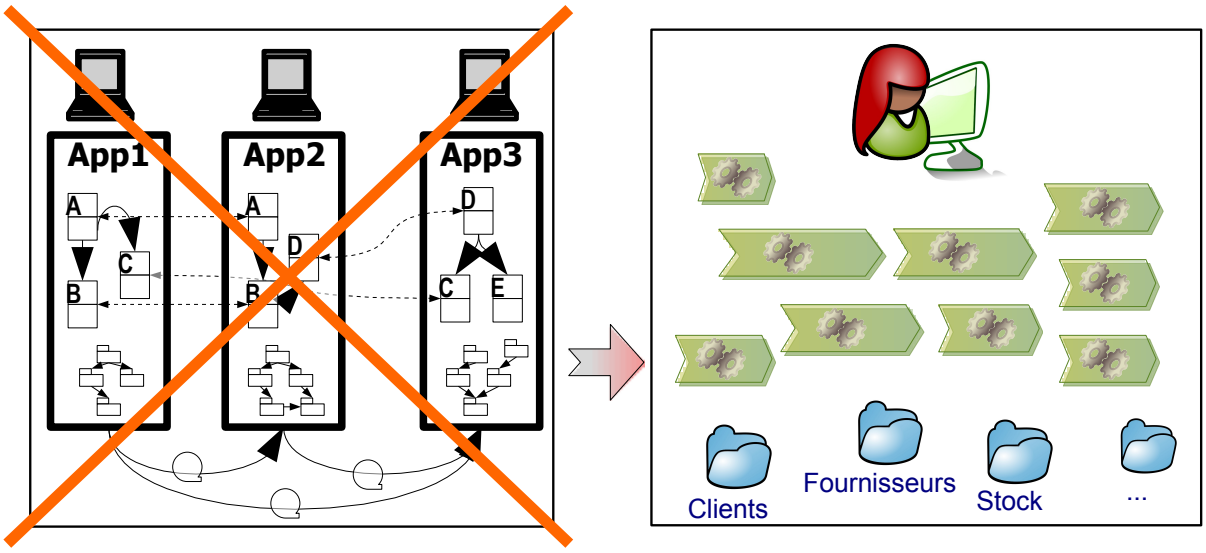
Pour résoudre ces ruptures, il est souvent décidé de restructurer le Système d'Information autour de bases de données transverses utilisés par l'ensemble des traitements informatiques. Ce choix, qui pourrait paraître purement technique, a en fait de nombreuses implications au niveau fonctionnel et impose le respect d'un corpus de principes de conception, sous peine soit de réintroduire les ruptures soit de rendre ingérable la complexité du Système d'Information.

L'objet de ce document est d'expliciter les principaux principes de ce corpus. Nous ne réclamons aucune paternité sur ces principes, que nous considérons en fait comme du bon sens – notre expérience sur plusieurs grands programmes de restructuration d'un système d'information d'entreprise en mode SOA nous a démontré qu'il n'est pas inutile de les rappeler en un ensemble cohérent et justifié.

Principe 1: Modularité et encapsulation

Le Système d'Information est partitionné en sous-ensembles fortement cohérents et faiblement couplés, les Services Fonctionnels (SF) :

- les Référentiels fournissent à l'ensemble du SI les services liés à leurs données, et
- les Pilotes fournissent l'implémentation d'un ensemble de traitements.



Ce principe est un principe lié au Système d'Information et n'est pas un principe métier. Il découle de deux raisons principales :

- complexité de réalisation : il est illusoire de vouloir construire un SI de la taille d'un SI d'une grande entreprise en un seul morceau et il est nécessaire de le découper en sous-ensembles plus faciles à appréhender ;
- maintenabilité : dans un SI de la taille d'un SI d'entreprise, il est indispensable de pouvoir modifier une partie du SI tout en contrôlant les impacts, en particulier la localisation de ces impacts, de manière à ne pas avoir à reconstruire et re-qualifier l'ensemble du SI à chaque évolution d'un des ses composants.

Pour cela, le SI est segmenté suivant des critères fonctionnels par identification de sous-ensembles faiblement couplés (une évolution d'un sous-ensemble impacte au minimum les autres sous-ensembles) et fortement cohérents (les données et les traitements à l'intérieur d'un sous-ensemble sont conceptuellement proches). Ces sous-ensembles sont appelés *Services Fonctionnels* (SF), car il s'agit ici d'une structuration purement fonctionnelle (les aspects applicatifs ou techniques ne doivent pas entrer en ligne de compte à ce niveau).

Conformément au choix initial de structurer le SI autours de bases de données transverses, on distingue deux types de Services Fonctionnels :

- les Référentiels, qui fournissent à l'ensemble du SI les services d'accès à leurs données, et
- les Pilotes, qui fournissent l'implémentation d'un ensemble de traitements.

Une autre raison de cette distinction est liée au niveau d'agilité intrinsèque des différents SF. Notre expérience nous montre que les traitements doivent changer fréquemment en fonction de l'évolution toujours plus rapide du marché et des exigences des clients. A l'opposé, la structuration des données de l'entreprise est liée au fondamentaux du segment de marché de l'entreprise et évolue peu et lentement. Il est donc pertinent, pour faciliter la maintenance du SI, de séparer les Référentiels ayant un cycle d'évolution lent des Pilotes nécessitant des évolutions rapides.

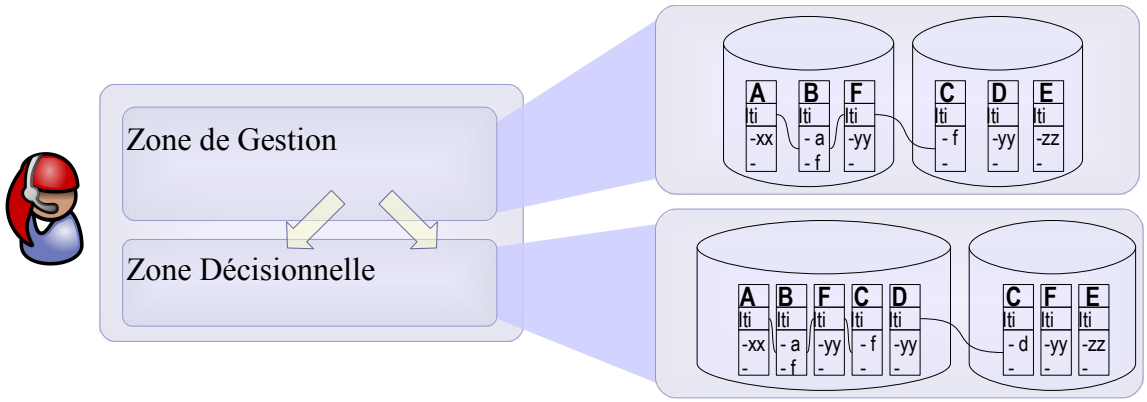
Afin de maximiser la maintenabilité du Système d'Information, il est essentiel de limiter et de contrôler les impacts d'une modification locale de l'implémentation d'un SF - par exemple lors de la correction d'un bogue, lors d'une évolution non fonctionnelle pour améliorer les performances ou lors d'un changement de plate-forme technologique. Dans ce cadre, les SF doivent masquer les détails internes de leur implémentation, en particulier la structuration interne du stockage de leurs données. Les Référentiels ne donnent donc accès à leurs données que via des offres de services précisément décrites et publiées. Ceci est résumé dans la règle suivante :

> ***Règle 1a**: Les Référentiels encapsulent leurs données et communiquent via des offres de services décrites et publiées.*

Principe 2: Unicité de la localisation d'une information

Une information est gérée en un point unique du Système d'Information.

Des copies non modifiables peuvent exister pour des besoins spécifiques (archivage ou recoupement en temps différé) ; ces copies ne sont pas utilisées par les traitements usuels.



L'objectif de ce principe est de garantir le maintien de l'intégrité et de la fraîcheur des informations dans le Système d'Information ; il est le fondement de l'approche permettant d'éviter les ruptures et l'incohérence des informations.

Il s'agit ici de bien distinguer le concept de valeur du concept d'information : une même valeur peut correspondre à des informations différentes. Par exemple, l'adresse courante d'un client et l'adresse donnée sur un bon de commande peuvent être la même valeur mais pas nécessairement la même information : si le client a coché la case 'livraison à mon adresse habituelle' alors il s'agit de la même information, sinon il s'agit de la même valeur mais de deux informations différentes – cette distinction est importante en particulier si le client vient de faire une demande de changement d'adresse par courrier postal et passe une commande par internet avant que le changement d'adresse ne soit effectif. Le principe ci-dessus s'applique bien aux informations et pas aux valeurs : les valeurs doivent être dupliquées quand il s'agit d'informations différentes.

La seconde partie du principe reconnaît le caractère particulier de la Zone Décisionnelle :

- les traitements de cette zone ne participent pas de manière synchrone à l'activité courante de l'entreprise
- ces traitements ont besoin d'une structuration spécifique des données pour permettre les recoupements, et
- ces traitements n'ont pas besoin d'informations fraîches (au contraire, ils utilisent en général des historiques, souvent des informations de millésimes précédents).

La duplication des informations est donc autorisée entre Zone de Gestion et Zone Décisionnelle. Afin d'éviter les pertes d'intégrité, les informations de la Zone Décisionnelle ne peuvent pas être utilisées par les traitements de la Zone de Gestion. Par contre un traitement de la Zone de Décisionnelle peut utiliser (en lecture ou en écriture) des informations de la Zone de Gestion. Il est en particulier possible qu'un traitement de la Zone Décisionnelle mette à jour des données de la Zone de Gestion (par exemple des données statistiques de vente d'un produit).

Une conséquence importante de ce principe est la suivante :

> ***Règle 2a :*** *Les modèles de données des référentiels de la Zone de Gestion sont disjoints deux à deux.*

Pour permettre de gérer les relations entre les données des différents référentiels, il est donc nécessaire d'introduire le concept d'*identifiant*. Un référentiel de la Zone de Gestion contient donc ses données et des identifiants permettant de pointer vers les informations des autres référentiels. Ces identifiants se doivent de respecter les propriétés spécifiques décrites dans la règle suivante :

> ***Règle 2b :*** *Chaque information du Système Informatique possède un Identifiant Transverse Informatique, qui vérifie les 4 propriétés suivantes :*
>
> *Non-signifiance : un identifiant n'a pas de signification*
> *Non-modificabilité : un identifiant ne peut être modifié*
> *Non-destruction : un identifiant ne peut être détruit même s'il n'est plus utilisé*
> *Non-réutilisation : un identifiant ne peut être réutilisé même après destruction de l'information*
>
> *Les relations entre informations se font exclusivement via les Identifiants Transverses Informatiques.*

La propriété de non-signifiance est une conséquence du principe d'unicité de la localisation d'une information (principe 2) : si l'identifiant était signifiant alors il y aurait duplication d'information et donc fort risque d'incohérence. Prenons par exemple le numéro de sécurité sociale d'une personne. Ce numéro ne peut pas servir comme identifiant informatique car on pourrait en déduire la date de naissance de la personne ; si jamais il s'avérait que la date donnée lors de l'attribution du numéro était fausse alors il faudrait corriger la date de naissance dans le référentiel des personnes - mais alors tous les autres applicatifs, qui utilisent le numéro de sécurité sociale pour en déduire l'âge de la personne, donneraient des résultats incohérents. Pour éviter ces incohérences, on pourrait imaginer de changer l'identifiant pour y mettre la bonne date de naissance mais il faudrait alors diffuser cette modification dans tous les référentiels qui font référence à cette personne. Il faudrait donc soit gérer dynamiquement la liste des référentiels qui font référence à cette personne, soit avertir tous les référentiels. Dans les deux cas, il s'agit de mécanismes complexes et difficiles à rendre résistants à tous les types de pannes possibles. Il est clair qu'il est beaucoup plus simple et robuste d'utiliser des identifiants non signifiants et d'invoquer le référentiel des personnes pour connaître la date de naissance.

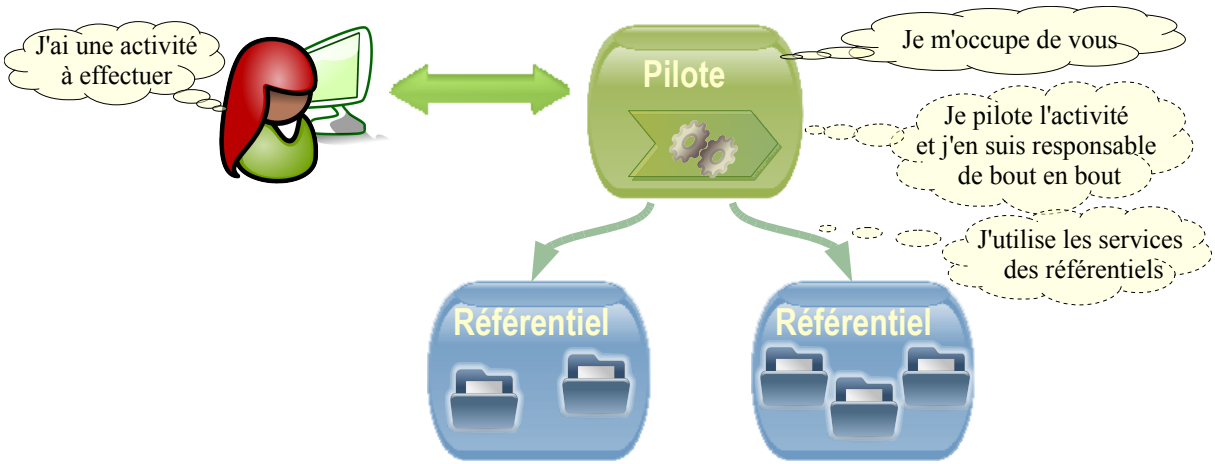
La propriété de non-modifiabilité permet de garantir la pérennité des identifiants. En effet, comme nous l'avons vu, un identifiant transverse informatique peut être stocké dans d'autres référentiels. Si on modifiait cet identifiant, alors les références contenues dans les autres référentiels deviendraient invalides et les relations entre ces informations ne pourraient plus être utilisées. Il faudrait donc avertir les autres référentiels de la modification pour qu'ils la prennent en compte. Comme ci-dessus, on pourrait diffuser la modification à l'ensemble des référentiels mais cela générerait, dans un système de la taille d'un SI d'entreprise, un gigantesque trafic de messages de mise à jour. On pourrait aussi ne diffuser qu'aux référentiels qui font référence à cet identifiant, ce qui imposerait de gérer une liste inverse des références ; autrement dit un référentiel devrait savoir à tout moment qui fait référence à chacune de ses informations – c'est clairement un mécanisme complexe et difficile à rendre robuste aux erreurs. Dans les deux cas, il faudrait gérer les cas d'indisponibilité des référentiels (panne ou maintenance programmée), ce qui rajouterait un niveau de complexité et de risque de dysfonctionnement.

Les propriétés de non-réutilisation et de non-destruction servent à garantir que les identifiants feront bien toujours référence à l'information à laquelle ils doivent faire référence - autrement dit, éviter que la commande de Mme Michu ne soit attribuée plus tard par erreur à M. Martin (c'est essentiel pour la traçabilité légale).

Il est important de faire la distinction entre identifiants internes informatique et identifiants utilisés dans la vraie vie. Ces derniers sont généralement ambigus soit parce que des être humains doivent s'en souvenir (il est plus simple de faire référence à Mme Martin par son nom que par son numéro d'identité fiscale), soit parce des raisons culturelles ou légales interdisent l'usage d'identifiants non ambigus (demander à un client de s'identifier par ses empreintes digitales ou par une analyse ADN n'est acceptable que dans des cas très particuliers). Les référentiels du SI contiennent des identifiants de la vie réelle pour les échanges avec les humains, mais n'utilisent que les identifiants internes pour les échanges informatiques au sein du SI.

Principe 3: Unicité de la localisation du pilotage des activités

L'utilisation du Système d'Information est modélisée sous forme d'*activités métier*. Une activité métier est une unité de travail, telle que vue par l'utilisateur final, et doit respecter la règle des 4 unités suivante : unité de lieu, unité de temps, unité d'acteur et d'action.
Toute activité métier est pilotée de bout en bout sans délégation par un unique Pilote, qui enchaîne des demandes de services à des Référentiels.



La première partie du principe se justifie par le besoin, pour un SI de la taille d'un SI d'entreprise, d'un minimum de normalisation et d'homogénéisation des concepts de modélisation au niveau métier pour éviter la création d'un capharnaüm préjudiciable à la fois aux informaticiens et, dans une certaine limite, aux utilisateurs (de part la cohérence ergonomique qu'elles apportent).

La seconde partie de ce principe est l'équivalent au niveau des traitements du principe d'unicité de la localisation des informations (principe 2). Sa justification en est similaire : il s'agit de modulariser le Système d'Information en limitant les couplages forts (on notera que la règle des quatre unités de lieu, de temps, d'acteur et d'action garantit déjà une forte cohérence de l'activité). En effet, éclater le pilotage d'une activité sur plusieurs SF signifierait que ces SF se passeraient le relais en cours d'une interaction, interaction qui est pourtant vue par l'utilisateur comme une unité de travail. Pour éviter les ruptures ergonomiques et sémantiques, ce changement de pilote nécessiterait une forte connaissance réciproque des SF, jusqu'au niveau de l'enchaînement des leurs écrans, et entraînerait donc un couplage fort qui conduirait à la complexification du Système d'Information et à sa rigidification – donc à des coûts et des délais de maintenance fortement accrus.

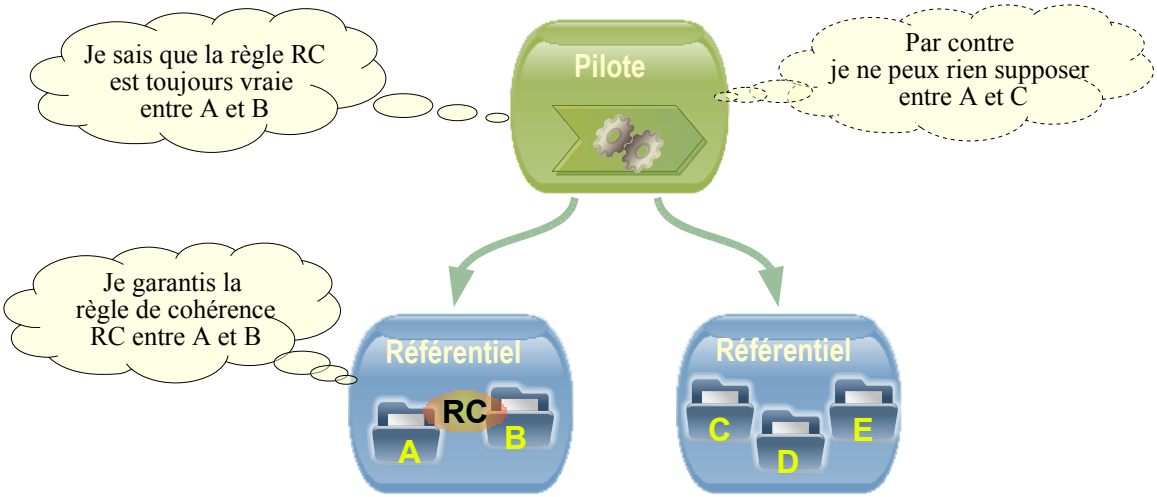
D'autre part, les cascades de passages de relais successifs de Pilote en Pilote rendraient très complexe le traitement des cas non nominaux – en particulier les erreurs. En effet si une erreur (fonctionnelle ou technique) se produisait dans le troisième Pilote de la cascade, alors il faudrait revenir dans le second Pilote au bon endroit dans le bon écran, où il est fort possible qu'il faille retourner dans le premier Pilote qui est celui qui porte l'enjeu de l'activité puis revenir dans le troisième Pilote pour finaliser le traitement. Pour garantir une ergonomie acceptable à l'utilisateur, il faudrait donc introduire un couplage fort entre les différents Pilote de la cascade. Il est donc bien plus simple de faire piloter une activité métier par un unique Pilote.

Bien entendu, ce principe ne doit pas être utilisé pour re-créer un système d'information cloisonné : le Pilote responsable de l'activité doit faire les demandes de services appropriés vers les Référentiels appropriés (y compris ceux créés par d'autre projets).

Certains traitements ne peuvent respecter pas la règle des quatre unités : il s'agit d'enchaînements d'activités soit sur des périodes temporelles différentes soit par des acteurs différents. Pour cela on introduit des Pilotes dont le rôle est de gérer les transitions entre acteurs et entre activités suivant les modalités en vigueur dans l'entreprise (souvent appelées règles de *workflow*). La règle d'unicité de la localisation des traitements s'applique aussi à ces Pilotes – même si une telle modélisation exige une attention toute particulière pour respecter les règles de faible couplage entre SF.

Principe 4: Garantie de la cohérence fonctionnelle

Une information est propriété de son Référentiel qui est seul responsable de la garantie de sa cohérence fonctionnelle.



Ce principe peut paraître une simple conséquence du principe d'unicité de la localisation de l'information (principe 2) ; c'est en fait un changement de paradigme fort mais subtil dont les conséquences sont nombreuses.

Dans un Système d'Information en mode cloisonné, les informations sont totalement gérées par une application, qui connaît tous les traitements pouvant agir sur celles-ci. Le concepteur d'une application peut donc, quand il spécifie un traitement, prendre en compte les besoins des autres traitements de l'application – par exemple,

interdire certaines modifications des données qui sont tout à fait correctes dans le cas particulier de ce traitement mais qui introduiraient des incohérences dans d'autres traitements de l'application. Cette approche a montré ses limites : quand l'application grossit, le risque d'oublier une de ces interdépendances cachées devient grand et augmente très significativement les coûts et durées des actions de maintenance évolutive de l'application – au point parfois de figer le Système d'Information par peur d'introduire des régressions dues à ces interdépendances implicites difficilement maîtrisables.

Dans un Système d'Information en mode cloisonné, structuré en nombreuses petites applications, cette approche tentante est souvent source de gains en durée de réalisation, pourvu que l'on accepte les ruptures applicatives, la duplication des informations et leurs incohérences potentielles – incohérences que l'on peut parfois amoindrir par des gros travaux périodiques dits de remise en cohérence (habituellement tous les quelques mois).

Dans le cadre d'un Système d'Information d'entreprise structuré autour de référentiels, la limite décrite ci-dessus prend vite toute sa mesure et il est illusoire de croire que l'on pourra construire (et encore moins maintenir) un SI basé sur des interdépendances implicites, qui seront nécessairement en grand nombre (du simple fait de la taille du SI d'entreprise). Il est donc nécessaire de rendre explicites toutes les interdépendances et le lieu le plus adapté pour cela est le Référentiel : les Référentiels sont donc les seuls garants de la cohérence transverse des informations.

Les conséquences sont multiples.

> ***Règle 4a :*** *Aucun traitement ne peut supposer le respect, même temporaire, d'une règle de cohérence d'un ensemble d'informations si cette règle n'est pas explicitement garantie par un Référentiel.*

En effet, comme les informations sont accessibles par tous les traitements, rien ne garantit qu'un autre traitement n'a pas modifié ces informations sans prendre en compte cette contrainte de cohérence – contrainte dont il n'a pas à avoir connaissance. Si cette contrainte est absolument nécessaire pour garantir l'intégrité du traitement (voire du SI en entier), alors il faudra durant la phase de conception allouer à un Référentiel la responsabilité de la règle de cohérence correspondante. Cette règle de cohérence devra être explicitement et précisément décrite dans la spécification du référentiel.

Il s'agit ici de trouver le bon équilibre entre trop peu de règles dans les référentiels, au risque de ne plus garantir la cohérence transverse du Système d'Information, et trop de règles dans les référentiels, au risque de faire porter aux référentiels de règles qui ne sont pas fondamentales. Il ne faut pas en particulier que les règles garanties par les référentiels ne soient en totale abstraction avec l'aspect dynamique du monde réel - par exemple le strict respect d'une règle du genre « une situation B est interdite si une situation A s'est produite dans le passé » n'est possible que si l'on oublie que la situation A était peut-être une erreur de saisie.

Pour cela, une bonne habitude consiste à considérer les Référentiels comme les garants des règles fondamentales d'un domaine fonctionnel et donc de s'attacher lors de l'urbanisation du SI et de la conception des Référentiels à identifier les fondamentaux fonctionnels du marché de l'entreprise. Notre expérience montre que ces fondamentaux sont assez stables dans le temps, contrairement aux traitements très liés aux évolutions du marché – cette stabilité est donc un discriminant souvent pertinent.

Une autre conséquence de ce principe s'attache à l'historique des informations :

> ***Règle 4b :*** *Aucun traitement ne peut faire d'hypothèse sur l'histoire d'une information, en dehors de l'historique géré par un Référentiel.*

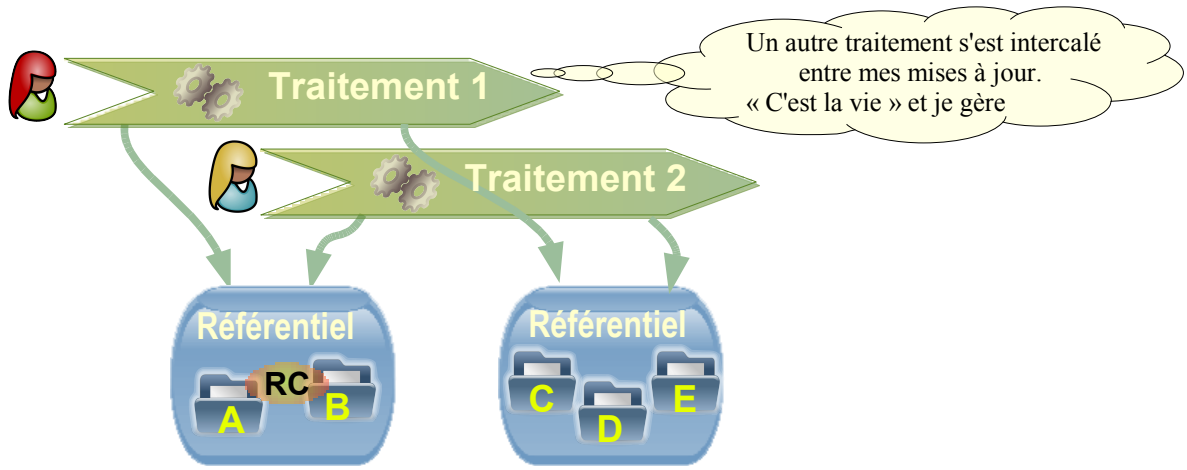
Cette règle peut sembler une conséquence simple du rôle transverse des référentiels (i.e. un autre traitement peut avoir modifié l'information) ; elle va cependant à l'encontre d'une habitude des concepteurs d'applications en mode cloisonné qui font souvent des hypothèses du genre : « si tel attribut est à telle valeur c'est que tel traitement l'a positionné et comme je sais que ce traitement vérifie ceci, je peux donc faire cela sans risque ». Dans une approche basée sur des référentiels transverses de telles hypothèses ne sont simplement plus valides car rien ne garantit que c'est bien tel traitement qui a modifié l'attribut.

S'il est nécessaire de savoir si un certain traitement métier a été appliqué sur une donnée alors il faut explicitement stocker cette information dans un référentiel (par exemple rajouter un attribut aux données d'un client signifiant que ses capacités de paiement ont été vérifiées). Un Pilote n'a pas à savoir qui, en terme applicatif, a été le support du traitement métier : le fait que l'information est stockée dans le référentiel lui suffit (sinon on introduirait des interdépendances cachées). Bien sûr, la signification de cette information doit être clairement définie par le Référentiel et acceptée par tous.

Ces deux règles s'appliquent même si le traitement est celui qui a créé ces données. Ceci peut paraître évident mais n'est pas intuitif, en particulier pour les concepteurs habitués à des applications en mode cloisonné propriétaires exclusifs de leurs données.

Principe 5: Asynchronisme des pilotes (ou l'illusion du temps absolu pour les pilotes)

Les SF Pilotes ne peuvent présupposer qu'ils seront synchronisés sur une même base temporelle, en particulier vis à vis de la mise à jour des données des référentiels.



Ce principe découle pour partie de la complexité, voire de l'impossibilité, de synchroniser un ensemble de processus aussi vaste celui d'une entreprise dans sa globalité et pour partie du besoin de faible couplage entre les différents SF.

En effet, les différents processus sont souvent initialisés de manière indépendante par des acteurs distincts qui ont des contraintes temporelles différentes – ces contraintes étant souvent hors du contrôle de l'entreprise (contraintes législatives ou imposées par un acteur externe pour des raisons commerciales).

Par exemple, un client peut avoir passé une commande en précisant que la livraison devrait se faire à son adresse habituelle. La préparation de la commande peut prendre quelques jours (par exemple suite à une rupture de stock). Entre temps, le client peut vouloir changer son adresse habituelle. Que doit faire le SI quand la commande sera prête ? Doit-il envoyer la commande à l'adresse habituelle au moment de la commande ou à l'adresse habituelle au moment de l'expédition ? Doit-il rejeter la demande de changement d'adresse tant qu'une commande est en cours ? De par notre expérience, la plupart des SI utilise la troisième solution (peut être pas dans cet exemple simpliste mais dans des cas plus complexes). Ceci introduit en fait un couplage fort entre les différents traitements, qui peut conduire soit à des incohérences soit au blocage de certains processus (dans l'exemple, un bon client qui passe de nombreuses commandes risque de jamais pouvoir changer d'adresse). Une meilleure solution est d'accepter le changement d'adresse mais d'expédier la commande à l'adresse habituelle au moment de la commande (tout simplement parce que c'est que le client a demandé) – il faut bien sûr que ceci soit clairement indiqué sur le bon de commande et il est de bon usage de fournir au client un moyen de re-router une livraison.

Il est important de noter que ceci est une conséquence intrinsèque de la mise en place de référentiels transverses ; les applications en mode cloisonné qui gèrent leurs propres copies des informations n'ont pas ce genre de problèmes, mais au prix d'incohérences entre informations et d'un service dégradé pour l'entreprise, ses clients ou ses partenaires. Il ne sert donc à rien d'essayer de combattre ce phénomène, mais il s'agit bien de l'accepter car la solution est simple : il suffit de garder un historique de toutes les modifications des informations des référentiels. Avec cet historique, le traitement de changement d'adresse, par exemple, peut s'exécuter à tout moment car le Pilote en charge de l'expédition de la commande demandera au référentiel non pas la dernière adresse habituelle du client mais bien son adresse au moment de la commande.

Ceci se résume dans la règle suivante :

> ***Règle 5a :*** *Toutes les modifications des informations des référentiels sont historisées..*

Cette historisation des informations impose à tous les Référentiels de partager une base de temps commune, i.e. une horloge commune. D'un point de vue technique les mécanismes correspondants existent et sont simples à mettre en œuvre (utilisation du protocole NTP).

Cette règle s'applique aussi, en conjonction avec la règle de non-destruction des identifiants, à la suppression des données :

> ***Règle 5b :*** *Les informations ne sont pas détruites mais marquées comme invalides à partir d'une certaine date.*

Cette règle soulève la question des purges et des archivages. Les concepts de purge et d'archivages sont liés aux contraintes physiques des supports de stockage de masse : ces supports avaient auparavant des capacités très limitées nécessitant de purger et d'archiver les données les plus anciennes ou peu utilisées. Ceci imposait la mise en place de mécanismes complexes, en particulier pour rapatrier les données archivées en cas de besoin et les re-purger ensuite. Ces mécanismes, déjà compliqués dans le cadre d'une application dans un SI cloisonné, deviennent extrêmement complexes dans le cadre de l'ensemble d'un SI d'entreprise. La très forte augmentation des capacités des systèmes de stockage permet actuellement de se passer de ces mécanismes dans la plupart des cas et donc de simplifier très significativement la conception du SI. D'autre part, la généralisation des obligations d'auditabilité (traçabilité industrielle ou contraintes de type Sarbanes-Oxley) imposent la non-destruction de la plupart des données historiques de l'entreprise. Une attention particulière doit être portée dans les cas où la destruction réelle des informations est une obligation légale forte ; il s'agit alors de vérifier l'impact de cette obligation sur l'ensemble des processus de l'entreprise.

Une autre conséquence de ce principe est la préconisation de permettre l'activation simultanée des traitements conversationnels et non conversationnels (les batchs). En effet, il est habituel de réserver les traitements non conversationnels pour la nuit, en dehors des heures de travail quand les ressources matérielles (les machines) ne sont pas utilisées par les traitements conversationnels. Ceci permet d'optimiser l'utilisation des ressources matérielles. Cette règle de bon usage des ressources s'est souvent transformée au fil du temps en une règle fonctionnelle et les concepteurs ont souvent tiré parti de cette exclusion pour faciliter la rédaction des spécifications fonctionnelles. Ceci est clairement un couplage fort entre traitements, dont nous avons vu les inconvénients.

De plus, dans le cadre de référentiels transverses trans-géographiques, le territoire couvert par un Pilote est dispersé sur plusieurs fuseaux horaires, ce qui impose un large étalement des horaires de disponibilité des traitements conversationnels. Les exclusions fonctionnelles entre traitements conversationnels et non conversationnels imposent donc

- soit de réduire très fortement la fenêtre temporelle disponible pour les traitements non conversationnels,
- soit de demander aux employés de certains pays de travailler hors des horaires habituels dans leur pays.

Ce phénomène est encore amplifié dans le cadre des télé-procédures où l'expérience a montré que les internautes exigent une plage d'ouverture des services très étalée, voir continue (i.e. 24h sur 24h et 7 jours sur 7).

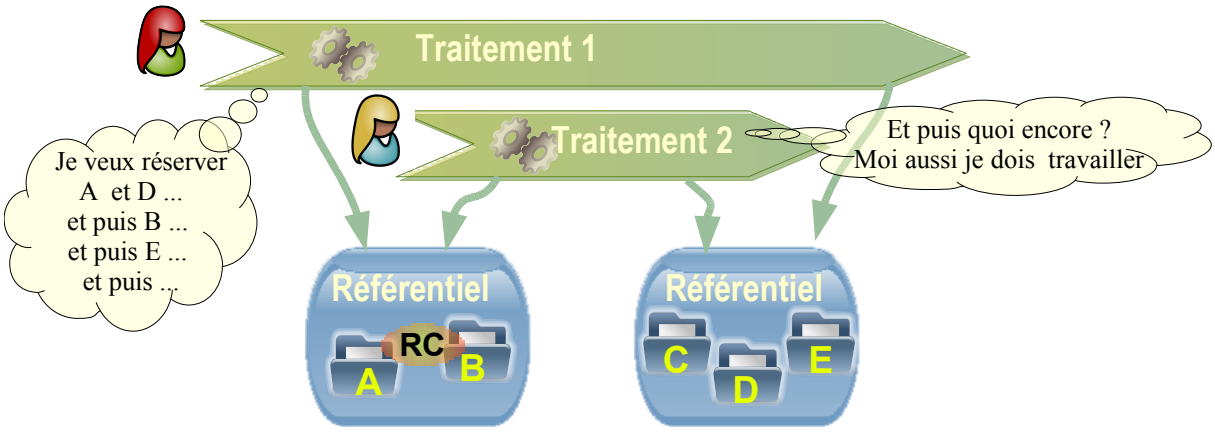
Ceci justifie donc la règle suivante :

> ***Règle 5c :*** *Les activités doivent être modélisées de manière à permettre l'activation simultanée des activités conversationnelles et des activités non conversationnelles.*

Cette règle qui peut paraître contraignante est en fait simple à mettre en œuvre grâce à l'historisation des données (règle 5a) et au concept classique de date de valeur.

Principe 6: Non-exclusivité des données (même de manière temporaire !)

Un Pilote ne peut réserver, même de manière temporaire, un accès exclusif à une donnée d'un Référentiel.



Ce principe est très similaire au principe 5 d'asynchronisme des Pilotes, avec une dimension supplémentaire liée à la complexité des données.

Dans les applications en mode cloisonné, la gestion des potentiels conflits d'accès entre plusieurs utilisateurs sur une même donnée est généralement faite via des mécanismes de réservation : l'utilisateur annonce son intention de travailler sur une donnée et l'application en interdit l'accès (au moins en modification) aux autres utilisateurs. Ce mécanisme de verrouillage des données est généralement mis en œuvre quand l'activité de l'utilisateur est relativement longue : de quelques heures à quelques jours.

Dans le cas d'un système basé sur des référentiels transverses, une telle réservation impliquerait qu'aucun autre traitement de l'ensemble du SI de l'entreprise ne pourrait modifier cette donnée. Cette réservation, qui était voulue et acceptable sur une petite sous-partie du système d'information, prend une ampleur difficilement contrôlable quand elle s'applique sur l'ensemble du SI. En effet la sémantique de la réservation devient vite complexe à définir précisément et entraîne rapidement un couplage fort entre traitement - couplage dont nous avons vu les inconvénients majeurs.

Prenons l'exemple d'un client dont le département financier serait en train de ré-évaluer la solvabilité. Cette modification peut prendre plusieurs minutes voir plusieurs heures, en particulier si l'employé est interrompu (téléphone) ou si l'employé doit chercher des informations supplémentaires. On pourrait donc vouloir 'réserver' les données du client pour éviter que quelqu'un d'autre ne les modifie en même temps ou n'applique des traitements sur ces données en cours de modifications.

Quel serait le périmètre d'application de cette 'réservation' : le client, son adresses, ses moyens de paiement, ses commandes – en lecture ou en écriture ? Devrait-on refuser une commande d'un client quand on est en train de ré-évaluer sa solvabilité (potentiellement à la hausse) ? Devrait-on refuser un paiement ? On voit que la détermination du périmètre d'application devient très complexe dans le cadre d'un SI d'entreprise.

D'autre part, où pourrait être stockée cette 'réservation' ? Si elle était stockée dans le Pilote de l'activité, alors soit il faudrait introduire un couplage fonctionnel fort entre les SF, soit les traitements des autres SF n'appliqueraient pas cette 'réservation' et pourraient donc modifier les informations, ce qui rendrait inutile la réservation. Il faudrait donc stocker cette réservation dans le référentiel propriétaire de l'information. Mais comment faire si les informations à 'réserver' s'étendent sur plusieurs Référentiels, par exemple les référentiels Clients, Commandes et Paiements (changer la solvabilité peut avoir un impact sur les intérêts appliqués en cas de paiement tardif). Nous savons bien que ceci conduit lentement mais surement vers des problèmes d'étreintes fatales (*deadlocks*) et de blocages.

Enfin, se pose la question de la durée de la réservation : une réservation bloquée depuis longtemps est-elle due au fait que le traitement qui a initialisé la réservation est toujours valide (e.g. attente d'information par courrier) ou bien au fait que l'utilisateur a été interrompu et va bientôt continuer ou bien au fait qu'il a complètement oublié l'activité en cours. Pour gérer le dernier cas, il serait tentant de mettre en place un mécanisme de libération de réservation. Le mécanisme simple qui consiste à tout libérer tous les soirs n'est adapté qu'aux cas où l'objectif de la réservation est limité dans le temps, ce qui exclut tous les cas de réservation liée à la cohérence des données. Dans les autres cas, il faudrait mettre en place des mécanismes complexes, par exemple envoyer des messages à l'utilisateur pour lui demander de libérer les données ou annuler toutes les modifications faites par cet utilisateur.

Comme on le voit, cette simple volonté de réserver une donnée engendre une énorme complexité fonctionnelle et/ou des contraintes métier fortes. Il faut donc comparer le gain obtenu par rapport à une situation où la donnée n'est pas réservée. Or les objectifs annoncés de telles réservations sont doubles :

- Garantie d'une cohérence : Il s'agit de garantir une règle de cohérence fonctionnelle considérée comme importante par le traitement qui réserve l'information et dont il ne veut pas qu'elle soit perturbée par un autre traitement. Comme cette réservation est temporaire, rien ne garantit que cette règle sera respectée ensuite par les autres traitements : la réservation n'apporte donc aucune garantie réelle. Comme nous l'avons vu dans le principe de gestion de la cohérence (principe 4), si une règle de cohérence doit être respectée par tous alors elle doit être garantie de manière permanente par un référentiel et la réservation est alors inutile.
- Gestion d'une incohérence temporaire : Il s'agit de permettre de manière temporaire le non respect d'une règle de cohérence garantie par le référentiel. Il est donc nécessaire de bloquer l'accès à l'information pour 'cacher' ce non-respect temporaire et éviter que d'autres traitements s'appliquent sur des informations incohérentes. Dans ce cas, on voit bien que l'on est en train d'essayer de dévoyer le principe 4 ci-dessus : la garantie de la cohérence par les référentiels. Il est alors beaucoup plus simple de garder cet état intermédiaire incohérent dans le Pilote et de mettre à jour les référentiels dès que les données sont

cohérentes du point de vue du référentiel. Il est important ici de noter qu'il ne s'agit pas d'attendre que les données soient cohérentes du point de vue du pilote – nous avons vu au paragraphe précédent que c'était inutile – mais bien du point de vue du référentiel de manière à ne pas re-créer des applications en mode cloisonné et à rendre au plus tôt les données disponibles à tous les traitements.

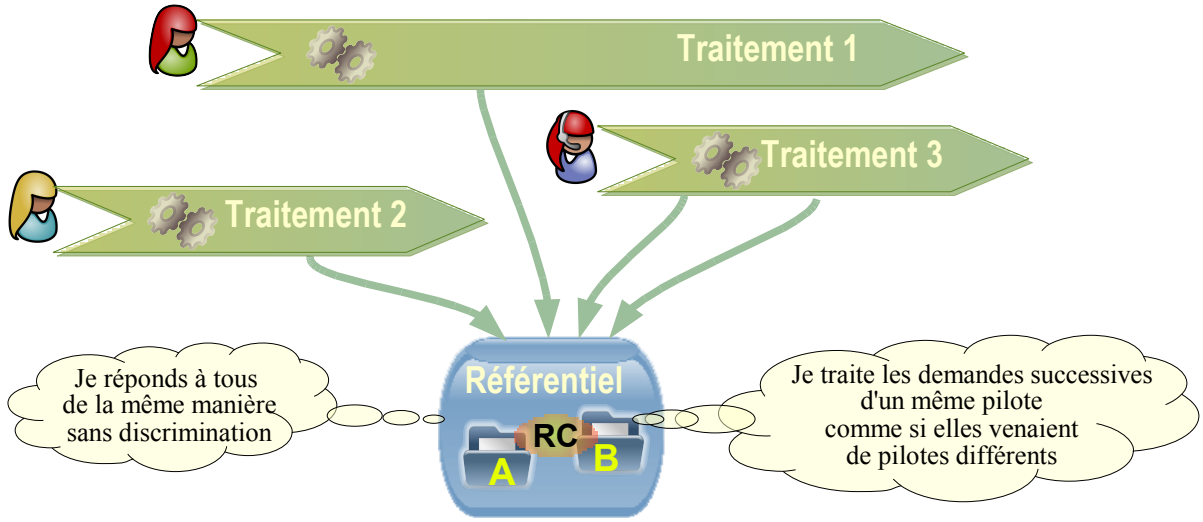
Les objectifs de la réservation de données, qui semblaient importants à première vue et qui pouvaient être utiles dans système d'information en mode cloisonné, ne sont donc plus pertinents dans une approche basée sur des référentiels transverses.

Il est à noter que ce principe entraîne des évolutions dans l'ergonomie de certains traitements : la structuration du système d'information autour de grands référentiels transverses impose une évolution du dialogue homme-machine pour passer d'une logique où l'utilisateur se voit refuser une activité a priori car la donnée est verrouillée à une logique où il se voit demander son avis a posteriori en cas de conflit (concept d'*optimistic locking*).

Enfin, il est important de noter que ce principe s'applique aussi pour le traitement qui a créé la donnée. Ceci peut paraître évident mais n'est pas intuitif pour les concepteurs habitués à des applications en mode cloisonné propriétaires exclusifs de leurs données.

Principe 7: Services sans états

Les Référentiels fournissent à tous des services sans état – laissant toujours le référentiel dans un état cohérent.



Le concept de « service sans état » signifie que chaque activation du service est traitée de manière indépendante sans référence aux précédentes activations des services du SF (autrement bien sûr que via les données stockées dans le référentiel).

L'alternative à ce principe serait d'offrir des services avec sessions, c'est à dire des services qui s'inscrivent dans le cadre d'une session et dont les résultats dépendent des services précédemment invoqués dans la session. Le Référentiel devrait garder et gérer localement un état pour chacun des clients connectés. Un exemple de session est illustré par un site de commerce en ligne où l'utilisateur se connecte et ouvre une session, se 'promène' sur le site et dépose des articles dans son caddie ; le contenu du caddie est maintenu entre les différentes invocations du service de dépose d'un article mais il disparaît à la fin de la session. Dans le cadre des interactions avec un humain, le concept de session est parfois nécessaire pour offrir une ergonomie appropriée aux activités des utilisateurs.

Dans le cadre des services offerts par les Référentiels, il s'agit d'une session purement informatique entre deux éléments logiciels. De telles sessions et les protocoles d'interaction associés sont relativement complexes à spécifier car il faut définir les ressources qui seront visibles dans la session mais pas hors de la session et définir les services liés à la gestion de la session (ouvrir, fermer, annuler, reconnecter, etc) ; il faut aussi gérer le lien entre les ressources et les services de gestion de session, sans oublier les cas inhabituels (par exemple, que faire si une session est ouverte sans activités depuis longtemps : la laisser ouverte indéfiniment au risque de bloquer des ressources inutilement si l'initiateur de la session a oublié de la fermer ou la fermer au risque de perdre les données de la session ?).

Au niveau technique, la gestion des sessions consomme des ressources matérielles. Cette consommation, qui était relativement facile à maîtriser dans un système cloisonné au niveau applicatif et géographique, devient difficilement contrôlable dans un système basé sur des référentiels transverses. En effet, chaque Référentiel devrait alors être capable de gérer potentiellement des sessions pour tous les Pilotes et pour tous les traitements de l'ensemble des utilisateurs. Le nombre de session à gérer simultanément par un Référentiel serait une combinaison du nombre d'utilisateur (plusieurs dizaines à plusieurs centaine de milliers d'utilisateurs pour une grande entreprise) et du nombre de traitements qu'un utilisateur peut dérouler en parallèle (habituellement 2 à 3). On voit que les ressources matérielles à mettre en œuvre seraient très significatives.

Or l'utilisation de sessions correspond aux objectifs suivants :

- permettre une réservation temporaire d'une donnée ou d'un ensemble de données pendant la durée de la session
- permettre de créer des états temporairement incohérents d'une donnée ou d'un ensemble de données pendant la durée de la session

Nous avons vu ci-dessus que ces objectifs ne sont plus pertinents dans un système d'information basé sur des référentiels transverses. Il n'est donc pas utile, et même contre-productif, d'autoriser les Référentiels à fournir des services à état. Ceci justifie la première partie du principe ci-dessus.

La seconde partie du principe est une conséquence simple du rôle de garant de la cohérence fonctionnelle des données par les Référentiels : à tout moment les informations du Référentiel, telles que visibles par les autres SF via les services, doivent respecter les règles de cohérence garanties par le Référentiel.

Ce principe a une conséquence importante sur la structure et la granularité des services exportés par un SF :

> ***Règle 7a :*** *Les services fournis par les Référentiels sont atomiques et de granularité moyenne.*

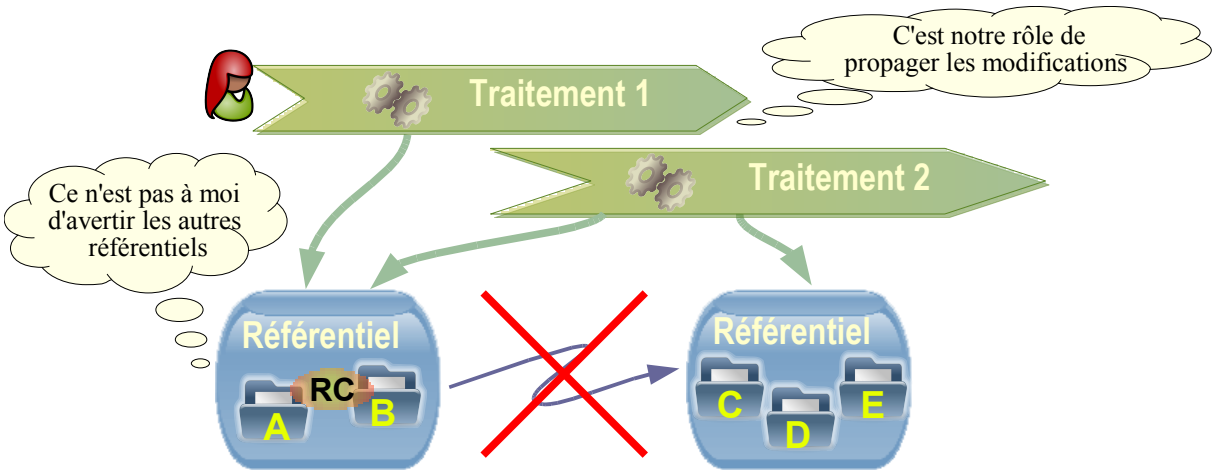
Les services sont atomiques dans le sens où chaque invocation de service est traitée de manière complète et insécable. Si au niveau technique un Référentiel peut traiter en parallèle plusieurs invocations de service, au niveau fonctionnel tout se passe comme si les invocations étaient traitées les unes après le autres - en particulier, si l'exécution d'un service nécessite la mise à jour de plusieurs objets du référentiel, alors le Référentiel garantit qu'aucun service de lecture ne pourra s'insérer au milieu de ces écritures.

Les services sont de granularité moyenne dans le sens où une granularité trop fine ne permettrait pas de garantir que les règles de cohérence fonctionnelle sont toujours garanties. Par exemple, si une règle de cohérence porte sur plusieurs objets, alors un service permettant la mise à jour simultanée de ces objets est nécessaire – car sinon il faudrait plusieurs invocations de services pour retrouver un état cohérent et on créerait un état potentiellement incohérent entre ces différentes invocations.

Il est important de noter, dans cette règle, que atomique ne veut pas dire minuscule mais bien insécable. Il n'y a donc pas de contradiction entre services atomiques et services de granularité moyenne : dans un système d'information centré sur des référentiels transverses les services sont de 'gros atomes'.

Principe 8: Référentiels passifs

Un Référentiel n'a pas vocation à avertir les autres Référentiels des modifications de ses données.



Les raisons de ce principe sont multiples. Tout d'abord nous avons vu que le pilotage des activités et des processus métiers était dévolu aux Pilotes. Avertir pour action un autre SF d'une modification d'une donnée est un morceau de processus qui doit donc être piloté par un Pilote non dans un Référentiel.

D'autre part, la diffusion d'information par ce genre de mécanismes (dit de push) imposerait la mise en place de protocoles complexes pour traiter correctement de manière robuste les impondérables de la vie réelle. En effet, lors de la mise à jour de la donnée, il est possible qu'un des SF à avertir ne soit pas disponible (panne machine ou maintenance programmée – il ne faut pas oublier que la haute disponibilité coûte très cher en développement et en exploitation et qu'il faut la réserver aux composants pour lesquels il existe une réelle justification métier). Il faudrait donc définir un mécanisme permettant de ré-émettre les avis de modification – tout en garantissant que les avis ne peuvent pas être dupliqués (imaginez les conséquences de la duplication d'un ordre de débit d'un compte).

Il faudrait aussi définir les règles de diffusion, car avertir tous les SF de toutes les mises à jour de toutes les données demanderait une bande passante et une capacité machine astronomique – en effet il ne faut pas oublier que dans une approche basée sur des référentiels, le taux de mise à jour par référentiel est bien plus élevé que dans le cas d'un système cloisonné applicativement et géographiquement. Afin de respecter le principe de couplage faible entre processus de SF différents, il serait nécessairement à la charge des différents SF d'indiquer au référentiel les données dont les modifications les intéressent ; ce mécanisme d'abonnement rajouterait encore à la complexité des protocoles à définir.

Il est donc beaucoup plus facile de mettre en place, dans les référentiels, des services qui permettent aux autres Pilotes de connaître les données qui ont été modifiées sur une période de temps donnée – charge aux Pilotes d'utiliser ce service et d'accéder aux historiques des modifications (cf. règle 5a).

La fréquence d'appel à ces services dépend des besoins métier – dans le cas d'un système d'information de gestion, une fréquence horaire est suffisante ; un besoin de fréquence plus élevée est souvent symptomatique d'un couplage trop fort entre deux SF et donc certainement d'une erreur d'urbanisation.

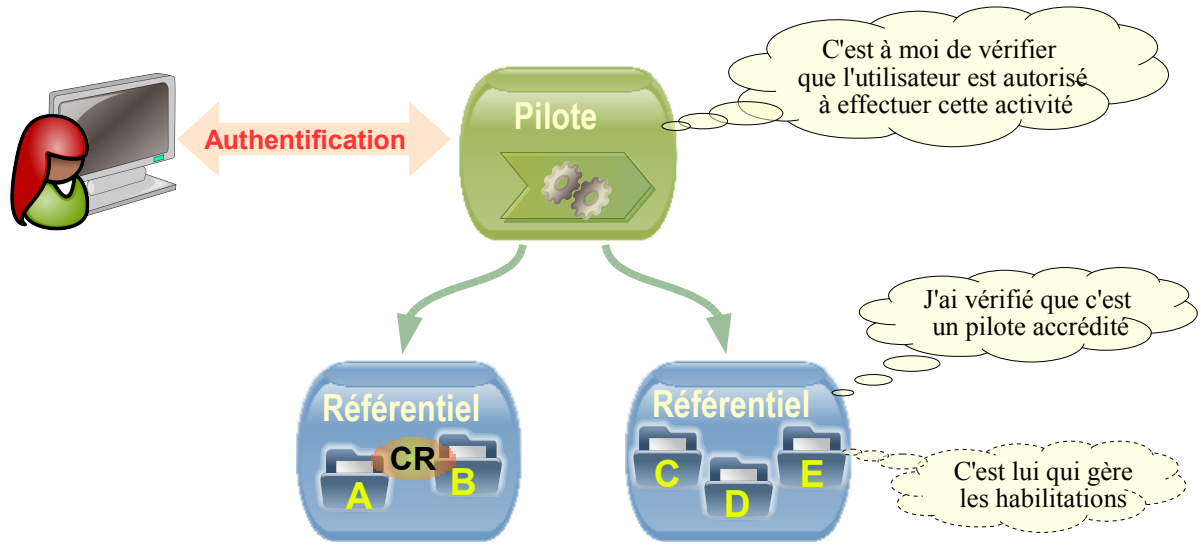
Ceci se résume dans la règle suivante :

> ***Règle 8a :*** *Les Référentiels doivent fournir des services permettant aux Pilotes de connaître les données qui ont été modifiées entre deux dates données..*

Il est à noter que ces services sont très simples à définir et à implémenter dès lors que les informations sont historisées (règle 5a).

Principe 9: Garantie de la sécurité par les Pilotes

La sécurité du SI est basée sur une infrastructure de confiance : les habilitations des utilisateurs sont vérifiées par les pilotes et les Référentiels se contentent de vérifier l'accréditation des Pilotes.



La principale raison de ce principe est que le contrôle d'accès aux informations doit être lié à l'activité de l'utilisateur et pas à l'information elle-même. En effet, le Pilote en charge de l'information peut avoir besoin

pour piloter une activité, de consulter certaines informations, que l'utilisateur n'a pas à connaître. Par exemple, un agent du fisc qui vérifie le dossier de M. Martin peut avoir besoin de savoir si la pension alimentaire déclarée a bien été versée, par exemple en vérifiant que cette pension a été déclarée par l'ex Mme Martin – par contre cet agent n'a pas à consulter le dossier fiscal de l'ex Mme Martin.

Ceci nécessite une infrastructure de confiance permettant une authentification centralisée de l'utilisateur et une gestion homogène des habilitations par les Pilotes. Les mécanismes techniques pour cela sont bien connus (e.g. LDAP, SSO, ..).

Conformément au principe 7, les Référentiels répondent aux requêtes de tous les Pilotes sans discrimination, il convient donc pour garantir la sécurité du SI de vérifier que personne n'y a introduit des Pilotes frauduleux. Pour cela, les Référentiels vérifient que les Pilotes sont bien accrédités. Différents moyens techniques bien connus existent pour cela, qui vont des certificats serveurs à la localisation des Pilotes dans des zones contrôlées par des pare-feux.

Remerciements : Tous mes remerciements à mes collègues et à mes clients pour les discussions qui ont conduits aux neufs principes, en particulier Jean-Marie Lapeyre et Denis Oddoux du Ministère des Finances, Jean-Paul Figer et Stéphane Girard de Capgemini.