

Processus de reengineering et approche SOA

Philippe Dugerdil

Septembre 2005

Contribution au projet ISNet 82

Haute école de gestion
(Univ. of Applied Sciences)
Informatique de gestion
7, rte de Drize
CH-1227 Geneva
Switzerland
+41 22 388 17 00

www.hesge.ch/heg

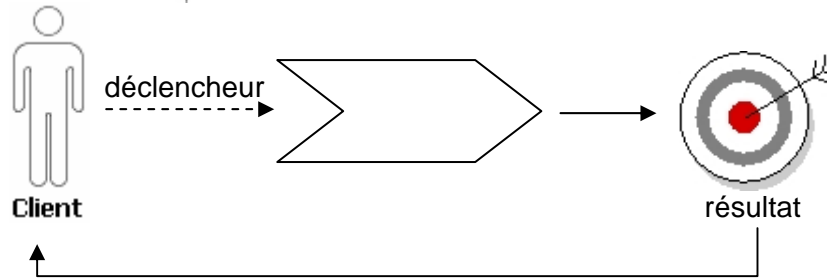
1 Introduction

Dans le cadre du projet ISNet 62, nous avons élaboré une méthode de reverse-engineering pour grands systèmes d'information. Cette technique est basée sur l'analyse des processus métier dont le système d'information fait partie. En effet, nous avons montré que la retro-architecture (reverse-architecting), est indissociable de la compréhension de la structure du programme. Or, selon notre modèle, cette compréhension consiste à donner un sens « métier » aux éléments logiciels identifiés. En d'autres termes, la compréhension de programme et le reverse-engineering sont les deux faces d'une même pièce : l'un ne va pas sans l'autre. Toutefois, quand il s'agit d'analyser un système mal documenté, nous tombons rapidement dans un cercle vicieux. Pour identifier les composants logiciels il faut connaître les concepts métier correspondants et pour sélectionner les concepts métier pertinents il faut connaître les composants à documenter. Pour sortir de cette situation paradoxale, nous avons proposé de nous appuyer sur l'analyse des processus métier que ces systèmes automatisent. En effet, la description de ces processus représente une information fiable puisque ce sont les utilisateurs réguliers du système qui peuvent nous la fournir (par opposition aux développeurs du système qui, lorsqu'il s'agit de vieux systèmes, peuvent très bien ne plus être disponibles). Cependant, la profondeur technique de cette description est forcément limitée. Elle doit donc être accompagnée de techniques d'analyse de code plus traditionnelles.

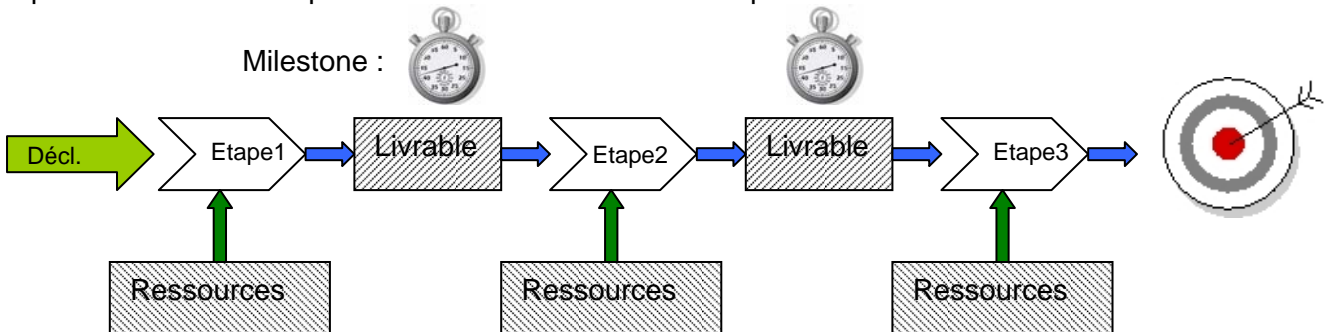
D'un autre côté, l'approche SOA représente une nouvelle approche de l'architecture des systèmes d'information informatisés d'entreprise. En effet, le maître mot aujourd'hui semble être la flexibilité et l'adaptabilité au niveau de l'entreprise et de ses services [17][8] et cela demande naturellement une adaptation des systèmes informatiques sous-jacents. Si on en croit la littérature spécialisée, la solution pourrait venir du paradigme SOA. Il s'agit en effet plus d'un paradigme que d'une technologie car, au-delà du simple schéma « Publish-Find-Bind » chère aux Web services, c'est la conception même des applications qui s'en trouve modifiée [7][11]. Toutefois, il s'agit de ne pas s'engouffrer tête baissée dans cette nouvelle direction et de déclarer tout de go que le paradigme SOA est « la » solution aux maux dont on veut aujourd'hui affubler l'informatique. En particulier, la qualité globale des systèmes risque de s'en trouver modifiée. C'est à ce niveau que se situe la réflexion consignée dans ce rapport.

2 Processus métier

Avant d'approfondir l'usage de SOA dans le cadre du reengineering, il est utile de fixer le vocabulaire en posant quelques définitions concernant la modélisation métier. Un processus métier est « une collection de tâches inter-reliées, dont l'exécution est lancée en réponse à un événement, qui conduit à un résultat spécifique pour le client de ce processus » [15]. Il y a deux points essentiels dans cette définition : « résultat spécifique » et « client ». Le premier indique qu'un processus doit être focalisé sur l'atteinte d'un résultat qui lui donne son sens. Ce résultat peut être un produit ou un service et doit être clairement identifiable et énumérable (« comptable »). Ce dernier point évite que le résultat ne soit exprimé en termes conceptuels ou trop vagues. Les activités du processus doivent toutes concourir à l'atteinte du résultat. Ensuite un processus doit avoir un client. Si personne n'est intéressé par un résultat, le processus n'a pas d'intérêt. Le client n'est pas forcément un individu, cela peut être une organisation, un département. Cependant le client doit posséder la capacité de juger si le résultat est satisfaisant. Enfin, la notion d'événement déclencheur permet de situer le début d'un processus, ce qui n'est pas toujours évident. C'est souvent le client qui déclenche un processus.



Un processus est constitué de tâches ou activités. Dans le cas le plus simple, la fin de l'une d'elles provoque l'initialisation de la suivante. Dans les cas plus complexes, on peut avoir des tâches parallèles qui se rejoignent pour lancer une tâche suivante, le tout accompagné d'événements externes et de conditions d'exécution. Une tâche est une activité exécutée par un acteur (dans le sens UML : un rôle joué par une ou plusieurs personnes). Cet acteur peut parfois être informatique. Une tâche a une étendue temporelle.



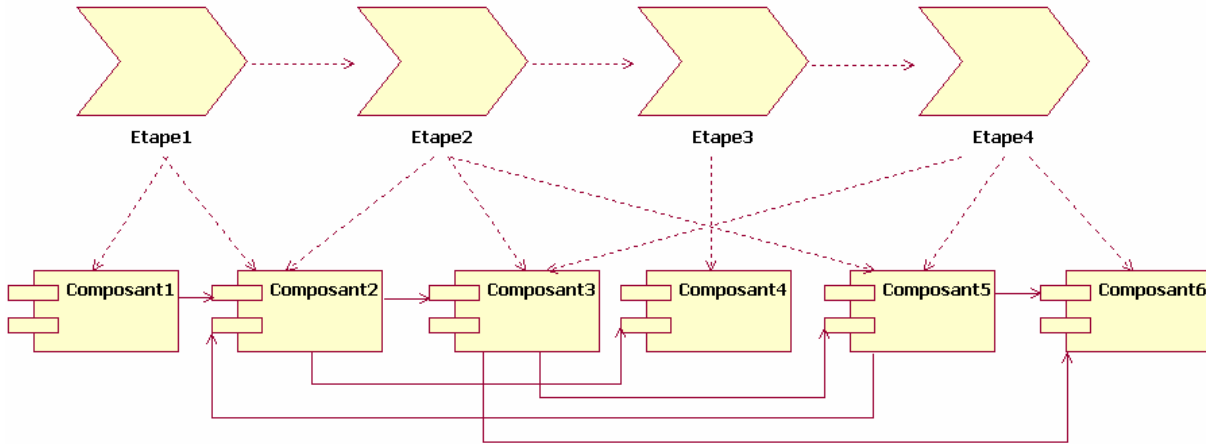
Une activité produit un résultat intermédiaire ou livrable qui est éventuellement exploité par les tâches suivantes. Une tâche, pour son exécution, nécessite des ressources humaines, matérielles et immatérielles (équipements, documents, procédures, sources d'information). Certains livrables peuvent correspondre à un milestone dans le processus.

3 Perspective SOA

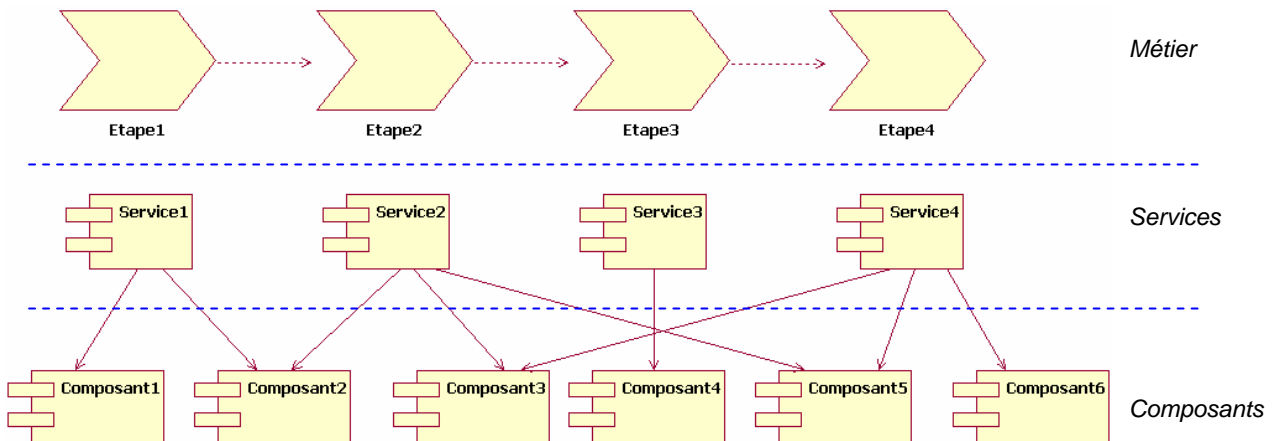
Le mouvement autour de l'architecture orientée services va au-delà de perspectives purement technologiques. En effet, la pression pour la réduction des coûts et l'augmentation de l'efficacité des services informatiques dans les entreprises est forte. Aujourd'hui, on sait que l'une des clés de cette efficacité est d'améliorer l'alignement de l'informatique avec la stratégie métier de l'entreprise (« IT-Business Alignment » [3]). Mais il est connu que l'informatique peut être un frein la mise en place de nouvelles organisations d'entreprise par la rigidité de l'infrastructure en place [7]. En conséquence, il est nécessaire d'adopter une architecture logicielle qui évite cette rigidité. La réponse, tout au moins du point de vue conceptuel, semble venir de SOA [6][8]. Il est important d'insister sur le fait que SOA est avant tout une approche organisationnelle (architecturale) des applications informatiques. Par ailleurs, la technologie des « web services » est l'une des façons de réaliser une approche SOA mais elle n'est de loin pas suffisante¹. En particulier, il est intéressant de noter que quelques entreprises ont mis en place de telles architectures SOA avant même l'avènement des web services [7]. Le point important de cette approche est de concevoir des services métier sous forme de fonctionnalités informatiques réalisées et invoquées de manière indépendante. En conséquence, l'approche SOA est indissociable de la notion de processus métier. C'est en effet dans ce contexte qu'un service SOA prend son sens et qu'il

¹ Cette technologie, bien qu'étant la plus prometteuse, n'est pas la seule. Historiquement, des approches SOA ont été tentées via le recours à un bus asynchrone (IBM MQ series par exemple) ou aux protocoles IIOP, Java RMI ou encore .Net.

montre un potentiel de réutilisabilité. Comme nous l'avons vu précédemment, l'objectif est de faciliter la mise en place de nouvelles organisation et processus métier. Le niveau de flexibilité demandé est donc fondamentalement au niveau métier. Ce n'est que par l'analyse de ces processus et l'identification de fonctions (services) réutilisables au niveau métier que la granularité de ces services pourra être déterminée et les interfaces spécifiés. Toute autre approche risquerait de produire des composants ne permettant pas ce niveau de flexibilité.



Dans le schéma simplifié ci-dessus, il est facile de comprendre qu'une réorganisation des étapes du processus ne serait pas chose facile du point de vue informatique : les composants mis à contribution par les différentes étapes ne leur son pas dédiés. Par exemple, en admettant que l'étape 2 soit réorganisée, il est difficile d'en déterminer à priori l'impact informatique. La transformation vers une approche SOA consisterait à ajouter une couche supplémentaire qui rende visible les frontières d'étapes de processus. Dans ce cas, chaque service est indépendant et correspond à une seule étape du processus. C'est à lui d'accéder convenablement aux composants de bas niveau :

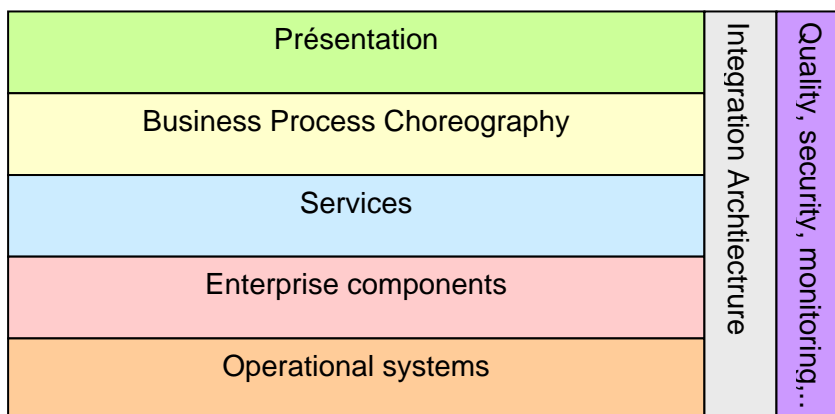


Il est intéressant de noter qu'une telle approche s'inspire indirectement des notions, classiques en génie logiciel, de haute cohésion et de faible couplage. Seulement, elles doivent se comprendre ici par rapport aux étapes du processus métier. Tout d'abord, les composants informatiques impactés par une modification d'une étape de processus seront regroupés au sein d'un même service. Ensuite, le couplage doit être faible entre les éléments impliqués dans deux étapes différentes. Ainsi, les services pourront être invoqués de différentes manières par différentes application en fonction de l'organisation métier à

mettre en place. En quelque sorte l'application finale² est une couche d'invocation de services selon la séquence déterminée par le processus métier que l'on veut mettre en place. C'est donc elle qui fait le lien entre des services a priori indépendants. On parle parfois de « chorégraphie » des services pour illustrer le fait que la responsabilité principale de cette couche est d'orchestrer l'invocation de services indépendants les uns des autres. En principe elle n'implante pas elle-même de règles métier.

Un service peut être invoqué par un grand nombre d'applications différentes, via un interface parfaitement défini aussi bien du point de vue syntaxe que sémantique. Son fonctionnement doit donc être indépendant de l'application cliente et ne doit pas faire d'hypothèse quand aux applications susceptibles de l'invoquer. Cela a deux conséquences pratiques importantes. Tout d'abord, un service ne peut pas « faire confiance » à l'application cliente. En particulier, avant tout travail, il devra s'assurer de la validité des données transmises et détecter toute anomalie susceptible de perturber son fonctionnement. Cependant, dans le cas d'une application interactive, on a coutume pour augmenter la rapidité de réaction de l'interface utilisateur d'effectuer certaines validations syntaxiques au niveau de la couche de présentation. Ainsi, il est probable que certaines validations seront réalisées deux ou plusieurs fois à différents niveaux de l'architecture, avec tous les problèmes de cohérence associés. Par ailleurs, il faut relever que les appels aux services sont de granularité beaucoup plus large que des appels de composants traditionnels: un appel de service correspond à une partie substantielle d'un processus métier. De plus, les appels sont en principe indépendants les uns des autres, quelque soit l'application cliente. La conséquence pratique est qu'en principe le serveur de services ne maintient pas de contexte de session entre deux appels. Ainsi, on ne peut pas implanter de transactions entre les services. Toutefois, l'objectif d'un service étant de représenter une étape complète dans un travail avec un livrable intermédiaire. Il est donc raisonnable qu'il constitue une unité du point de vue transactionnel.

On le voit, l'approche SOA ouvre des perspectives intéressantes du point de vue de la construction des systèmes d'information mais amène également de nouveaux défis en termes de conception. En particulier, cette approche demande de développer une vision holistique de l'entreprise. Ainsi, des disciplines naguère quelque peu exotiques telles de l'Entreprise Architecture chère à Zachman [21] deviennent fondamentales. En effet, la spécification et l'implantation de services doit non seulement cibler les processus métier actuels, mais s'avérer flexibles vis-à-vis des demandes potentielles futures. En conséquence, il s'agit désormais de s'intéresser aussi bien aux processus métier qu'au contexte de l'entreprise et à sa stratégie. Afin d'atteindre une telle flexibilité, IBM propose une architecture en 5 couches [1][22] :



² Ce terme est à prendre avec précaution car il a perdu de son sens traditionnel.

Dans la couche la plus basse se trouvent les systèmes traditionnels, les applications spécifiques, les ERP et autres progiciels, les applications de business intelligence et les applications « legacy ». Dans la couche du dessus, on trouve les composants métiers et les composants techniques transversaux à toute l'entreprise. Ils sont standardisés et doivent tous répondre aux mêmes standards de qualité. Dans la couche service, ces composants sont agrégés pour former des services de haut niveau suivant les processus métier de l'entreprise. La couche de « chorégraphie » est celle qui contrôle et réalise un processus métier via l'appel des services sous forme de workflow applicatif. C'est en particulier dans cette couche que les modifications seront effectuées si un changement de processus métier conduit à une modification de workflow. Finalement, on trouve la couche présentation qui ne fait pas strictement partie d'une architecture SOA. Elle est indiquée par souci de complétude.

Les deux couches verticales sont nécessaires au fonctionnement du tout. La couche d'intégration, qu'on appelle parfois ESB (Enterprise Service Bus) contient la technologie permettant d'identifier et de retrouver les services et composants, de les invoquer et d'effectuer les nécessaires conversions de protocole. Finalement, la couche qualité assure le monitoring de la qualité de service, de la sécurité et tout autre aspect transversal aux applications et nécessaire à la qualité de service.

Pour l'anecdote, il est intéressant de relever qu'une approche de conception conjointe des systèmes d'information et des organisations a été proposée par David Taylor en 1995, approche qu'il a baptisée « convergent engineering » [18]. Cependant elle était basée sur la technologie objet et souffrait donc de la différence de granularité entre les structures organisationnelles et les objets techniques. Plus tard, Hubert a exploité ces idées dans le cadre d'une méthode de conception des architectures logicielles basées sur les composants java distribués (EJB) qu'il a tout naturellement baptisée « convergent architecture » [10]. Mais la notion de processus métier n'était pas explicite. Aujourd'hui, avec la technologie des services, les idées de David Taylor prennent une nouvelle dimension.

4 Reengineering orienté services

4.1 Modélisation du processus métier pour la rétro-architecture

La méthode de rétro-analyse architecturale que nous avons élaborée (ISNet62) passe par l'identification de tâches associées à un processus métier. En effet, dans le cas de la modernisation d'un système d'information « legacy³ » c'est à dire d'un système utile, mais mal documenté, difficilement maintenable et implanté dans une technologie obsolète, il s'agit tout d'abord de savoir ce que le système fait exactement et comment il le fait. La première étape de notre méthode consiste donc à identifier et décrire globalement chaque processus implanté par le système d'information. Il s'agit en particulier d'identifier [15] :

1. Le nom du processus
2. L'évènement déclencheur;
3. Le résultat attendu ;
4. Le client du processus ;
5. Les autres intervenants : personnes intéressées mais pas clientes (stakeholders) ;
6. Les activités majeures (4-7 activités « macroscopiques ») ;
7. Les acteurs des activités avec leurs rôles ;
8. Les ressources des activités ;

³ «A legacy Information System is any information system that significantly resists modification and evolution to meet new and constantly changing business requirements...To complicate matters, these IS are mission-critical- that is essential to the organization's business - and must be operational at all times" [5].

9. Les durées et autres éléments temporels ;
10. L'environnement du processus : les autres processus éventuellement associés.

L'identification des activités majeures peut se baser sur les livrables intermédiaires et/ou milestones importants dans le processus. Toutes ces informations sont obtenues auprès des personnes concernées de près ou de loin par l'utilisation du système. Il s'agit donc de l'ensemble des acteurs et autres intervenants (administrateurs, responsables, autorités).

En parallèle à cette modélisation, il s'agit d'identifier les tables de bases de données avec leurs attributs, ainsi que les fichiers avec leurs champs et enregistrements. Il faut alors créer un glossaire de l'ensemble de ces informations (entités avec attributs et type)

4.2 Analyse des étapes du processus basé sur les cas d'utilisation.

L'analyse des étapes du processus métier est conduite via l'analyse des cas d'utilisation du système d'information. Il est souvent plus facile de procéder en remontant la chaîne à partir du résultat global qu'en descendant à partir de l'évènement déclencheur. En conséquence, le travail démarre par l'identification du résultat global du processus, de son client et des conditions de déclenchement. Ensuite, en partant du résultat global :

1. Identifier l'étape permettant de produire le résultat et décrire la tâche effectuée en termes métier.
 - a. Identifier le ou les acteurs métier impliqués (business workers).
 - b. Répertorier les ressources consommées / consultées nécessaires à l'accomplissement de l'étape.
 - c. Répertorier les éventuelles ressources supplémentaires produites par l'étape.
 - d. Établir les liens entre les ressources produites par l'étape courante et les ressources consommées / consultées par l'étape suivante (en aval dans la chaîne d'activités).
2. Identifier les dépendances en termes métier
 - a. Identifier, du point de vue métier, comment le résultat est construit ainsi que son contexte.
 - b. Quels sont les composantes de ce résultat (type d'entité, attribut).
 - c. A partir de quelle information (type d'entité, attribut) peut-on le calculer, quels sont les résultats intermédiaires requis et quelles sont les règles de calcul⁴.
 - d. Quels sont, éventuellement, les milestones / livrables intermédiaires importants conduisant à la production du résultat.
 - e. Identifier, dans le glossaire issu de l'analyse des données, les entités pouvant potentiellement correspondre aux informations consommées, consultées, transformées et produites par l'étape.
 - f. Établir les liens entre les entités produites / transformées par l'étape courante et les entités consommées /consultées par l'étape suivante (en aval dans la chaîne d'activités).
3. Décrire le (ou les) cas d'utilisation associé à l'étape
 - a. Nom, acteur principal et acteurs secondaires
 - b. Résultat à valeur ajoutée pour l'acteur principal
 - c. Déclencheur
 - d. Autres intervenants
 - e. Flot de base et d'extension

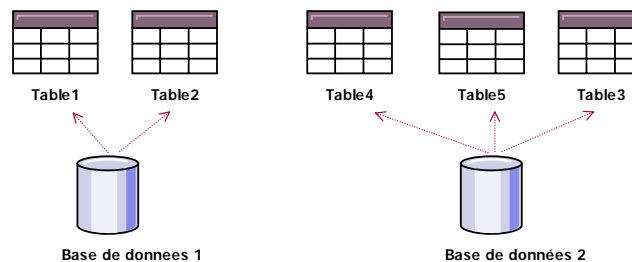
⁴ Fréquemment, le problème dans un système legacy est que les règles métiers sont codées dans le logiciel et que personne ne sait plus avec exactitude comment le résultat est calculé. Toutefois, une connaissance conceptuelle globale est souvent encore disponible : on connaît les étapes essentielles du calcul. Par exemple, dans le cas d'une ouverture de compte bancaire, un spécialiste pourra dire qu'il existe un montant minimum en compte et des limites de retrait mensuelles, sans pouvoir exprimer exactement ces montants.

4. Construire le MOA du use-case
 - a. Les entités sont-elles semblables à celles identifiées au point 2 ?
 - i. Si ce n'est pas le cas, identifier les entités dans le glossaire.
 - b. Créer un boundary par écran
 - c. Identifier les entités manipulées dans le glossaire.
 - d. Pour les données absentes du glossaire (information calculées) créer une entité par use-case pour les contenir et enregistrer cette entité dans le glossaire.
5. Pour chaque résultat intermédiaire utilisé par l'étape (ressource consommée / consultée), répéter les étapes 1-4.
6. Synthétiser et ordonner les activités du processus global à partir de l'analyse des cas d'utilisation.
 - a. Créer un cas d'utilisation de sous-fonction pour chaque fonction se retrouvant dans plusieurs use-cases (écran d'interrogation utilisé de manière récurrente).
 - b. Valider que l'ensemble des résultats intermédiaires et des milestones du processus métier est produit par l'un ou l'autre des cas d'utilisation identifiés.
 - c. Vérifier la séquence du processus en analysant la séquence de production des résultats intermédiaires.
 - d. Regrouper / ordonner les étapes en fonction des cas d'utilisation.

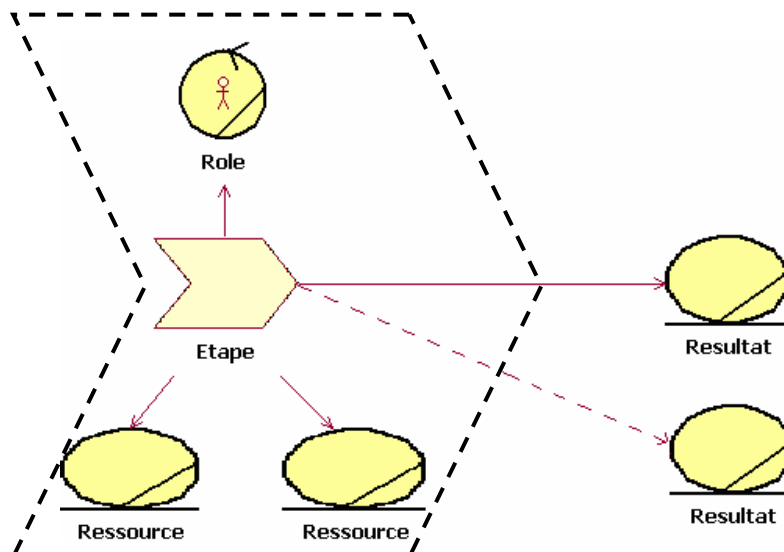
4.3 Synthèse: synergie entre analyse des processus et des cas d'utilisation

Dans les schémas qui suivent, nous utilisons principalement la symbolique UML et le profil de modélisation métier.

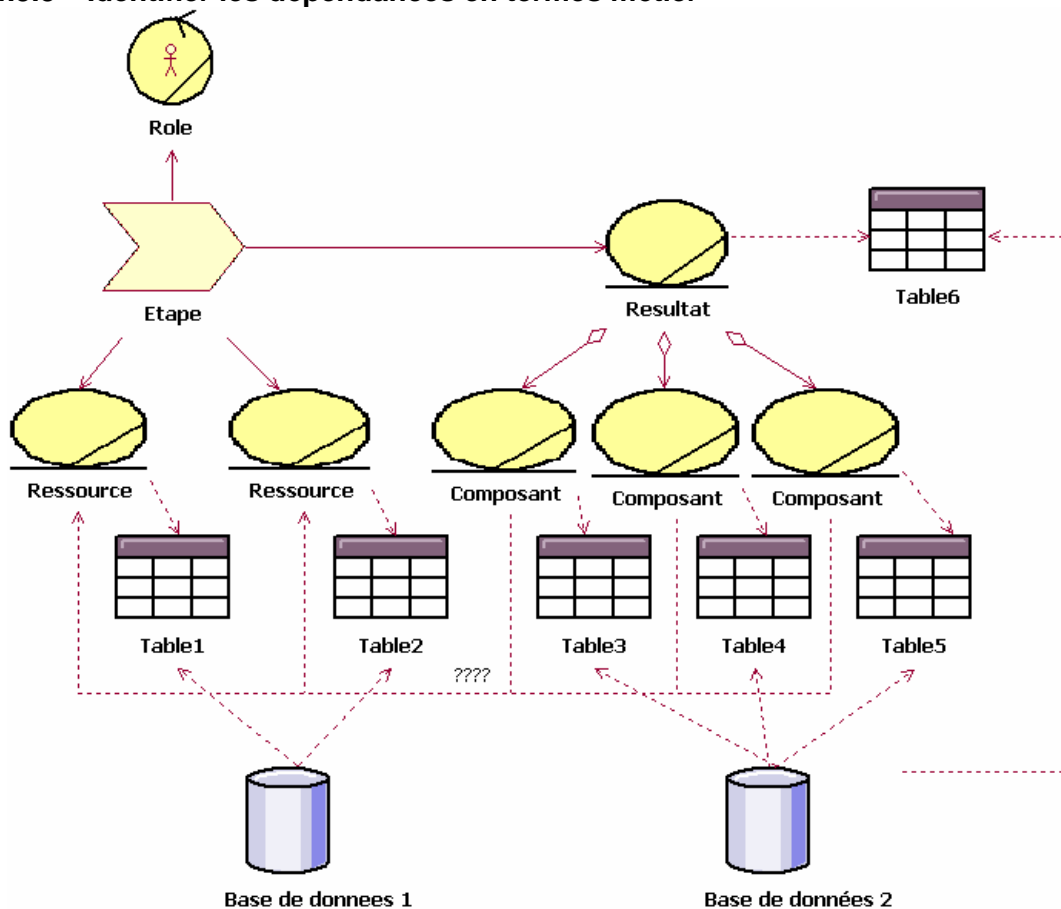
4.3.1 Etape préliminaire : identifier les tables de bases de données et les fichiers avec leurs enregistrements



4.3.2 Identifier l'étape permettant de produire le résultat

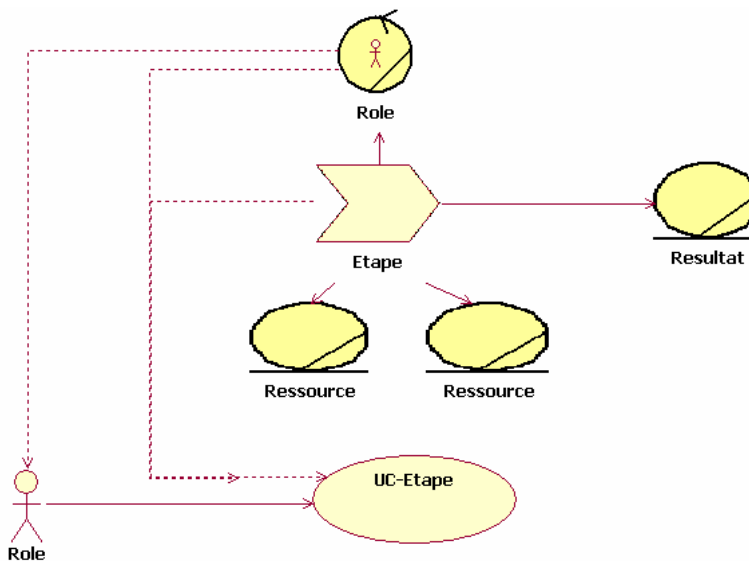


4.3.3 Identifier les dépendances en termes métier



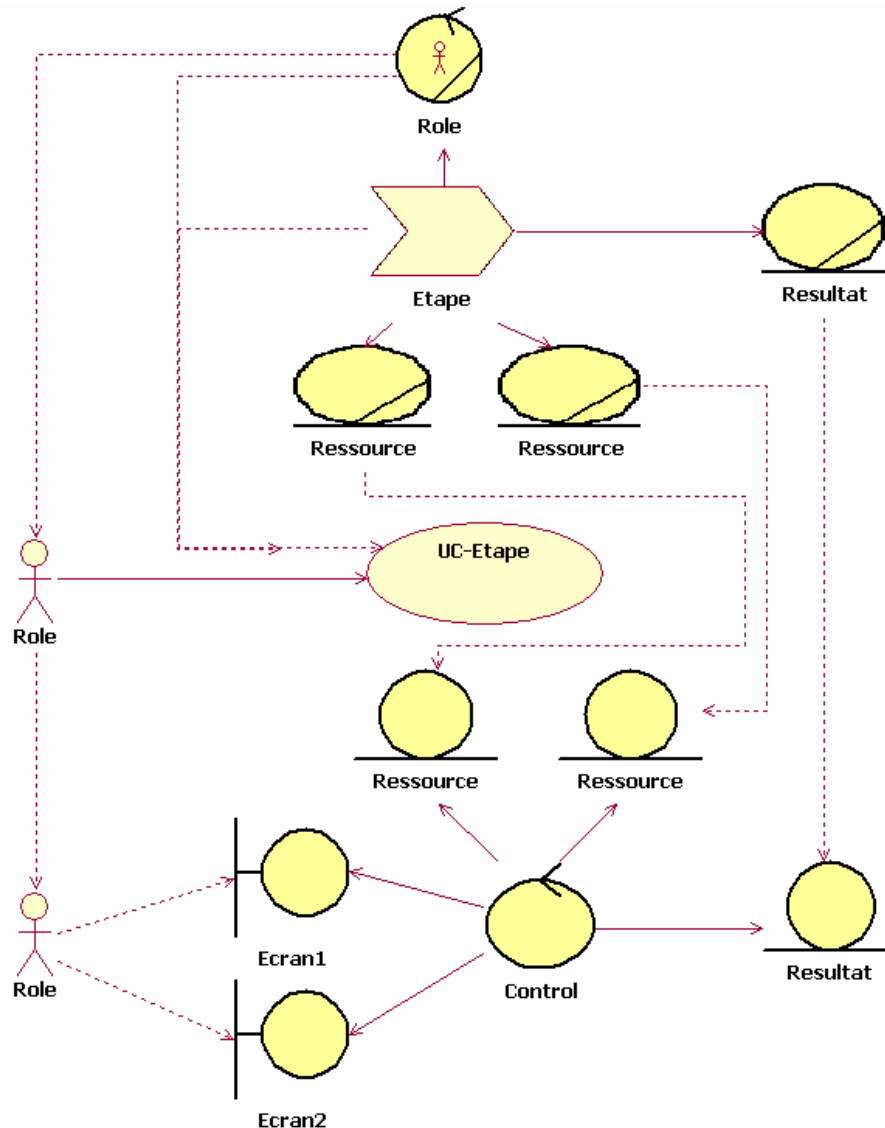
Il s'agit ici en particulier de retrouver le lien entre les entités constituant les ressources d'une étape du processus et les composants du résultat de l'étape. Ce lien peut être direct ou indirect, dans le cas où les attributs du résultat sont calculés. Il s'agit également de faire le lien entre les informations impliquées et les tables de bases de données.

4.3.4 Décrire le (ou les) cas d'utilisation associé à l'étape



Il s'agit ici de la technique classique de mapping des cas d'utilisation métier aux cas d'utilisation système documentée dans le Rational Unified Process (RUP) : les business workers deviennent des acteurs système et leur implication dans le processus métier les conduit à utiliser le système d'information au travers d'un cas d'utilisation système.

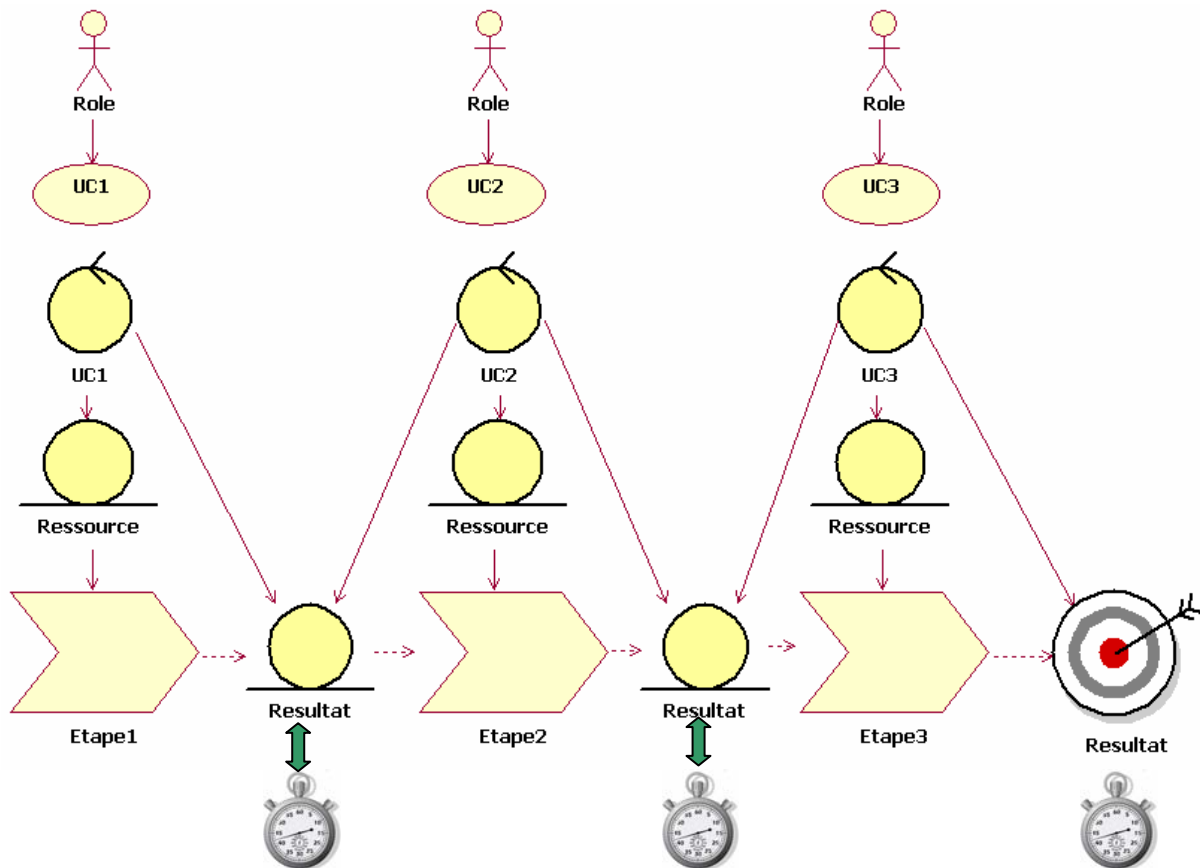
4.3.5 Construire le MOA du use-case



Dans cette étape, on utilise à nouveau les techniques classiques de RUP et en particulier les heuristiques de passage des cas d'utilisation système aux objets d'analyse. Les entités métier deviennent des entités système le use-case est coordonné par un objet contrôleur et les écrans sont représentés par des objets de type « boundary ».

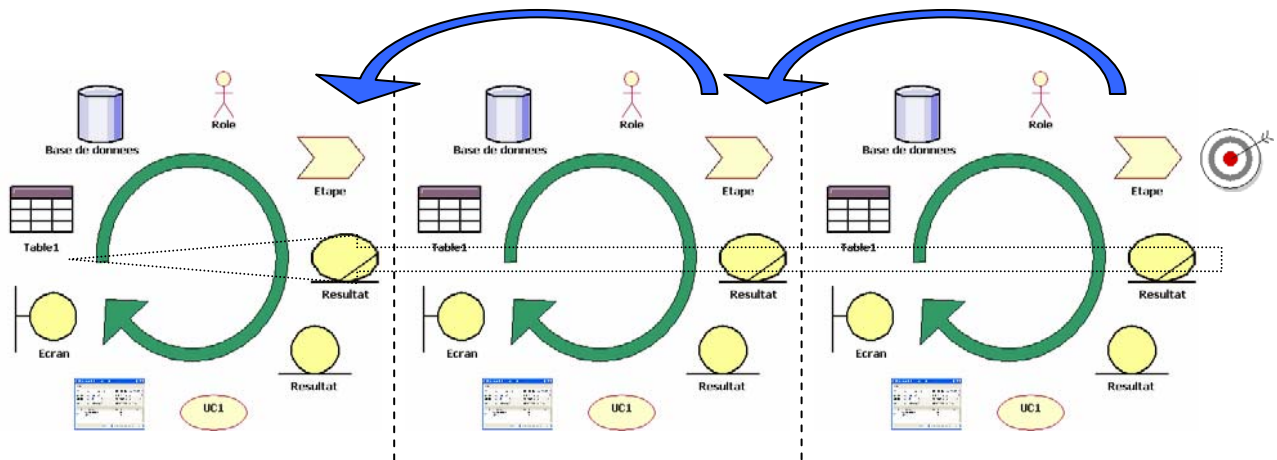
4.3.6 Synthétiser et ordonnancer les activités du processus global

C'est la dernière étape de cette analyse conduite par les processus métier. Les use-cases construits de manière incrémentales sont ordonnés et leurs résultats mis en correspondance et validés. On obtient finalement une représentation de la séquence d'étapes du processus métier avec les use-cases système et les objets d'analyse correspondants.



Il est important de noter que notre approche est fondamentalement itérative et incrémentale : l'identification d'un livrable permet l'identification d'une tâche qui, par ailleurs, utilise d'autres entités pour produire un résultat. Ces dernières sont produites par d'autres tâches et ainsi de suite. Lors de l'analyse de chaque tâche, on identifie les cas d'utilisation du système. Sur les écrans associés apparaissent des attributs d'entités qui devraient correspondre à celles qui ont été identifiées dans l'analyse de la tâche. Si ce n'est pas le cas, il s'agira de retrouver les tâches intermédiaires qui permettent de faire le lien. Il est à noter que dans certains cas il peut exister des tâches intermédiaires complètement automatisées (tâche batch) qui produisent des résultats utilisés par les use-cases. Le contrôle de cohérence sur les entités est fondamental. Il permet de s'assurer que l'ensemble des résultats intermédiaires et du résultat final peut être produit par la suite logique des étapes du processus. Par ailleurs, la nature des entités et l'appartenance des attributs aux entités peut être difficile à déterminer si on ne connaît pas la structure de la base de données. En conséquence, une analyse des données préalable accélérera l'identification de ces entités.

Schématiquement, le processus se déroule incrémentalement en partant du résultat principal du processus :



4.3.7 Identification des composants systèmes pour description architecturale

La description complète de cette étape sort du cadre de la présente étude et a fait l'objet du projet de recherche ISNet62. Nous nous contenterons d'en tracer ici les grandes lignes. Il faut relever que nous nous plaçons délibérément dans le contexte d'un système peu ou pas documenté mais dont le code source est disponible.

Le problème est donc de faire le lien entre une description de haut niveau et les composants du code source qui permettent d'implanter ces concepts. Ce problème est très complexe et a fait l'objet de multiples publications. En principe, il s'agit de mettre en œuvre un algorithme permettant de regrouper les déclarations et éléments du code source selon différents critères. Parmi ces techniques les plus fréquemment citées on trouve :

- Le slicing : recherche des déclarations qui influencent le contenu d'une variable ou qui conduisent au calcul d'une valeur d'attribut [20],
- Le clustering : regroupement basé sur l'analyse des dépendances entre les éléments de programme, ainsi que l'exploitation de principes comme la forte cohésion, le faible couplage) [14].
- L'analyse des concepts formels (formal concept analysis) : technique d'analyse de données mathématiquement fondée qui permet en particulier de retrouver les ensemble d'éléments (les objets) partageant des caractéristiques communes (attributs) [13][16].

Toutes ces démarches visent à partitionner l'ensemble des déclarations du code source en sous-ensemble ayant un sens du point de vue de l'architecture à reconstruire. Le problème est bien entendu le choix du ou des critères discriminants permettant de trouver les frontières « naturelles » entre les composants. Notre hypothèse est que cette frontière peut être trouvée en suivant les « frontières » entre les tâches élémentaires du processus métier. Les arguments en faveur de cette hypothèse sont que, quelque soit la méthode de développement utilisée, il est très probable que le développement ait suivi les étapes naturelles du travail des utilisateurs ou des étapes « naturelles » de production des livrables intermédiaires. Même si un grand nombre de maintenances sont venues perturber l'architecture originale, ces maintenances s'organisent autour de fonctionnalités, lesquelles sont identifiées au sein d'activités métier.

En résumé et en première approximation, nous proposons que le programme soit découpé selon les étapes du processus en s'appuyant sur les entités manipulées par les cas d'utilisation de chacune de ces étapes. Il s'agit cependant de se concentrer sur les entités ou attributs d'entités spécifiquement associées aux cas d'utilisation de l'étape. Les entités partagées ne permettent justement pas de faire la distinction entre deux composants de haut niveau.

La démarche que nous avons retenue est itérative et incrémentale : à partir des éléments de haut niveau que sont les attributs des entités de chaque étape, nous proposons de mettre en œuvre la technique d'analyse de concept formels pour retrouver les éléments de code auxquels ils sont associés. Le problème est bien entendu qu'un module peut implanter les services associés à un cas d'utilisation particulier et également des services utilisés par d'autres modules, services qui n'ont peut être rien à voir avec le cas d'utilisation original. Dans ce cas, le composant est hybride et ne peut pas strictement être isolé du reste du programme. L'analyse de concept peut, dans ce cas, être utilisée pour restructurer les modules [19].

5 Qualité de l'architecture résultante

5.1 Introduction

Une fois l'architecture de l'application originale obtenue, plusieurs solutions s'offrent à l'ingénieur de maintenance. Il peut vouloir réutiliser certains composants de traitement spécifiques dans le cadre d'une nouvelle application ou au contraire porter globalement l'ancienne application vers des architectures et technologies modernes. La décision dépasse le seul point de vue technologique et doit prendre en compte la stratégie d'entreprise et son contexte « business » de manière à déterminer le devenir des processus métier. En effet, de la solution choisie dépendra en particulier la flexibilité de la nouvelle application. En dehors de toute autre considération sur la qualité de l'application originale, si la flexibilité requise par l'application est faible, alors l'adaptation éventuelle de l'architecture n'est pas nécessaire et un portage de l'ancienne application dans un nouveau langage peut s'avérer une solution viable. Si en revanche le reengineering doit s'accompagner d'une augmentation de la flexibilité, alors l'analyse des processus métier et leur évolution probable est nécessaire. En réalité, cette approche doit se placer dans le contexte plus vaste des qualités non fonctionnelles du logiciel, c'est-à-dire l'ensemble des caractéristiques d'une application, qui, au-delà de la stricte fonctionnalité, font qu'elle satisfait l'ensemble des personnes impliquées. On les appelle souvent les *attributs qualité* du logiciel.

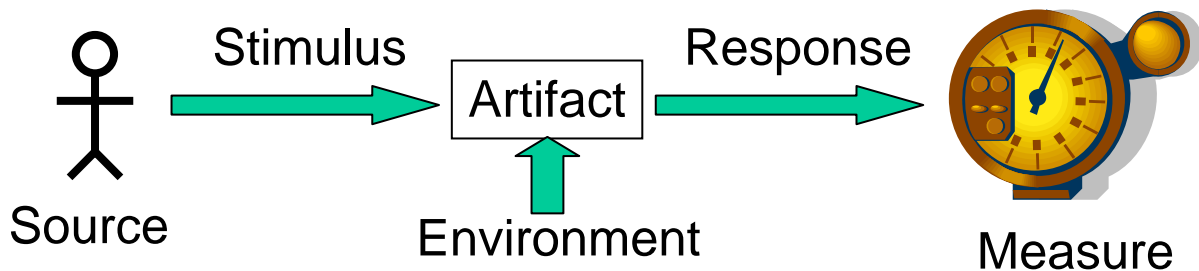
5.2 Attributs qualité

Les attributs qualité sont l'ensemble des caractéristiques d'une application, qui, au-delà de la stricte fonctionnalité, font qu'elle satisfait l'ensemble des personnes impliquées. Par exemple, il s'agit de qualités telles que :

- La flexibilité ;
- La maintenabilité / modifiabilité ;
- La portabilité ;
- La performance ;
- La sécurité ;
- L'utilisabilité ;
- La disponibilité ;
- La testabilité.

L'atteinte de la plupart de ces qualités est essentiellement conditionnée par l'architecture de l'application [2]. En effet, il semble raisonnable d'admettre que la fonctionnalité d'une

application ne dépend pas de la manière dont elle est programmée⁵. En revanche des qualités telle que la performance, la portabilité, la flexibilité le sont clairement. Il est par exemple beaucoup plus facile de porter une application si les éléments dépendant d'une plateforme spécifique sont encapsulés dans des composants localisés que si ces dépendances sont dispersées dans tout le logiciel. Il est important de relever ici que l'atteinte d'une qualité donnée s'accompagne souvent d'une dégradation d'autres qualités. Il s'agit donc, en général, de faire des compromis [12]. Pour juger de l'atteinte d'une qualité non fonctionnelle donnée, il faut se donner un cadre conceptuel de mesure (framework). Cependant, la plupart de ces qualités ne sont pas directement mesurables. Pour pallier ce problème le SEI propose d'utiliser une notion de scénario permettant au minimum de fixer les conditions d'évaluation d'un attribut qualité [2].



Dans cette figure, *l'artifact* est l'élément dont la qualité doit être évaluée. Cela correspond souvent au système en entier. La *source* est la personne ou le système qui sollicite l'artifact pour tester ses qualités. Le *stimulus* est l'action prise ou l'information introduite par la personne ou le système pour solliciter l'artifact. L'*environnement* est l'état du système et de son contexte, par exemple: fonctionnement en mode normal ou dégradé. La *réponse* est l'action ou le comportement de l'artifact provoqué par l'application du stimulus. Finalement, la *mesure* est la valeur attendue pour la réponse. La métrique doit bien entendu être adaptée au type de réponse à recevoir.

Exemple de scénario associé à la modifiabilité du système

Source: Développeur
Stimulus: Besoin de modifier / corriger une fonctionnalité
Artifact: Système en entier.
Environment: Système fonctionnel, en production.
Response: La modification doit être effective pour les fonctions concernées et neutre pour les autres. Le système doit être testé et remis en production.
Response measure: Maximum deux jours entre le début du projet de modification et la mise en production

6 Migration vers une architecture orientée services

L'architecture orientée services représente une architecture logicielle particulière. En conséquence elle amène des avantages non fonctionnels mais également des contraintes. Vis-à-vis des qualités énoncées précédemment, il est clair que l'attribut qualité bénéficiant principalement de cette approche est la flexibilité et, dans une certaine mesure, la modifiabilité. En effet, le but d'une approche SOA est de permettre une certaine souplesse dans l'organisation des services associés à un processus métier, Cela représente donc l'aspect flexibilité. De surcroît, comme les traitements d'information sont partitionnés selon

⁵ En admettant bien entendu que cette fonctionnalité soit correctement programmée, conformément au cahier des charges.

les étapes des processus métier, une telle architecture favorise la modifiabilité du logiciel pour autant que le stimulus soit une modification localisée dans une étape du processus, donc dans un seul service. La situation est plus difficile si une modification implique plusieurs services. Ainsi l'architecture basée services pourrait répondre favorablement, par exemple, au scénario suivant :

Source: Développeur
Stimulus: Besoin de réordonner les services correspondant aux étapes d'un processus.
Artifact: Système en entier.
Environment: Système fonctionnel, en production.
Response: La modification doit être effective pour les services concernés. Le système doit être testé et remis en production
Response measure: Maximum un jour entre le début du projet de modification et la mise en production

Dans ce cas, seule la couche de « chorégraphie » est impliquée et, en principe, la modification est légère. Toutefois, si une modification s'adresse au fonctionnement propre d'un service, alors c'est l'architecture du service lui-même qui devient importante. On se retrouve donc avec une contrainte sur les composants impliqués dans la couche inférieure.

Cependant, l'approche SOA peut être moins favorable qu'une approche traditionnelle sur d'autres dimensions de la qualité comme la performance, la sécurité ou la disponibilité. C'est pourquoi il convient, lors d'une opération de reengineering de spécifier convenablement les attentes en termes de qualités non fonctionnelles, par exemple au travers des scénarios. Ainsi, le choix d'une architecture pertinente est conditionnée par les qualités à atteindre. C'est l'essence du framework « Horseshoe » du SEI [4].

7 Conclusion

Dans le cadre de ce rapport nous avons montré que l'approche préconisée pour l'analyse des systèmes basés sur les services était proche de la méthode que nous avons développée pour le reengineering dans le cadre du projet ISNet62.

En particulier, nous avons proposé que l'identification à priori des processus métier d'une entreprise représente une base fiable pour l'analyse de l'architecture d'un système d'information. A partir de cette étape nous avons proposé, dans le cadre du projet ISNet62, de descendre graduellement les niveaux de granularité du système en veillant à chaque étape de faire le lien entre les concepts métier et le système logiciel.

Cependant, cette proximité de concepts entre notre approche de reengineering et les méthodes de conception basés SOA ne doit pas induire le lecteur à penser que l'approche SOA est la meilleure qui soit pour le reengineering d'un système. Tout d'abord, l'identification et l'encapsulation d'un service dans un logiciel existant est souvent une opération délicate. Ensuite il n'est pas sûr que cela corresponde aux objectifs du reengineering. Le point essentiel est d'être clair sur les qualités non fonctionnelles à atteindre via l'opération de reengineering. Ce sont elles qui, in fine, conditionneront le choix d'une architecture cible.

Aujourd'hui, il semble que la tendance « business » soit à la flexibilité des processus métier. Cette contrainte doit être clairement exprimée au moyen d'un scénario pour savoir de quelle flexibilité on parle. S'il s'agit de flexibilité dans l'ordonnancement des services dans un processus, alors l'approche SOA peut être une solution intéressante. En revanche, si la flexibilité est d'une granularité plus fine, alors le problème est reporté au niveau de

l'architecture des services eux-mêmes et l'approche SOA ne fournit dans ce cas qu'une partie seulement de la solution.

8 Bibliographie

- [1] Arsanjani A. – Service Oriented Modeling and Architecture. IBM. June 2004. www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/
- [2] Bass L., Clements P., Kazman R. – Software Architecture in Practice, 2nd edition. Addison-Wesley Inc. 2003.
- [3] Benson R.J., Bugnitz T.L. Walton W.B. – From Business Strategy to IT Action. John Wiley & Sons, 2004.
- [4] Bergey J., Smith D., Weiderman N., Woods S. - Options Analysis for Reengineering (OAR): Issues and Conceptual Approach. Software Engineering Institute, Carnegie Mellon University, Tech. Note CMU/SEI-99-TN-014, Sept. 1999.
- [5] Brodie M. L., Stonebraker M. – Migrating Legacy Systems. Morgan Kaufman, 1995.
- [6] Cullen A. - SOA Will Change How IT Works. Forrester Research, May 31, 2005.
- [7] Heffner R. – Service Orientation: Key Application Design Principles. Forrester Research, August 7, 2003.
- [8] Heffner R. – The Big Strategic Impact of Organic Business and Service Oriented Architecture. Forrester Research, June 17, 2004.
- [9] Heffner R. – Concurrent Business Engineering. Forrester Research, June 20, 2005.
- [10] Hubert R. – Convergent architecture. Addison-Wesley, 2002.
- [11] Huhns M.N., Singh M.P. – Service Oriented Computing. Key Concepts and Principles. IEEE Internet Computing, pp 75-81, January-February 2005.
- [12] Kazman R., Klein M., Clements P. – ATAM: Method for architecture Evaluation. Technical Report CMU/SEI-2000-TR-004, Aug. 2000.
- [13] Linding C., Snelting G. – Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis. Proc IEEE Int. Conference on Software Engineering (ICSE'97), 1997.
- [14] Mitchell B.S. – A Heuristic Search Approach to Solving the Software Clustering Problem. PhD Thesis, Drexel Univ. March 2002.
- [15] Sharp A., McDermott P. – Workflow Modeling. Artech House Inc. 2001.
- [16] Siff M., Reps T. – Identifying Modules via Concept Analysis. IEEE Trans. On Software Engineering 25(6) Dec. 1999.
- [17] Spratt D. – the business case for Service Oriented Architecture. CDBI Journal, November 2004
- [18] Taylor D. Business Engineering With Object Technology. John Wiley & Sons, 1995.
- [19] Tonella P. – Concept Analysis for Module Restructuring. IEEE Trans. On Software Engineering, 27(4), April 2001.
- [20] Verbaere M. – Program Slicing for Refactoring. MS Thesis, Oxford University, Sept 2003
- [21] Zachman J.A. - A framework for information systems architecture. IBM System Journal 26(3), 1987.
- [22] Zimmermann O. et al. – Elements of Service-Oriented Analysis and Design. IBM. June 2004. www-128.ibm.com/developerworks/webservices/library/ws-soad1/