Chapitre 1 : Concepts de base

Définition générale

Le SOA (Service Oriented Architecture) n'est pas une démarche entièrement nouvelle. L'idée de concevoir des architectures orientée services est née au début des années 1990 avec l'apparition des solutions Client / Serveur.

Aujourd'hui, les besoins d'ouverture du système d'information, les générations récentes des serveurs d'application (J2EE, .NET) et les Web services (SOAP, WSDL) donnent une nouvelle importance au SOA.

Il s'agit de bâtir des architectures applicatives adaptées aux serveurs d'application, capables d'exposer des services interopérables (legacy, progiciels) et de collaboration entre les entreprises (services métiers).

SOA et service

Cadre méthodologique de construction des systèmes d'information et d'implémentation du logiciel sous la forme de services. On peut mettre en place du SOA sans utiliser de Web services, ni retenir l'approche objet.

Un service expose une interface qui définit le traitement offert sous la forme d'un message d'entrée et d'un autre de réponse. Pour son implémentation, le service peut solliciter une ou plusieurs entités physiques de logiciel (ie. des composants). Le service exprime donc un niveau « logique » d'accès aux traitements et pas un niveau « physique » d'implémentation.

On distingue le concept de service métier spécifié au niveau du cahier des charges (fonctionnel, métier) et le concept de service issu d'un travail d'architecture applicative qui prépare l'implémentation des services métiers sous la forme de services.

Web service[1]

Technologie d'interopérabilité entre des systèmes distribués et hétérogènes mettant en œuvre XML et SOAP comme formats standards d'échange des messages. Il est fortement recommandé de mettre en place SOA lorsque l'on construit des Web services.

[1] Voir cadre de référence Web service.

Approche orientée objet

Méthodologie de construction des systèmes d'information et d'implémentation du logiciel sous la forme d'objets. SOA et approche objet sont complémentaires[2] . En effet, on utilise un principe de décomposition du service sous la forme de méthodes attachées à des objets.

Composant

Elément logiciel exécuté par un serveur d'application. Par exemple, dans l'infrastructure J2EE, un composant est un EJB, un javabeans, une servlet ou une classe RMI[3].

Cette définition est différente de celle retenue dans l'approche basée sur les composants (Component-Based Development). En effet, en CBD, le composant peut être implémenté dans différentes technologies et exprime donc aussi un niveau logique de représentation des traitements.

En SOA, on distingue clairement les deux niveaux logique et physique : **le service est le concept du niveau logique, le composant est le concept physique d'implémentation**.

Sur le site http://longhornblogs.com, Clemens Vasters, consultant, propose une définition intéressante :

*"A **service** is strictly not a unit of code; it is rather a boundary definition that might be valid for several different concrete implementations."*

*"The term **component** is a development and deployment concept and refers to some form of compiled code"*

[2] Suivant le même principe que la programmation structurée est complémentaire de l'approche objet pour la spécification des méthodes.

[3] Le concept de Web service n'est pas directement assimilé à un composant. En effet, il s'appuie sur une implémentation en composant qui est soit de type servlet (JAX-RPC), soit de type EJB. Voir aussi le cadre de référence Web services.

Définition détaillée du service

Un service est un traitement qui respecte les cinq propriétés que nous détaillons ci-après : couplage faible, activable à distance et interopérable, asynchrone, expose un contrat d'utilisation (interface), respecte le pattern d'architecture SOA.

Les propriétés obligatoires sont le couplage faible, l'exposition d'un contrat d'utilisation et le respect du pattern d'architecture SOA.

Propriété 1 : Couplage faible

En théorie

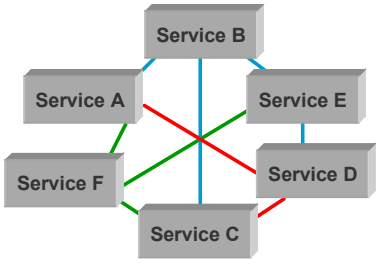
Un service ne peut pas appeler un autre service. Il délègue cette fonction à un traitement spécialisé dans l'enchaînement (fonction d'orchestration).

Dans la pratique

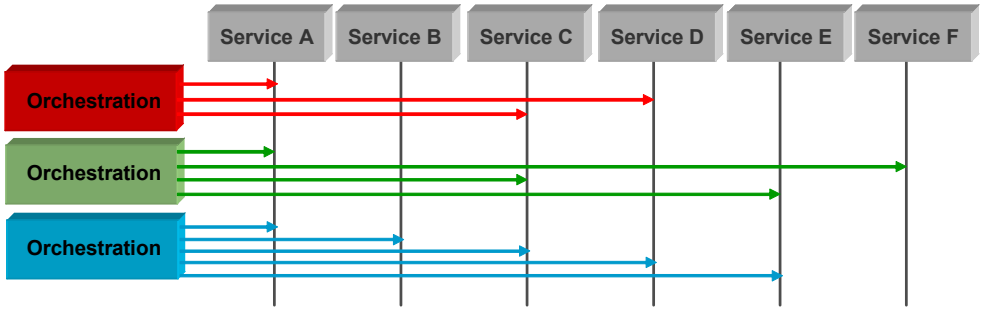
Cette propriété s'applique grâce aux principes de décomposition SOA et à l'usage des façades (voir suite du document).

C'est une propriété obligatoire.

Cette organisation des services n'est pas en couplage faible :



Cette organisation des services est en couplage faible :



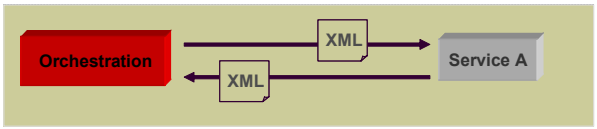
De manière complémentaire, on associe également la propriété du couplage faible à l'indépendance de l'activation du service vis-à-vis des technologies d'implémentation :

En théorie
Un service est activable indépendamment de sa technologie. Pour ce faire, l'activation se réalise par l'envoi (et la réception) d'un message XML. Il ne s'agit donc pas d'un appel binaire de type DCOM ou RMI-IIOP (Corba).

Dans la pratique
Cette propriété s'applique si on implémente le service à l'aide d'une invocation par message XML (XML-RPC et plus précisément SOAP-XML : voir Guide Cadre de référence Web services).

C'est une propriété obligatoire.

Activation de service agnostique sur le plan technique :



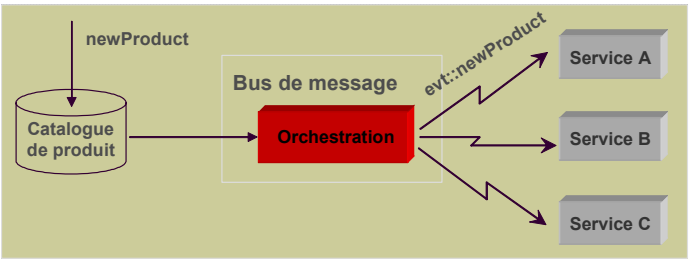
On peut aussi associer le couplage faible au mode « découplé » c'est-à-dire à une activation des services selon un mode asynchrone et gestion d'événements.

En théorie
Un service peut être activité suivant un mode asynchrone. Dans ce cas, le service s'abonne à un événement auprès d'une fonction d'orchestration

Dans la pratique
Cette propriété renvoie au concept d'architecture EDA (Event Driven Architecture) qui est un mode d'implémentation asynchrone de l'architecture SOA.

C'est une propriété non obligatoire.

Exemple basique d'une architecture EDA (Event Driven Architecture)



Lorsque l'activation des services se réalise à l'aide de XML et plus particulièrement SOAP (Web services), on dispose de plusieurs standards émergeants qui contribuent à l'EDA comme par exemple WS-Eventing et WS-Addressing (voir Guide Cadre de référence Web services).

Propriété 2 : Activable à distance et interopérable

En théorie
Un service expose une interface d'utilisation qui est la même indépendamment de sa localisation sur les réseaux. L'appel au service fonctionne quel que soit le langage et le système d'exploitation du consommateur[4].

Pour favoriser l'interopérabilité, l'usage des Web services[5] est intéressant (WSDL-SOAP).

Dans la pratique
Tous les services ne sont pas implémentés en WSDL-SOAP. Pour des raisons de performances et de gestion transactionnelle, l'utilisation d'autres protocoles d'activation restent nécessaires (EJB IIOP-RMI, DCOM, MOM[6]…).

Un service dont l'activation se réalise toujours au sein d'une même machine virtuelle utilise un appel local (pas d'activation à distance nécessaire).

C'est une propriété non obligatoire.

Propriété 3 : Asynchrone

En théorie
Un service fonctionne de manière asynchrone. C'est-à-dire qu'il ne bloque pas le consommateur pendant qu'il s'exécute. Ce principe est intéressant pour réduire les goulets d'étranglement (performance, robustesse).

Dans la pratique
Tous les services ne peuvent pas être appelés en mode asynchrone *via* un MOM.

De plus, compte tenu de l'état de l'art, cette propriété est contradictoire avec la précédente : les MOM exposent des protocoles propriétaires de communication. Par exemple, avec JMS[7], il faut que le fournisseur et le consommateur dispose chacun d'une couche de communication Java.

L'architecture EDA (Event Driven Architecture) des Web services (WS-Eventing, WS-Addressing…) est encore émergente (Septembre 2004). Les implémentations interopérables et stables de ce mode de fonctionnement n'apparaîtront vraisemblablement pas avant 2006 (*2005 étant plutôt l'année des prototypes sur ce sujet*).

C'est une propriété non obligatoire bien que son existence s'avère une condition *sine qua non* pour le déploiement à grande échelle des services (le découplage des systèmes grâce à l'asynchronisme est un point déterminant des architectures).

[4] C'est-à-dire l'application client qui active le service.

[5] WSDL et SOAP. Voir cadre de référence Web service.

[6] Message Oriented Middleware.

[7] Java Message Service.

Propriété 4 : Expose un contrat d'utilisation

En théorie

Un service expose un contrat d'utilisation[8] décrit en deux parties[9]. Une partie abstraite qui déclare les messages d'entrée et de réponse du traitement offert.

Une partie concrète qui décrit les standards et protocoles techniques utilisés pour l'activation du service[10] : XML-RPC, SOAP-HTTP, SOAP-JMS, protocole binaire… Selon les choix d'implémentation et de déploiement, il est possible d'avoir plusieurs parties concrètes pour une même partie abstraite.

On nomme aussi le contrat d'utilisation par le terme d'interface de service.

Dans la pratique

Cette propriété est respectée entièrement lorsque l'on met en œuvre les Web services. En effet, le document WSDL permet d'isoler les parties abstraite et concrètes.

Dans le cas de service implémenté dans d'autres techniques, le contrat d'utilisation peut se limiter à une simple API dans le langage d'implémentation. Par exemple une API java qui fait l'objet d'une documentation en javadoc[11].

C'est une propriété obligatoire sauf pour la dissociation de la partie abstraite et de la partie concrète qui reste particulière aux Web services.

On rappelle que le contrat d'utilisation comporte plusieurs natures d'information (syntaxique, sémantique, qualité de services) qui nécessitent des standards et langages différents de spécification formelle[12]. La représentation formelle permet d'automatiser les phases de validation des contrats en exploitation, c'est-à-dire de s'assurer du respect des clauses des contrats selon les contextes d'usage.

On précise maintenant la définition des natures d'information qui composent le contrat d'utilisation du service :

- Le contrat syntaxique présente le nom du traitement offert et ses paramètres d'entrée et de réponse, c'est-à-dire les messages. Il correspond à l'interface. Il présente aussi les informations sur le binding, c'est-à-dire le ou les protocoles de communication et d'encodage des données utilisés. On peut avoir plusieurs bindings différents pour un même service.

---

[8] Le terme contrat de service est réservé au contrat de nature juridique qui précise les règles de fonctionnement entre le fournisseur et le consommateur.

[9] Le détail du contenu du contrat est précisé au chapitre suivant, notamment les préconditions et les postconditions.

[10] Notion de Binding.

[11] Mécanisme de documentation des classes et des méthodes en Java J2EE.

[12] Se reporter au Guide Cadre de référence Web services 'Validation des contrats d'utilisation' pour une présentation exhaustive des possibilités de représentation formelle des contrats (Schematron, Xpath, WS-Policy…)

- Le contrat sémantique fournit une description informelle du traitement et notamment les pré-conditions (équation logique des valeurs des paramètres d'entrée permettant le déclenchement) et les post-conditions qui expliquent, pour chaque pré-condition, le mode de valorisation du message de réponse. On complète la description informelle par une description formelle grâce à l'OCL (Object Constraint Langage) ou Xpath ou encore en facettes de schéma XML (3).
- Le contrat de qualité de service (Quality Of Service) qui contient les engagements de temps de réponse, de disponibilités (4)…

Propriété 5 : Respecte le pattern d'architecture SOA

En théorie

Le pattern d'architecture SOA consiste à créer une architecture applicative qui décompose les traitements sous la forme de services rattachés à des paquets de classes. Ces paquets forment des Catégories[13] (objet métier, sujet métier), chacune dotée d'une façade d'accès qui contient l'ensemble des services qu'elle expose (on parle aussi de port). Un service a le droit d'interagir uniquement avec les classes de sa catégorie[14].

En SOA, l'encapsulation ne se fait pas uniquement au niveau le plus élémentaire de la classe[15] mais aussi au niveau supérieur des catégories UML.

Dans la pratique

Cette propriété en entièrement respectée.

Il s'agit de la démarche de conception et d'architecture SOA que nous détaillons dans les chapitres suivants.

C'est à partir de cette propriété que l'on découvre les services.

C'est une propriété obligatoire.

APERCU RAPIDE DU PATTERN D'ARCHITECTURE APPLICATIVE SOA

On présente ici une synthèse sur les principes importants de communication entre les services et les catégories. Sur la figure de gauche ci-dessous, on voit plusieurs violations des principes du couplage faible car les services s'appellent directement entre eux aussi bien au niveau d'une catégorie qu'entre deux catégories.

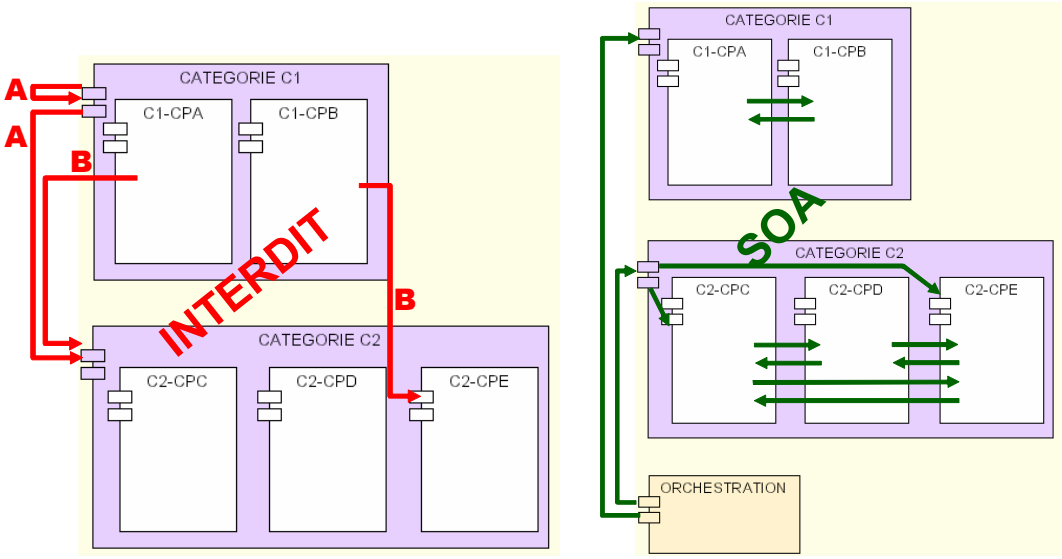
La figue de droite montre l'approche à retenir et qui contribue à l'architecture SOA. Une fonction d'orchestration apparaît et prend en charge les appels entre services. Ainsi, chaque catégorie reste isolée des autres (découplage du système).

[13] Au sens de Grady Booch, contributeur à UML.

[14] En adoptant cette approche, on impose que le service soit mono sujet de gestion. Sa granularité dépend donc, en grande partie, de celle des catégories UML (voir chapitre suivant).

[15] Au sens de l'approche objet.

Figure 1 : Aperçu du principe du pattern d'architecture applicative SOA



Les opérations des services métiers sont des façades qui assemblent des services rattachés aux catégories. Ainsi, dans le cas d'une implémentation technique dans les standards Web services (SOAP-XML, WSDL), l'opération SOAP ne contient pas de logique métier étendue mais uniquement la logique d'agrégation de services rattachés à des catégories (fonction d'orchestration dans la figure de droite ci-dessus).

A partir de ces principes, on construit un modèle logiciel (architecture logique) par exemple sous la forme d'un framework de classes (java, .Net) qui réifie les concepts manipulés : processus, services métiers et opérations, phases… On obtient ainsi une continuité entre la modélisation amont (processus, services métiers…) et l'architecture du logiciel.

Tous ces principes sont détaillés dans la suite de ce document.

Cas du service métier

Mode de découverte et cas d'usage

Le traitement offert par un service métier est un service de type particulier ; il respecte l'ensemble des propriétés du service sauf celle qui concerne le pattern d'architecture SOA.

En effet, le service métier n'est pas découvert à l'occasion de la mise en place du pattern d'architecture SOA. Il est découvert directement au niveau de la modélisation des processus (diagramme d'activités). Il correspond à un périmètre fonctionnel que l'on souhaite exposer à des consommateurs[16] indépendamment des choix d'architecture applicative.

Type d'usage

Le service métier intervient dans les types d'usage suivants :

- **Usage métier et fonctionnel** : exposition d'une fonctionnalité à des organisations clientes tierces qui consomment le service. Si ces organisations sont internes à l'organisation du fournisseur, on parle aussi d'usage fonctionnel[17].
- **Usage technique** : exposition d'une fonctionnalité qui permet la communication entre des systèmes informatiques hétérogènes (par exemple, contexte de type EAI ou ESB)[18].

Dans le cas de l'usage métier et fonctionnel, le processus modélise un traitement métier. Dans le cas de l'usage technique, le processus modélise une interface applicative entre plusieurs systèmes informatiques (type EAI, ESB).

Terminologie retenue

Dans la pratique, l'implémentation des services métiers se fera de plus en plus grâce aux technologies Web services (WSDL, SOAP)[19]. **Afin d'être conforme avec ce nouveau standard, sans pour autant l'imposer, nous retenons les principes suivants** :

- Un service métier expose une ou plusieurs opérations[20]. Le contrat d'utilisation du service métier, c'est-à-dire son interface, peut donc être plurielle puisque plusieurs opérations peuvent être exposées. Ceci à la différence du service issu du pattern d'architecture applicative qui n'expose qu'un seul et unique traitement.

---

[16] Le service métier étant implémenté à l'aide de services (voir suite), on ne peut pas construire des services métiers sans retenir la démarche méthodologique SOA.

[17] Exemple d'usage fonctionnel : l'exposition de services d'accès à un annuaire LDAP pour des organisations internes à l'entreprise.

[18] Bien qu'il s'agisse d'une préoccupation technique on conserve le terme générique de service métier puisqu'il s'agit d'une préoccupation du « métier » de la Direction Informatique. Ce choix évite de créer un nouveau terme.

[19] Voir cadre de référence Web service.

[20] On rappelle que « opération » est le terme standard retenu par les Web services pour nommer « un service » dans le Web service. Un Web service peut contenir plusieurs opérations (donc plusieurs services) : voir cadre de référence Web service.

- Une opération correspond à une fonctionnalité offrant un traitement de bout en bout[21] et qui respecte les propriétés du concept de service sauf celle relative à l'architecture applicative. Une opération est donc obligatoirement en couplage faible vis-à-vis des autres opérations et s'expose au travers du contrat d'utilisation du service métier. Compte tenu de la vocation du service métier, la propriété d'activation à distance de l'opération s'impose généralement. Celle de l'interopérabilité dépend de l'implémentation ou non en Web service[22]

Si l'implémentation en Web service est retenue, on obtient alors une équivalence immédiate des concepts : un service métier est implémenté sous la forme d'un Web service et les opérations correspondent directement à celles exposées par le Web service.

Il est possible de concevoir et d'implémenter un service métier qui n'expose qu'une seule opération.

Particularités du service métier

Comme nous le verrons dans le chapitre suivant, la spécification des services métiers nécessite une très bonne maîtrise des principes méthodologiques suivants :

MODELISATION DU PROCESSUS - DIAGRAMME D'ACTIVITES

L'opération d'un service métier est presque toujours une étape d'un processus plus large qu'il faut modéliser avec précision. La relation entre le fournisseur et le consommateur est spécifiée au travers de la modélisation du processus. On utilise le diagramme d'activités UML pour cette modélisation (ou MOT en Merise).

PRECONDITIONS ET POSTCONDITIONS

Les pré-conditions permettent d'énoncer les règles à respecter pour le déclenchement de l'opération. Pour chaque pré-condition on précise également la postcondition qui définit les conditions d'émission du résultat de l'opération.

La spécification des pré-conditions et des post-conditions constitue des éléments importants pour le contrat d'utilisation du service.

Ce concept est aussi utilisable pour la construction des services au niveau du pattern d'architecture SOA.

CONTEXT- AWARE

Une même opération peut avoir des comportements différents selon le consommateur et le contexte d'exécution. Par exemple, en fonction d'un canal internet ou call-center le résultat de l'opération peut varier. Plus le nombre de consommateurs et de contextes augmente, plus le paramétrage de l'opération est déterminant. Au moment de la spécification des opérations, il faut donc définir les différents contextes d'usage afin de modéliser les paramètres nécessaires. Une fois paramétrée, l'opération est capable de tenir compte de son contexte d'exécution pour adapter son comportement : c'est le concept de context-aware.

Le context-aware peut également être utilisé pour certains services au niveau du pattern d'architecture applicative SOA.

---

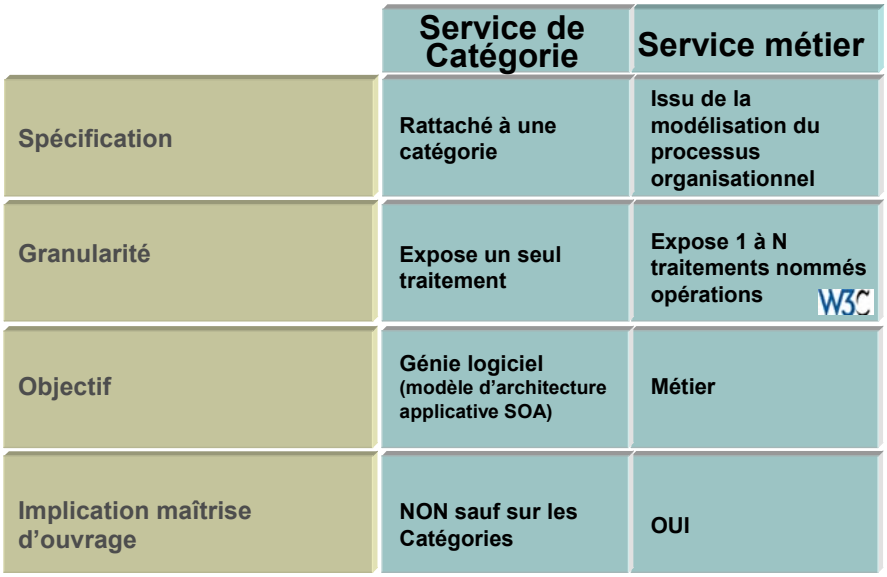
[21] En anglais on parle de « self contained » pour exprimer l'autonomie de fonctionnement du traitement.

[22] Les Web services avec SOAP-WSDL favorisent les déploiements interopérables.

Différences entre Service et Service métier

La figure ci-dessous synthétise les différences majeures entre le concept de service et de service métier que nous venons de décrire dans les sections précédentes.

Figure 2 : Différences entre Service de Catégorie et Service métier



Le tableau ci-dessous présente la liste complète des propriétés par type de service.

Tableau 1 : Synthèse des propriétés par type de service

	Service métier	Service exposé par une Catégorie	Service interne à une Catégorie
Pré-condition	Oui	Possible	Non
Post-condition	Oui	Possible	Non
Façade (réifié par)	Oui	Oui	Non
Mode dégradé	Oui	Non	Non
Paramétrage (variantes)	Oui	Non	Non
SLA	Oui	Non	Non
Données Pivot XML	Oui	Non	Non
Données Pivot Interne	Non	Oui	Non
Couplage faible	Oui	Oui	Non
Agnostique	Oui	Possible	Non
Attaché à Catégorie	Non	Oui	Oui
Organisationnel	Oui	Non	Non
Mono-traitement	Non	Oui	Oui
Contrat formel	Oui	Non	Non
Consommable	Oui	Non	Non

Définition détaillée du composant

Un composant est une entité de traitement qui respecte les propriétés que nous détaillons ci-après : entité physique, réifie le pattern d'architecture applicative SOA, typé N-Tiers, distribuable, expose une interface.

Les propriétés obligatoires sont les suivantes : entité physique, typé N-Tiers pour les composants internes à une catégorie, réifie le pattern d'architecture applicative SOA, expose une interface de services sans pour autant garantir le couplage faible entre les services.

Propriété 1 : Entité physique

En théorie	*Dans la pratique*
Le périmètre du composant est celui d'une entité physique de logiciel qui s'exécute en exploitation. Dans une infrastructure J2EE il peut s'agir d'un EJB, d'une servlet…	**C'est une propriété obligatoire.**

Propriété 2 : Réifie le pattern d'architecture applicative SOA

En théorie	*Dans la pratique*
Le périmètre d'un composant est régulé par les concepts issus de la démarche SOA. Les cas suivants sont possibles : Soit il réifie la façade d'une catégorie. Par exemple, il s'agit d'un EJB Session qui expose tous les services publiés par la catégorie qu'il réifie. Soit il correspond à un composant à l'intérieur d'une catégorie. Dans ce cas, il ne peut regrouper que des classes appartenant obligatoirement à cette catégorie. Le périmètre du composant ne peut jamais s'étendre au-delà de sa catégorie d'appartenance. Soit il réifie un concept d'orchestration situé au niveau du processus, de la phase ou du service métier (c'est-à-dire de l'opération).	**C'est une propriété obligatoire.**

Propriété 3 : Typé N-Tiers

En théorie

Le traitement assuré par un composant interne à une catégorie est dédié à une couche du modèle N-Tiers qui généralement se compose en trois niveaux : présentation (IHM), applicatif (métier) et accès aux données (persistance).

On distingue également un type de composant particulier qui permet d'implémenter les façades des catégories. Ces façades sont transverses aux couches N-Tiers.

Dans la pratique

C'est une propriété obligatoire pour les composants internes à une catégorie.

Le composant qui réifie une catégorie peut exposer des services placés aux différents niveaux de l'architecture N-Tiers.

Propriété 4 : Distribuable

En théorie

Distribuable sur le réseau sans impacte sur son code applicatif et sans modification des relations existantes avec les autres composants. Il s'agit d'une propriété de transparence de la localisation physique du composant sur le réseau.

En corollaire du point ci-dessus, au-delà de son périmètre fonctionnel, ne peut activer que d'autres composants

Dans la pratique

Il est possible de construire des composants dont la distribution sur le réseau est d'emblée écartée pour des raisons de besoins et/ou de performance des implémentations.

C'est une propriété facultative.

Propriété 5 : Expose une interface de services

En théorie

Expose une interface de services, dont les services sont physiquement localisés dans le périmètre du composant[23].

S'il s'agit d'un composant d'une catégorie, ses services ne sont pas directement visibles à l'extérieur de la catégorie.

Dans la pratique

Les propriétés associées au concept de service au sens SOA ne sont pas obligatoirement toutes respectées au niveau des services exposés par un composant. Plus précisément, voici les règles à suivre pour les propriétés obligatoires attachés au concept de service au sens SOA :

Couplage faible : on admet que les services des composants au sein d'une même catégorie puissent interagir directement entre eux. C'est le corollaire de l'acceptation d'un couplage fort entre les méthodes au sein des classes d'une même catégorie. Néanmoins, l'architecture peut mettre en œuvre des principes d'orchestration interne à la catégorie permettant d'assurer un couplage faible entre les composants d'une même catégorie. L'apport reste toutefois à justifier par rapport au travail d'architecture supplémentaire à produire. Généralement le couplage faible au niveau des services exposés par les catégories est suffisant.

Expose un contrat d'utilisation : propriété à respecter. C'est un équivalent à la propriété d'exposition d'une interface décrite ici.

Respecte un pattern d'architecture applicative : Cette propriété est respectée puisque le composant réifie les concepts SOA.

[23] A la différence du concept de service dont l'interface exposée (le contrat d'utilisation) peut s'étendre sur plusieurs composants; voir aussi définitions présentées dans le premier chapitre.

Différence entre Service et Composant

La figure ci-dessous synthétise les différences majeures entre le concept de service et de composant que nous venons de décrire dans les sections précédentes.

Figure 3 : Différences entre Service et Composant

	Service	Composant
Couplage faible	OUI	Pas obligatoire au sein d'une catégorie
Contrat d'utilisation	OUI + outil de gestion	OUI + simple API (type javadoc)
Modèle d'architecture SOA	OUI-> Appartient à une Catégorie	OUI-> concrétise Catégorie, Orchestration
Entité physique d'exploitation	NON	OUI
N-Tiers	NON	OUI
Distribuable sur le réseau	Facultatif -> sauf si web service	Facultatif

Aperçu général

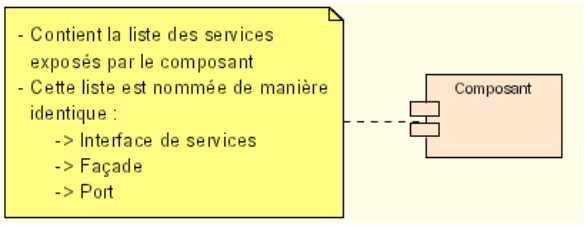
Dans l'ensemble de cette étude, nous retenons les notations UML 1.x[24] sauf indications contraires.

Les illustrations présentées maintenant anticipent la description des règles de construction SOA détaillées dans les chapitres suivants. Néanmoins, ces illustrations permettent d'appréhender rapidement et de manière générale les grands principes SOA retenus.

Représentation UML d'un composant

En UML 1.x la représentation du composant est illustrée ci-dessous. La partie droite permet de représenter, de manière abstraite, la liste des services exposés par le composant. Le vocabulaire utilisé pour nommer cette liste est multiple : interface de services, façade ou port.

Figure 4 : Représentation UML d'un composant



Il est important de distinguer les deux cas d'application suivants :

- Si le composant réifie la façade d'une catégorie, alors la liste des services est celle exposée par la catégorie et respecte les propriétés du concept de service.
- Si le composant est une unité de découpe au sein de la catégorie, alors la liste des services respecte les propriétés du concept de composant.

Représentation en UML 2.x

UML 2.x est encore récent (*début 2004*). Il permet notamment de mieux représenter graphiquement les interactions entre les interfaces de services de plusieurs composants.

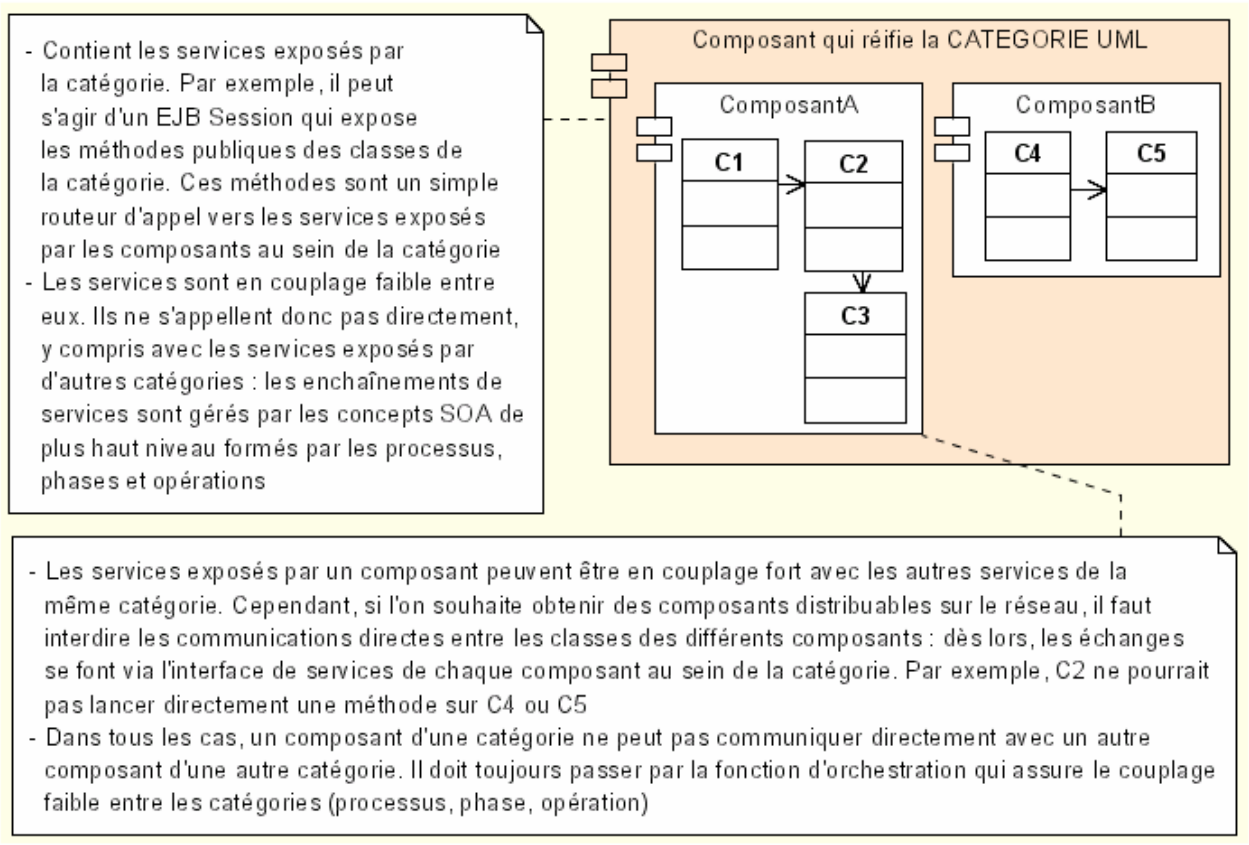
Son usage n'est pas détaillé dans cette étude et n'impacte pas les règles SOA décrites.

Représentation du composant au niveau de la catégorie UML

Cette figure montre comment la catégorie est réifiée sous la forme d'un composant, elle-même décomposée en deux composants internes qui organisent son implémentation.

[24] UML 2.x apporte des compléments de notations intéressants pour notamment les diagrammes de composants, les diagrammes de séquence et d'activités. Néanmoins, cela n'impacte pas les principes SOA détaillés ici.

Figure 5 : Représentation du composant au niveau de la catégorie UML



Les composants internes à la catégorie sont typés selon les couches N-Tiers : présentation, applicatif (métier), base de données (persistance).

Représentation du composant au niveau processus et service métier

Chaque processus et service métier est réifié sous la forme d'un composant.

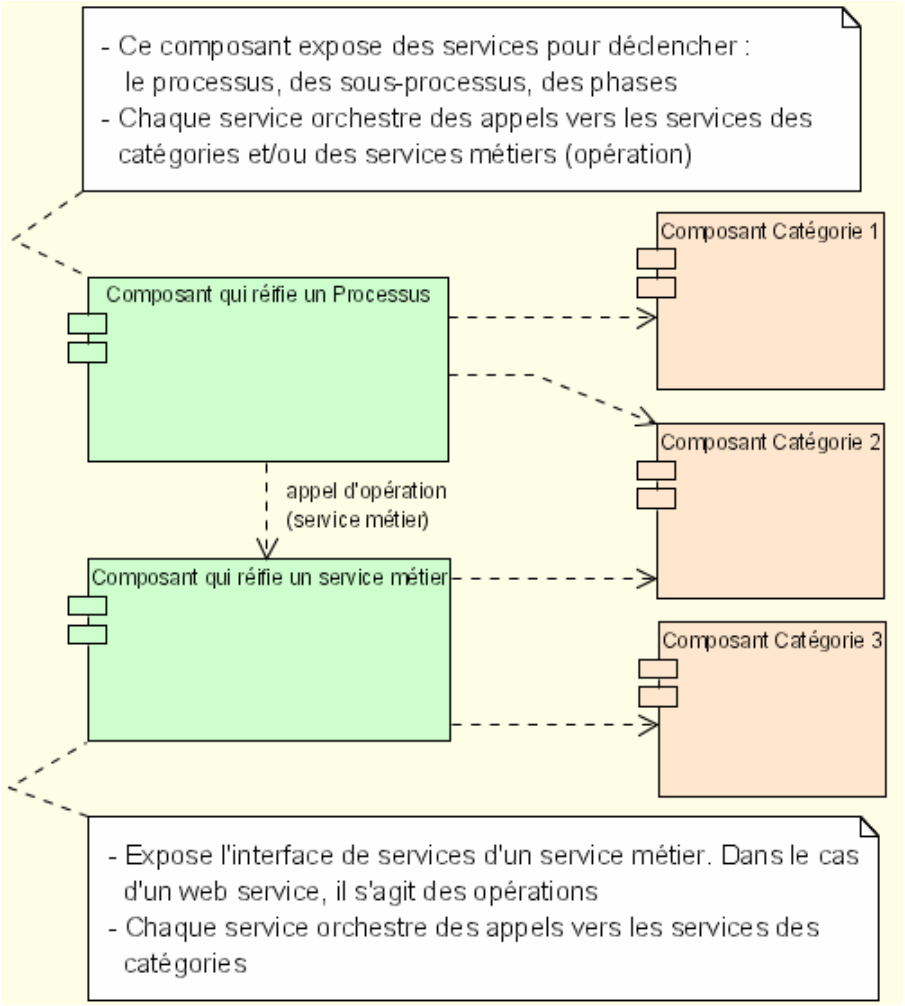
Les services exposés par ces composants jouent le rôle d'orchestrateur d'appel vers les services implémentés dans les catégories. C'est de cette façon que la propriété de couplage faible entre les services est assuré : un service d'une catégorie n'appelle jamais un autre service directement mais délègue ce rôle soit à un processus (ou une phase dans le processus), soit à une opération au sein d'un service métier.

Si le processus contient des sous-processus et/ou des phases, les services correspondants sont implémentés sous la forme d'orchestrateur dans le composant.

Le processus (et le sous-processus) peut aussi orchestrer des appels vers les opérations des services métiers.

La figure suivante illustre ces mécanismes.

Figure 6 : Représentation du composant au niveau processus et service métier



Utilisation de stéréotypes UML

Afin de faciliter la lecture des modèles UML, il est intéressant d'introduire des stéréotypes qui permettent de représenter les types des composants (tableau ci-dessous).

Tableau 2 : Liste des stéréotypes UML utilisés pour le typage des composants

<< façade catégorie >> Composant	Contient les services exposés par la catégorie. Ces services peuvent être de type présentation ou métier. Les services d'accès aux données sont encapsulés par la couche métier.
<< présentation catégorie >> Composant	Représente un composant de type graphique de la catégorie. Son interface expose un ou plusieurs services de présentation.

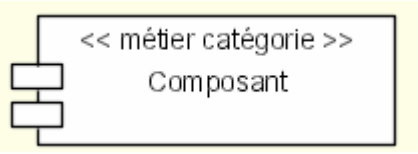
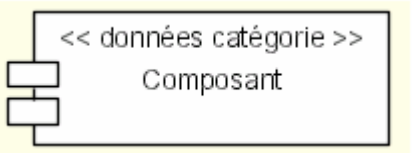
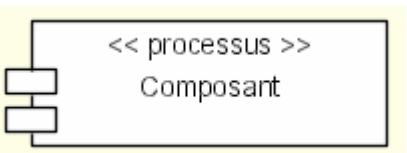
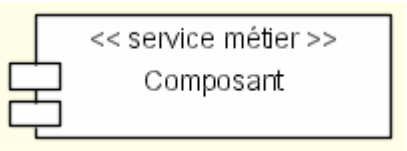
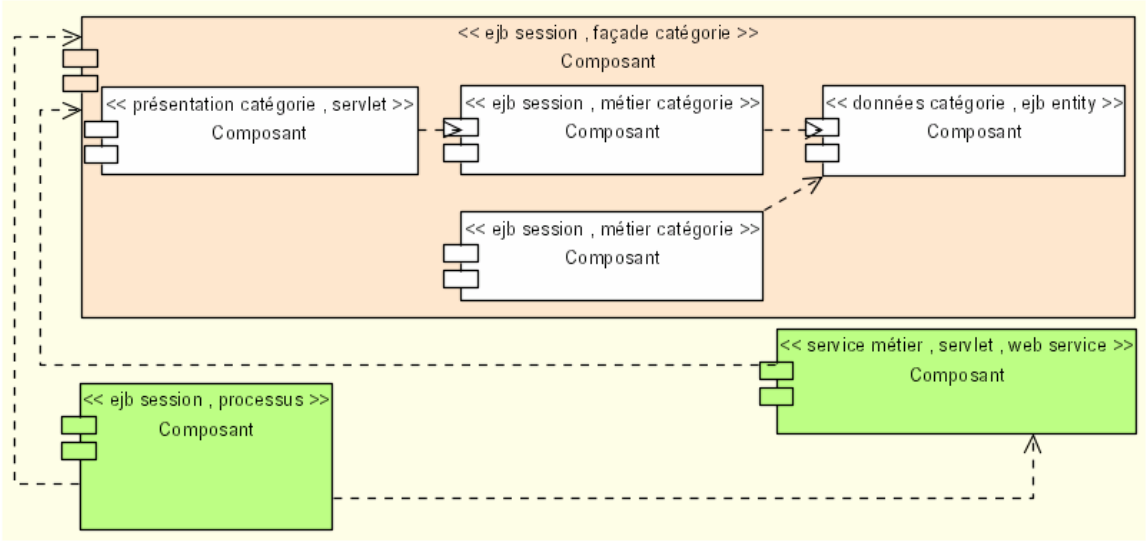
	Représente un composant de type métier de la catégorie. Son interface expose un ou plusieurs services de type applicatif.
	Représente un composant de type données de la catégorie. Son interface expose un ou plusieurs services de gestion de la persistance.
	Contient les services exposés pour la demande d'exécution d'un processus (et sous-processus, phases).
	Contient les services exposés par un service métier. Dans le cas d'une implémentation en Web service, il contient les opérations du Web service

Figure 7 : Exemple 1 d'utilisation de stéréotypes pour les types de composants



Comme la figure ci-dessus le montre, on peut aussi étendre les stéréotypes avec les types de composants utilisés dans le serveur d'application.

Par exemple, en J2EE on dispose des stéréotypes suivants : <<ejb session>>, <<ejb entity>>, <<servlet>>, <<rmi>>, <<javabeans>>, <<local>>.

Dans la figure, on note que le composant qui réifie le service métier est stéréotypé en <<servlet>> mais aussi en <<Web service>>. Cela permet de représenter un mode d'implémentation en Web service JAX-RPC[25]. Si l'implémentation du Web service est réalisée en EJB, alors il faut indiquer à la place de <<servlet>> le stéréotype <<ejb session>>.

[25] Voir cadre de référence Web service.

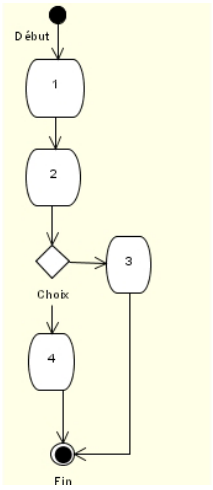
Niveau d'utilisation du SOA dans le cycle projet

Processus conceptuel

Au stade conceptuel, la modélisation du processus ne tient pas compte des acteurs ni des systèmes utilisés (outils informatiques). Il s'agit d'une description « réglementaire » de haut niveau.

L'utilisation des principes de décomposition SOA à partir de cette représentation débouche sur la spécification de services éloignés du contexte organisationnel et systèmes. Les services sont alors « théoriques » et doivent ensuite être restructurés pour tenir compte de l'organisation. Leur valeur est du niveau documentaire. Ainsi, au stade conceptuel, la rentabilité du SOA est délicate et sa mise en œuvre nécessite une forte maturité méthodologique.

Exemple de processus conceptuel



Processus macro organisationnel

Au niveau macro organisationnel, le processus tient compte des types d'acteur concernés par le processus ainsi que les types d'outils informatiques.

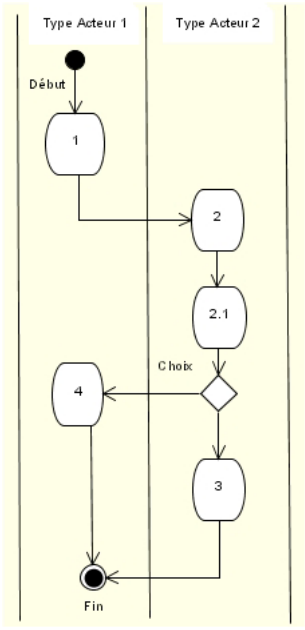
Le SOA appliqué au niveau macro organisationnel permet d'obtenir des services qui représentent un compromis intéressant entre la préoccupation conceptuelle et la dérivation dans la structure organisationnelle.

Ces services sont re-travaillés au niveau suivant de la modélisation (réutilisation du macro organisationnel pour établir le niveau organisationnel). Ils servent donc utilement à l'effort de modélisation du système d'information.

Néanmoins, la modélisation macro organisationnelle est souvent à la charge d'équipes transverses (urbanisation) et peu d'entreprises engagent des moyens adaptés pour mener ces travaux dans de bonnes conditions (compétences, budgets).

Sur la figure ci-contre, on montre le même processus que celui présenté plus haut, étendu à la prise en compte des types d'acteurs. On pourrait aussi représenter les types de systèmes (outils informatiques).

Exemple du même processus détaillé en macro organisationnel



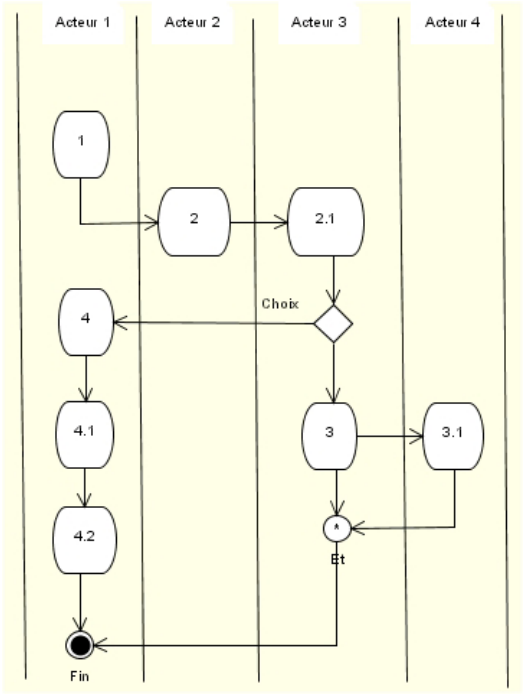
Processus organisationnel

Ici le processus tient compte de la réalité de l'organisation et des contingences des outils informatiques. C'est le niveau le plus évident pour l'entreprise qui débute avec le SOA.

La décomposition SOA permet d'obtenir des services qui tiennent compte des contingences d'implémentation tout en offrant un bon niveau de stabilité vis-à-vis des évolutions organisationnelles et systèmes. Pour ce faire, les services sont organisés autour des éléments les plus stables du SI, c'est-à-dire les catégories UML (objets métiers ou sujets métiers).

Cette modélisation se réalise dans le cadre d'un projet et il faut disposer des bonnes pratiques de conception notamment pour déterminer les catégories UML[26]. Sur la figure, on reprend le processus présenté plus haut afin de le détailler au niveau organisationnel. On représente ici les acteurs, on peut aussi représenter les systèmes (outils informatiques).

Exemple du même processus détaillé en organisationnel



Implémentation technique

Enfin, le SOA se projette au niveau de l'implémentation du logiciel, par exemple dans les serveurs d'application J2EE et .NET.

Pour ce faire, on élabore un pattern d'organisation des classes qui forment un modèle logique. Ce modèle permet de réifier les concepts SOA manipulés et prépare l'implémentation du logiciel. Par exemple, un processus est réifié sous la forme d'une classe dont les méthodes représentent le point d'entrée dans le processus mais aussi ses sous-processus et ses phases.

Enfin le modèle logique est projeter sous la forme de composants dans le serveur d'application : javabeans, EJB Session, EJB Entity…ou équivalent en .NET

Dans la suite de cette étude, la préconisation retenue est d'introduire le SOA à partir des processus organisationnels puis de le poursuivre jusqu'au niveau de l'implémentation technique.

[26] Il faut aussi prévoir un rôle transverse aux projets pour l'administration des catégories UML et des services ; voir annexe Administration et méta modèle SOA.

Définition des rôles

Fournisseur

Le fournisseur de services est l'organisation qui délivre un service.

Consommateur

Le consommateur de service est l'organisation qui consomme un service, c'est-à-dire qui implémente son contrat d'utilisation (Interface au sens d'une API ou d'un WSDL[27] pour un Web service par exemple) dans son système. Lorsqu'il s'agit de l'utilisation d'un service métier (usage métier), le consommateur est souvent nommé « organisation cliente ».

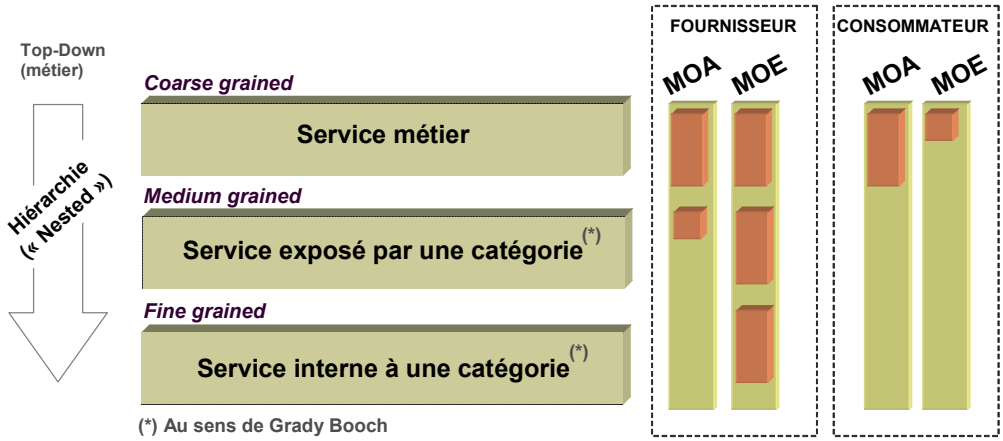
[27] Web service Description Language (voir Guide Cadre de référence Web services).

Synthèse des concepts clefs

Types de services

Il faut comprendre que le concept de service intervient à plusieurs niveaux dans l'architecture du Système d'Information. Plus précisément, on retient les trois niveaux suivants :

Figure 8 : Les types de services



- **Service métier**. C'est le « plus haut » niveau. Il s'agit du service qui fait sens pour le métier, donc pour la maîtrise d'ouvrage du fournisseur mais aussi pour la maîtrise d'ouvrage du consommateur. Cette dualité des maîtrises d'ouvrage est aussi un point particulier qu'il faut bien noter en SOA. Nous y reviendrons dans la suite.

- **Service exposé par une Catégorie**[28]. Il s'agit du niveau de l'architecture applicative du logiciel. Ces services peuvent être visibles de la maîtrise d'ouvrage du fournisseur car ils sont au carrefour de la préoccupation métier et d'architecture du logiciel. Généralement, on constate que dans les entreprises ayant déjà une expérience en SOA, les maîtrises d'ouvrage sont impliquées. Ces services forment le grain privilégié d'administration et de réutilisation des services.

- **Service interne à une catégorie**. Il s'agit des services qui implémentent ceux exposés par les catégories. Ces services sont proches de l'implémentation logiciel notamment du concept de composants que nous détaillons plus loin dans le séminaire. Ils ne sont pas visibles de la maîtrise d'ouvrage.

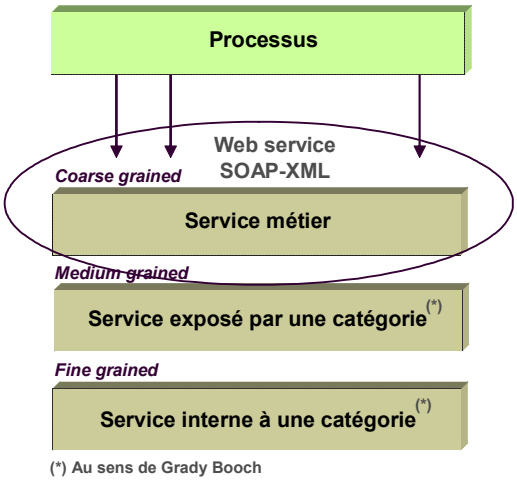
[28] Au sens de Grady Booch, voir Chapitre Conception.

La figure ci-dessous complète la typologie et positionne le concept de processus. Un processus permet d'enchaîner plusieurs services métiers afin de réaliser un traitement de bout en bout sur le plan organisationnel. Concrètement, il est modélisé sous la forme d'un Diagramme d'activités UML ou d'un MOT en Merise.

Compte tenu de la vocation des Web services (exposition vers des consommateurs hétérogènes) et de la représentation des messages en XML (risque de dégradation des performances à cause notamment du parsing), il faut limiter leurs usages aux services métiers (voir aussi Guide Cadre de référence Web services).

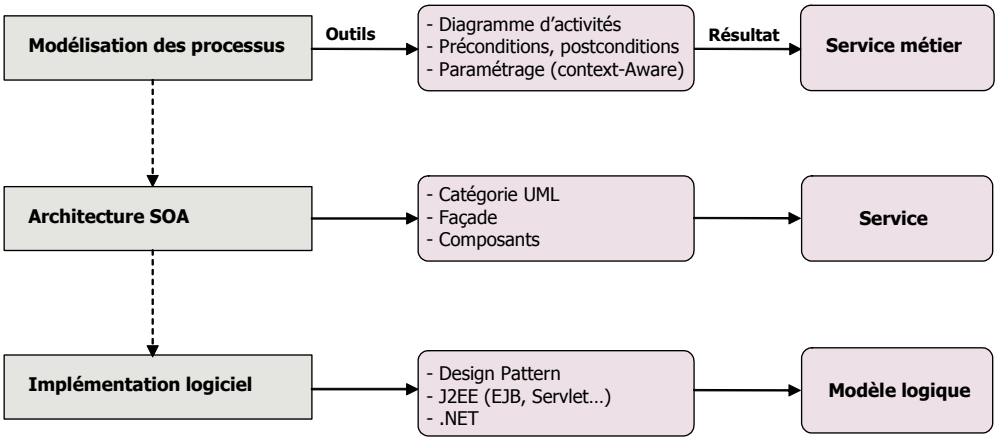
Un Web service est donc 'coarse grained', pas 'medium grained' ni 'fine grained'. C'est une règle de départ que l'on peut dégrader selon les besoins d'un projet mais en justifiant les choix.

Figure 9 : Processus et Web service



La figure suivante synthétise les concepts clefs que nous venons de décrire (commentaires en page suivante).

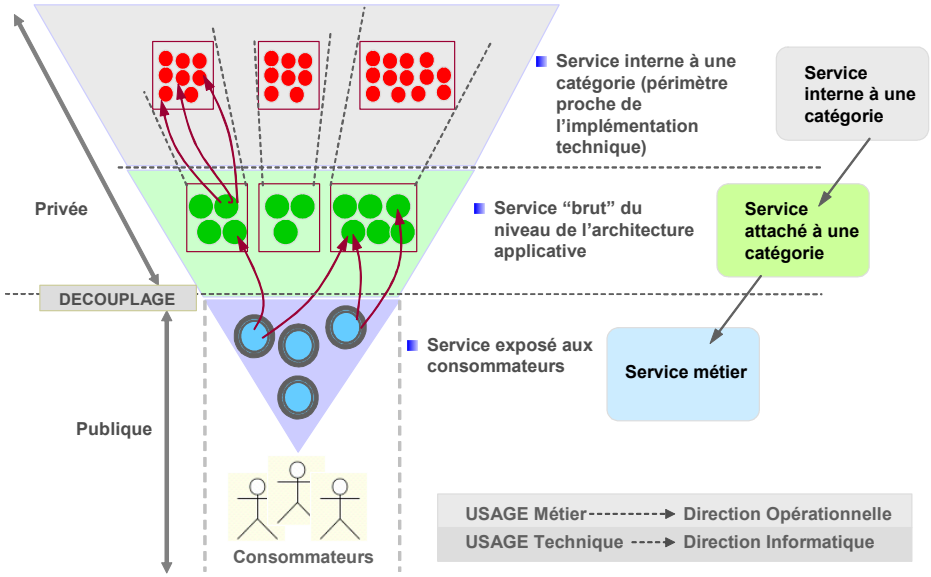
Figure 10 : Synthèse des concepts clefs



- Le **service métier** est découvert au niveau de la modélisation des processus afin d'exposer des traitements dont l'usage est soit métier ou fonctionnel[29], soit technique[30]. Un service métier est formé d'une ou de plusieurs opérations et chaque opération respecte les propriétés du concept de service. Cette approche permet d'être conforme avec la terminologie standard des Web services sans pour autant imposer leurs implémentations
- Le **service** est découvert au niveau du travail d'architecture applicative. Il est le résultat du pattern d'architecture applicative SOA Ce travail consiste à créer les Catégories[31] (objets métiers ou sujets métiers) et de décomposer les processus[32] en services rattachés à ces catégories. Le niveau de détail pour la modélisation des processus utilisé par le pattern dépend du choix d'introduction du SOA dans le cycle de construction : conceptuel, macro organisationnel ou organisationnel.
- Les concepts manipulés par le SOA (catégorie, processus, service métier, opération, phase,) sont réifiés au niveau de l'implémentation à l'aide de **composants** immédiatement exécutables dans les serveurs d'application retenus (.NET, J2EE). Les règles d'organisation de ces composants sous la forme de classes et de services techniques donne naissance à la notion de **modèle logique**.

Les figures suivantes détaillent les principes de la typologie de services et mettent en évidence le SOA métier (construction des services métiers) et le SOA de l'architecture applicative (construction des services exposés par les catégories).

Figure 11 : Typologie de service en SOA



[29] Usage métier s'il s'agit d'exposer un service vers des organisations tierces. Fonctionnel s'il s'agit d'exposer des services au sein de l'organisation du fournisseur.

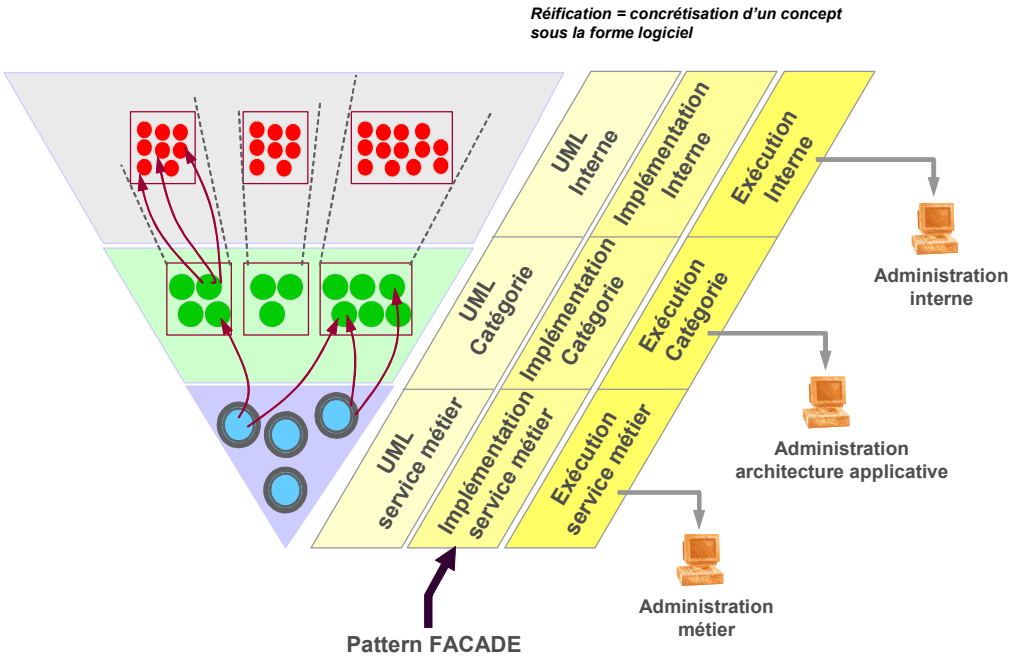
[30] Usage technique s'il s'agit d'un service utilisé pour l'interopérabilité entre des systèmes applicatifs (type EAI ou ESB).

[31] Au sens de Grady Booch, Contributeur à UML.

[32] Donc plus précisément, les opérations et les phases qui composent les processus.

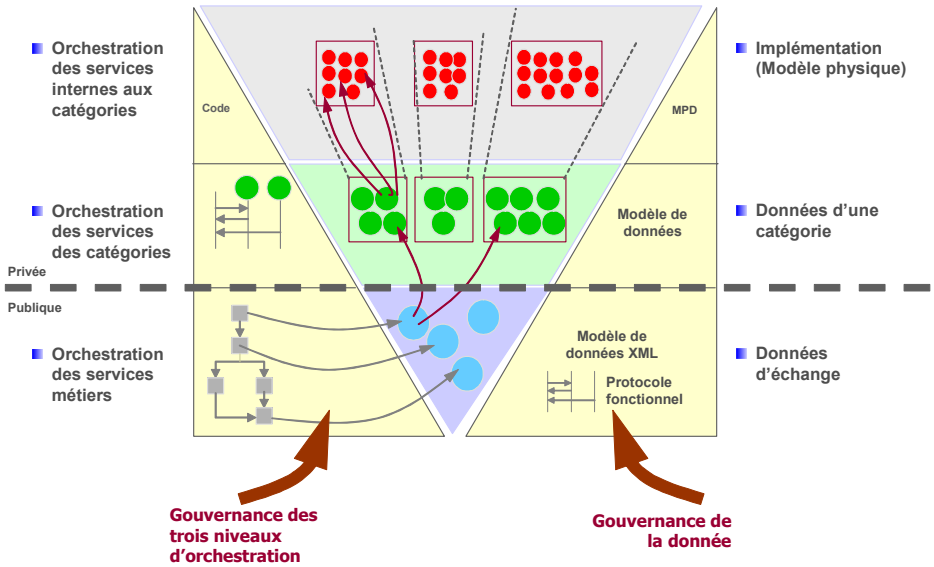
Cette figure insiste sur l'importance du principe de réification de chaque type de service de bout en bout, c'est-à-dire de la modélisation UML à son implémentation en production.

Figure 12 : Principe de réification des types de service en SOA



La figure suivante montre les deux niveaux de gouvernance, d'une part sur l'orchestration des services, et d'autre sur l'administration des données.

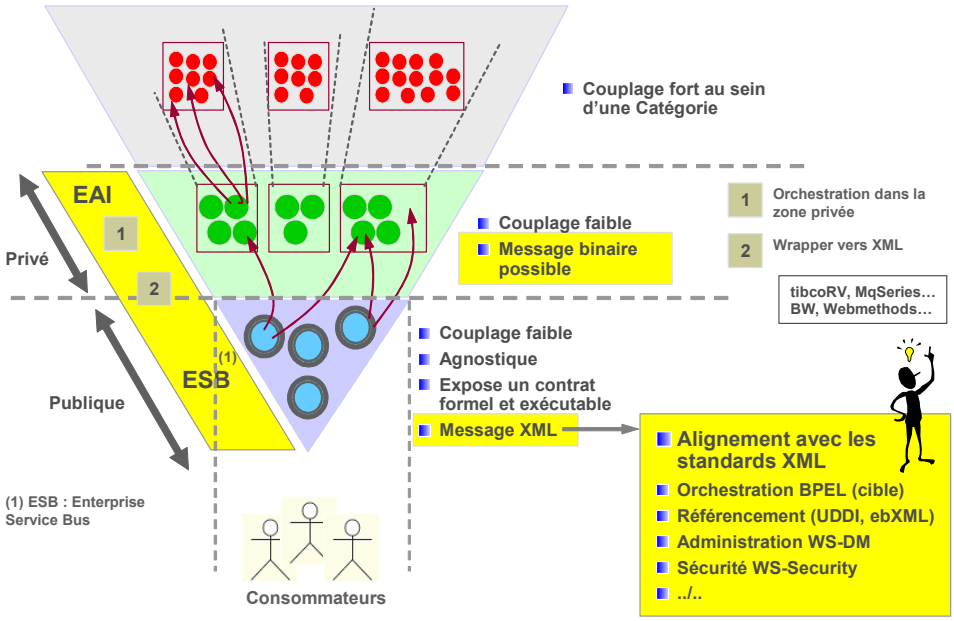
Figure 13 : Niveau de gouvernance en SOA



Pour le thème de la gouvernance de la donnée XML, se reporter aux slides du séminaire SOA et Web services.

Enfin, cette figure montre que les nouvelles technologies à base d'XML (SOAP, BPEL, WS-**) sont utilisées pour les services métiers. L'usage de ces technologies pour les services de catégories et d'implémentation ne fait généralement pas sens (pas de valeur ajoutée) et pose des difficultés de performance (à l'état de l'art de Février 2005).

Figure 14 : Type de services en SOA et technologie EAI, ESB

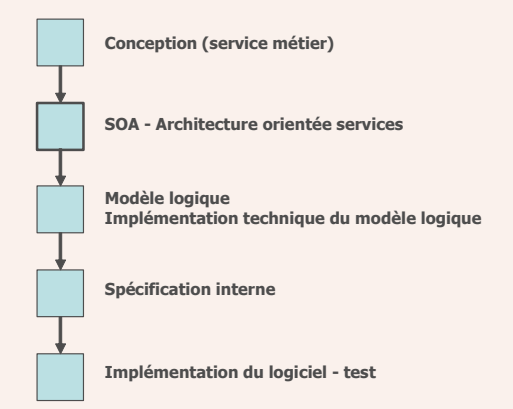


Cycle de construction simplifié SOA

La figure suivante présente le cycle simplifié pour la construction des services. Les itérations et phases de prototypage ne sont pas indiquées ni les tests et recettes. Elles ne modifient pas les objectifs de chaque étape.

L'usage du SOA est valable dès les études de cadrage qui travaillent sur un sous-ensemble représentatif du système d'information.

Figure 15 : Cycle simplifié de construction des services



Nous présentons maintenant une définition succincte de chaque étape avant de les détailler dans la suite de ce document.

Conception

Cette étape modélise les processus et les données du système. La notion de service métier est introduite dès ce niveau.

→ *Correspond au niveau du modèle organisationnel des traitements (Diagramme d'activités UML) et du modèle des données (Diagramme de classes UML).*

SOA - Architecture orientée services (niveau architecture applicative)

Les traitements issus de la modélisation sont organisés sous la forme de services selon les principes de l'architecture orientée services.

Le cas particulier des traitements d'orchestration (processus, enchaînement de services…) fait l'objet d'une description spécifique.

Modèle logique & implémentation technique du modèle logique

Cette étape construit un modèle logiciel (framework) qui supporte l'ensemble des concepts issus de la modélisation. C'est une organisation de classes, d'interfaces, de façades, de packages… qui normalisent l'organisation du logiciel.

Spécification interne

Il s'agit de la spécification détaillée des services et des traitements d'orchestration.

Implémentation du logiciel – test

L'implémentation du logiciel s'appuie sur un framework technique qui assure la prise en charge de plusieurs fonctions transversales.

Efforts pour la prise en main du SOA

Sur le plan méthodologique

SOA impacte plusieurs niveaux méthodologiques que nous détaillons dans la suite de ce document :

- L'identification des services métiers dès la phase de conception, c'est-à-dire lors de la rédaction du cahier des charges et de la modélisation des processus : pré-conditions, post-conditions, contrat d'utilisation du service (syntaxique, sémantique, qualité de service), paramétrage du service, mode dégradé…

- La cartographie du diagramme de classes sous la forme de catégories[33] qui représentent les objets métiers ou sujets métiers à partir desquelles sont construits les services au niveau de l'architecture applicative (sorte de moins variants stables dans le temps et partageables par les projets).

- La découpe des services métiers (et du processus de manière générale) sous la forme de services rattachés aux catégories. Il s'agit de la construction de l'architecture applicative.

- Dans le cas d'une implémentation orientée objet du logiciel, la découpe des services sous la forme de méthodes rattachées aux classes d'une même catégorie.

- La réification des concepts de modélisation dans un modèle logique qui prépare l'implémentation au sein des serveurs d'application J2EE et .NET.

Il s'agit aussi de définir les règles et les outils d'administration des services en vue de favoriser la réutilisation entre les projets.

Sur le plan technologique

L'implémentation du SOA met en œuvre plusieurs mécanismes techniques que nous présentons dans la suite du document :

- L'activation à distance de services (attention au sens de l'invocation en mode message et pas au sens d'objets distribués), ce qui n'est pas entièrement nouveau mais qui s'appuie aujourd'hui sur des implémentations en couplage faible avec des API XML (SOAP notamment).

- L'usage des façades pour l'implémentation des fonctions d'orchestration qui contribuent au couplage faible entre les services.

- La prise en main des frameworks techniques en vue de masquer une partie de la complexité des implémentations dans les serveurs d'applications de nouvelles générations J2EE et .NET.

[33] Au sens de Grady Booch, voir Chapitre Conception.

Apports du SOA

Apports métiers

Le SOA favorise la découverte et la spécification de services métiers au niveau de la modélisation des processus.

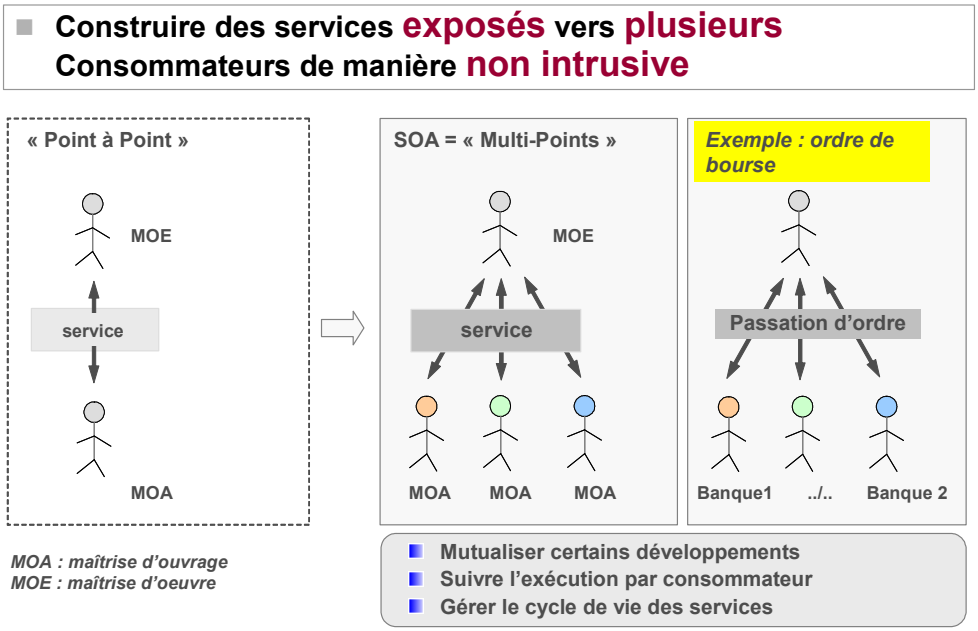
Il permet d'appréhender de manière rationnelle la notion « d'entreprise étendue »[34] :

- L'entreprise expose des services métiers à des organisations tierces.
- L'entreprise intègre dans son propre système d'information des services offerts par d'autres.

La SOA s'appuie sur des outils de management dont certaines fonctionnalités sont directement destinées aux Maîtrises d'Ouvrage, par exemple pour agir sur le paramétrage des services métiers et sur l'administration des contrats d'utilisation des services. Le niveau d'implication des MOA lors de la spécification des services métiers mais aussi en exploitation courante est un point déterminant du SOA. Voir aussi à ce sujet le site http://soa.orchestranetworks.com .

Il est très important de noter (Février 2005) que les entreprises perçoivent l'intérêt du SOA dans un contexte d'usage multi-points des services métiers. La figure suivante illustre ce mode d'usage. Dans ce contexte, il convient d'appréhender correctement la mutualisation de certains développements, la gestion des versions (favoriser la compatibilité ascendante par exemple) et suivre l'exploitation afin de ventiler les dépenses auprès des consommateurs.

Figure 16 : SOA multi-points (apports métiers)



[34] Voir aussi cadre de référence Web services.

Apports techniques

Le SOA permet :

- De rationaliser les développements orientés objets en apportant une structuration du logiciel au dessus des arbres de classes *via* les catégories[35] et les façades :

    - Permet de contrôler les risques de couplage fort entre les méthodes des classes. Plus précisément, le couplage entre les méthodes n'est possible que dans le périmètre d'une catégorie.

    - Permet d'intégrer un modèle d'architecture en couches (N-Tiers) indispensable pour la qualité du logiciel.

- De tenir compte des besoins d'intégration du legacy et d'interopérabilité entre les plateformes J2EE et .NET notamment en cas d'implémentation par les Web services.

- D'augmenter la flexibilité et la robustesse générale du logiciel par la création d'une architecture applicative dont les services se projettent de manière adaptée dans l'infrastructure technique (J2EE, .NET) et qui favorise la réutilisation. Par exemple en J2EE, il s'agit de déterminer les règles de projection des services en composants EJB, Servlet, javabeans…mais aussi en Web services (SOAP-XML).

[35] Au sens de Grady Booch, voir Chapitre Conception.

Chapitre 2 : Conception

C'est au niveau de la modélisation des processus que l'on spécifie les services métiers exposés aux consommateurs. Plus précisément, il s'agit des opérations qui composent ces services. Pour ce faire, il faut d'une part définir le périmètre fonctionnel de l'opération[36] mais aussi spécifier ses conditions de déclenchement et de restitution des résultats (pré-conditions et post-conditions).

Puisque l'opération est potentiellement utilisée par plusieurs consommateurs, il faut aussi tenir compte de ses multiples contextes d'usage qui sont souvent hétérogènes et évolutifs. L'opération doit être configurable à partir d'un ensemble de paramètres qui définit son comportement selon ses contextes d'utilisation[37].

Ce chapitre décrit les règles de conception des services métiers et de leurs opérations.

SOA impacte la modélisation des processus au niveau suivant :

- Découverte des opérations qui constituent des sous-ensembles du processus d'origine.
- Spécification des conditions de déclenchement et de restitution des opérations sous la forme de pré-conditions et de post-conditions.
- Spécification des messages d'entrée et de réponse et plus particulièrement, standardisation des types de données qui composent ces messages.
- Spécification du niveau d'adaptabilité du service par la modélisation d'un ensemble de paramètres de configuration (nommé modèle d'adaptation).

Ces impacts doivent être compris par la maîtrise d'œuvre mais aussi par la maîtrise d'ouvrage. Le travail de conception mené en SOA est complémentaire de la modélisation habituelle des processus ; il est déterminant pour l'exposition de services métiers ainsi que pour l'obtention d'une architecture applicative orientée services.

Diagramme d'activités

Cette section présente un rappel sur les diagrammes d'activités UML et précise la façon dont ils sont utilisés dans le cadre méthodologique SOA

Le diagramme d'activités représente un modèle de traitement modélisé sous la forme d'un enchaînement d'activités ; c'est un processus (ou sous-processus).

Une activité est une série de tâches qui se déroulent dans une même unité de temps (non interruptible) et réalisées par un seul acteur (activité humaine) ou un seul système (activité informatique).

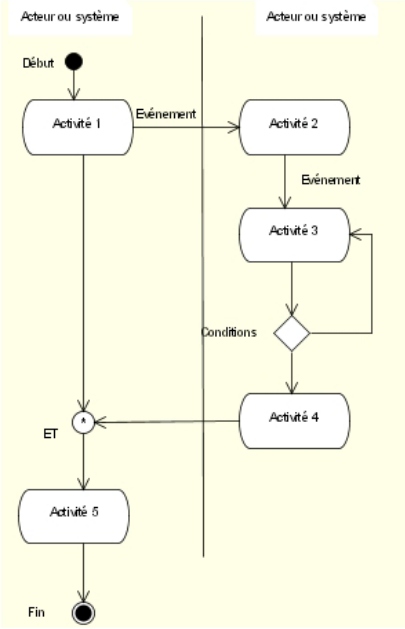
Une activité est déclenchée par un événement utilisateur ou système. En terme logiciel, l'événement déclencheur de l'activité n'appartient pas à l'activité. Il est géré par une logique applicative localisée à l'extérieure de l'activité. Dans le cas le plus simple, l'événement est une demande d'exécution immédiate.

[36] Il s'agit d'un sous-ensemble du diagramme d'activités UML (voir suite).

[37] Principe du context-aware.

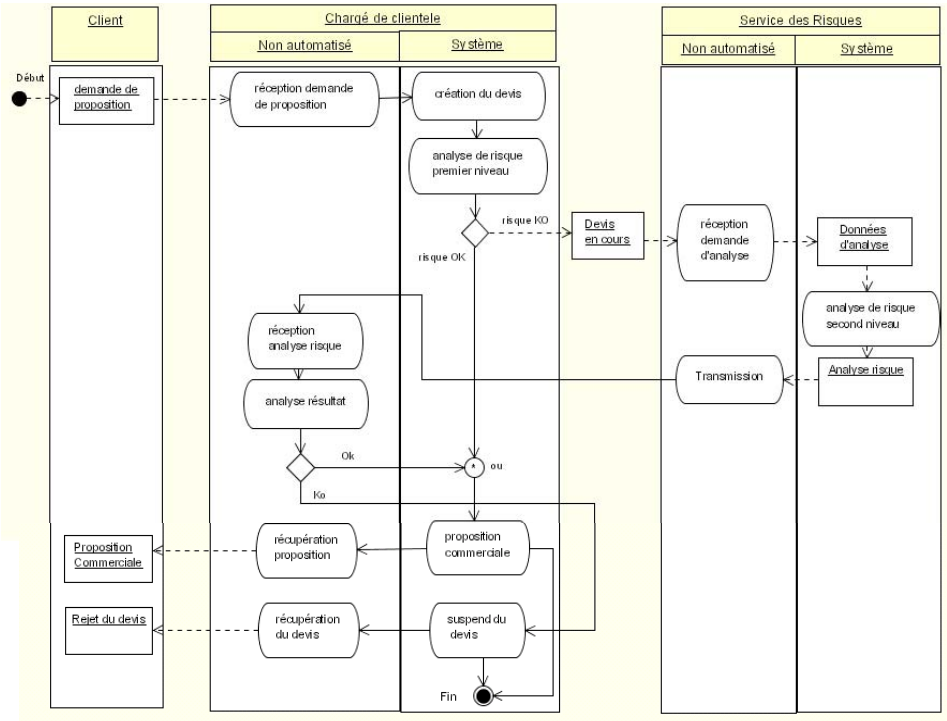
Les activités sont placées sur le diagramme selon des couloirs (ou colonnes) d'activités qui représentent chacune un acteur ou un système (*swim lane* UML)[38].

Figure 17 : Diagramme d'activités



Dans la pratique, pour un couloir d'activités de type acteur, il est souvent nécessaire de faire la différence entre les activités manuelles et celles qui sont réalisées par le système (figure suivante).

Figure 18 : Sous couloirs d'activités



[38] Ce diagramme est l'équivalent du modèle organisationnel de traitement (MOT) de Merise.

On note aussi sur cette figure que les flux échangés entre les activités peuvent être qualifiés. Ici la communication entre le système utilisé par le Chargé de clientèle et le Service des risques est un échange du document « Devis » en l'état « en cours ».

Service métier et opération

Principes de base

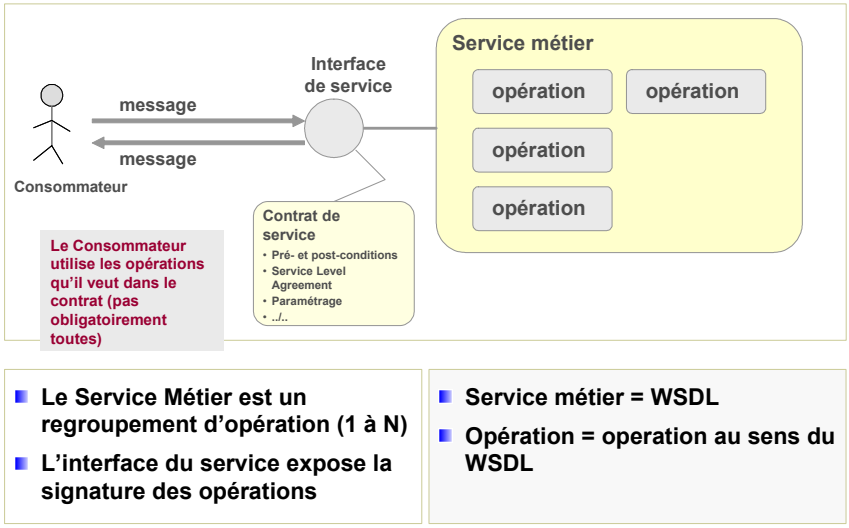
Afin de modéliser les traitements exposés par le service métier, on introduit dès la conception des processus la notion d'opération.

Comme nous l'avons déjà indiqué dans le chapitre précédent, on retient la terminologie standard des Web services qui nomme les traitements exposés au consommateur par le terme d'« opération ». Ce choix n'oblige évidemment pas à réaliser une implémentation du logiciel sous la forme de Web services. A ce niveau de la modélisation des processus, les choix techniques d'implémentation ne sont pas obligatoirement connus.

Néanmoins, compte tenu de l'impact des Web services sur la construction des services métiers, il est important de retenir un vocabulaire standard plutôt que de créer une terminologie spécifique.

Un service métier est le regroupement d'une ou plusieurs opérations qui forment une cohérence fonctionnelle (figure suivante) :

Figure 19 : Structure du service métier en opérations



Une même opération peut être exposée dans plusieurs Web services. Les règles retenues pour guides les regroupements sont les suivantes :

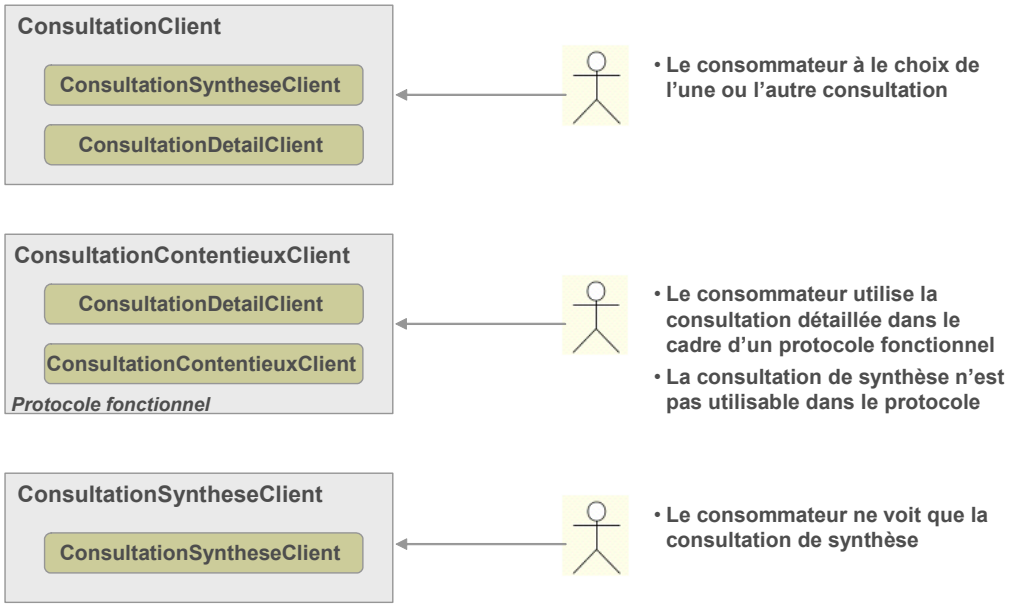
- **Publication -** Le service métier expose toutes ses opérations
- **Cohérence –** Les opérations dans un même service métier doivent faire sens fonctionnellement pour le Consommateur (ex : protocole fonctionnel -> enchaînement d'opérations)
- **Réutilisation** – Une opération peut être réutilisée dans plusieurs services métiers. Dans ce cas, faire attention aux impacts en cas de mise à jour de cette opération car plusieurs services métiers sont concernés

- **Version** – La gestion de version porte sur le service métier et pas sur l'opération : quand l'opération change tous les services métiers qui la déclare changent de versions (selon les cas en compatibilité ascendante ou pas) (*)

(*) La gestion d'un compteur de version sur chaque opération est souvent un principe trop ambitieux qui nécessiterait une coordination avec la version des services métiers. On se limite à gérer un historique des modifications sur chaque opération et on retient un seul compteur de version attaché au niveau service métier

Par exemple, on peut concevoir un service métier de consultation d'un portefeuille de valeurs mobilières qui contient une opération de consultation synthétique (cumul des lignes du portefeuille) et une autre opération de consultation détaillée par ligne qui compose le portefeuille :

Figure 20 : Service métier et opérations



Modélisation au niveau du diagramme d'activités

Les opérations ne se connaissent pas entre elles. La logique d'enchaînement entre les opérations est placée dans une fonction d'orchestration matérialisée ici par le concept de processus[39].

L'opération ne gère pas d'état. C'est-à-dire que l'exécution d'une opération ne dépend pas de son état précédent. Ceci n'empêche pas que l'opération puisse interagir avec des données persistantes qui influencent son comportement.

Une opération est un regroupement de plusieurs activités d'un processus ou d'un sous processus obligatoirement contiguës et non interruptibles. Il s'agit donc d'une série d'activités de type système.

L'opération est formée par :

- Un message d'entrée qui contient les paramètres d'appel.
- Un message de réponse qui contient les paramètres de résultats.
- Des exceptions de notification des erreurs[40].

Une opération est déclenchée par les événements de la première activité qui la compose. Elle s'exécute suivant un mode « question – réponse » synchrone[41] et son message de retour est donc toujours envoyé à l'activité initiatrice de l'évènement déclencheur de l'opération

L'opération regroupe toutes les activités qui participent à l'exécution du traitement comprises entre la première activité déclenchée et celle initiatrice de la réponse. Une même activité peut participer à la définition de plusieurs opérations.

Exemple d'une composition simple

Dans la figure ci-après, l'opération est composée d'une seule activité système (activité 2) et dispose d'un message d'entrée et de réponse.

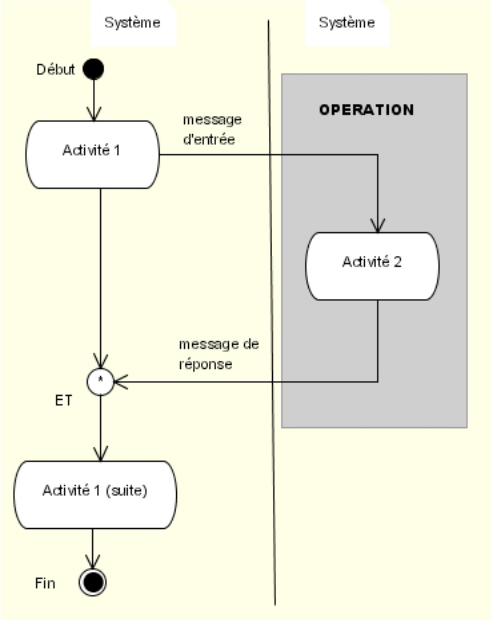
On note que l'activité 1 lance l'opération et reçoit la réponse (fonctionnement synchrone).

[39] Voir aussi annexe Fonction d'orchestration.

[40] En cas d'implémentation Web service, SOAP Fault – Voir cadre de référence Web service.

[41] Dans le cas de service métier asynchrone, il faut veiller à ce que l'implémentation du logiciel intègre un bus de type MOM (Middleware Orienté Message). Si cette implémentation s'appuie sur les Web services, il faut alors exécuter du SOAP sur JMS (Java Message Service). Aujourd'hui, ce mode d'utilisation reste marginal car il impose que les applicatifs fonctionnent tous en environnement Java et disposent d'une couche de communication JMS, ce qui est plus structurant que la simple pratique de HTTP. Cette remarque est valable pour le cas d'usage d'autres types de MOM (approche propriétaire).

Figure 21 : Découpe en opération (1)



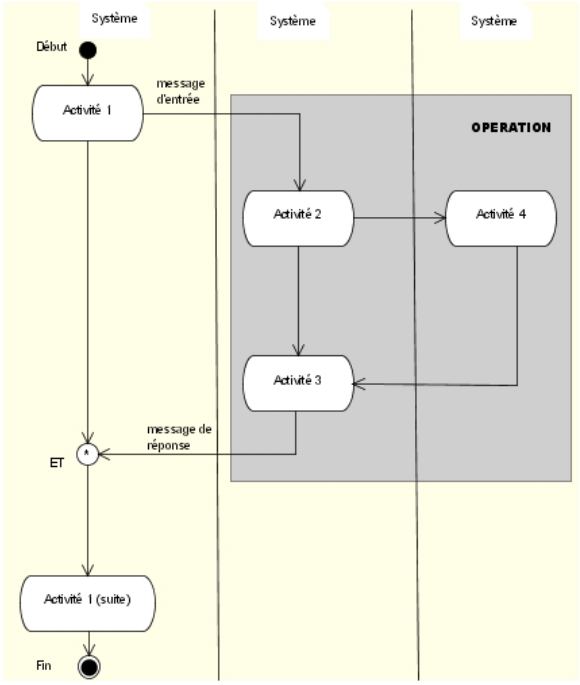
Exemple d'une composition complexe

La figure ci-après montre un cas plus complexe de composition d'une opération. Cette opération met en œuvre trois activités qui s'exécutent sur deux systèmes différents. Toutes ces activités sont contiguës et non interruptibles. L'opération dispose toujours d'un message d'entrée et de réponse.

Dans l'étape suivante d'architecture applicative, le lancement de l'activité 4 par l'activité 2 sera structuré sous la forme d'un appel de service et implémenté, si nécessaire, à l'aide d'un protocole d'activation distant.

On pourrait aussi, dès ce niveau de modélisation, identifier deux opérations : une première qui regroupe les activités 2 et 3 ; une seconde pour l'activité 4. Il s'agit d'un choix de conception qui n'est pas déterministe. Tout dépend de ce que l'on souhaite exposer sous la forme d'un service métier, y compris s'il s'agit d'un usage technique qui pourrait faire sens au niveau de l'interface applicative entre l'activité 2 et l'activité 4.

Figure 22 : Découpe en opération (2)



Précondition – postcondition - exception

Vue par le consommateur, l'utilisation de l'opération est décrite par un ensemble de conditions d'usage que l'on nomme pré-conditions[42].

Si le consommateur respecte les pré-conditions alors le fournisseur certifie que le résultat renvoyé par l'opération respectera des conditions de restitution que l'on nomme post-conditions.

PRECONDITION

Une pré-condition est une règle qui doit être vérifiée pour que l'opération soit lancée. Elle renvoie une valeur booléenne « vrai » si elle est vérifiée, « faux » dans le cas contraire.

La règle s'applique sur les valeurs des paramètres du message d'entrée et d'autres données propres au contexte de l'opération.

La somme des pré-conditions constitue un ensemble exhaustif des cas de déclenchement de l'opération.

[42] Il s'agit ici d'un élément très important du contrat d'utilisation du service (voir section suivante). Au niveau de l'implémentation du logiciel, on parle de programmation défensive pour exprimer le fait que le fournisseur du service crée une sorte de façade spécialisée dans la vérification des conditions de déclenchement de son service.

POSTCONDITION

Une postcondition est associée à une ou plusieurs pré-conditions. Elle décrit les propriétés que le résultat envoyé doit satisfaire (ex : une liste de valeurs possibles) et la liste des exceptions susceptibles d'être levées.

EXCEPTION

Une exception correspond à l'émission d'un résultat en erreur. Il peut s'agir d'erreur fonctionnelle ou technique.

Les exceptions sont levées par l'exécution des post-conditions.

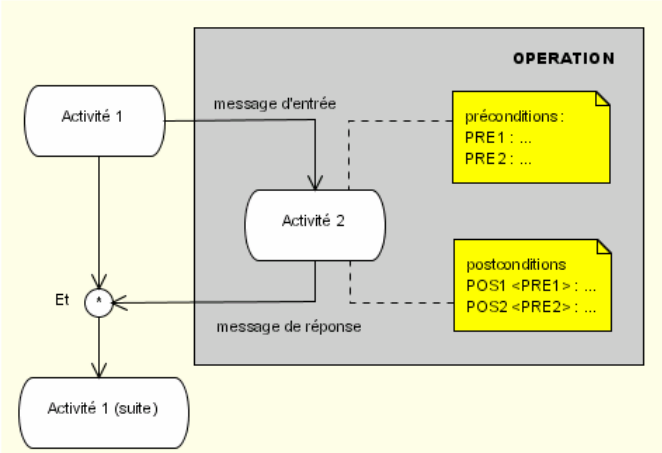
La spécification des pré-conditions, post-conditions et exceptions est un travail déterminant à mener dès la phase de conception des opérations.

MODE DE REPRESENTION ET OCL

Les pré-conditions, post-conditions et exceptions peuvent figurer, sous une forme synthétique, au niveau du diagramme d'activités *via* des notes de documentation.

Ces notes contiennent une description informelle et/ou formelle grâce au langage d'expression de contraintes OCL (Object Constraint Langage)[43].

Figure 23 : Exemple de représentation des pré et post-conditions



Elles sont ensuite rédigées de manière détaillée afin qu'elles puissent apparaître dans le contrat d'utilisation du service.

[43] Dans la pratique on utilise peu l'OCL car il n'est pas très lisible pour les maîtrises d'ouvrage. Une spécification informelle est souvent plus adaptée. Voir aussi guide Cadre de référence Web services Paragraphe sur la validation du contrat d'utilisation du service qui recense les différents langages et standards pour la représentation formelle des pré-conditions et post-condition : OCL mais aussi Schematron, Xpath, WS-Policy…

Message d'entrée et de réponse

Une opération est formée par un seul message d'entrée et un seul message de réponse.

Ces messages correspondent à un ensemble de types simples ou complexes dont la modélisation est issue d'un diagramme de classes UML ou d'une représentation en XML[44].

Puisque ces types de données sont partagés entre le fournisseur et le consommateur, il est déterminant de favoriser la standardisation aux niveaux suivants :

- **Codification** : un même type ne doit pas être codifié de manière différente d'une opération à l'autre. Le respect de ce principe peut conduire à la mise en place de meta-type qui masque des codifications hétérogènes préexistantes à la construction des opérations[45].
- **Sémantique** : une administration des données doit garantir l'homogénéisation de la terminologie.
- **Implémentation** : l'usage de XML Schéma s'impose comme standard pour la gestion des types. Si l'on utilise une représentation UML, il est conseillé de s'appuyer sur un profile XML Schéma. Cependant, la richesse importante de ce standard conduit à des risques de divergences d'interprétation technique selon les environnements utilisés par le fournisseur et les consommateurs[46]. Il faut donc définir des règles d'usage d'un sous-ensemble de XML Schéma et des bonnes pratiques qui favorisent l'interopérabilité[47].

La pertinence de la modélisation des messages et la standardisation des types de données sont des actions déterminantes pour rationaliser les développements et garantir la qualité de la relation avec les consommateurs.

[44] On peut aussi générer les documents XML à partir d'une représentation UML sous condition d'utiliser un profile UML adapté (voir aussi cadre de référence Web service, mapping UML / XML).

[45] Voir cadre de référence Web services, Meta-type.

[46] Par exemple, un type XML Schéma d'énumération de données peut être interprété différemment selon que l'implémentation soit Java, C# ou Perl.

[47] Voir cadre de référence Web services, Règles de modélisation des messages XML.

Paramétrage de l'opération

Une même opération peut être exposée à plusieurs consommateurs dont les contextes d'exécution sont potentiellement hétérogènes.

Ces contextes sont en partie connus au moment de la spécification de l'opération et évoluent dans le temps.

Pour être en mesure de gérer de manière rationnelle ces contextes il faut[48] :

- Modéliser une structure de paramètres qui permette de configurer l'opération de manière adaptée selon les contextes d'exécution. Ces paramètres concernent la présentation ergonomique de l'opération, son comportement fonctionnel, sa qualité de service…
- Construire l'opération de manière générique en tenant compte des paramètres modélisés.

Par exemple, pour une opération de consultation détaillée d'un portefeuille de bourse on peut disposer des paramètres suivants :

- Nombre de ligne maximum par consultation.
- Format des montants (décimalisation intégrée ou séparée).
- Présentation d'une marque (logo) ou pas (marque blanche).
- Présentation de clauses juridiques personnalisées.
- Heures d'ouverture du service (heures ouvrées, 24/24…).

L'effort de spécification mené pour décrire la structure de paramètres de l'opération renforce la couverture fonctionnelle exprimée ainsi que son exhaustivité. En effet, la démarche d'anticipation de l'usage de l'opération dans des contextes d'exécution hétérogènes, et en partie existants au moment de la spécification, permet d'atteindre un bon niveau de généricité.

Il s'agit ici d'un enjeu de flexibilité et de maîtrise des coûts de possession des opérations. Plus le paramétrage est pertinent, moins les actions de maintenance évolutives seront nécessaires[49].

[48] Cette préoccupation de gestion de contexte est aussi parfois nommée « Context-Aware ».

[49] Voir annexe Granularité des services et dans le cadre de référence Web services, Paramétrage des Web services.

Contrat d'utilisation

Le contrat d'utilisation du service métier, et plus précisément de ses opérations, est un document rédigé par le fournisseur et remis au consommateur[50].

Ce document exprime les engagements du fournisseur et les devoirs du consommateur pour l'utilisation de chaque opération. Une première version est rédigée au moment de la modélisation des opérations, puis il est complété dans les phases suivantes du cycle de construction[51].

Un exemple de plan-type détaillé de contrat d'utilisation de service métier est présenté en annexe. On y trouve notamment les clauses suivantes :

- Objectifs et principes de fonctionnement de l'opération.
- Description du message d'entrée et du message de réponse.
- Description des pré-conditions et post-conditions.
- Description des exceptions.
- Règles générales de fonctionnement.
- Contacts pour le support.
-

Correspondance avec les Web services

Si le service métier est implémenté en Web service, alors ses opérations correspondent directement à la notion d'opération des Web services.

Si le service métier n'expose qu'une seule opération alors le Web service correspondant n'expose également qu'une seule opération.

[50] Y compris dans le cas d'une implémentation en Web service puisque le document WSDL de description des opérations n'est pas suffisant pour exprimer l'ensemble des clauses du contrat comme par exemple les préconditions et postconditions.

[51] Notamment pour l'ajout des propriétés techniques liées aux choix d'implémentation et de déploiement. Il s'agit par exemple de la notion de Binding qui permet de fixer le protocole de communication (SOAP sur HTTP...), les mécanismes d'encodage technique des données…

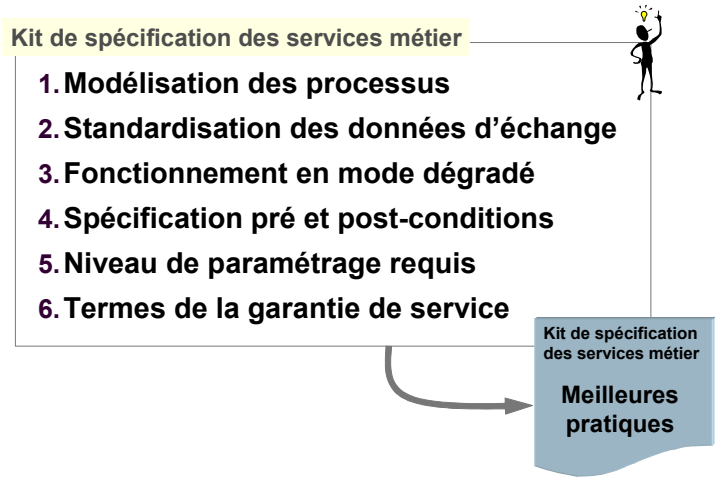
Synthèse

Le concept d'opération respecte les propriétés suivantes :

- Couplage faible, c'est-à-dire qu'une opération n'appelle pas directement une autre opération[52].
- Se décrit par l'intermédiaire d'un contrat d'utilisation qui est remis au consommateur.
- Dispose d'une structure de données de paramétrage qui permet de gérer les contextes hétérogènes d'exécution.

Dans la pratique, le service métier se compose d'une à plusieurs opérations. On identifie un kit de spécification du service métier autour de 6 points de vigilance rappelés dans la figure ci-dessous.

Figure 24 : Le kit de spécification du service métier



[52] C'est le processus qui s'en charge.

Phase

Lors de la modélisation du processus, les activités qui ne participent pas à la conception d'une opération sont regroupées sous la forme de phases.

Si le processus est entièrement structuré sous la forme d'une ou de plusieurs opérations alors la notion de phase n'intervient pas. Dans ce cas, cela signifie que tous les traitements du processus sont exposés à des organisations clientes. Ce mode d'action est possible sur certains processus mais pas au niveau de l'ensemble des processus de l'entreprise.

Une phase est un regroupement d'activités qui ne participent pas aux opérations, qui sont contiguës et qui forment une étape cohérente du processus modélisé. En distinguant la notion de phase on favorise aussi une bonne structuration du diagramme d'activités qui est utile pour la maîtrise de la complexité.

Selon les objectifs que l'on se fixe lors de la modélisation, il est envisageable de concevoir les phases comme de possibles futurs services métiers, c'est-à-dire des opérations potentielles. Il suffit pour cela de veiller aux respects des propriétés de la notion d'opération et notamment l'expression des pré-conditions et post-conditions.

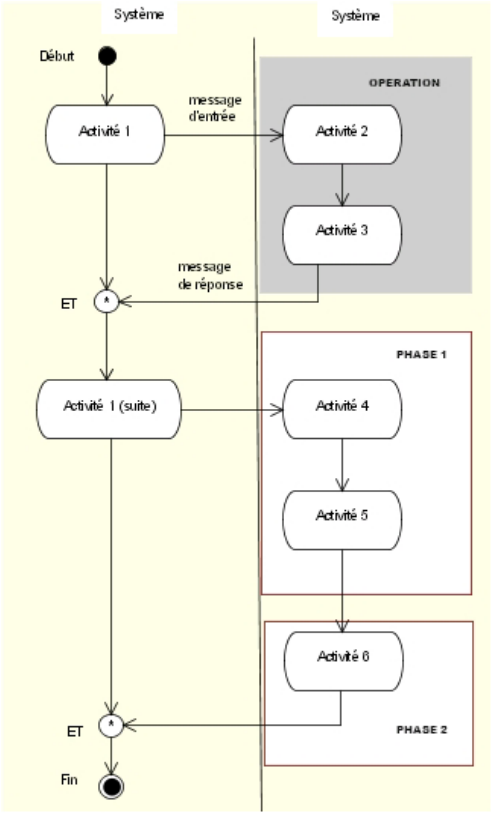
La phase est alors un traitement qui pourra être exposé ultérieurement sous la forme d'un service métier.

En final, un diagramme d'activités est composé de regroupements d'activités sous la forme d'opérations et/ou d'au moins une phase pour les activités qui ne participent pas aux opérations. L'association des concepts d'opération et de phase assure la complétude sur le diagramme d'activités : il n'y a pas d'activité qui ne soit pas associées à une opération ou une phase.

Une activité appartient à une seule phase ; il n'y a pas de chevauchement entre les phases.

La figure suivante montre un exemple où l'on modélise deux phases pour les activités 4 à 6. Ces activités ne sont donc pas exposées sous la forme d'opération.

Figure 25 : Découpe en phase



Dans certains cas, le concept de phase représente les états transactionnels complexes. Dans notre exemple, on peut considérer que la phase 1 forme un état transactionnel global pour les activités 4 et 5 qui la compose.

Type Interactif et données

L'opération et la phase exposent de manière exclusive soit un flux interactif (IHM), soit un flux de données.

Il est important de distinguer ces deux types afin que les choix d'implémentation puissent se faire correctement dans les étapes suivantes du cycle de construction.

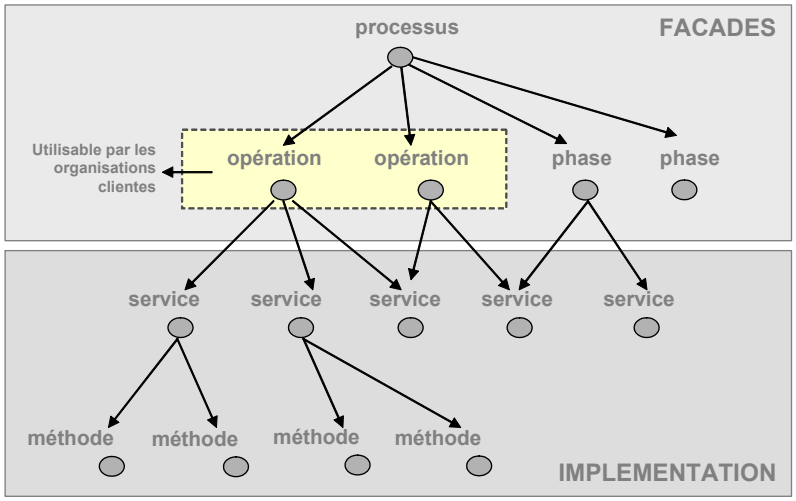
Par exemple, le choix d'implémenter une opération IHM sous la forme d'un Web service conduit à l'usage du standard WSRP[53] (Web service remote portlet).

[53] WSRP permet de concevoir la couche IHM des applicatifs grâce à XML. L'objectif est aussi d'orchestrer, à partir de cette description, des appels SOAP vers les opérations des Web services. WSRP respectent les principes techniques du MVC : le contrôleur exprime la logique d'enchaînement des écrans grâce à XML Schema ; les vues sont formées par XSL ; et le modèle fait référence aux opérations des Web services.

Processus

Un processus est représenté par un diagramme d'activités qui modélise un traitement de bout en bout pour une préoccupation de gestion (*self-contained*). Il enchaîne le déclenchement d'opérations et de phases. Il assure un rôle d'orchestrateur. Par défaut, il n'est pas exposé aux organisations clientes.

Figure 26 : Orchestration d'opération par le processus

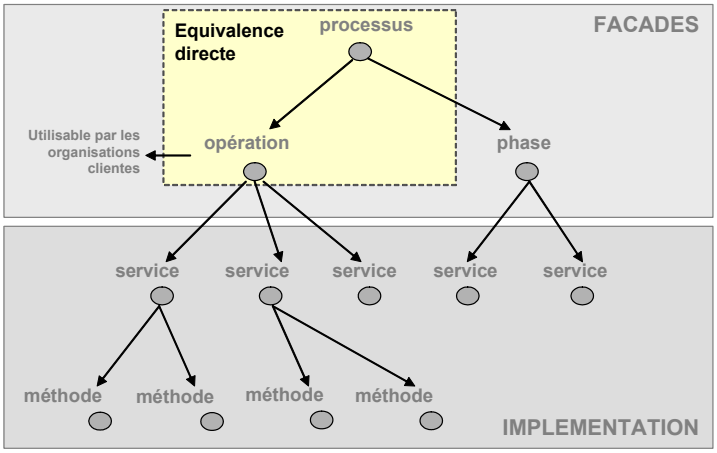


Dans le cas où l'on souhaite exposer le processus aux organisations clientes, il faut distinguer les trois scénarios décrits ci-après.

Scénario 1 : Bijection entre processus et opération

C'est le cas le plus simple. Il consiste à créer une bijection entre le processus modélisé et une seule opération. En d'autres termes, le processus décrit la logique d'une unique opération avec en plus l'enchaînement possible de phases. Ce choix est restrictif car il interdit l'orchestration d'opérations mais il permet de respecter la propriété de couplage faible : un service métier (opération) n'appelle pas un autre service métier.

Figure 27 : Bijection entre le processus et l'opération

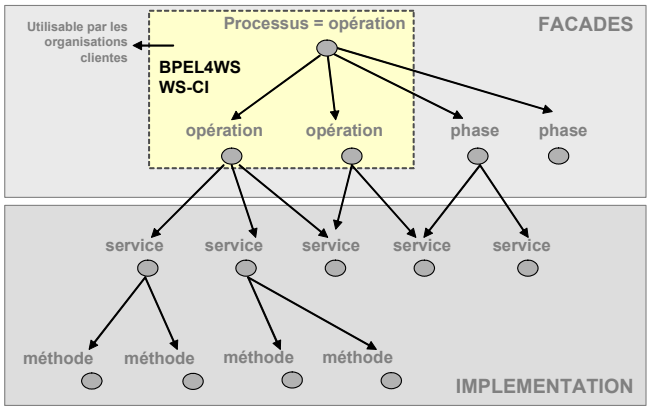


Scénario 2 : Utilisation d'un standard d'orchestration des opérations

C'est le cas plus complexe. On souhaite orchestrer l'appel à des opérations au niveau même d'une opération qui représente la logique du processus. Plus précisément, le processus devient lui-même une opération qui en orchestre d'autres. Ici on rend caduque la propriété de couplage faible entre opérations.[54].

La façon la plus pertinente pour implémenter ce type d'approche consiste à utiliser un standard de chorégraphie ou d'orchestration de Web services tels que WS-Choreography[55] ou BPEL4WS[56]. Compte tenu de la complexité et de l'état de maturité de ces standards, leurs usages restent encore expérimentaux (*début 2004*).

Figure 28 : Utilisation d'un standard d'orchestration des opérations



Scénario 3 : Court-circuit des façades des opérations

De manière générale, il est important de noter que le fait qu'une opération ne puisse pas en déclencher directement une autre n'empêche pas qu'une opération puisse réutiliser la logique applicative d'une autre opération.

En effet, sur le plan de l'implémentation, le respect du pattern d'architecture SOA conduit à ce que l'opération ne soit qu'une façade d'appel vers des services rattachés à des catégories UML et donc réutilisables entre les opérations[57]. Dans ce cas, on peut toujours revenir au premier scénario, c'est-à-dire considérer l'existence d'une bijection entre le processus modéliser et l'opération puis gérer l'orchestration en « court-circuitant » les façades des opérations que l'on doit enchaîner[58], ce qui revient à faire des appels directs aux services des catégories (en rouge sur la figure ci-dessous).

---

[54] S'il s'agit d'une implémentation en Web service SOAP, cela revient à déclencher à partir d'une opération SOAP une ou plusieurs autres opérations SOAP, ce qui pose des problèmes de maintien de contexte et de gestion transactionnelle (voir Annexe Fonction d'orchestration).

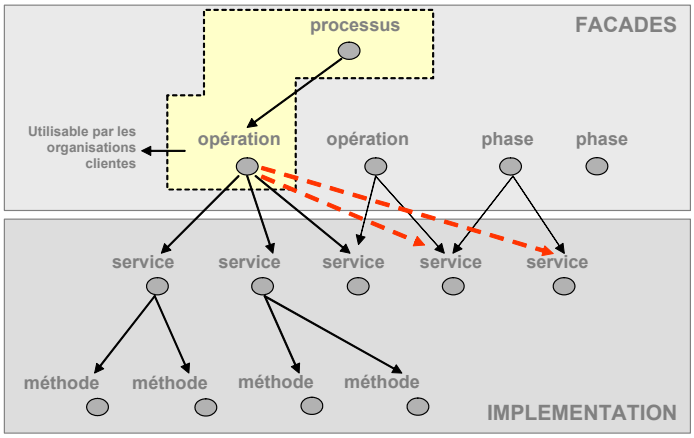
[55] Spécifié par le W3C. WS-Choreography permet de modéliser les interactions entre plusieurs opérations de Web services. Cette modélisation se place du point de vue « broker » c'est-à-dire au centre des échanges entre les opérations.

[56] Spécifié par l'OASIS. BPEL4WS (Busines Process Execution Langage for Web services) permet de modéliser les interactions entre plusieurs Web services du point de vue d'un participant.

[57] Voir suite du document.

[58] C'est alors un travail d'intégration à mener. Il ne s'agit plus, comme avec WS-Choreography ou BPEL4WS de « simplement » modéliser les enchaînements grâce à l'utilisation des interfaces WSDL.

Figure 29 : Court-circuit des façades des opérations



(Février 2005) : la typologie des processus est précisé selon la classification suivante :

- Processus d'orchestration entre Consommateur et Fournisseur (nommé aussi Chorégraphie)
- Processus d'orchestration privé au Consommateur et privé au Fournisseur
- Processus interne à un Service Métier

Les figures suivantes illustrent cette typologie.

Figure 30 : Processus de chorégraphie

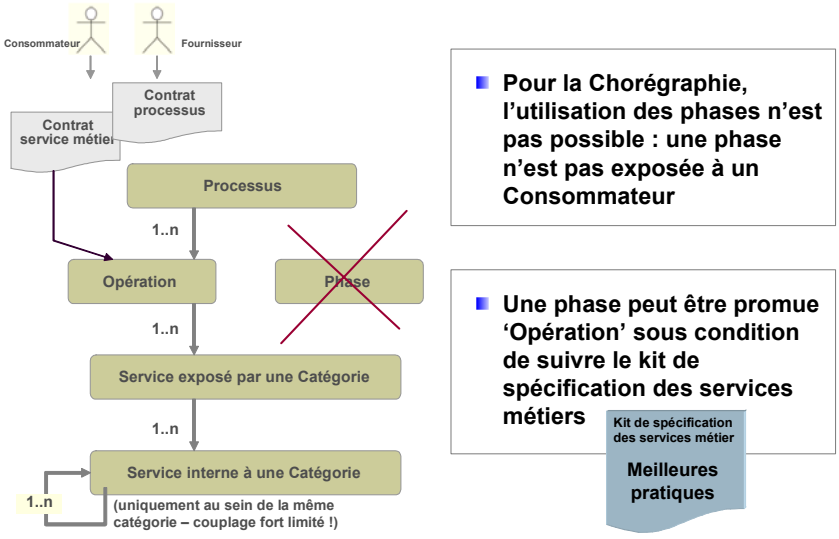
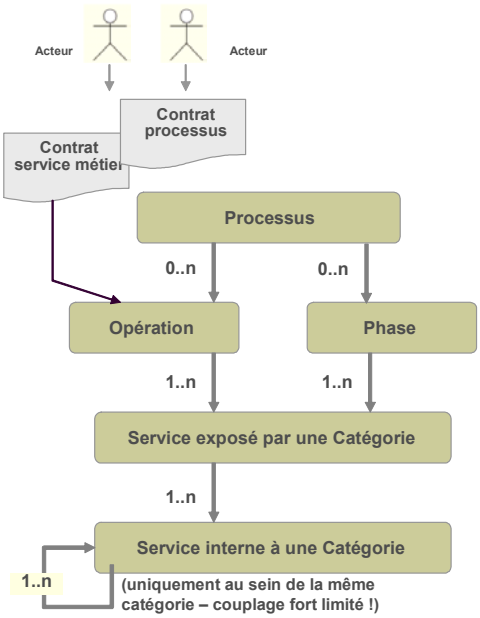


Figure 31 : Processus d'orchestration



- **L'utilisation de la phase est possible uniquement dans un périmètre organisationnel fixé (ex : à l'intérieur d'un domaine fonctionnel – bloc d'urbanisation)**
- **Au-delà de ce périmètre il faut obligatoirement passer par un service métier**

- **Une phase peut être orchestrée de manière standard (*via* BPEL) sous condition qu'elle soit décrite par un WSDL**
- **Rappel : WSDL ne signifie pas SOAP ni le passage par le kit de spécification des services métiers !**

Principes de modélisation des données

Diagramme de classes et catégories

Comme nous le détaillons dans le chapitre suivant, la démarche SOA conduit à la modélisation d'une cartographie des données sous la forme de Catégories, concept identifié à l'origine par Grady Booch contributeur à UML (*Class Category*). Chaque catégorie représente un objet métier (ou sujet métier) à partir de laquelle on construit les services.

Quand les catégories font l'objet d'une administration de données, il est alors possible de les réutiliser puis de les enrichir au moment de la modélisation des diagrammes de classes[59].

Messages XML

Les opérations des services métiers échangent des messages dont le format d'implémentation est le plus souvent XML. Il faut en tenir compte au moment de la conception notamment en mettant en place un profile UML adapté à la dérivation XML Schéma.

Nous présentons dans le guide Cadre de référence Web services, un paragraphe complet qui traite des meilleures pratiques pour la modélisation des messages en XML et plus particulièrement en XML Schéma.

[59] Voir annexe administration et méta modèle SOA.

Chapitre 3 : Architecture SOA

Principes

La démarche d'architecture orientée services décompose les traitements[60] sous la forme de services rattachés à des unités cartographiques du modèle de données. Ces unités cartographiques correspondent au concept de **Catégorie**[61].

Chaque catégorie représente un objet métier ou sujet métier (les termes sont équivalents).

Cette approche forme le pattern d'architecture SOA qui est une propriété que le concept de service doit obligatoirement respecter.

Le pattern d'architecture SOA définit un mécanisme de décomposition successif des processus sous la forme de services rattachés aux catégories.

Le concept de Catégorie est introduit par Grady Booch, contributeur à UML.

Catégorie

Définition

On regroupe les classes du diagramme de classes selon les grands objets de gestion du système d'information. Ce regroupement forme la notion de catégorie, que l'on nomme aussi objet métier ou sujet métier, selon le choix de vocabulaire retenu par l'entreprise.

Chaque catégorie respecte obligatoirement les cinq propriétés indiquées dans le tableau suivant.

Tableau 3 : Propriétés de la catégorie

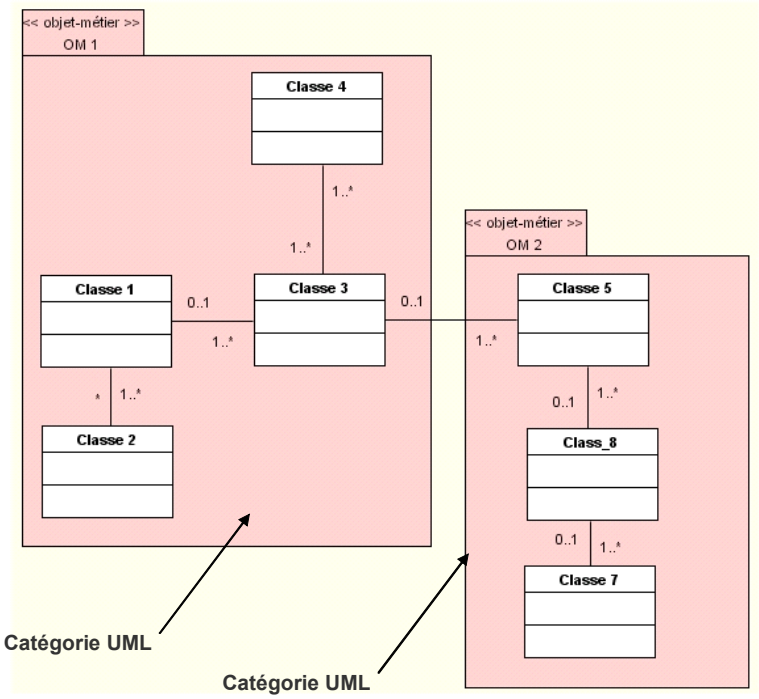
Stable	La catégorie n'est pas circonstancielle à un projet particulier. Sa frontière ne varie pas en fonction des évolutions des traitements.
Consistante	La catégorie forme un réel poids sémantique. Elle est composée d'une classe pivot qui représente son concept métier et des classes dérivées qui la décrit.
Mono préoccupation	La catégorie est singulière. Elle ne contient que les classes qui décrivent son seul et unique concept métier.
Contiguë	Les classes qui composent la catégorie sont toutes en relation entre elles. Il n'y a pas de classes isolées.
Nommé	Chaque catégorie dispose d'un nom unique la définissant.

[60] Modélisé par les processus.

[61] Au sens de la définition de Grady Booch. Les termes Catégorie et Catégorie UML sont équivalents. Le concept de Catégorie est aussi utilisable lorsque l'on met en œuvre Merise.

On représente la catégorie sous la forme d'un package UML avec un stéréotype indiquant soit objet-métier, soit sujet-métier ou catégorie selon le choix de vocabulaire retenu.

Figure 32: Cartographie en catégories



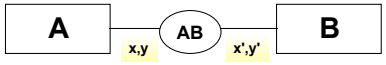
Règles de découpe du diagramme de classes

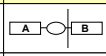
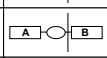
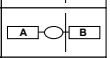
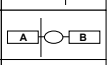
Ce paragraphe recense les règles de bonnes pratiques pour déterminer le périmètre des catégories :

- **Classe maîtresse** : analyse du diagramme de classes afin de déterminer la classe maîtresse[62] de chaque catégorie. On considère qu'une classe est maîtresse s'il est possible d'agréger autour d'elle un ensemble d'autres classes qui se trouvent structurellement sous son contrôle. En d'autres termes, on recherche le niveau de *stress* (effet de propagation) engendré par une évolution de la classe maîtresse vis-à-vis de ses congénères. Toutes les classes impactées par une évolution de la classe maîtresse participent à l'agrégat de données.
- **Règles de composition et de complétude** : les catégories ne doivent pas se chevaucher entre elles. Chaque classe doit appartenir de manière exclusive à une seule et unique catégorie.
- **Règle d'autonomie** : l'existence d'une occurrence d'une catégorie est, autant que faire se peut, indépendante des autres catégories. Le tableau ci-dessous donne les règles de découpe en fonction des cardinalités entre les classes :

[62] Nommée aussi classe pivot.

Tableau 4 : Cardinalités et catégories



x,y \ x',y'	0,1	1,1	0,n	1,n
0,1	coupure possible des 2 côtés	A=B		
1,1		A=B		
0,n			coupure possible des 2 côtés	
1,n				A=B

A=B

D'après les cardinalités, A et B devraient appartenir à la même catégorie.

On peut couper du côté de B; c'est à dire que A et l'association appartiennent à une même catégorie.

On peut couper du côté de A

- Cas « A=B » : l'analyse des cardinalités incite à réunir A et B dans une même catégorie. L'analyse des cardinalités ne donne aucune indication sur la façon de découper s'il y a lieu, mais invite à vérifier que A et B appartiennent au même agrégat.
- Cas (0,1; 1,n) ou (1,1; 1,n) : la coupure du côté B est possible mais il faut vérifier si elle est judicieuse, car une occurrence de A n'est pas concevable sans une occurrence de B. Couper à cet endroit peut induire un fort taux de sollicitation entre les deux catégories.
- Dans le cas des ternaires ont applique les critères de cardinalités des liaisons 2 à 2.

- **Règle de continuité** : une catégorie est composée d'un seul et unique agrégat. Il n'est pas possible d'avoir des classes ou groupes de classes non rattachées à l'agrégat de données[63].
- **Règle de consistance** : l'envergure de la catégorie en terme de contenu doit être pertinent. Par exemple, il ne serait pas judicieux de définir une catégorie composée d'une seule règle de gestion et d'un agrégat de données limité à une unique classe.
- **Règle de durabilité** : la catégorie est dotée d'une durée de vie plus grande que celle d'un projet. Elle peut et doit être réutilisée et enrichie par les nouveaux projets.
- **Règle de granularité** : en pratique, on constate que la granularité d'une catégorie est de moins d'une trentaine de classes (Grady Booch conseille moins d'une douzaine de classes…).
- **On ne fait pas de catégorie « sous-type » visant à prendre en compte des besoins de factorisation**. La factorisation reste interne à une catégorie.

[63] Si nécessaire, il faut adapter le modèle de données pour respecter cette propriété.

Dans le cas de la modélisation de classes techniques (framework technique), la démarche de cartographie est identique. La catégorie ne représente plus une préoccupation métier mais une préoccupation technique propre au framework.

Apports des catégories

La cartographie du diagramme de classes en catégories forme l'ossature du pattern d'architecture SOA. Cette démarche est déterminante à plusieurs niveaux :

- Permet la décomposition des processus sous la forme de services rattachés aux catégories[64].
- Favorise la maîtrise de la complexité par une organisation du système autour des éléments les moins instables représentés par les regroupements de classes. Cette organisation s'opère à un niveau supérieur au grain élémentaire des classes. C'est un facteur essentiel pour la maîtrise des développements orientés objets.
- Autorise l'organisation des équipes de conception et de développement par catégorie afin de contrôler les interactions entre des intervenants multiples.

Lien avec l'approche objet

Nous venons de voir qu'une catégorie est un regroupement de données autour d'une préoccupation stable du système d'information. C'est donc la définition d'une sorte de « macro objet » dont le périmètre est plus important qu'une classe prise isolément.

Il est possible de concevoir une catégorie à partir d'un modèle de données qui n'est pas orienté objets comme par exemple un MCD. Dans ce cas, la catégorie est formée par un ensemble d'entités et d'association.

Comme nous le verrons dans la suite, chaque catégorie est associée à une façade qui contient l'ensemble des services que cette catégorie expose. Ici également, cette façade n'est pas obligatoirement construite selon une classe au sens de l'implémentation objet. Il peut s'agir d'un module qui contient une procédure par service exposé. Dans tous les cas, on comprend que la façade assure un mécanisme d'encapsulation des données.

Exemple

Dans cet exemple, on montre un diagramme de classes qui est cartographié à l'aide de six catégories.

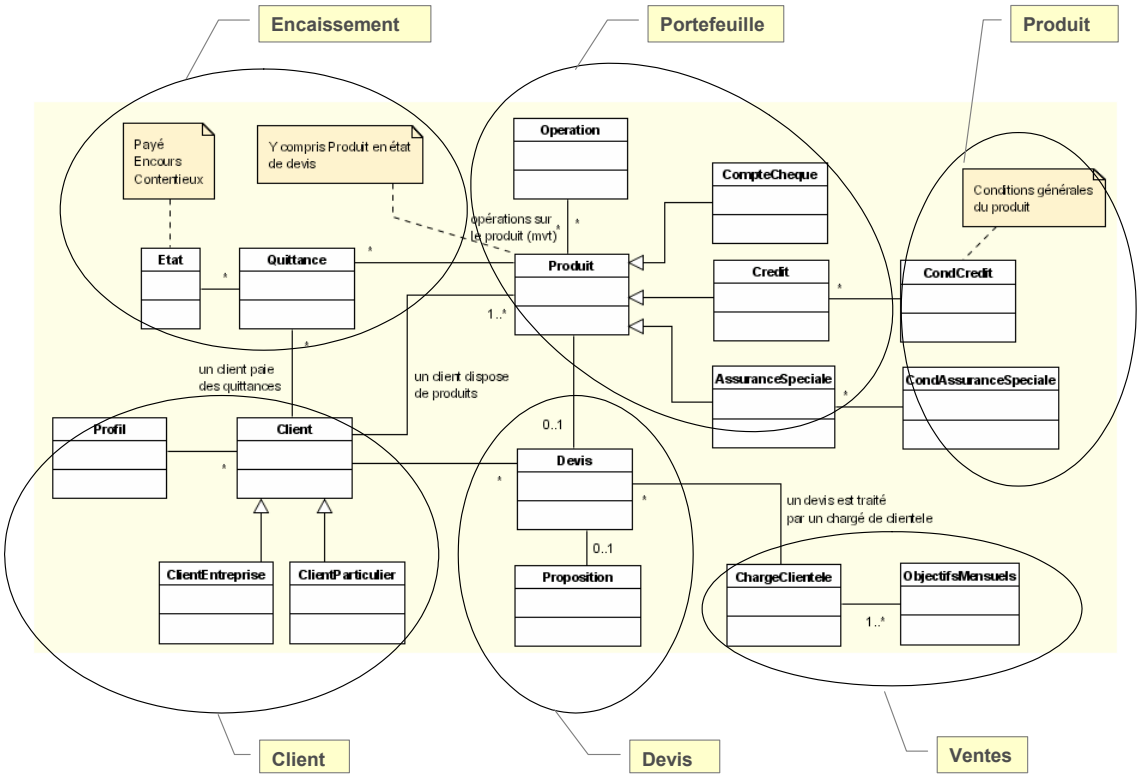
Ces catégories respectent les propriétés que nous avons indiqué précédemment : stable, consistante, mono préoccupation, contiguë et nommé.

Le choix de la création des catégories est en partie empirique. Par exemple, le regroupement des concepts de devis et de proposition commerciale pourrait être discuté.

En final, l'important est de promouvoir les catégories d'un projet à l'autre même si elles présentent certaines imperfections selon les points de vue.

[64] Voir paragraphe suivant pour les règles détaillées de cette décomposition.

Figure 33 : Illustration d'une cartographie en catégories



Service et catégorie

Un service est un traitement qui appartient sémantiquement à une et une seule catégorie. Par ailleurs, il est physiquement rattaché à cette catégorie *via* un package.

Le concept de service utilisé ici correspond à la typologie des services au niveau de l'architecture applicative, c'est-à-dire ceux qui se situent 'au dessous' des services métiers : voir le Chapitre sur la définition détaillée du concept de service.

Chaque catégorie correspond à un package au niveau UML, puis dans le langage d'implémentation (par exemple un package java).

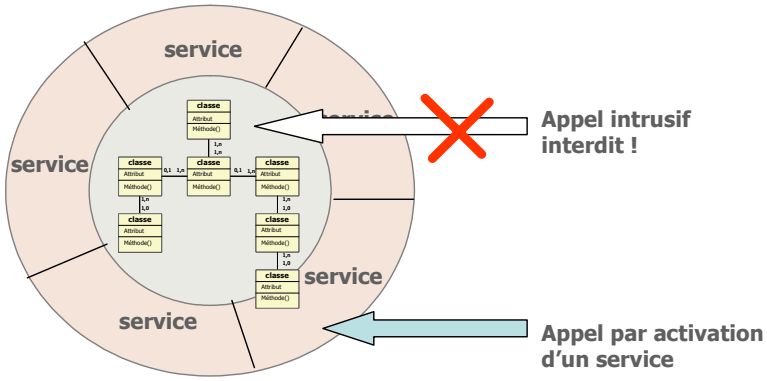
Par définition[65], le service respecte les principes du couplage faible avec les autres services :

- Il n'appelle pas directement d'autres services. Comme nous le précisons plus loin, les enchaînements entre les services sont centralisés dans une fonction spécialisée d'orchestration.
- Il est sans état persistant. Son exécution ne dépend pas d'un état antérieur, ce qui n'empêche pas le service d'interagir avec des données persistantes qui peuvent influencer son comportement.

La catégorie est donc composée de classes encapsulées par ses services. En d'autres termes, les services forment une façade d'accès aux classes de la catégorie et assurent la propriété d'encapsulation (figure suivante).

[65] Voir chapitre Concepts de base.

Figure 34 : Catégorie et principes d'encapsulation



L'identification d'un service et son rattachement à une catégorie se réalise suivant le principe du respect de la proximité sémantique. Il s'agit d'un travail de découverte et de classification.

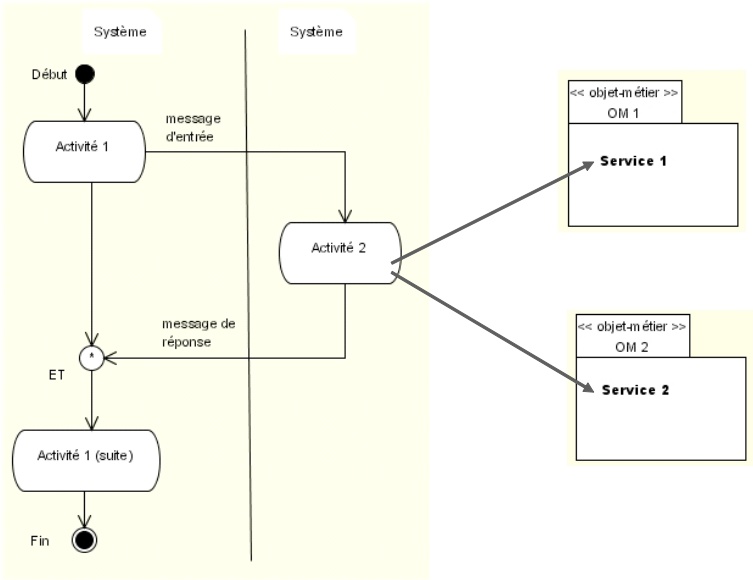
Décomposition de l'activité en services

Principes de base

La logique de l'activité (au sens du diagramme d'activités) est décomposée sous la forme de services rattachés aux catégories. C'est le même principe dans le cas d'une modélisation en MOT Merise.

Par exemple, la figure suivante montre que l'activité 2 est éclatée en deux services respectivement placés dans les catégories nommées objet métier OM 1 et objet métier OM 2.

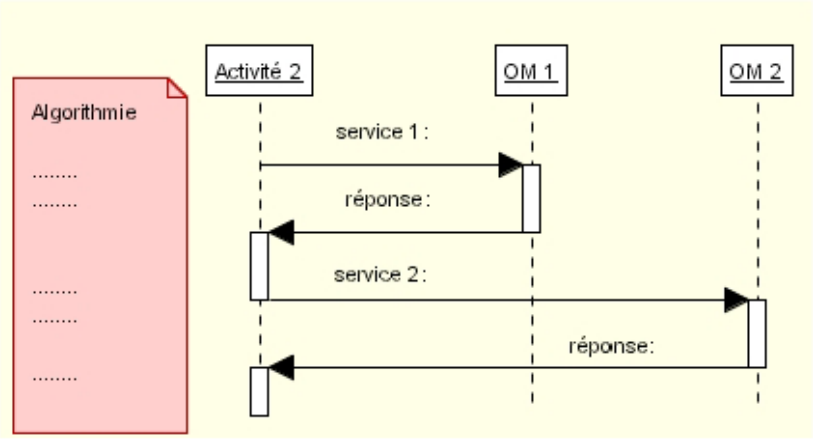
Figure 35 : Décomposition de l'activité en services



La décomposition en services autour des catégories n'impose pas l'usage d'une programmation objet. A ce stade, le service est rattaché sémantiquement à un groupe de classes mais pas à une classe en particulier. On peut donc pratiquer cette décomposition dans des contextes de programmation structurée non objet. Dans ce cas, l'encapsulation des classes de la catégorie par ses services est implémentée de

manière spécifique selon le langage de programmation utilisé[66]. Le modèle de représentation de la décomposition est un diagramme de séquence UML (figure suivante).

Figure 36 : Diagramme de séquence – Décomposition de l'activité en services



Opération et phase

La décomposition des activités se fait en réalité sur le périmètre des concepts d'opération et de phase. La première branche du diagramme de séquence est donc une opération ou une phase.

Cas des jointures

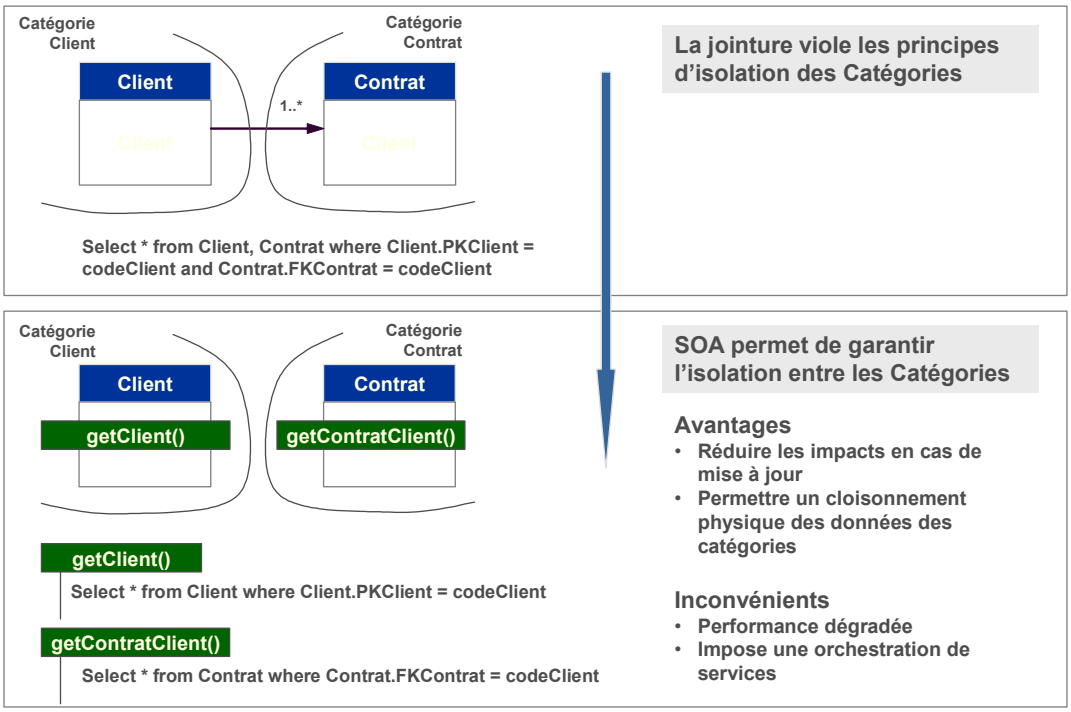
Dans la pratique, la création de jointure pour l'accès à des données localisées logiquement dans plusieurs catégories nécessite :

- Soit d'autoriser ces services à accéder directement en consultation aux données d'autres catégories (violation du principe d'encapsulation) afin d'exécuter la jointure.
- Soit de créer une vue (au sens SQL) et de localiser cette vue dans une catégorie qui en devient la propriétaire.

Une approche complémentaire consiste à « simuler sur le plan applicatif » la jointure à partir d'un service (voir figure ci-après).

[66] Il existe de telle implémentation par exemple en langage Cobol.

Figure 37 : Jointures et Catégorie



Taxonomie

Les processus, les opérations et les phases ne suivent pas la classification par catégories retenue pour les services.

Une taxonomie spécifique est généralement mise en œuvre. Néanmoins, elle peut suivre celle amenée par les catégories. Par exemple, on peut décider de placer chaque processus sous la responsabilité d'une catégorie UML.

Exemple 1

Dans cet exemple on montre les niveaux de décomposition d'un processus. Les points à noter sont les suivants :

- Le processus est décomposé en deux phases et une opération.
- L'opération est représentée sous la forme d'une branche.
- Pour simplifier le modèle, les deux phases ne sont pas représentées par une branche. Dans la pratique il serait plus conforme de prévoir une branche par phase.
- L'opération et les phases se décomposent en service autour des catégories formées par les six objets métiers client, encaissement, portefeuille, produit et devis.

Cet exemple concerne le cas de la souscription d'un produit d'assurance. Lors du processus de souscription deux niveaux d'analyse de risque sont réalisés :

- 1er par le chargé de clientèle
- 2ème par le Service des risques

Pour le risque de 2ème niveau :

- Existant : réalisé manuellement
- Cible : automatisé par un Service métier

Figure 38 : Exemple d'une décomposition en services (1)

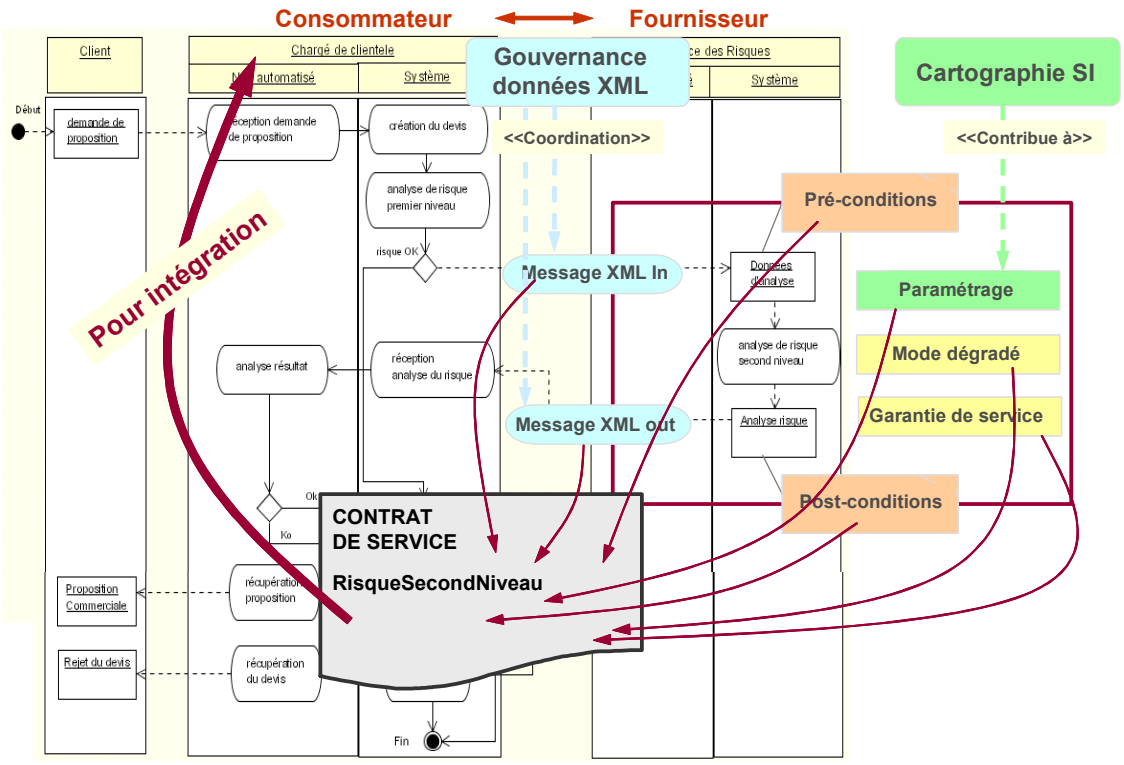
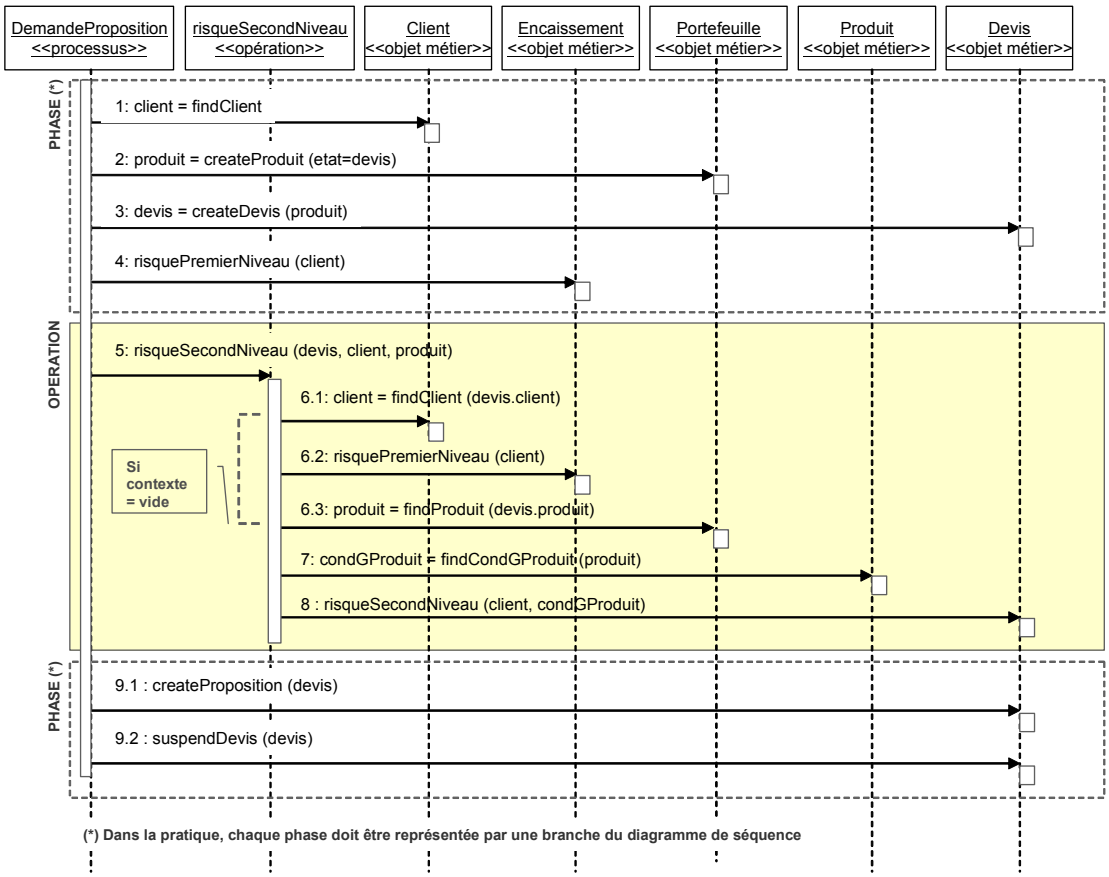
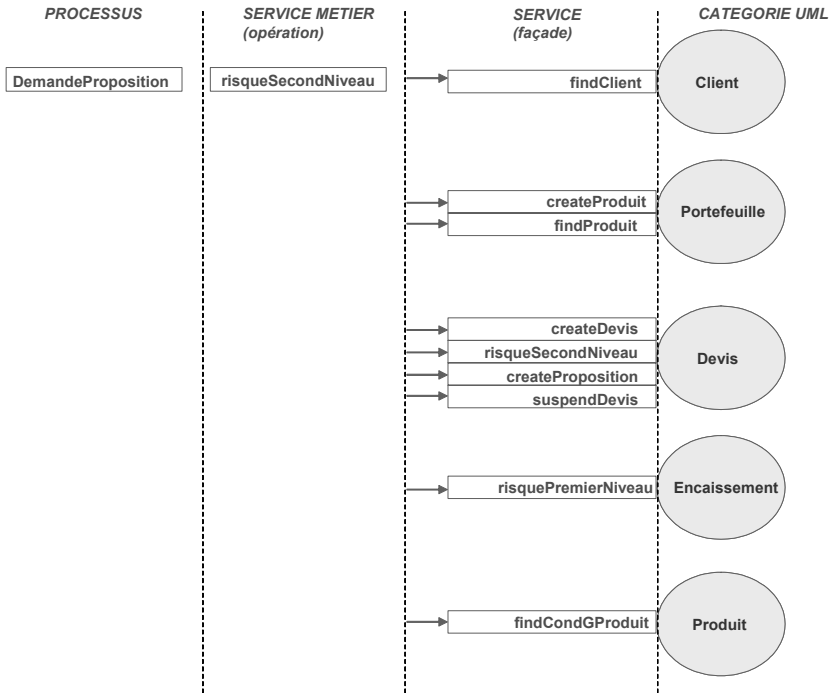


Figure 39 : Exemple d'une décomposition en services (2)



On obtient un catalogue de services rattachés aux catégories (figure suivante).

Figure 40 : Exemple de catalogue de services



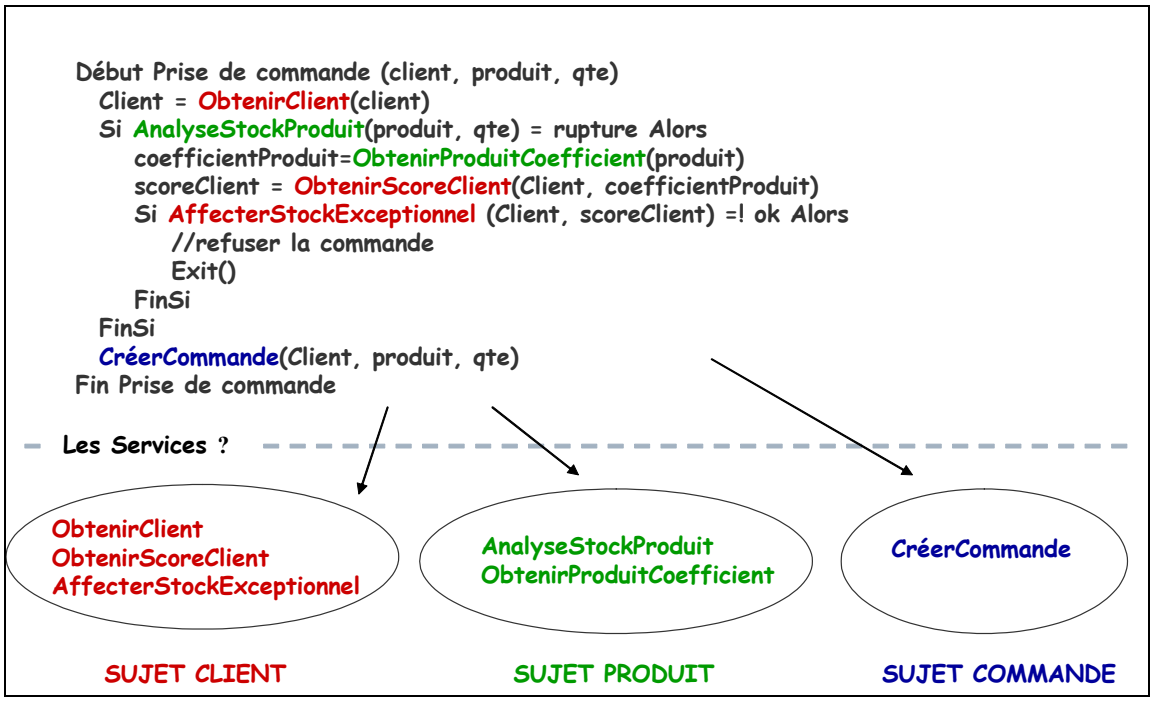
Exemple 2

Dans cet exemple, on montre tout d'abord une organisation du logiciel qui ne respecte par le pattern d'architecture SOA :

```
Début Prise de commande (client, produit, qte)

  select * from Client where nomClient=client
  Si Client n'existe pas Alors
      Saisir information du client
      insert into Client… values…
  FinSi

  select * from Produit where nomProduit = produit
  Si Produit.stock insuffisant<qte Alors
      select coefficientProduit from ProduitCoef, Produit
       where ProduitCoef.typeProduit = Produit.type
      scoreClient = Client.CA / coefficientProduit
      Si scoreClient > seuil Alors
          update Client set stockExceptionnel = … where
          nomClient=client
      Sinon
          //Refuser la commande
          Exit()
      FinSi
  FinSi
  insert into Commande… values…

Fin Prise de commande
```

- Mélange des sujets Client (rouge), Produit (vert) et Commande (bleu)
- Accès direct aux bases de données.
- Logique modulaire et monolithique.
- Pas de couplage faible.
- Tout est lié.
- Rien n'est réutilisable.
- Rien n'est exposable.
- Pas de logique « processus ».
- Les règles d'enchaînement sont diluées dans le code applicatif.

Le « refactoring » de cette organisation selon le pattern d'architecture SOA est présenté maintenant.



Avec cette organisation : le processus orchestre les appels aux services ; les services sont en couplages faibles ; les services encapsulent les catégories représentées par les sujets métiers Client, Produit et Commande.

Le catalogue de services que l'on obtient est alors le suivant :

SUJET CLIENT

```
Début ObtenirClient(client)
   select * from Client where nomClient=client
   Si Client n'existe pas Alors
   Saisir information du client
   insert into Client… values…
   FinSi
   Retourne Client
Fin ObtenirClient

Début ObtenirScoreClient(Client, coefficientProduit)
   scoreClient = Client.CA / coefficientProduit
   Retourne scoreClient
Fin ObtenirScoreClient

Début AffecterStockExceptionnel(client, scoreClient)
    Si scoreClient > seuil
      update Client set stockExceptionnel = … where
      nomClient=client
      Retourne ok
      Sinon
      Retourne ko
   FinSi
Fin AffecterStockExceptionnel
```

SUJET PRODUIT

```
Début AnalyseStockProduit(produit)
   select stock from Produit where
   nomProduit = produit
   Si Produit.stock<qte Alors
      Retourne rupture
   Sinon
      Retourne ok
   FinSi
Fin AnalyseStockProduit

Début ObtenirProduitCoefficient(produit)
   select coefficientProduit from ProduitCoef, Produit
   where ProduitCoef.typeProduit = Produit.type and
   Produit.nomProduit=produit
   retourne coefficientProduit
Fin ObtenirProduitCoefficient
```

SUJET COMMANDE

```
Début CréerCommande(Client, produit, qte)
   insert into Commande… values…
Fin CréerCommande
```

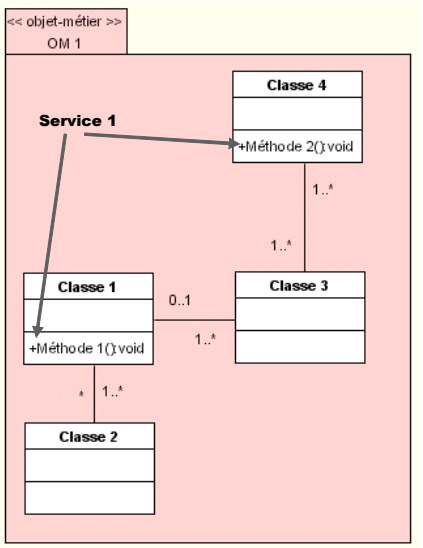
Décomposition du service en méthodes

Cette décomposition n'est possible que si l'on retient une implémentation dans un langage de programmation objet. Dans les autres cas, on se limite au niveau de décomposition en services vu précédemment.

On décompose chaque service sous la forme de méthodes (au sens de la programmation orientée objet) attachées aux classes de la catégorie propriétaire du service et jamais à d'autres classes. On assure ainsi que le couplage fort entre les méthodes est obligatoirement limité au périmètre d'une catégorie. C'est un point important pour la qualité du logiciel.

Dans la figure suivante, on décompose le service 1 de l'objet métier OM 1 en deux méthodes respectivement attachées à la classe 1 et 4.

Figure 41 : Décomposition du service en méthodes



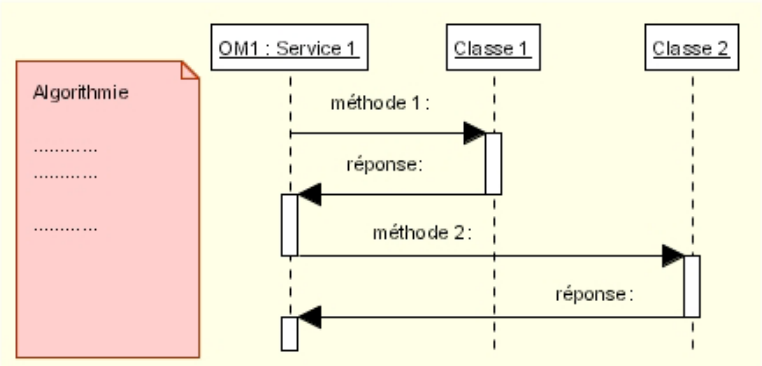
Cette décomposition retient le principe de la programmation objet.

Il est recommandé de le mettre en œuvre pour au moins un premier niveau de décomposition. Dans ce cas, un service correspond à une seule méthode attachée à une classe dont la logique fait appel à des modules applicatifs extérieurs au diagramme de classes. Ce mécanisme présente l'avantage de disposer d'une façade objet pour l'accès au service.

La méthode est la plus petite unité programmatique. Il est possible, selon les besoins de l'implémentation, de décomposer une méthode en d'autres méthodes. Dans ce cas, les méthodes s'appellent alors directement entre-elles sans devoir respecter le principes du couplage faible. Attention, ces appels restent limités au périmètre des classes d'une catégorie (objet métier). En effet, au-delà c'est les fonctions d'orchestration qui reprennent la main (processus, opération et phase).

Le modèle de représentation de la décomposition est un diagramme de séquence UML (figure suivante).

Figure 42 : Diagramme de séquence – décomposition du service en méthodes



Fonctions d'orchestration

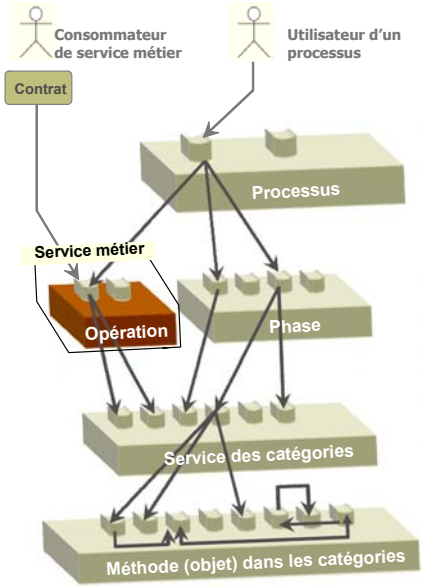
Grâce au travail de conception, on obtient une modélisation sous la forme de processus, d'activités, d'opérations, de phases, de services et de méthodes. La fonction d'orchestration implémente l'enchaînement de ces concepts.

Seule l'activité n'est pas retenue pour l'implémentation. Elle disparaît au profit des regroupements de plus haut niveau représentés par les opérations et les phases. Cette approche simplifie l'implémentation sans nuire à la consistance fonctionnelle :

- Une activité dont le poids sémantique est important et que l'on souhaite isoler au niveau de l'implémentation est assimilée à une phase lors de la conception.
- Une activité qui représente un traitement exposé à une organisation cliente est d'office assimiler à une opération.

On obtient alors les niveaux d'assemblage indiqués dans la figure suivante.

Figure 43 : Niveaux d'assemblage des concepts issus du travail de conception



Sur cette figure, on note que les méthodes s'appellent directement entre elles. Comme nous l'avons déjà indiqué plus haut, ce couplage fort est autorisé uniquement dans le périmètre des classes d'une même catégorie. Par ailleurs, pour ne pas surcharger la figure, on ne représente pas les contrats d'utilisation des services existants au niveau des services des catégories.

Les appels directs entre opérations correspondent à un cas particulier d'enchaînement que nous détaillons dans l'annexe Fonction d'orchestration.

Présenté de manière différente, le tableau ci-après détaille les niveaux d'orchestration entre les concepts manipulés. La colonne « fonction d'orchestration » schématise les principes d'enchaînement des concepts. Chaque sommation est un traitement spécialisé dans l'enchaînement.

Tableau 5 : Niveaux de décomposition des concepts de modélisation

Concept	Décomposition	Fonction d'orchestration
Processus (sous processus) -> Phase et Opération	Ensemble de phases et d'opérations avec une complétude sur l'ensemble du diagramme d'activité	$\text{Processus} = \sum_{i=1}^{n} \text{Phase}^{i} + \sum_{j=1}^{p} \text{Opération}^{j}$
Phase -> Activité	Ensemble d'activités qui forment une étape cohérente d'exécution dans le processus (sous processus) sans être candidat à une exposition sous forme d'opération d'un Web service. Si nécessaire peut représenter un état transactionnel combiné pour l'ensemble des activités qui composent la phase	$\text{Phase} = \sum_{i=1}^{n} \text{Activité}^{i}(\text{Processus})$
Opération -> Activité	Ensemble d'activités qui concourent à l'exécution d'une opération au sens d'un service métier. L'opération est donc exposée à des organisations clientes. Elle s'exécute suivant un mode « question-réponse » synchrone (sauf cas particulier d'implémentation d'un MOM de type JMS mais qui diminue l'interopérabilité entre des systèmes hétérogènes)	$\text{Opération} = \sum_{i=1}^{n} \text{Activité}^{i}(\text{Processus})$
Activité -> Service	Ensemble de services attachés à des objets métiers (élément de la cartographie du diagramme de classes)	$\text{Activité} = \sum_{\substack{i=1 \\ j=1}}^{n,p} \text{Service}^{i}(\text{Objet métier})^{j}$
Niveau optionnel – Valable uniquement si on implémente le logiciel à l'aide d'un langage objet		
Service -> Méthode	Ensemble de méthodes (au sens de la programmation objet) attachées aux classes de l'objet métier propriétaire du service	$\text{Service} = \sum_{\substack{i=1 \\ j=1}}^{n,p} \text{Méthode}^{i}(\text{Classe}^{j})$

La pratique correcte de ces cinq concepts[67] de modélisation favorise :

- L'organisation du système selon les principes SOA.
- La découverte, dès la phase de modélisation, des opérations qui seront exposées par les services métiers.
- L'identification des niveaux d'enchaînement entre les concepts manipulés, ce qui est essentiel pour l'implémentation des fonctions d'orchestration.

[67] Processus (sous processus), phase, opération, activité, service