Les Services

Le service est l'élément clé dans la SOA, cependant cette notion reste si difficile à définir et à avoir un consensus sur la réponse à la question « Qu'est-ce qu'un service ? ». Les discussions autour de ce sujet débouchent souvent sur des blancs, des réponses alambiquées et incertaines, ou des débats enflammées et souvent stériles.

Pour avoir une réponse assez complète à la question précédente nous évoquons les définitions suivantes :

I. Définition :

« Un service est un comportement défini par contrat, qui peut être réalisé et fourni par tout composant pour être utilisé par tout composant sur la base unique du contrat » ***[Bieber and Carpenter 2002].***

«Un Service est un composant logiciel distribué, exposant les fonctionnalités à forte valeur ajoutée d'un domaine métier » ***[XEBIA BLOG : 2009].***

Cependant ***Thomas Erl*** a donné dans son livre « ***SOA Principles of Service Design*** » référencé par [1] une nouvelle dimension à la définition du service, en spécifiant huit aspects (caractéristiques) à satisfaire au composant logiciel pour qu'il puisse considéré comme service dans l'architecture orientée service. Ces aspects sont : le contrat standardisé, le couplage lâche, l'abstraction, la réutilisabilité, l'autonomie, sans état, découvrabilité, et composabilité.

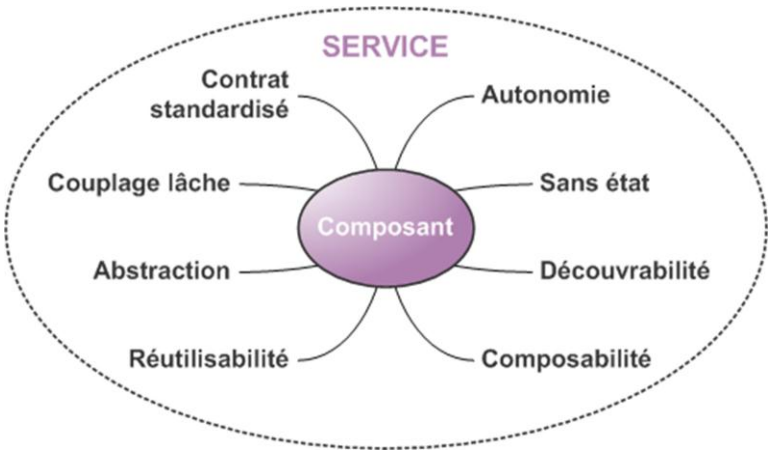


Figure 1 : Les aspects d'un service dans la SOA

Ainsi à travers ces explications nous aboutissons à la définition suivante :

Un service est un composant logiciel distribué satisfaisant les huit aspects (caractéristiques) suivants :

1) Il est exposé à travers un contrat standardisé.
2) Il permet un couplage lâche entre le consommateur et le fournisseur :
3) Il est abstrait (vu comme une boîte noire).
4) Il est réutilisable.
5) Il est autonome.
6) Il est sans état.
7) Il est découvrable.
8) Il est composable.

II. Les aspects (caractéristiques) d'un service :

1) Contrat Standardisé [2] :

L'ensemble des services d'un même Système Technique sont exposés au travers de **contrats** respectant les mêmes règles de **standardisation**.

Qu'est-ce qu'un contrat de service ?

Le contrat de service définit un accord entre le fournisseur et le consommateur. Il est composé :

- D'un **contrat syntaxique** qui propose une représentation technique du service :

  Il constitue le contrat d'utilisation du service *(son interface)*. Il présente le nom du traitement, ses paramètres d'entrée et de sortie et les contraintes structurelles *(format et contrainte sur les données)* qui s'y appliquent.

- D'un **contrat sémantique** qui fournit une description informelle du traitement :

  Il précise les règles et contraintes d'utilisation du service : La valorisation des messages de réponse *(0 représente-t-il le résultat d'un calcul, une erreur, une interruption de service ?)* ; Les exceptions ; les pré et post conditions techniques *(ex. : volume des données échangées)* ou métiers *(ex. : écriture comptable équilibrée)*.

- D'un **contrat de niveau de service** *(QoS & SLA)* qui précise les engagements du service :

  Il spécifie par exemple le temps de réponse maximum attendu, les plages horaires d'accessibilité, le temps de reprise après interruption, les procédures mises en œuvre en cas de panne, les procédures de prise en charge du support, …

Exemple :

Pour un service web (service implémenté sous forme de web service), son contrat peut être formé comme suit :

- Un document WSDL décrivant l'interface du service (les modalités d'accès), ceci fait partie du contrat syntaxique.
- Un ensemble de schémas XML (XSD) qui définissent les types de données échangées par le service :

  Les XSD font partie du contrat syntaxique. Elles définissent les formats de données et les contraintes structurelles.

  Les XSD font également partie du contrat sémantique. En effet, la richesse des types et restrictions XSD permet d'auto-documenter les valeurs possibles et permet souvent d'éviter des quiproquos.

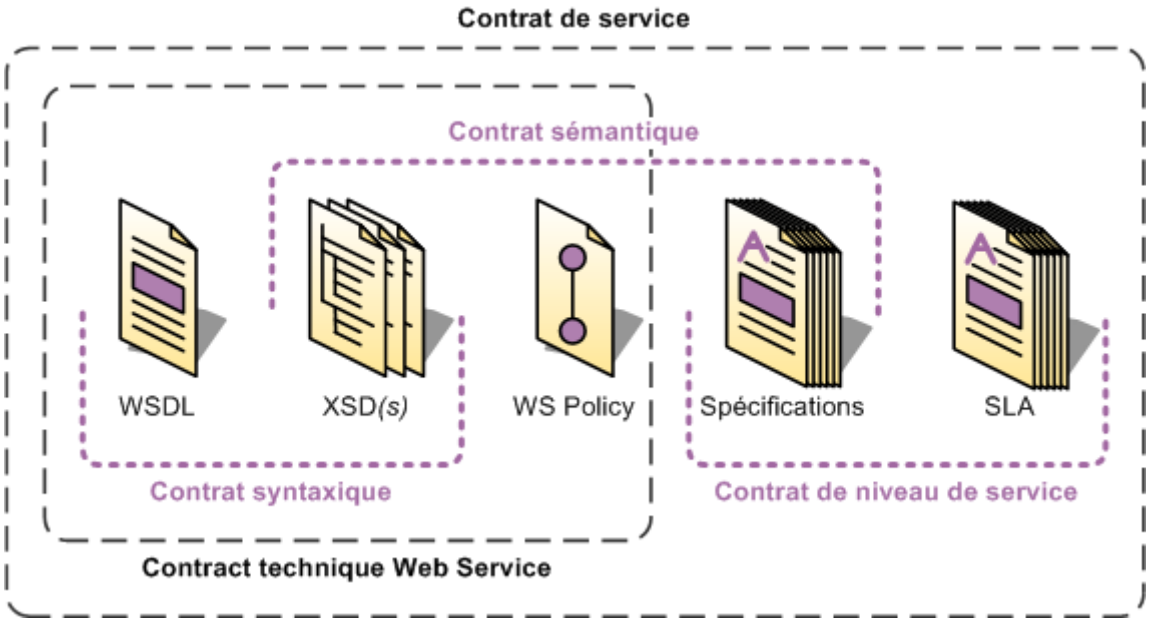
- D'un ensemble de ***policies*** qui définissent les règles d'utilisation du service :

  Les ***policies*** font parties du contrat sémantique. Les règles et contraintes qu'elles expriment adressent des domaines variés : Sécurité, encodage, langue, versioning, métiers, …

- De documentations complémentaires.

  Ces documentations complètent le contrat sémantique. C'est par exemple dans les spécifications que l'on décrira les prés et post condition d'utilisation du service *(elles ne sont pas toutes implémentables sous forme de policy)*.

  Ces documentations constituent le contrat de niveau de service *(QoS & SLA)*. Notons que l'on parle bien d'un contrat qui définit un ensemble d'indicateurs et de valeurs seuils. Il n'est pas question ici des moyens techniques à mettre en œuvre pour leur supervision *(traces, alertes, …)*.



Standardisation du contrat :

Le contrat de service est donc une notion complexe qui ne se limite pas à la *(simple)* définition d'interfaces. Un contrat de service est constitué de nombreux éléments *(techniques ou non)* qui forment un fond documentaire pour lequel il est préférable *(voire indispensable)* de respecter un formalisme commun. L'utilisation de ce formalisme commun est le meilleur moyen de construire un **modèle cohérent** et donc facile à comprendre et à partager.

En plus de cadrer la définition des contrats *(et donc de faciliter la communication)*, l'utilisation d'un formalisme commun et de règles de standardisations facilite la centralisation des éléments constituant les contrats. Cette centralisation encourage la **réutilisation**.

Parmi les constituants d'un contrat de service, les règles d'utilisation (policies) et les formats des données (XSD) sont des bons candidats à la réutilisation vu qu'on général les mêmes règles d'utilisations et les mêmes structures de données sont partagées entre les services dans un seul système.

Il est indispensable ainsi de séparer les différents éléments du contrat de service afin de permettre leur réutilisation.

2) Couplage lâche [3]:

L'objectif de la SOA est de permettre aux entreprises de s'adapter en permanence et être de plus en plus réactives aux variations des marchés et aux avancements technologiques. Afin d'arriver à cette fin il est assez primordial d'éviter les chausse-trappes des grands projets d'architectures distribuées ou intégrées, et en particulier le couplage technique et fonctionnel entre consommateurs et fournisseurs de services.

Le couplage technique : impose au consommateur de connaître le protocole d'échange du fournisseur. À grande échelle, un tel couplage complique, voire interdit l'évolution du socle technique, et risque de figer le SI dans une inertie sclérosante.

Le couplage fonctionnel : impose au client de connaître le format d'échange du fournisseur. Ainsi toute évolution du fournisseur a un impact potentiel sur chacun de ses consommateurs. Mal gérée, cette dépendance peut conduire à de véritables verrous fonctionnels dans le Système d'Informations – interdisant toute évolution ou, plus sûrement, augmentant de façon exponentielle le coût de ces évolutions.

Ainsi se pose la question « ***Comment garantir l'indépendance entre le consommateur (Interface client) et le Fournisseur (Service) ?*** » sachant que par nature, la logique d'un service est fortement liée à son environnement d'implémentation.

Examinons tout d'abord la nature de dépendances qui existent :

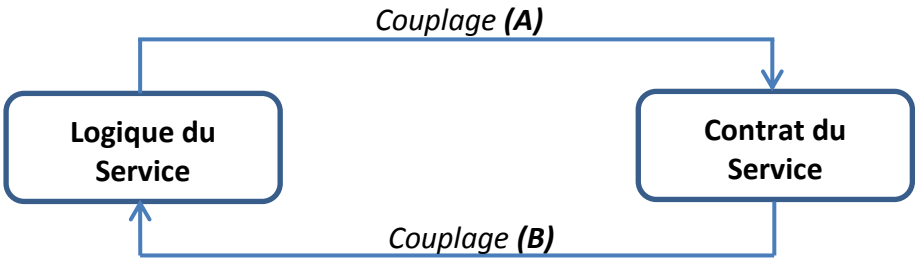
- La logique d'un service est directement dépendante de son implémentation *(1)*. Cette implémentation s'appuie sur un ensemble de ressources qui constituent l'environnement d'implémentation du service. La logique d'un service est donc couplée à ces ressources.
- De la même façon, la logique d'un service est fortement liée aux technologies sur lesquelles son implémentation s'adosse *(2)*.
- D'autre part, dans le cadre de services composites, la logique d'un service est également dépendante des services qui le composent *(3)*.
- Enfin, le contrat de service ainsi que sa logique peuvent être couplés aux processus qui les utilisent *(4)*.

Ces couplages sont inévitables.

Du côté consommateur, celui-ci est lié directement avec le service à travers son contrat. Ainsi nous pouvons déduire que c'est le quatrième type de couplage cité ci-dessus (Couplage Service/Contrat) qu'il faut réduire afin de permettre un découplage entre le consommateur et le service.

Le couplage Contrat / Service

Le contrat d'un service et sa logique peuvent être couplés de deux façons différentes :



- **Couplage du contrat de service vers la logique d'implémentation du service *(B)* :**

  Un tel couplage résulte d'une approche ***Contract last*** lors du design du service *(le contrat dérive de l'implémentation)*. Cette approche, nous allons le voir, est à proscrire car elle induit un couplage du contrat avec l'environnement du service.

- **Couplage de la logique du service au contrat (*A*) :**

  Ce couplage dénote une approche ***Contract first*** dans la conception du service *(le contrat est écrit préalablement à l'implémentation du service)*. Cette approche est toujours préférable et le couplage du service vers son contrat est considéré comme un couplage positif.

Ainsi il faut établir la dépendance dans le bon sens :

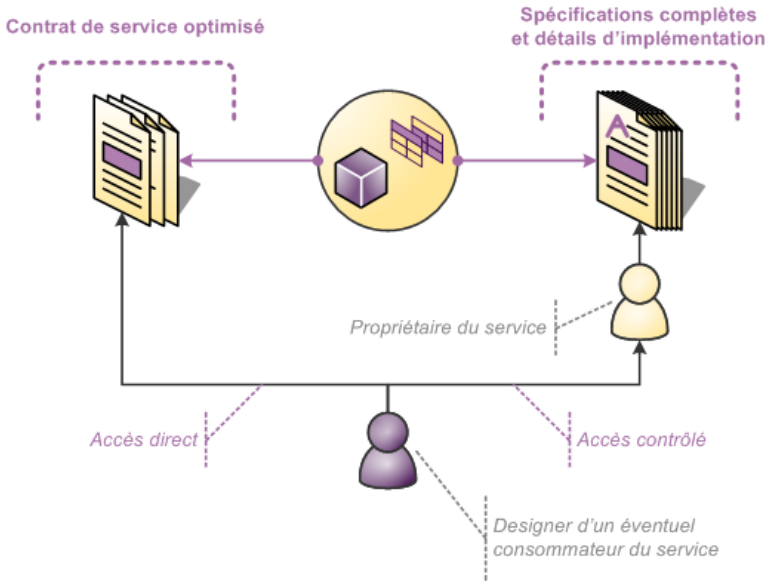
Le consommateur d'un service est lié au contrat de ce dernier, et ne doit être lié qu'à celui-ci *(Pas au service lui-même)*. C'est ce que l'on appelle le **couplage lâche**.

Or, si le contrat est couplé avec la logique du service *(B)*, le consommateur va hériter par transitivité de l'ensemble des dépendances de la logique du service avec son environnement. Nous nous retrouvons donc avec **un consommateur qui est fortement couplé au service (*Couplage Fort*)**.

3) Abstraction [4] :

Le principe d'abstraction consiste à fournir les services du SI sur un modèle **boîte noire**, or les seules informations fournies au consommateur sont celles contenues dans son contrat, ainsi il est quasiment indispensable de n'y mettre que les informations essentielles à son invocation.

Ainsi il est souvent inutile de fournir les informations complètes sur les détails d'implémentation du service, le fournisseur du service peut permettre un accès contrôlé à ce genre de détails à des concepteurs particuliers de consommateurs.



Cependant cette restriction d'accès aux informations du service ne doit pas violer le principe de la prédictibilité du service.

Prédictibilité du service :

La prédictibilité d'un service signifie que son comportement et la réponse qu'il donne suite à une requête ne doivent pas varier (suite à une concurrence d'accès par exemple).

Ce principe est induit par **le respect du contrat du service**.

Ainsi le principe de l'abstraction de service signifie qu'il faut mettre dans le contrat du service toutes et uniquement les informations nécessaires et suffisantes à son invocation.

4) Réutilisabilité [5] :

La réutilisabilité des services (et plus largement des ressources du SI) constitue une des pierres angulaires de la mise en œuvre d'une architecture orientée service. En effet, la mise en œuvre d'une SOA vise, entre autres, à **éviter le gaspillage** des ressources en éliminant les redondances inhérentes au modèle en **silo**. D'autre part, la réutilisation (et donc la réutilisabilité) est une condition première de **l'agilisation du SI** indispensable à la réduction du **time-to-market**, principal élément de ROI des SOA.

La réutilisabilité d'un service désigne la capacité du service à répondre aux besoins de plusieurs types de consommateurs (est-ce que plusieurs consommateurs vont utiliser ce service ?). Ceci implique que le service doit exprimer une logique agnostique afin qu'il puisse être positionné comme une ressource réutilisable.

Les services identifiés comme réutilisables sont ainsi mis à la disposition d'un large spectre de services composites, de processus, d'applications, … Au fil du temps, de tels services ont donc vocation à prendre part à un nombre croissant de compositions, d'orchestrations, de workflow, …

En effet les principaux freins qui peuvent influencer mal sur le degré de réutilisation de service sont :

a) La centralisation et la standardisation des ressources :

La **centralisation des contrats de service** garantit qu'un consommateur n'a accès au service que par son contrat, et la **centralisation de la logique métier** permet d'obtenir un inventaire de services hautement normalisé. **La standardisation de cet inventaire** est essentielle pour maximiser la réutilisabilité (et la réutilisation) des services disponibles au sein de l'entreprise.

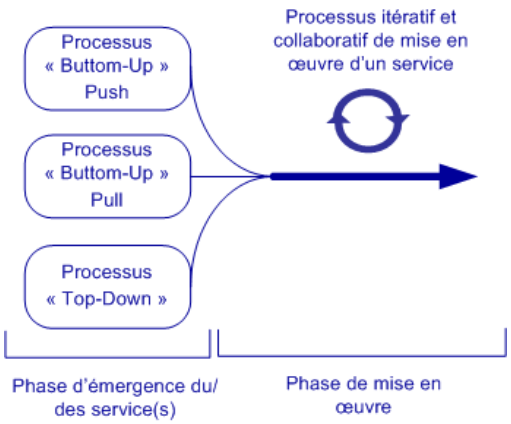
b) La démarche suivie dans la conception de service :

- **Le processus de conception « *Bottom-up* » :** un service émerge d'une démarche « **Push** » initiée par un fournisseur ou « **Pull** » initiée par un consommateur. Selon l'initiateur de cette démarche (fournisseur ou client), le résultat de ce processus est un service plus générique (**Push**) qui essaye de répondre aux besoins de tous les partenaires (risque d'être sans valeur ajoutée métier), ou bien un service qui répond au besoin d'un client particulier sans forcément prendre en compte des perspectives de réutilisation du service. Cela s'explique par le fait que ce genre de demande se conjugue généralement avec des impératifs « Time To market » imposés par le client du service.
- **Un processus de conception « *Top-down* » :** le service est le résultat d'une étude plus globale et systémique du SI ou d'un bloc du SI. Cette démarche garantit une **cohérence** globale de l'ensemble des services, néanmoins, sa mise en place s'avère un challenge inintelligible dont les bénéfices sont loin d'être acquis.

Quelle démarche favoriser donc ?

Dans l'absolu, il n'y a pas de démarche à favoriser en particulier. Les 3 peuvent coexister selon le contexte et l'offre de services existants. Néanmoins, en phase de conception et de mise en œuvre du service, il est primordial de prendre en considération les points suivants:

- Favoriser une démarche itérative et collaborative entre clients et fournisseurs de services.
- Penser un service donné en tant qu'**offre de services**, plutôt qu'un mono-service qui essaie de répondre à tous les besoins.
- Ne pas considérer l'étude d'urbanisation, s'il y en avait une, comme une cible en elle-même, mais plutôt comme un fil conducteur pour la cible.



c) La granularité de service :

La granularité d'un service est définie par le nombre de fonctions recouvertes par celui-ci. En effet ce facteur est très critique, et une mauvaise granularité du service va surement limiter son réutilisation.

M**auvaise granularité :** on parle de mauvaise granularité de service lorsque celui-ci couvre trop ou trop peu de fonctionnalités :

- Un service qui fait trop de choses causera des mauvaises performances et risquerait d'être moins réutilisable à cause des fonctionnalités non utiles pour le consommateur.
- Un service qui fait trop peu de choses sera sans valeurs ajoutée métier et risque d'être non réutilisable de son besoin à collaborer avec d'autres services pour répondre aux besoins du consommateur.

d) La résistance au changement :

Souvent le consommateur n'utilise pas un service parce qu'il existe, mais parce qu'il en a le droit et l'habitude ; et ce droit est donné par un/des urbanistes qui gèrent de facto les mises en relation. ». C'est en effet le cas dans certaines organisations. On retombe ici dans les travers liés à la propriété des composants et au financement au projet.

C'est malheureusement pour cela que dans de nombreuses structures (notamment les plus larges), la réutilisation effective des services reste un vœu pieux.

5) Autonomie [6] :

Parmi les caractéristiques citées déjà, nous avons parlé de la prédictibilité du service. Afin de garantir cette prédictibilité dans le cadre d'accès concurrents, deux principes doivent être appliqués lors de l'élaboration des services :

- Le service doit être autonome (au maximum) .
- Le service doit être sans état.

L'autonomie de service signifie que le service n'a pas besoin à d'autres services pour accomplir sa tâche (il est indépendant des autres), et par conséquent ça implique que le comportement d'un service ne dépend pas du contexte dans lequel il est invoqué (contexte fonctionnel ou contexte technique)

Afin qu'un service s'acquitte au mieux de ses engagements (comportement, fiabilité, performances, …), il doit exercer un contrôle fort sur son environnement d'exécution sous-jacent. Plus ce contrôle est fort, plus l'exécution d'un service est prédictible.

Ainsi c'est au moment de la décomposition des fonctions du SI en inventaire de services, qu'il faut définir les membres de ce registre comme des blocs indépendants. C'est donc un haut niveau d'autonomie individuelle des services qui est visé. Réduire l'accès partagé aux ressources d'un service et augmenter le niveau d'isolation physique des services sont deux leviers permettant d'augmenter cette capacité des services à fonctionner de façon autonome.

Cependant il est quasiment impossible de satisfaire cette caractéristique pour l'ensemble des services, c'est pour cela qu'on a fait introduire deux niveaux basiques d'autonomie :

- **Autonomie de niveau de service :** Les frontières des services entrant dans cette catégorie sont clairement définies et les services sont indépendants les uns des autres, mais il se peut que ces services partagent encore certaines ressources sur lesquelles ils s'appuient.
- **Autonomie pure :** La logique sous-jacente au service et les ressources qu'il utilise sont la propriété du service et sous son contrôle exclusif. C'est le niveau d'autonomie que l'on pourra atteindre lors de la création de « nouveaux » services quand la logique sous-jacente est construite spécifiquement pour supporter le service. Cependant ceci reste un objectif difficilement atteignable au sein d'un système déjà existant et avec lequel il faut vivre pendant longtemps.

6) Sans état (Stateless) [7] :

Garantir la prédictibilité d'un service dans le cadre d'accès concurrents, implique la satisfaction des deux principes :

- Un service doit (au maximum) être autonome.
- Un service doit être sans état.

Un service doit minimiser la consommation de ressources en déléguant la gestion des informations d'état quand cela est nécessaire. D'une manière plus générale, la gestion d'états (d'informations de contexte) au sein d'un service pose des problèmes :

- **De compréhension et de maintenabilité** : La gestion d'états va sensiblement augmenter la complexité cyclomatique de l'implémentation et donc rendre difficile sa lecture, sa documentation, sa testabilité (plus cette complexité cyclomatique est élevée, plus il est difficile d'obtenir une couverture de code acceptable), … Dans le cadre d'une composition de services, la complexité du composé étant directement liée à celle de ses composants, la complexité des services de plus haut niveau peut donc rapidement devenir ingérable.
- **De réutilisation** : La gestion d'états au sein du service brouille la lisibilité de son contrat puisque l'adaptation de son comportement en fonction de son état ne transparaît pas dans son contrat. Or la lisibilité et la transparence des contrats de service sont des facteurs clés de la réutilisation des services.

  D'autre part, l'utilisation d'états au sein d'un service présuppose souvent l'utilisation de ce service au sein d'un enchaînement d'invocations défini à l'avance. Il sera donc difficile de réutiliser le service au sein d'une autre orchestration ou composition.
- **De performances** : car la gestion des états est consommatrice de ressources systèmes, notamment en terme de stockage de ces états (en mémoire ou sur disque).

On préférera donc la conception et l'implémentation de services sans état. La responsabilité de la gestion d'états sera alors déléguée aux utilisateurs (consommateurs) des services : compositions, orchestrations, processus, … Ce transfert de responsabilité (déléguer la gestion d'états au client) rejoint les principes d'une approche REST des services.

7) Découvrabilité [8] :

Le positionnement des services comme ressources réutilisables au sein de l'entreprise passe par :

- La **prédictibilité des services** proposés (ce qui implique que les services soient autonomes et sans état).
- La **découvrabilité des services** existants.

Un service est complété par un ensemble de métas-données de communication au travers desquelles il peut être découvert et interprété de façon effective.

Sachant que la découvrabilité au runtime reste un espoir inabouti (ou une promesse non tenue), il ne reste donc que la découverte par un acteur humain (découverte en phase de conception).

Ainsi cette découvrabilité des services existants passe par la mise en œuvre d'un **repository de services** qui vient outiller l'inventaire des services disponibles. Ce repository stocke l'ensemble des métadonnées nécessaires à :

- La recherche des services de l'inventaire.
- La récupération de l'ensemble des artefacts relatifs aux services de l'inventaire (Spécifications, SLAs, Policies, Schémas XML, WSDL, Interfaces, …).

Ainsi, le concepteur d'un consommateur de service (Service composé, application composite, orchestration, processus, …) pourra s'appuyer sur ce repository de la façon suivante :

- Le concepteur recherche dans le repository un service possédant les fonctionnalités dont il a besoin (1).
- En se basant sur les métadonnées contenues dans le repository, le concepteur est capable de découvrir et d'identifier un service potentiellement capable de répondre à ces besoins (2).
- Le concepteur peut alors accéder au contrat du service : syntaxique, sémantique et de niveau de service (3). il est alors capable, en se basant sur les différents artefacts constituant le contrat du service (spécifications, SLAs, policies, syntaxe, …), de déterminer si le service découvert correspond bien à ces attentes.

8) Composabilité [9] :

Un service doit être conçu de façon à participer à des compositions de services.

Ce dernier aspect constitue en quelque sorte l'aboutissement des sept précédents. En effet, l'ensemble des principes présentés dans cette série vise *in fine* à la réutilisation des services. Or, cette réutilisabilité n'a de sens que si les services sont effectivement réutilisés en prenant part à des compositions de services.

C'est grâce à la composabilité que sera mis en œuvre le principe de « ***separation of concerns*** » au sein d'une architecture orientée services. L'objectif est ici de déterminer la « bonne » granularité de services afin de décomposer la solution à un problème métier de haut niveau en un ensemble de « plus petites unités réutilisables » de traitement : les services.

L'idée est de pouvoir recomposer notre logique métier à l'infini au sein de processus ou de services composites de haut niveau.

La mise en œuvre de cet aspect dans une architecture de services pose donc le problème du bon niveau de granularité pour un service :

- Un service trop large ne pourra pas être réutilisé, car il implémente un enchainement de traitements qui n'ont, a priori, de sens que dans le contexte où le service a été écrit. Un service trop large n'est utilisable que par une seule (ou quelques) application(s).
- A l'opposé, un service trop fin ne sera pas réutilisé, car il implémente un traitement atomique qui n'apporte pas de valeur ajoutée. Un service trop fin propose un niveau de détail qui n'est pas pertinent d'un point de vue métier.

D'autre part, il faut garder à l'esprit que, d'un point de vue technique (runtime), un service ne sera composable que s'il est autonome et stateless (c'est-à-dire réutilisable).

Références :

[1] Thomas Erl «SOA Principles of Service Design» 2007 ISBN: 0132344823,9780132344821.

[2] Le Blog Xebia : « https://blog.engineering.publicissapient.fr/2009/03/04/soa-du-composant-au-service-le-contrat-standardise/».

[3] Le Blog Xebia : « https://blog.engineering.publicissapient.fr/2009/03/19/soa-du-composant-au-service-le-couplage-lache/».

[4] Le Blog Xebia : «https://blog.engineering.publicissapient.fr/2009/04/03/soa-du-composant-au-service-labstraction/».

[5] Le Blog Xebia : «https://blog.engineering.publicissapient.fr/2009/04/16/soa-du-composant-au-service-la-reutilisabilite/ ».

[6] Le Blog Xebia : «https://blog.engineering.publicissapient.fr/2009/04/29/soa-du-composant-au-service-lautonomie/ ».

[7] Le Blog Xebia : « https://blog.engineering.publicissapient.fr/2009/06/04/soa-du-composant-au-service-sans-etat-stateless/».

[8] Le Blog Xebia : «https://blog.engineering.publicissapient.fr/2009/06/12/soa-du-composant-au-service-la-decouvrabilite/ ».

[9] Le Blog Xebia : «https://blog.engineering.publicissapient.fr/2009/08/11/soa-du-composant-au-service-la-composabilite/ ».