Conception orientée service - Service Oriented Analysis and Design

Constantin Drabo

2022-06-17

Contents

Chapter 1

Les concepts

Dans ce chapitre nous allons faire le tour des différents concepts qui entrent dans la compréhension de la conception orientée service. Il s'agira de définir la notion de **SOAD** (*Service Oriented Analysis and Design*), la définition du **SOA** (*Service Oriented Application*).

1.1 Définition

L'analyse et la conception orientée services (**SOAD** : *Service Oriented Analysis and Design*) est une méthodologie qui fait référence à la modélisation et à la conception d'application d'architecture orientée services (**SOA**: *Service Oriented Application*). Une approche SOAD dans la conception SOA nécessite les éléments clés suivants:

- **modèle de processus**
- **instructions**
- **normes**
- **artefacts**
- **qualité de service**

Modèle de processus: elle consiste en la définition du processus et de la notation en faisant un mélange de l'analyse orientée objet (**OOAD**), la modélisation des processus métier (**BPM**) et les éléments d'architecture d'entreprise.

Instruction: c'est la manière structurée de conceptualiser les services

Normes : fournir des facteurs de qualité bien définis et les meilleurs pratiques de service, de capacité, de données et de granularité des contraintes. Les rôles doivent être bien définis et indiquer si c'est un développeur, un architect ou un analyste qui est responsable de chaque fraction de travail.

Artefacts: elle consite à définir ce qui n'est pas un bon service, comme les services qui ne sont pas réutilisables, et qui ne sont donc pas considérés comme des résidents SOA.

Qualité de service: elle facilite la modélisation de bout en bout et fournit un support complet d'outils.

1.2 Qu'est-ce-que la SOA

La **SOA** (_Service Oriented Architecture) est un modèle de conception qui rend des composants logiciels réutilisables, grâce à des interfaces de services qui utilisent un langage commun pour communiquer via un réseau.

Un **service** est une unité autonome de fonctionnalité logicielle, ou d'un ensemble de fonctionnalités, conçue pour réaliser une tâche précise comme récupérer des informations ou exécuter une opération.

Il contient les intégrations de code et de données nécessaires pour exécuter une fonction métier distincte et complète. Vous pouvez y accéder à distance, et interagir avec lui ou le mettre à jour de manière indépendante. En d'autre

termes, l'architecture SOA permet à des composants logiciels déployés et gérés séparement de communiquer et de fonctionner ensemble sous la forme d'applications logicielles communes à différents systèmes.

L'approche **SOA** intègre nativement les principes de *modularité*, d'*interfaçage*, de *contractualisation*, et d'*interopérabilité*. Elle assure ainsi une adaptation rapide du système d'informationn au regard des évolutions des besoins de l'entreprise. Elle permet également de capitaliser la mise en place de bonnes pratiques par l'élaboration d'une architecture de reference. Cette architecture de référence pourra également être utilisée pour répondre à des problématiques de convergence de systèmes.

L'architecture SOA se base sur ressources de l'entreprise pour fournir un système informatique efficient.

- *Pratique* : elle utilise les best-practices de l'entreprise pour construire une architecture efficace
- *Plateforme* : augmenter l'efficacité opérationnelle
- *Utilisateur* : elle donne aux utilisateurs le pouvoir de prendre de meilleures décisions
- *Processus* : aligner l'IT au processus métiers de l'entreprise

Selon le **Gartner Group Research**, une application qui obéit à la SOA doit respecter les cinq(5) principes suivants:

1. *le système doit être modulaire.* Cela offre l'avantage évident de pouvoir diviser et régner (résoudre un problème complexe en assemblant un ensemble de petits composants simples qui fonctionnent ensemble)
2. *les modules doivent être distribuables.* Ils doivent être capables de fonctionner sur des ordinateurs différents et communiquent entre eux en envoyant des messages sur un réseau lors de leur exécution.
3. *les interfaces d'un module doivent être clairement définies et documentées.* Les développeurs de logiciels écrivent ou génèrent des métadonnées d'interface qui spécifient un contrat explicite afin qu'un autre développeur puisse trouver et utiliser le service (cela permet un couplage faible)
4. *un module qui implémente un service peut être remplacé par un autre module qui offre le même service et la même interface*, car l'interface conçue est distincte du module. Il s'agit d'un aspect du couplage faible qui permet une maintenance et des améliorations continues.

5. *les modules du fournisseur de services doivent être partageables.* Ils sont conçus et déployés de manière à pouvoir être invoqués successivement par des modules consommateurs de services disparates engagés dans des activités métiers diverses, bien que partiellement liées. ### Comment fonctionne l'architecture orientée service

Pour implémenter l'architecture orientée service (**SOA**) il existe une panoplie de technologies disponibles. Le choix technologique dépend du besoin de chaque entreprise. En général, une architecture orientée service utilise les services web pour son implémentation. Les services web sont une technologie qui permet de rendre les blocs fonctionnels accessibles à travers Internet. Parmi les possibilités de web service, nous avons l'implémentation du standard **SOAP** (*Simple Object Access Protocol*) , l'utilisation de l'architecture **REST** (*Representational State Transfer*). Il est également possible d'utiliser des technologies basées sur l'échange de message telle que **ActiveMQ**, **Apache ActiveMQ Artemis**, **Kafka**, etc. Les déveoppeurs peuvent également utiliser des modèles appelés **ESB** (*Enterprise Service Bus*) pour réaliser l'intégration entre unn composant centralisé et les systèmes backe-end, puis les rendre disponibles en tant qu'interfaces de services.

1.2.1 SOA vs approche monolithique

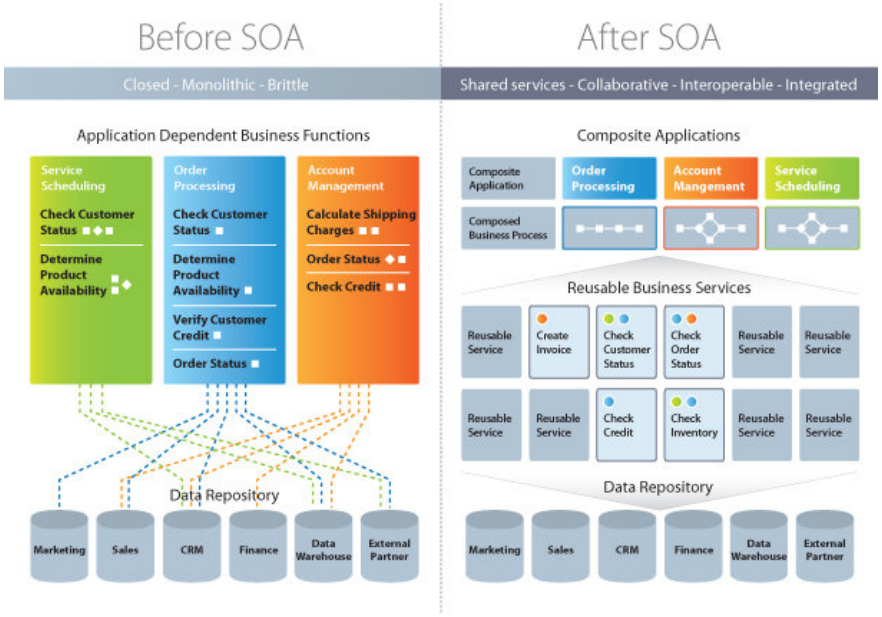
Il existe plusieurs avantages à l'architecture orientée service spécialement pour un service web basé métier. Dans les lignes qui suivront, nous allons exposer quelques unes de ces bénéfices.

Utiliser une architecture orientée service pour créer un code réutilisable : Non seulement cela réduit le temps consacré au processus de développement, mais il n'y a aucune raison de réinventer la roue de codage chaque fois que vous devez créer un nouveau service ou processus. L'architecture orientée services permet également d'utiliser plusieurs langages de codage car tout passe par une interface centrale.

Utiliser une architecture orientée service pour promouvoir l'interaction : Avec l'architecture orientée services, une forme standard de communication est mise en place, permettant aux différents systèmes et plates-formes de fonctionner indépendamment les uns des autres. Grâce à cette interaction, l'architecture orientée services est également capable de contourner les pare-feu, permettant « aux entreprises de partager des services vitaux pour les opérations ».

Utiliser une architecture orientée service pour la scalabilité : Il est important de pouvoir faire évoluer une entreprise pour répondre aux besoins du client, mais certaines dépendances peuvent entraver cette évolutivité. L'utilisation de l'architecture orientée services réduit l'interaction client-service, ce qui permet une plus grande évolutivité.

Utiliser une architecture orientée service les coûts : Avec l'architecture orientée services, il est possible de réduire les coûts tout en « maintenant un niveau de sortie souhaité ». L'utilisation de l'architecture orientée services permet aux entreprises de limiter la quantité d'analyses requises lors du développement de solutions personnalisées.



1.2.2 Les rôles au sein d'une architecture orientée service (SOA)

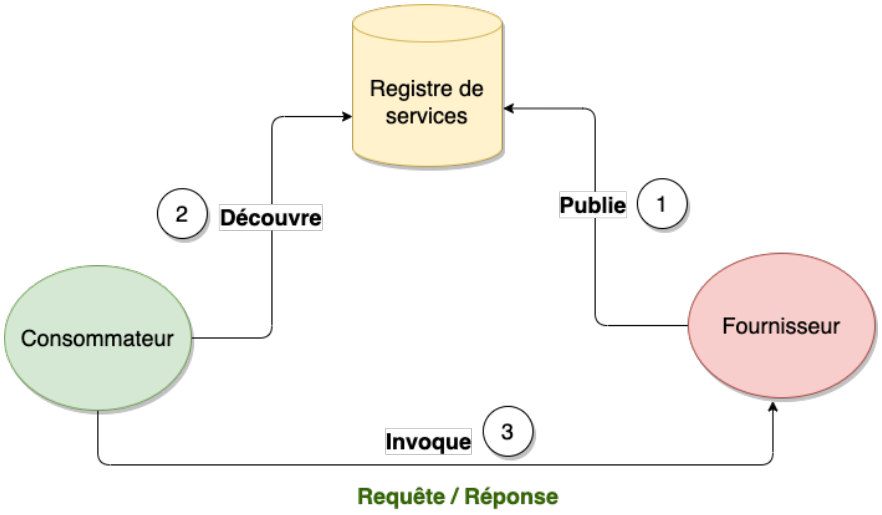
Trois (3) rôles fournissent les principaux composants d'une architecture orientée service :

- le fournisseur de service
- le broker ou registre de service
- le demandeur ou consommateur de service

Fournisseur ou producteur : un fournisseur crée des services web qu'il met à la disposition dans un **registre de services**. Il est responsable des conditions d'utilisation de ces services.

Broker ou registre de service : un borker ou registre de services est chargé de fournir les informations sur le service au **demandeur**. Le broker peut être public ou privé.

Demandeur ou consommateur : le consommateur de services cherche un service dans un broker ou registre de services, puis se connecte à un **fournisseur de service** pour obtenir le service en question.



1.2.3 Les extensions de l'architecture SOA - Architecture évenementielle (Event-Driven Architecture)

Dans un système orienté événements, la structure centrale de la solution repose sur la capture, la communication, le traitement et la persistence des événements. C'est ce qui différencie ce type de système du modèle traditionnel orienté requête.

Evénement : un événement désigne tout phénomène ou changement d'état significatif au niveau du matériel ou d'un logiciel système. Il ne faut pas confondre un événemet et une notification d'évenement, c'est-à-dire une notification ou un message envoyé par le système pour signaler à une autre partie du système qu'un événement s'est produit. Les événements peuvent être causés par des actions internes ou externes. Ils peuvent être provoqués par des utilisateurs (clics de souris ou frappe sur le clavier, par exemple), provenir d'une source externe (un capteur) ou être générés par le système (lors du chargement d'un programme, par exemple).

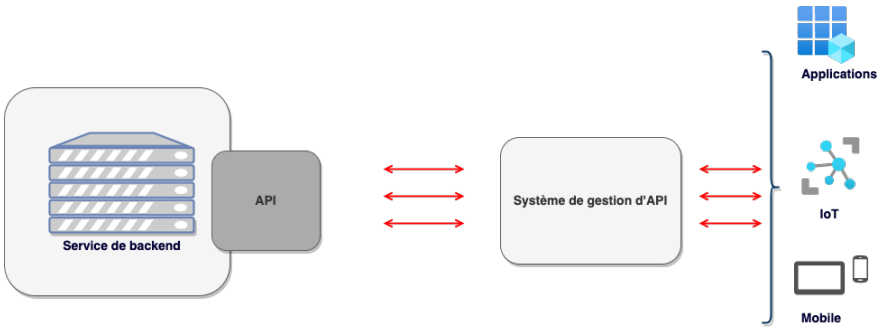
Ce type d'architecture implique des **producteurs** et des **consommateurs** d'événements. Un producteur d'événements détecte ou reconnaît un événement et le représente sous forme de message. Il ignore quels seront les consommateurs et les conséquences de chaque événement. Lorsqu'un évenement a été détecté, il est transmis du producteur au consommateur via des **canaux d'événement**, où une plateforme de traitemennt les prend en charge de façon asynchrone. Les consommateurs doivent être informés lorsqu'un événement se produit. Ils peuvent traiter l'événement ou être seulement affectés par ce dernier. La plateforme de traitement des événements exécute ma reponse adaptée à chaque événement et envoie l'activité en aval au consommateurs concernés. Cette activité permet de visualiser le résultat d'un événement.

Une architecture orientée événements peut être basée sur un modèle de **publication/abonnement** ou sur un modèle de **flux d'événements** .

- le modèle de **publication/abonnement** : ce modèle est une infrastructure de messagerie basée sur des abonnements à flux d'événements. Lorsqu'il est utilisé, chaque fois qu'un événement se produit ou est publié, il est envoyé aux abonnés qui doivent en être informés.
- le modèle de **flux d'événement** : avec ce modèle de flux d'événements, les évenements sont enregistrés dans un journal. Au lieu d'être abonnés à un flux d'évenements, les consommateurs peuvent accéder à n'importe quelle partie du flux et le rejoindre à tout moment.

1.2.4 Les extensions de l'architecture SOA - Les API (Application Programming Interface)

Une **API** ou **Interface de Programmation d'Application**, est un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciel d'applications. Les API permettent à votre produit ou service de communiquer avec d'autres produits et services sans connaître les détails de leur implémentation. Les API sont parfois considérés comme des contrats avec une documentation qui constitue un accord entre les parties : si la **partie 1** envoie une requête à distance selon une structure particulière, le logiciel de la **partie 2** devra répondre selon les conditions définies.



Il existe plusieurs type d'API : les **API privées** , les **API publiques** et les **API partenaires**.

L'**API privée** est utilisables qu'en interne de l'entreprise. Cette approche permet d'avoir un contrôle total sur l'API.

L'**API publique** est accessible à tous. Cette approche autorise les tiers à développer des applications qui interagissent avec votre API et peut devenir source d'innovations.

L'**API partenaire** est partagée avec certains partenaires de l'entreprise. Cette approche peut générer de nouveaux flux de revenus sans compromettre la sécurité.

1.2.5 Les extensions de l'architecture SOA - Les microservices

Les **microservices** sont une interprétation moderne des architectures orientées services utilisées pour créer des systèmes logiciels distribués. Ils ont une nouvelle approche de réalisation et de mise en oeuvre de la SOA qui met l'accent également sur le déploiement continu et d'autres pratiques agiles (**DevOps**).

Les architectures de microservices fonctionnent d'une manière très similaires à la SOA, dans le sens où elles utilisent des services faiblement couplés. Par contre, elles poussent la destructuration de l'architecture classique encore plus loin.

Les services qui compose l'architecture de microservices utilisent une structure de messagerie commune, telle que le **API RESTful**. Ils se servent des API RESTful pour communiquer entre eux simplement sans convertir leurs données ni recourir à des couches d'intégration supplémentaires. L'utilisation des API RESTful permet et favorise même l'accélération de la distribution es nouvelles fonctions et mises à jour.

Chaque service est distinct. Vous pouvez remplacer, améliorer ou supprimer chacun d'entre eux sans affecter les autres services de l'architecture. Cette architecture légère vous aide à optimiser les ressources distribuées ou Cloud et à faire évoluer chaque service de façon dynamique.

Les égalements ci-après caractérisent les microservices :

- des interfaces fines (vers des services déployables indépendamment)
- développement axé sur les affaires (par exemple, conception axée sur le domaine)
- architectures applicatives cloud
- programmation polyglotte et persistence
- déploiement de conteneurs légers
- livraison continue décentralisée
- DevOps avec une surveillance holistique des services.

1.3 Preview book

As you work, you may start a local server to live preview this HTML book. This preview will update as you edit the book when you save individual .Rmd files. You can start the server in a work session by using the RStudio add-in "Preview book", or from the R console:

```
bookdown::serve_book()
```

Chapter 2

Modèle de cycle de vie

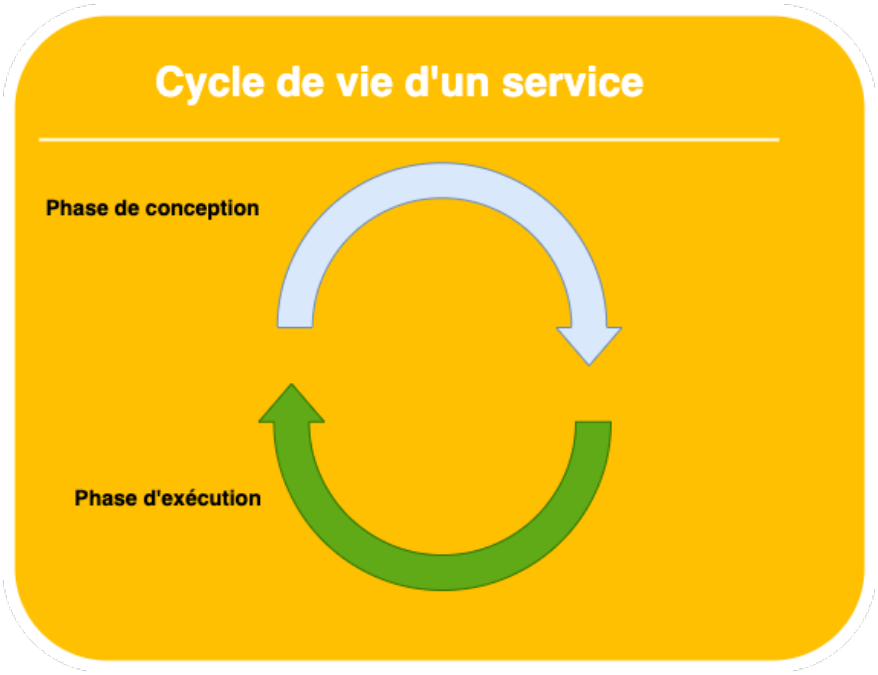
Ce chapitre va aborder le cycle de développement et le cycle de vie des applications orienté service.

Un service est quelque chose qui nait à un certain moment pour accomplir une tache précise qui peut s'achever ou pas. Un service est une entité qui vit pour une certaine raison, peut durer un certain temps et peut ou non être retiré en raison des circonstances.

Un service est une matérialisation de solution logicielle qui est créée dans le but de résoudre un problème commercial ou technologique, mais qui est également un élément de mise en œuvre qui doit coexister avec d'autres actifs logiciels.

Dans la suite du document, nous allons faire ressortir les préoccupations organisationelles des entreprises, façonnées par la discipline de modélisation de la SOA et allons terminer par l'implémentation de la solution informatisée.

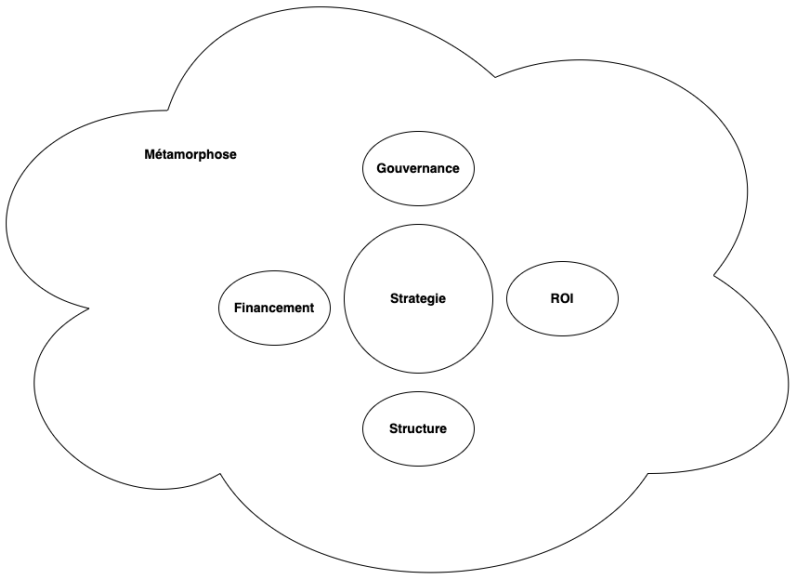
Il n'y a rien de **cyclique** dans le cycle de vie des services, et même si nous le souhaitons, les services ne durent pas éternellement. Alors que les services continuent d'évoluer et d'apporter de la valeur à leurs consommateurs et contribuent de manière significative dans les environnement de production au sein des entrerprises, ils sont toujours sujets à la perfection et aux amélioration. Cela nécessite généralement une investigation pour déterminer si un service doit être ramener à la forge de conception pour une refaire l'architecture ou initier une reconstruction. Ce processus répétitif est appelé le cycle de vie du service. La figure ci-dessous illustre le cycle de vie d'un service :



2.1 Principe du modèle de cycle de vie orienté service

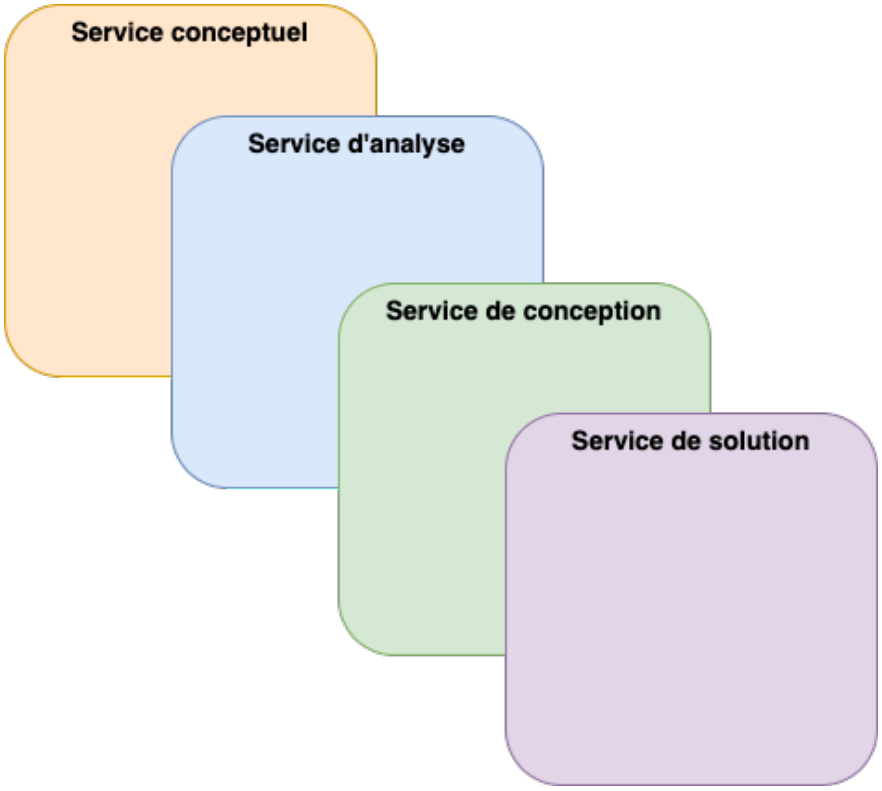
Les principes du cycle de vie d'un service sont les bonnes pratiques rudimentaires pour instituer une méthodologie du cycle de vie d'une SOA afin de gérer les initiatives métiers d'une organisation et le processus de développement d'un service. Ils abordent les fondamentaux de la gestion des frameworks, la planification, les préoccupations structurelles en rapport avec les sujets suivants : la métamorphose du service, la stratégie, la gouvernance, la structure, le financement, le retour sur investissement (ROI).

Le modèle du cycle de vie de la SOA est basé sur le ***principe de la stratégie*** qui a un impact sur tous les aspects du cycle de vie du service, telles que la planification d'initiatives de cycle de vie et la fourniture d'une feuille de route détaillée. Le ***principe de la gouvernance*** se focalise sur les bonnes pratiques de l'orienté service et les standards pour faciliter la gestion et l'exécution du cycle de vie pour superviser les activités. le ***principe de structure*** développe les éléments du cadre du cycle de vie. Le système de ***financement*** du projet est un autre important principe qui introduit un nouveau regard sur le développement des services et la budgétisation des opérations. Le ***principe du retour sur investissement*** met l'accent sur l'importance du suivi des revenus générés par les opérations de service. Enfin, le ***principe de métamorphose***, qui incarne la stratégie de cycle de vie orientée service, décrit l'évolution du service au cours de sa durée de vie.



La métamorphose

Le paradigme de la métamorphose de l'orienté-service est décrit comme l'aspect le plus important de la modélisation orienté-service. Le cycle de vie orienté service est centré sur la perspective de développement et d'exploitation d'un service et sur son passage dans le temps au cours de sa durée de vie. Ici, la principale préoccupation est le traitement d'un service lors des événements du cycle de vie. Cela concerne les mécanismes qui ont été commandés pour utiliser une exécution sans faille du service et pour assurer la continuité des activités dans l'environnement de production. La figure ci-dessous illustre les quatre principaux états de transformation du cycle de vie orienté service:



Au départ un service apparaît simplement comme une idée et un concept. Plus tard,il devient une unité d'analyse. Lorsque la phase d'analyse est terminée, le service se transforme en une entité de conception. Enfin, le cycle de vie de développement orienté service génère un service de solution physique prêt à être déployé dans des environnements de production.

Chapters and sections are numbered by default. To un-number a heading, add a `{.unnumbered}` or the shorter `{-}` at the end of the heading, like in this section.

Chapter 3

Principes de développement des applications orientées service

Cross-references make it easier for your readers to find and link to elements in your book.

3.1 Chapters and sub-chapters

There are two steps to cross-reference any heading:

1. Label the heading: `# Hello world {#nice-label}`.
    - Leave the label off if you like the automated heading generated based on your heading title: for example, `# Hello world` = `# Hello world {#hello-world}`.
    - To label an un-numbered heading, use: `# Hello world {-#nice-label}` or `{# Hello world .unnumbered}`.
2. Next, reference the labeled heading anywhere in the text using `\@ref(nice-label)`; for example, please see Chapter **??**.
    - If you prefer text as the link instead of a numbered reference use: any text you want can go here.

3.2 Captioned figures and tables

Figures and tables *with captions* can also be cross-referenced from elsewhere in your book using `\@ref(fig:chunk-label)` and `\@ref(tab:chunk-label)`, respectively.

See Figure 3.1.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Don't miss Table 3.1.

```
knitr::kable(
  head(pressure, 10), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

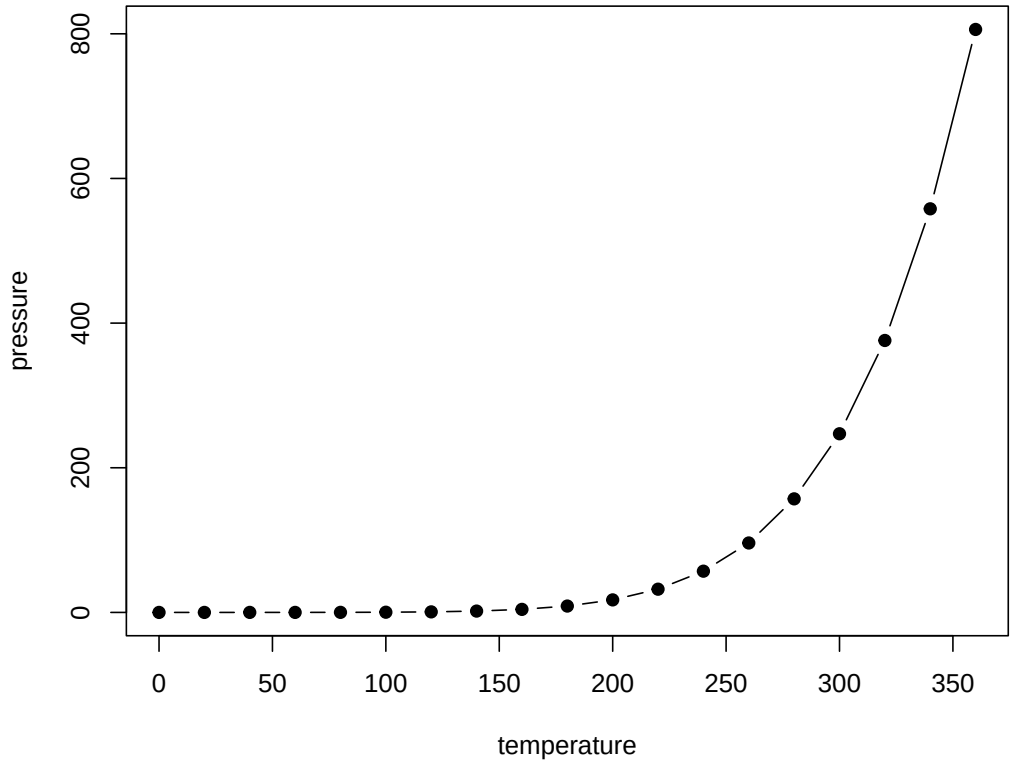


Figure 3.1: Here is a nice figure!

Table 3.1: Here is a nice table!

temperature	pressure
0	0.0002
20	0.0012
40	0.0060
60	0.0300
80	0.0900
100	0.2700
120	0.7500
140	1.8500
160	4.2000
180	8.8000

Chapter 4

Modèles

You can add parts to organize one or more book chapters together. Parts can be inserted at the top of an .Rmd file, before the first-level chapter heading in that same file.

Add a numbered part: `# (PART) Act one {-}` (followed by `# A chapter`)

Add an unnumbered part: `# (PART\*) Act one {-}` (followed by `# A chapter`)

Add an appendix as a special kind of un-numbered part: `# (APPENDIX) Other stuff {-}` (followed by `# A chapter`). Chapters in an appendix are prepended with letters instead of numbers.

Chapter 5

Protocoles et normes

5.1 Footnotes

Footnotes are put inside the square brackets after a caret ^[ ]. Like this one [1].

5.2 Citations

Reference items in your bibliography file(s) using `@key`.

For example, we are using the **bookdown** package (Xie, 2022) (check out the last code chunk in index.Rmd to see how this citation key was added) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015) (this citation was added manually in an external file book.bib). Note that the `.bib` files need to be listed in the index.Rmd with the YAML `bibliography` key.

The RStudio Visual Markdown Editor can also make it easier to insert citations: https://rstudio.github.io/visual-markdown-editing/#/citations

[1] This is a footnote.

Chapter 6

Approche et mise en oeuvre

6.1 Equations

Here is an equation.

$$f\left(k\right) = \binom{n}{k} p^k \left(1-p\right)^{n-k} \tag{6.1}$$

You may refer to using `\@ref(eq:binom)`, like see Equation (6.1).

6.2 Theorems and proofs

Labeled theorems can be referenced in text using `\@ref(thm:tri)`, for example, check out this smart theorem 6.1.

Theorem 6.1. *For a right triangle, if c denotes the* length *of the hypotenuse and a and b denote the lengths of the* ***other*** *two sides, we have*

$$a^2 + b^2 = c^2$$

Read more here https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html.

6.3 Callout blocks

The R Markdown Cookbook provides more help on how to use custom blocks to design your own callouts: https://bookdown.org/yihui/rmarkdown-cookbook/custom-blocks.html

Chapter 7

Sharing your book

7.1 Publishing

HTML books can be published online, see: https://bookdown.org/yihui/bookdown/publishing.html

7.2 404 pages

By default, users will be directed to a 404 page if they try to access a webpage that cannot be found. If you'd like to customize your 404 page instead of using the default, you may add either a `_404.Rmd` or `_404.md` file to your project root and use code and/or Markdown syntax.

7.3 Metadata for sharing

Bookdown HTML books will provide HTML metadata for social sharing on platforms like Twitter, Facebook, and LinkedIn, using information you provide in the `index.Rmd` YAML. To setup, set the `url` for your book and the path to your `cover-image` file. Your book's `title` and `description` are also used.

This `gitbook` uses the same social sharing data across all chapters in your book- all links shared will look the same.

Specify your book's source repository on GitHub using the `edit` key under the configuration options in the `_output.yml` file, which allows users to suggest an edit by linking to a chapter's source file.

Read more about the features of this output format here:

https://pkgs.rstudio.com/bookdown/reference/gitbook.html

Or use:

```
?bookdown::gitbook
```

Bibliography

Xie, Yihui. 2015. *Dynamic Documents with R and knitr.* 2nd ed. Boca Raton, Florida: Chapman and Hall/CRC. ISBN 978-1498716963.
URL: *http://yihui.org/knitr/*

Xie, Yihui. 2022. *bookdown: Authoring Books and Technical Documents with R Markdown.* R package version 0.26.
URL: *https://CRAN.R-project.org/package=bookdown*