IUT Nantes
Pôle Sciences et technologie
Nantes Université

SAÉ 401 : Développement d'une application complexe

R4.12 - Automates et Langages :

Modélisation de micro-services à l'aide d'automates

Groupe_1_1 : Kyllian Arnaud, Jauzua Destain, Pol Lamothe,
Brieuc Le Carluer, Thomas Souchet

17 avril 2025

BUT2 Informatique, année académique 2024-2025

Table des matières

1 Introduction

Dans le cadre de la SAE du semestre 4, il nous a été demandé dans la ressource R4.01, Architecture logicielle, de développer trois micro-services chacun sous la forme d'une API REST. Une API REST (Representational State Transfer) est une architecture d'API qui repose sur le protocole HTTP pour permettre la communication entre clients et serveurs. Elle est basée sur la notion de ressources, identifiées par des URLs, et manipulées à l'aide des méthodes HTTP standard selon la définition de Red Hat [1]. Une API REST est dite *stateless*, c'est à dire que chaque requête contient toutes les informations nécessaires, et le serveur ne garde pas de mémoire de l'état du client entre les appels.

En tenant compte du cahier des charges de la ressource R4.01, nous avons décidé de développer les trois micro-services suivants :

— Un service de consultation et d'opérations sur des cartes Pokémon reposant sur l'API externe : Pokémon TCG API [1]

— Un service de gestion des utilisateurs et de leur collection de cartes Pokémon

— Un service qui fait office de proxy pour les deux autres, ce service est le point d'entrée de notre application

Concrètement notre application permet aux utilisateurs de créer leur collection de cartes Pokémon en ouvrant des paquets de cinq cartes aléatoires appelés *booster* chaque paquet de cartes correspond à une série appelée *set*.

Dans le cadre de la ressource R4.12 Automates et Langages, nous avons modélisé un de nos micro-services en utilisant des automates à états comme abordé dans le cours de Christian Attiogbé [2]. En effet, les automates permettent, en amont de la phase de développement, de réfléchir au fonctionnement de l'application en représentant un scénario d'exécution de celle-ci. Nous avons choisi de faire la modélisation du premier micro-service présenté ci-dessus, c'est-à-dire celui qui interagit avec une API externe pour fournir des données sur les cartes Pokémon. La modélisation se décompose en deux automates, un pour le fonctionnement du serveur et un pour les actions du client.

2 Modélisation à l'aide d'automates

2.1 Hypothèses pour la modélisation

Pour faciliter la modélisation du fonctionnement d'une API, voici les hypothèses que nous utilisons :

— Le caractère `!` représente l'émission d'un message et le caractère `?` représente l'attente d'un message ou d'une requête. Il y a donc synchronisation entre les deux.

— Les primitives `Send(...)` et `Read(...)` sont utilisées respectivement pour émettre un message et pour lire dans la base de données.

— Les conditions sont indiquées entre crochets devant le nom des actions : `[condition] Action`

— Le caractère `/` lorsqu'il ne fait pas partie d'une URL (s'il y a un espace avant et après) fait office de `OU` logique, c'est donc soit l'action à gauche du caractère qui est utilisée soit celle à droite.

1. https://pokemontcg.io/

2.2 Automate client

Pour modéliser les actions du client (voir Fig. 1), nous avons défini un scénario qui limite les actions de l'utilisateur avec un ordre logique. Premièrement, l'utilisateur ne connaît pas les identifiants des *sets* ou des cartes, il commence donc par récupérer tous les *sets* disponibles. Ensuite, il a le choix entre plusieurs actions, il peut récupérer les informations sur un *set* en particulier (état $S2.1$), il peut récupérer toutes les cartes d'un *set* (état $S2.3$) et il peut ouvrir un *booster* à partir d'un *set* (état $S2.2$).

Après avoir ouvert un *booster* ou consulter les cartes d'un *set*, il peut choisir de faire des actions en rapport avec les cartes comme récupérer les informations sur une carte en particulier (état $S3.1$) ou trouver les évolutions possibles d'une carte (état $S3.2$).

Enfin, si l'utilisateur a ouvert un *booster*, il peut consulter le prix total des cartes qu'il vient d'obtenir (état $S3.3$).

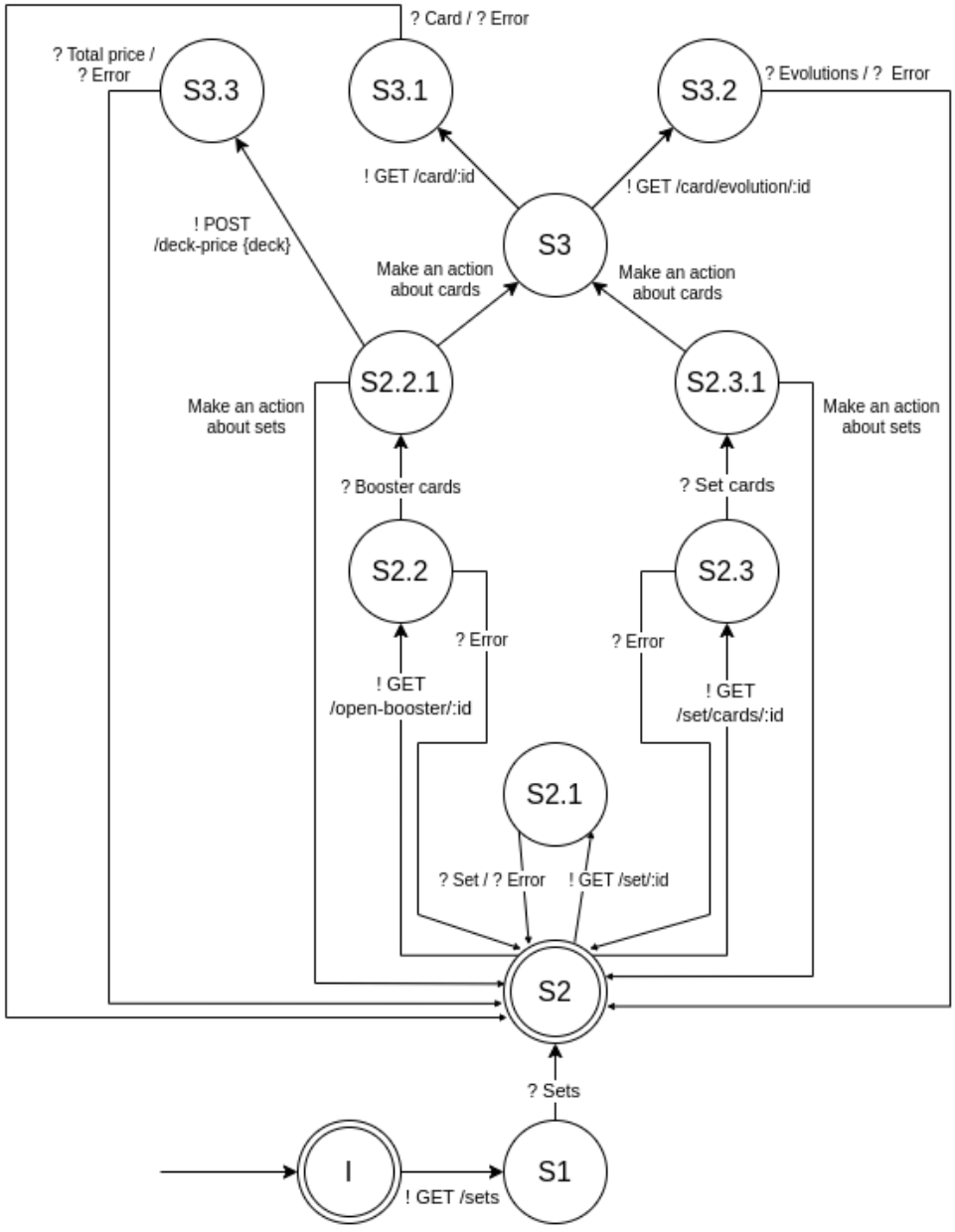

FIGURE 1 – Automate du client

2.3 Automate serveur

Pour modéliser le fonctionnement du serveur (voir Fig. 2), nous avons défini le point d'entrée de celui-ci (état I), à partir de cet état les sept routes sont disponibles, dans chacune d'elles le serveur peut lire des informations depuis la base de données et il renvoie une réponse valide ou une erreur. Cet automate est synchronisé avec l'automate client de la partie précédente.

Les routes sont les suivantes :

— `GET /sets` récupérer la liste des *sets* disponibles ;

— `GET /set/:id` consulter les informations d'un *set* ;

— `GET /set/cards/:id` récupérer la liste des cartes d'un *set* ;

— `GET /open-booster/:id` ouvrir un *booster* à partir d'un *set* ;

— `GET /card/:id` consulter les information d'une carte ;

— `GET /card/evolution/:id` récupérer la liste des évolutions possibles d'une carte ;

— `POST /deck-price {deck}` consulter le prix total d'une liste de cartes, le paramètre `deck` est la liste des identifiants des cartes.

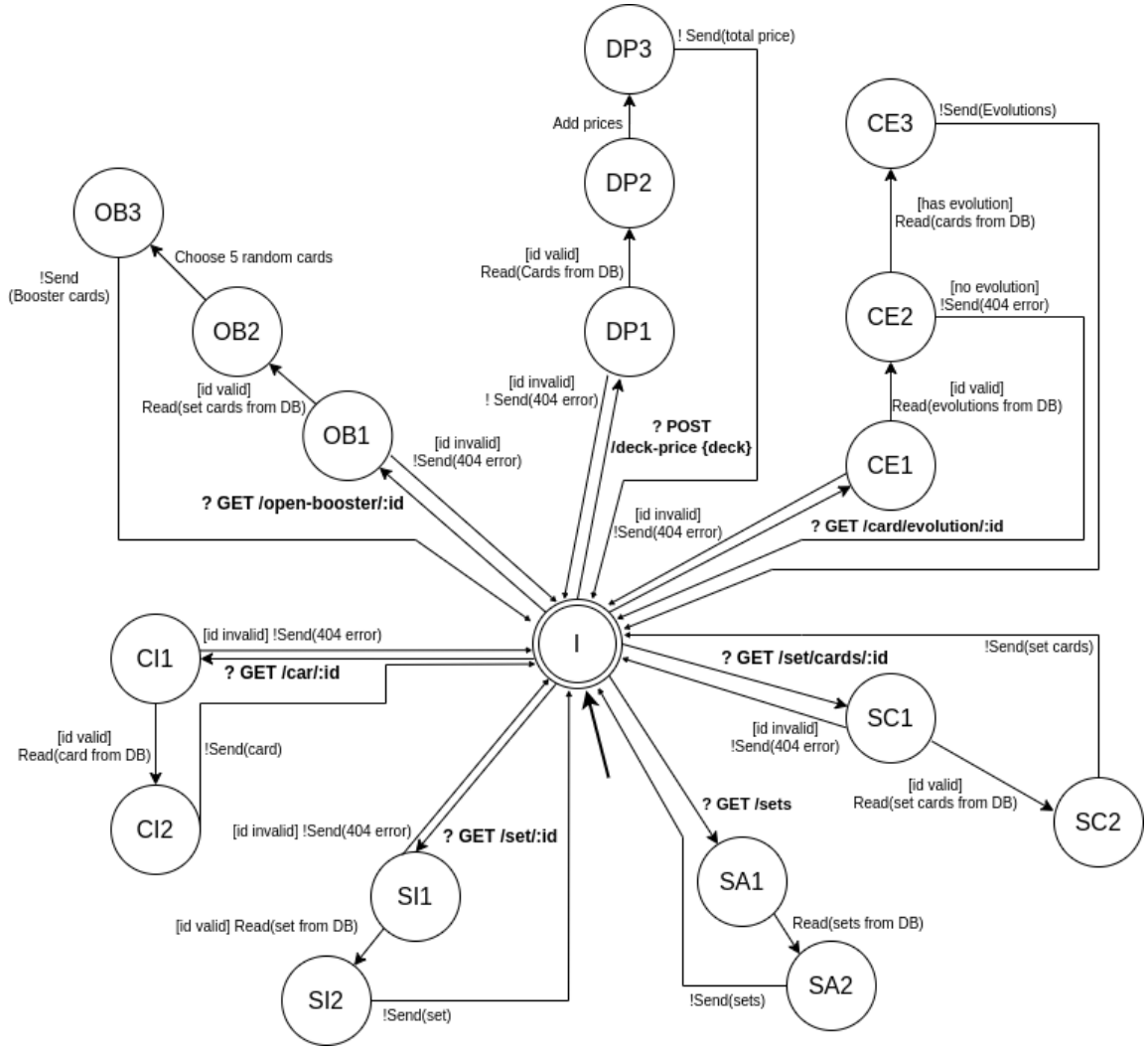

FIGURE 2 – Automate du serveur

3 Conclusion

En conclusion, la modélisation à l'aide d'automates à états de notre micro-service a été, dans notre cas, particulièrement utile pour la mise en place des tests. En effet, l'automate du client nous a donné des scénarios à tester précisément en s'assurant à chaque étape que l'automate se trouve dans le bon état et que l'on ne se retrouve pas dans un état invalide. L'automate du serveur nous a fourni une documentation facilement compréhensible de notre API et nous permettra d'ajouter des fonctionnalités en maintenant une structure cohérente.

Cette méthode de modélisation s'est révélée particulièrement efficace et pour de prochain projet, nous pourrions exploiter davantage cette méthode en modélisant entièrement une application à l'aide dun automate pour en générer du code automatiquement à partir a partir des traces du comportement de l'automate, cela peut être fait avec des frameworks comme SpringBoot [2], par exemple, comme illustré dans cet article [3].

Références

[1] Red Hat. *Qu'est-ce qu'une API REST ?* Red Hat, consulté le 17 avril 2025. Disponible sur : https://www.redhat.com/fr/topics/api/what-is-a-rest-api

[2] J. Christian Attiogbé. *Bases de la Théorie des langages et des automates*. Support de cours, IUT de Nantes, 2025.

[3] Pradeep Thomas. Business Flows and State Machines – Part II. *Medium*, 2020. Disponible sur : https://medium.com/@pradeep_thomas/business-flows-and-state-machines-part-ii-56fdd8414268

2. https://spring.io/projects/spring-boot