Service-oriented Architecture

MarsY, to the space and beyond

Team A

(comme astronaute)

Antoine BUQUET

-

Benoit GAUDET

-

Ayoub IMAMI

-

Mourad KARRAKCHOU

Sommaire

I. Architecture

1. Explication de l'architecture

Services	*weather-service*
	mission-service
	rocket-department-service
	telemetry-service
	payload-service
	executive-service
Mocks	*rocket-hardware-mock-service*
	payload-hardware-mock-service
	stage-hardware-mock-service
Database	*database-telemetry*
	database-payload

Figure 1. - Composition de l'architecture

L'architecture de notre projet est composée d'un certain nombre de micro-services que nous pouvons classer selon trois différentes catégories : les services en tant que tel, les mocks et enfin les bases de données.

- Les services sont les composants individuels de l'application. Ils sont responsables de tâches spécifiques, telles que la gestion de la mission, la collecte de données météorologiques ou la surveillance des métriques de la fusée.

- Les mocks sont des services fictifs utilisés pour simuler le comportement physique réel de la fusée ou de sa cargaison par exemple. Cela permet de tester d'autres services en envoyant des données régulièrement.

- Les bases de données sont utilisées pour stocker les données de l'application. Il en existe deux dans notre projet : une base de données de télémétrie et une base de données pour la cargaison.

2. Diagramme d'architecture

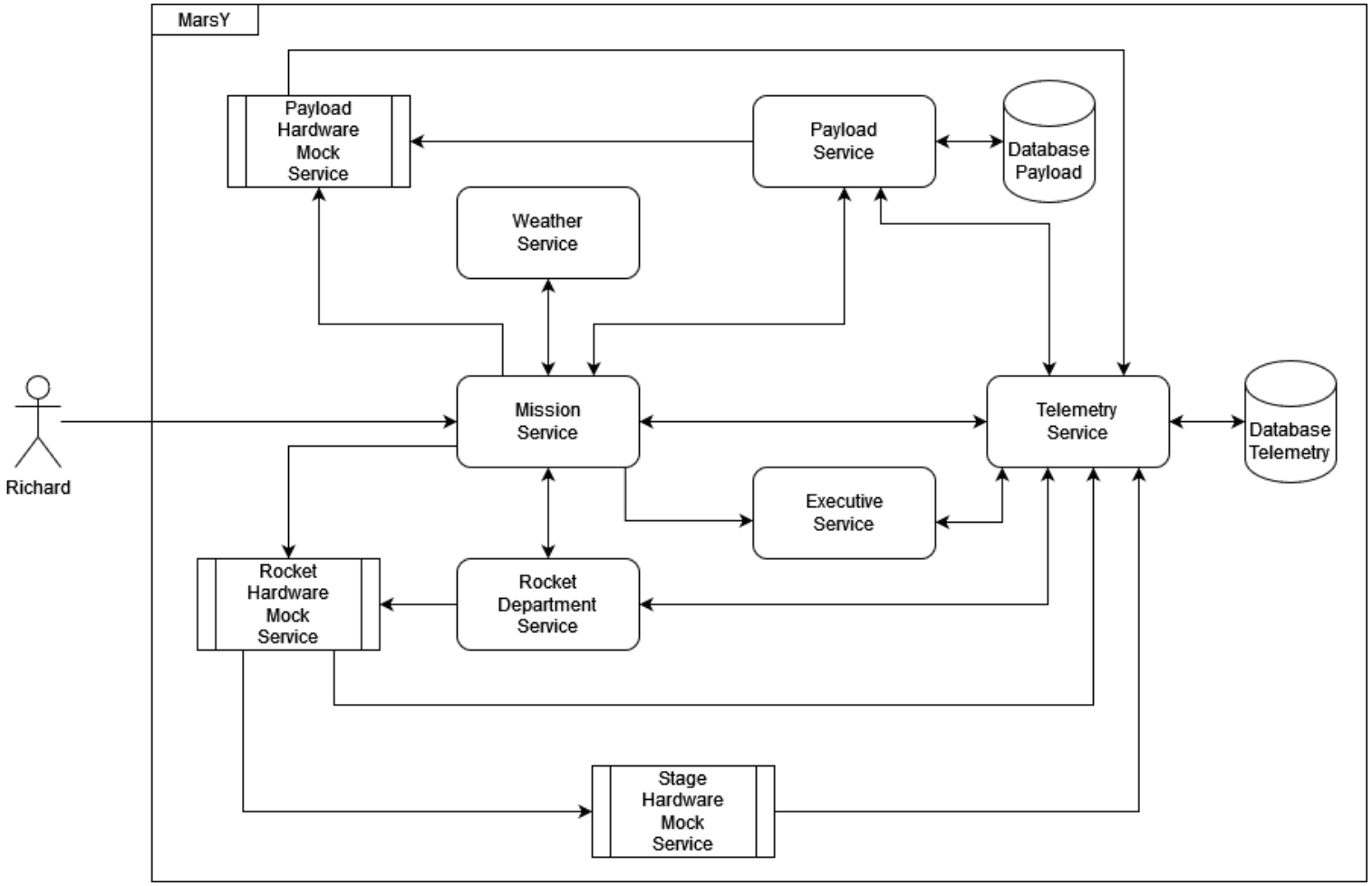

Figure 2. - Diagramme d'architecture

3. Description des services (même que dans le README)

Services

- *weather-service* : s'assure que les conditions météorologiques sont sûres sur l'ensemble des sites
    - Répond GO ou NO-GO au *mission-service* lorsqu'il demande l'état des conditions météorologiques avant le lancement de la fusée

- *mission-service* : supervise toute la mission
    - Demande au *weather-service* et au *rocket-department-service* un GO ou NO-GO pour lancer la fusée
    - Envoie un GO ou NO-GO au *rocket-department-service* en fonction des réponses du *weather-service* et du *rocket-department-service*
    - Informe le *payload-service* et l'*executive-service* du début de la mission
    - Demande au *telemetry-service* d'être averti si la fusée présente une grave anomalie
    - Donne l'ordre de détruire la fusée en cas de grave anomalie

- *rocket-department-service* : gère la fusée et de ses systèmes
    - Répond GO ou NO-GO à *mission-service* lorsqu'il demande l'état de la fusée avant le lancement
    - Lance la fusée après avoir reçu la réponse GO de *mission-service*
    - Demande au *telemetry-service* d'être informé de certains événements au cours de la mission
    - Découpe la fusée en plein vol lorsqu'il n'y a plus de carburant dans le premier étage
    - Ralenti la fusée lors du passage dans la zone Max Q pour éviter d'endommager le matériel et la cargaison

- *telemetry-service* : gère les communications de données entre la fusée et le sol
    - Reçoit des demandes de notification d'événements des autres services
    - Reçoit, surveille et enregistre les données de *rocket-hardware-mock-service*, *stage-hardware-mock-service* et *payload-hardware-mock-service*
    - Informe le *mission-service*, l'*executive-service*, le *payload-service* et le *rocket-department-service* de certains événements

- *payload-service* : responsable de la cargaison du client et de son largage sur la bonne la trajectoire ou orbite
    - Demande au *telemetry-service* d'être informé lorsque la fusée atteint la bonne hauteur pour larguer la cargaison
    - Largue la cargaison sur la bonne trajectoire ou orbite
    - Reçoit et enregistre les données du *payload-hardware-mock-service* via *telemetry-service*
    - Vérifie que l'altitude de la cargaison est toujours correcte après son largage

- *executive-service* : dirige et pilote stratégiquement les activités et les objectifs de l'entreprise
    - Demande au *telemetry-service* d'être informé du retour du premier étage de la fusée sur Terre

Mocks

- *rocket-hardware-mock-service* : représentation fictive du matériel de la fusée
    - Commence à envoyer des données au *telemetry-service* après le début de la mission
    - Envoie des données sur l'altitude, la vitesse, le carburant de chaque étage, l'état et l'accélération de la fusée toutes les demi-secondes

- *stage-hardware-mock-service* : représentation fictive du matériel des étages de la fusée
    - Commence à envoyer les données d'un étage spécifique au *telemetry-service* après que l'étage ai été détaché
    - Envoie des données sur l'altitude, la vitesse, le carburant et l'accélération de l'étage toutes les secondes

- *payload-hardware-mock-servic*e : représentation fictive du matériel de la cargaison
    - Commence à envoyer des données au *telemetry-service* après avoir été lâché par la fusée
    - Envoie des données sur la position actuelle de la cargaison au *telemetry-service* toutes les secondes

Databases

- *database-telemetry* :
    - Enregistre les données de la fusée reçues par *rocket-hardware-mock-service*
    - Enregistre les données du premier étage de la fusée reçues par le *stage-hardware-mock-service*
    - Enregistre les demandes de notification reçues par les autres services

- *database-payload* :
    - Enregistre les données de la cargaison reçues par le *payload-hardware-mock-service*

II. Scénarios

1. Réussite de la mission

Tout commence par la réception d'une demande pour commencer la mission, puis le *mission-service* signale le début de la mission. Ensuite, *mission-service* initie un sondage auprès de *weather-service* pour obtenir le statut météorologique "GO". De manière similaire, *mission-service* effectue un sondage auprès de *rocket-department-service* pour vérifier le statut de la fusée. Pendant ce temps, le *rocket-hardware-mock-service* est contacté pour activer la journalisation du matériel de la fusée. Le *rocket-hardware-mock-service* répond en initialisant ses composants, en commençant la journalisation, et en envoyant des données de fusée à telemetry-service. Une fois que tous les services sont prêts et que les réponses du sondage sont positives, le *mission-service* est autorisée à lancer la fusée. Le *rocket-department-service* gère le lancement de la fusée et demande à *telemetry-service* de le notifier lorsque le carburant du premier étage est épuisé, lorsque la fusée atteint la phase Max Q, et lorsqu'elle la quitte. Enfin, le *rocket-department-service* confirme le lancement de la fusée.

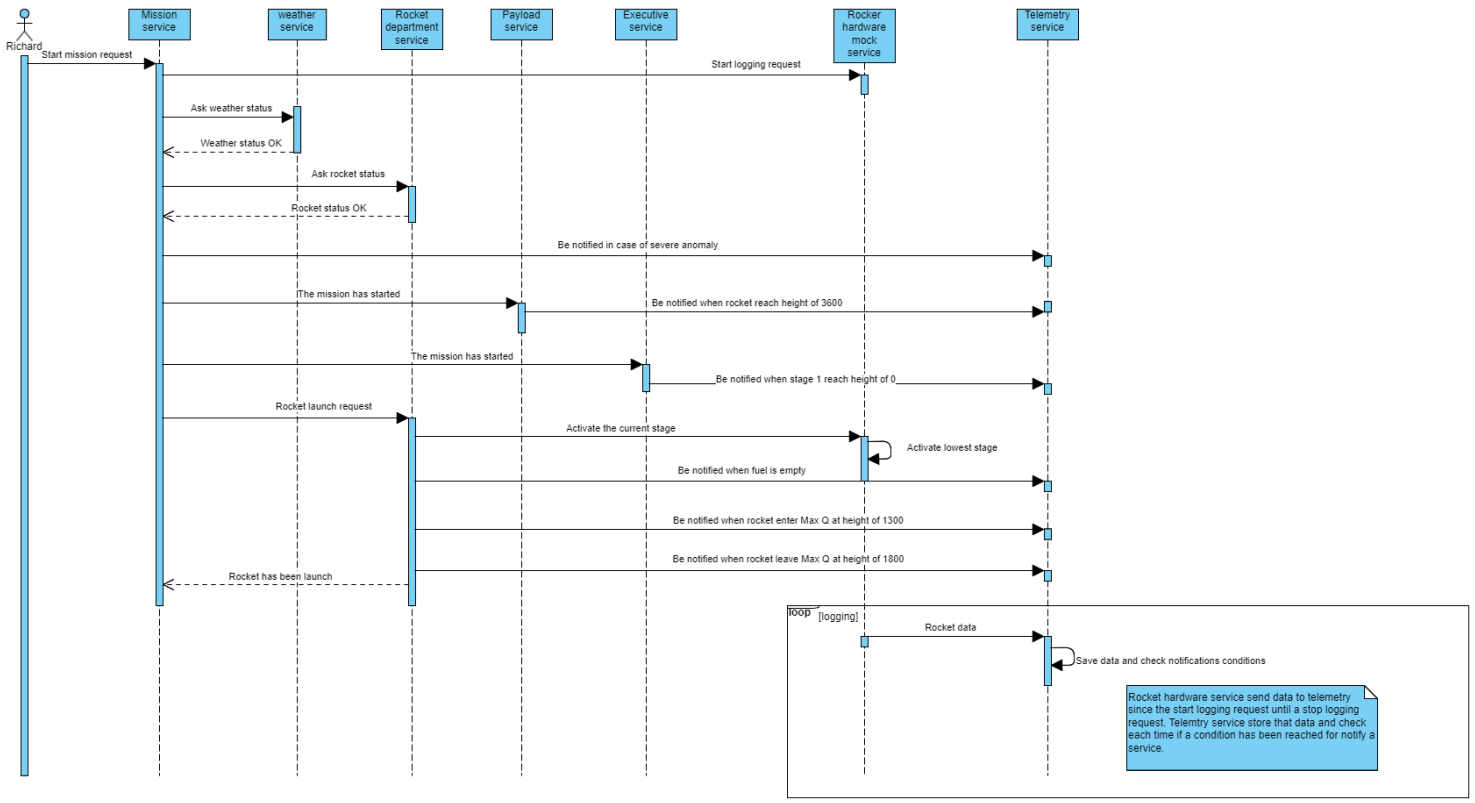

Figure 3. - Diagramme de séquence du début de la mission jusqu'au décollage de la fusée (disponible ici en grand format)

Après avoir reçu la demande de démarrage de la mission, le *rocket-department-service* informe *rocket-hardware-mock-service* d'activer le fuel de l'étage le plus bas de la fusée, ce qui entraîne une augmentation de la vitesse. et donc le décollage.

Pendant le vol de la fusée, un premier événement apparaît lorsque le carburant du premier étage de la fusée est épuisé. *telemetry-service* prévient *rocket-department-service* de l'évènement. La fusée va ensuite se séparer de cet étage, avant d'activer le deuxième étage pour continuer à avancer.
Ensuite, lorsque la fusée atteint la zone de Max Q (où la pression est maximale), *telemetry-service* prévient *rocket-department-service*, qui fera ralentir la fusée pour éviter de potentiel dégats sur la fusée ou sa cargaison.
Finalement, lorsque la fusée sort de la zone de Max Q, *telemetry-service* prévient *rocket-department-service* que la fusée quitte cette zone, qui accélère à nouveau.

Pour finir, lorsque la fusée atteint la hauteur de 3600 mètres, *telemetry-service* prévient *payload-service* qui va lâcher la cargaison à la bonne altitude. Suite à cette étape, *payload-service* indique à *mission-service* que la cargaison a correctement été livré. Il pourra par la suite affirmer le succès de la mission et demander à *rocket-hardware-mock-service* d'arrêter la journalisation des informations de la fusée.

La cargaison va continuer d'envoyer ses informations à *telemetry-service* afin de certifier au client que la cargaison reste bien sur la bonne orbite.

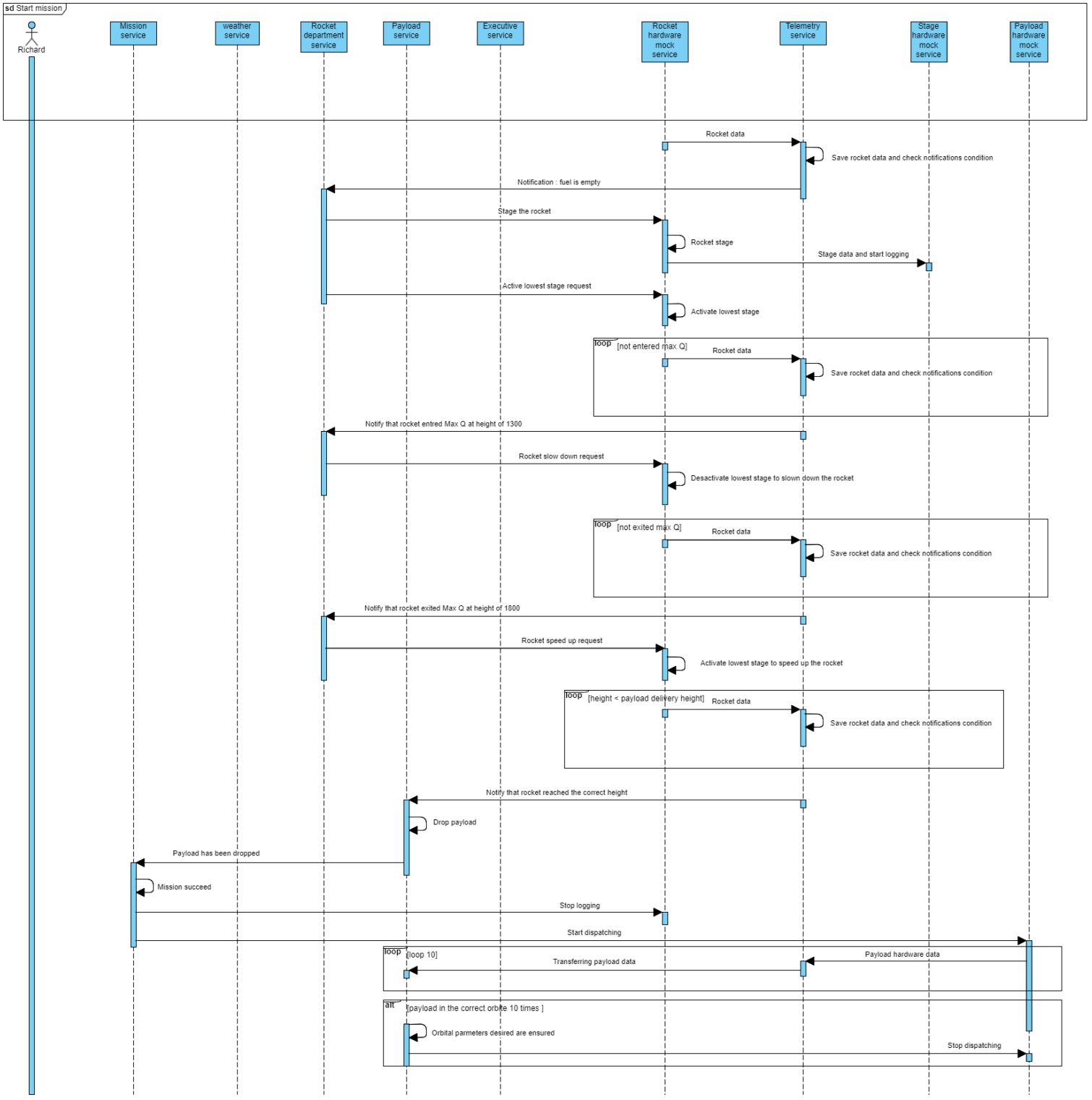

Figure 4. - Diagramme de séquence de la mission finit avec succès (disponible ici en grand format)

Lorsque le premier étage de la fusée est largué, il va envoyer ses informations à *telemetry-service*. Lorsque le étage atteint une altitude inférieur à 300 m, *stage-hardware-mock-service* va activer son parachute et permettre au stage d'atterrir sans encombre.
Ensuite, lorsque le stage a atterri, *telemetry-service* va prévenir *executive-service* que le stage a correctement atterrit sur Terre.

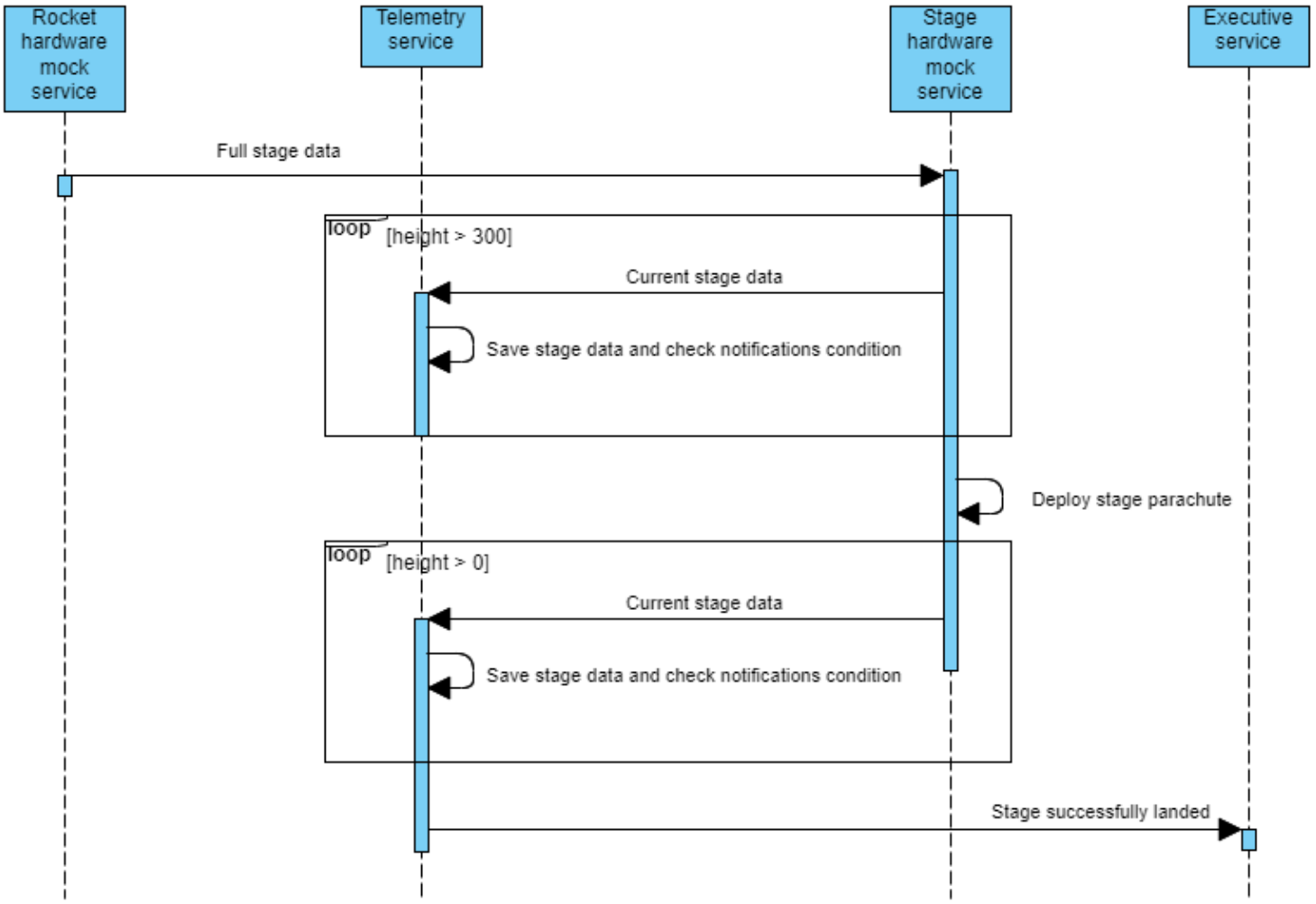

Figure 5. - Diagramme de séquence de l'atterrissage du premier stage

2. Échec de la mission dû à un sabotage de la fusée

Le début de la mission est le même que durant le scénario de la mission réussite. Les explications détaillées de ce scénario commencent après le lancement de la fusée, pendant que la fusée est en vol et qu'elle n'a pas encore terminé sa mission.

Durant le vol de la fusée, le maléfique Elune Mars est venu saboter la fusée. *rocket-hardware-mock-service* détecte alors que la fusée à une sévère anomalie. Lorsque *telemetry-service* reçoit les données de la fusée, il prévient *mission-service*. *mission-service* va alors envoyer l'ordre à la fusée de se détruire afin d'éviter tout dégât collatéral que l'anomalie pourrait provoquer.

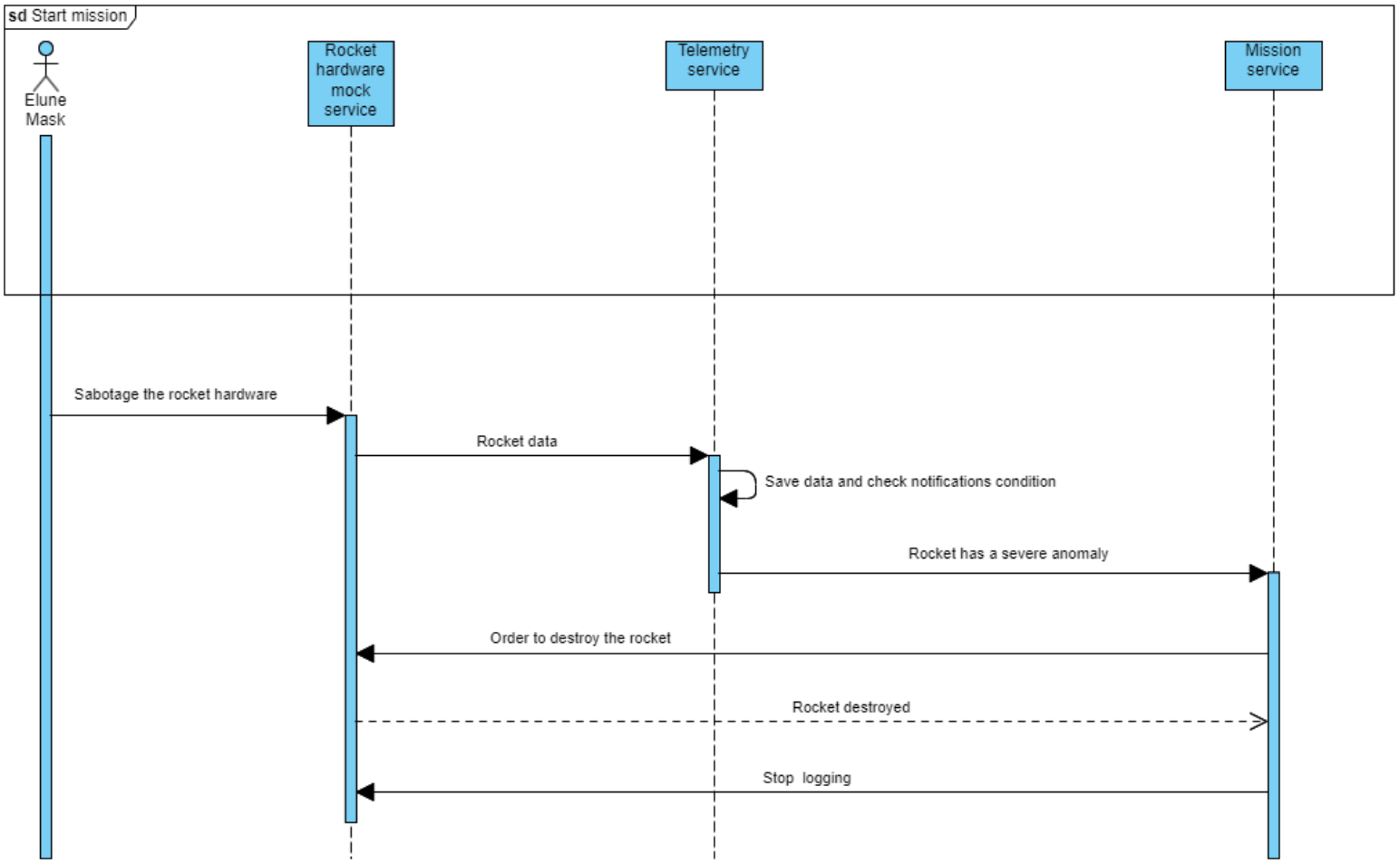

Figure 6. - Diagramme de séquence de la fusée ayant une sévère anomalie

III. Compréhension et interprétation du sujet

1. Compréhension du sujet et choix de conception

Le projet MarsY consiste à modéliser la gestion d'une fusée en utilisant plusieurs micro-services. En plus de la création de nos services, nous avons dû simuler plusieurs composants matériels et prendre des décisions sur leur fonctionnement. L'objectif était de maintenir la cohérence avec le fonctionnement réel de ces composants tout en répondant aux besoins de nos différents services. Pour le service rocket-hardware-mock, nous avons décidé qu'il renverrait diverses informations sur la fusée : le niveau de carburant de chaque étage, sa hauteur, sa vitesse et son accélération. Ces informations sont transmises toutes les demi-secondes au service de télémétrie pour simuler un signal récurrent envoyé par la fusée vers la Terre. Un autre composant que nous avons simulé est le service stage-hardware-mock, représentant le premier étage de la fusée qui retourne sur Terre. Nous avons choisi de lui attribuer son propre service car nous avons considéré que les deux composants de la fusée étaient totalement séparés pendant cette phase. Pour simuler cette séparation lors du largage du premier étage, le service rocket-hardware-mock envoie une requête à stage-hardware-mock pour l'informer que le largage a eu lieu et qu'il est désormais en autonomie. Ce service suit le même schéma d'envoi d'informations périodiques vers le service de télémétrie et il en va de même pour le troisième composant matériel, le payload-hardware-mock. Ce dernier envoie des positions plus détaillées, car il est nécessaire de pouvoir suivre la trajectoire du payload par la suite.
Nous avons également dû faire des choix concernant la découpe des services que nous devions implémenter. Nous avons créé nos différents services en suivant les différents acteurs qui constituent l'expédition : Telemetry, Payload, Weather, Rocket, Executive, et Mission. Les décisions que nous avons prises étaient basées sur les responsabilités de chaque service et la cohérence des actions qui y sont effectuées. Par exemple, nous avons envisagé de créer deux services distincts, un pour le lancement de la fusée (rocket-launch) et un pour le largage d'un étage (rocket-stage). Cependant, nous avons opté pour un seul et unique service rocket, car nous estimons que ces deux phases auront actuellement trop peu de responsabilités pour justifier la création de deux services distincts. Il était donc cohérent qu'un seul service gère les différentes phases de la fusée.

Enfin, il était nécessaire de faire des choix concernant le déroulement de la mission elle-même. Nous avons décidé que la fusée larguerait le payload à une hauteur de 3600 et qu'elle cesserait d'accélérer entre 1300 et 1800 mètres afin de passer sans problème la zone de Max Q. Pour le payload, nous avons convenu que nous pourrions considérer qu'il était sur la bonne trajectoire si 10 données consécutives émises par le payload-hardware-mock étaient comprises entre 3500 et 3700. Tous

ces choix ont principalement été faits dans le souci de garantir la clarté du déroulement de la simulation.

Concernant les choix technologiques, les micro-services sont développés en Java avec l'aide de Maven et du framework Spring Boot. Cela nous a permis de grandement faciliter le processus de mise en place de l'application au début du projet. L'architecture interne des micro-services est représentée par les couches de contrôleurs, composants, connecteurs et enfin répertoires.
La communication entre les différents micro-services s'effectue à travers les API des contrôleurs et l'aide du protocole HTTP REST. Nous avons choisi cette manière de faire également pour sa simplicité et son efficacité au début du projet et cela correspond parfaitement à nos besoins en termes d'échange de données entre micro-services.
La persistance a été réalisée grâce au module Spring Data JPA relié à des bases de données PostgreSQL. Cette approche nous permet d'éviter d'écrire tout le code nécessaire pour se connecter aux bases de données, faire les transactions ou les requêtes SQL et autres car JPA le fait déjà et nous offre une interface simple à utiliser.

2. Forces des micro-services

Les micro-services permettent d'avoir une architecture qui est plus à même de répondre à une demande nécessitant la gestion d'un grand nombre de requêtes. Pour l'instant, les users stories implémentés aurait pu être géré avec une application monolithe. Lorsque les services commenceront à être surchargés, la mise en place de micro-service permettra de dupliquer les services nécessitant plus de ressources. Pour prendre un exemple concret, le service de télémétrie permet de stocker les données d'une fusée et de monitorer ces données avec la mise en place d'un système de notification notamment. Celui-ci est beaucoup plus susceptible de devenir surchargé lorsqu'on pourra monitorer plusieurs fusées ou des données plus complexes. Il sera alors possible de le dupliquer sans que le weather service, par exemple, qui lui n'est pas surchargé ne le soit aussi.

Le 2ème atout des micro-services est que le métier est réparti entre différents services et que la défaillance d'un des services n'entraîne pas forcément la défaillance des autres. En effet, si par exemple le payload service est défaillant et ne peut plus larguer la cargaison, il faut que la fusée continue de pouvoir être monitorer. Évidemment certains services sont centraux et sans eux, il n'est pas possible que la mission continue en toute sécurité. Néanmoins, le fait de limiter ces points d'engorgement permet d'avoir des micro-services pouvant continuer en attendant que les autres soient réparés.

3. Faiblesses et limites

Bien que l'utilisation des micro-services nous permette de séparer notre code en petits services indépendants, elle nous force aussi à correctement définir des interfaces et des routes de communication. Nous avons choisi d'utiliser le protocole HTTP pour la communication entre nos différents services car il est simple à mettre en place et nous permet de transférer des données.

Toutefois, il aurait pu être intéressant d'utiliser des WebSocekts puisque ce protocole est bidirectionnel, il serait donc utile pour les demandes et les réceptions de notification d'événements. Mais les WebSockets sont aussi et surtout persistants, dès lors qu'une connexion est établie, elle reste ouverte jusqu'à ce qu'une des parties la ferme. Cela permet aux micro-services d'échanger des données en continu, sans avoir à établir une nouvelle connexion pour chaque message, là où dans notre architecture, nous utilisons des requêtes pour chaque message de la fusée au service de télémétrie.

Enfin, comme visible sur la *Figure 2*, le service de télémétrie est central à notre application et peut constituer un Single Point Of Failure. Si celui-ci venait à être dysfonctionnel, plus aucune communication sur les données de la fusée vers la base spatiale sur Terre serait possible. La seule option envisageable serait alors de détruire la fusée pour éviter son comportement imprévisible lié à la perte de connaissance de ses métriques.