

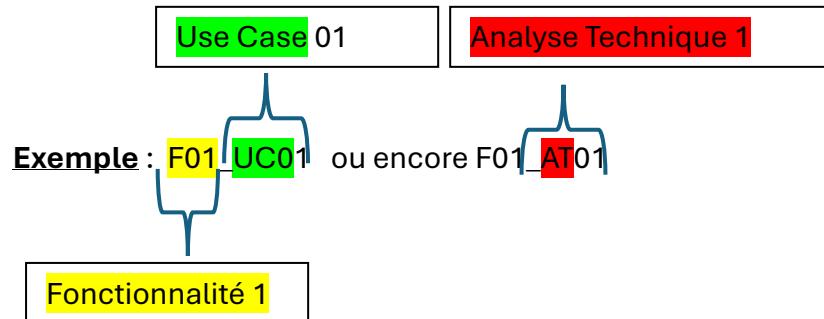
Table des matières

Avant-propos	4
Business Process Modeling.....	5
Fonctionnalités :	5
F01 Login Authentication	5
<i>F01_UC00 : Login</i>	5
F02 Gestion des zones de travail et plant (Work Area)	6
<i>F02_UC03 : édition d'un plant</i>	7
<i>F02_UC04 : Création d'une zone de travail</i>	8
<i>F02_UC05 : Edition d'une zone de travail</i>	8
F03 Gestionnaire de tâches	9
Modèle entité-association	10
Schéma physique de la base de données.....	10
Structure des autres systèmes de persistance de l'information (si applicable).....	10
Prototypage Les prototypes des principaux écrans de l'application	11
Choix techniques	11
F01 Login Authentication	11
Solution retenue.....	11
Justification	11
F02 Gestion des zones de travail et plant	12
Solution retenue.....	12
Justification	12
F03 Gestionnaire des tâches	12
Solution retenue.....	12
Justification	12
Gestion de l'exécution et modélisation des tâches	15
Exemples de spécificités selon le type de tâche.....	15
Utilisation d'un champ JSON	15
Alternative : Modèle polymorphe avec tables spécialisées.....	16
Alternative : Modèle EAV / Propriétés dynamiques (Key/Value)	16
Recommandation adaptée.....	16
Architecture de l'application	17

Conclusion de la phase d'analyse technique	17
Annexes.....	18
Figure 1: F01_UC00:BPMN	19
Figure 2:F01_ UC01: Log off.....	20
Figure 3 : F01_ UC02 : Register	21
Figure 4: F01_ UC02 : Password recovery.....	22
Figure 5 : F02_UC01 : Création d'un plant.....	23
Figure 6 : F02_UC02 : Suppression d'un plant	24
Figure 7 : F02_UC03 : édition d'un plant	25
Figure 8: F02_UC04 : Création d'une zone de travail	26
Figure 9: F02_UC05 : Edition d'une zone de travail.....	27
Figure 10 : F02_UC06 : Suppression d'une zone de travail.....	28
Figure 11 : F03_UC01 : Séquencement d'un groupe de tâches	29
Figure 12: F03_UC02 : Cycle de vie d'une tâche	30
Figure 13 : F03_UC03 : Tâche de transport.....	31

Avant-propos

Pour maintenir un fil conducteur entre le rapport et l'ensemble des analyses que j'ai menées, chaque étude de fonctionnalité fera référence à la documentation dans le bookstack via l'identification technique.



Business Process Modeling

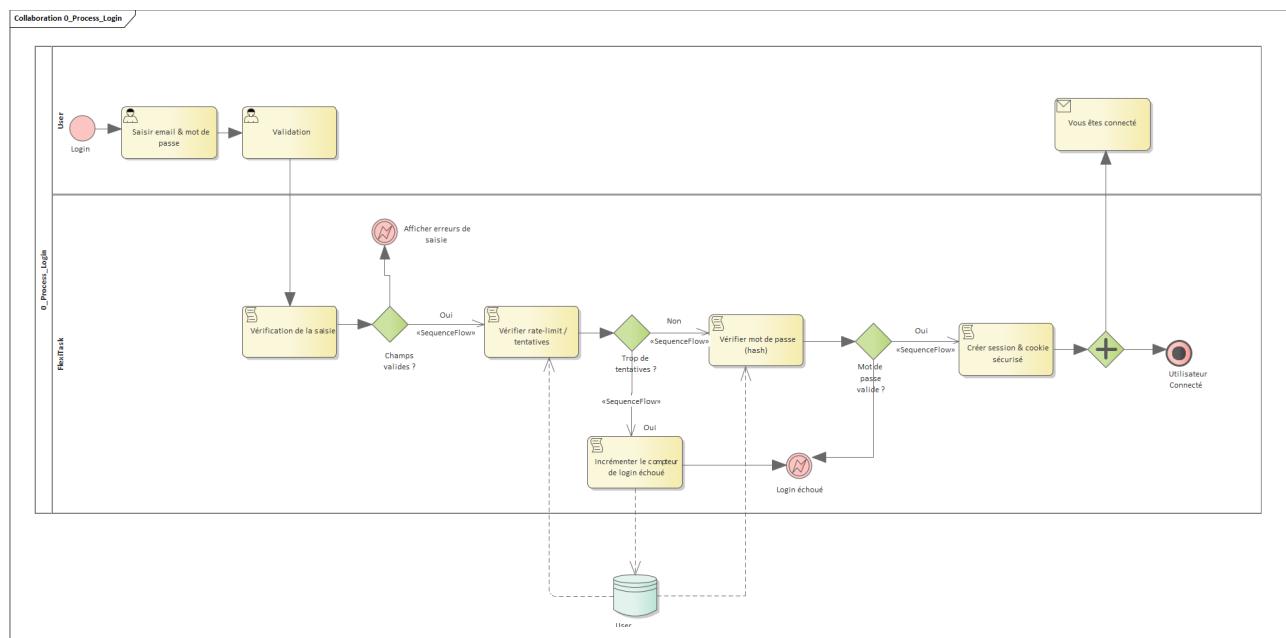
Je vais représenter ici l'ensemble des process modeling dans le cadre du poc et même un petit peu plus large, car l'ensemble des tâches qui devront être exécutée sont dépendent d'un moteur d'exécution de tâches que je défini dans la [F03_UC02](#)

Fonctionnalités :

F01 Login Authentication

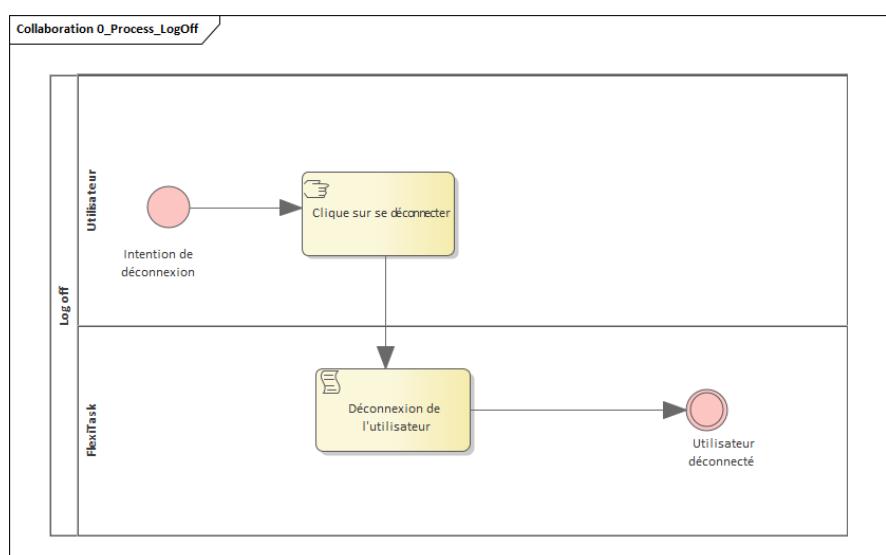
La fonctionnalité Login & Authentication permet d'assurer que seuls les utilisateurs autorisés puissent accéder à l'application en vérifiant leur identité via un nom d'utilisateur et un mot de passe, garantissant ainsi la sécurité et la confidentialité des données.

F01_UC00 : Login



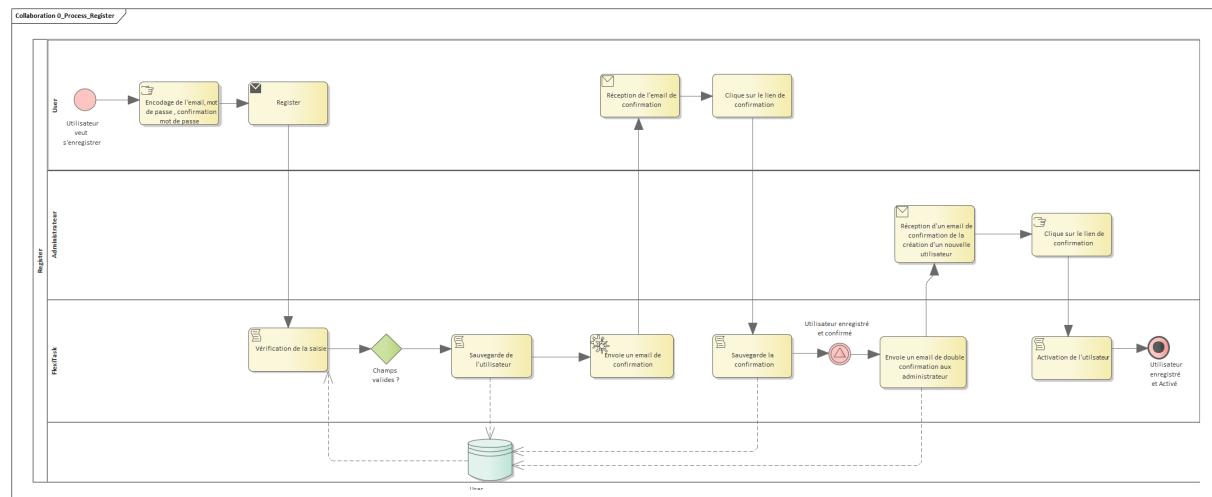
F01_UC01: Log off

La fonctionnalité log off intervient lorsque l'utilisateur souhaite quitter l'application ou mettre fin à sa session en cours. Elle garantit que l'accès à l'interface ne reste pas ouvert sur un poste laissé sans surveillance, ce qui pourrait entraîner un accès non autorisé à des données sensibles.

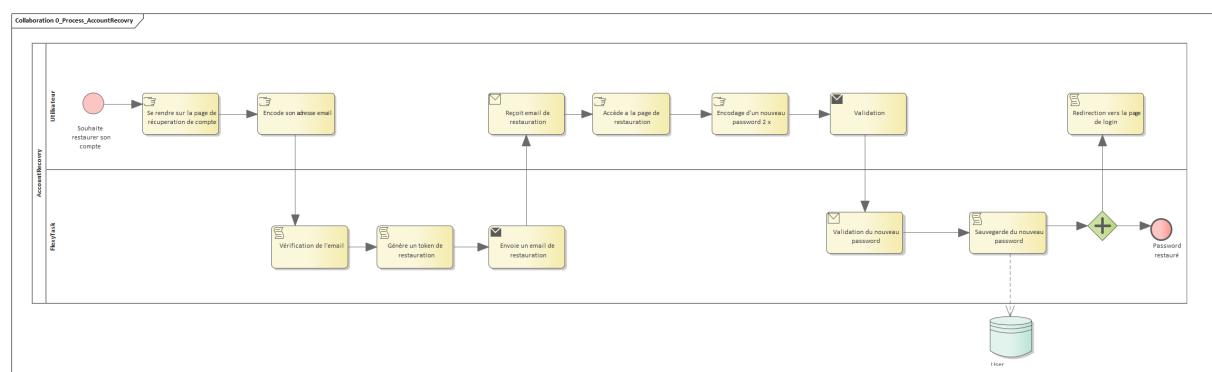


F01_UC02 : Register

La fonctionnalité Register permet à un nouvel utilisateur de créer un compte dans l'application. Elle garantit que l'identité de l'utilisateur est authentique en passant par une validation par e-mail, et que seules les personnes autorisées peuvent activer un compte.



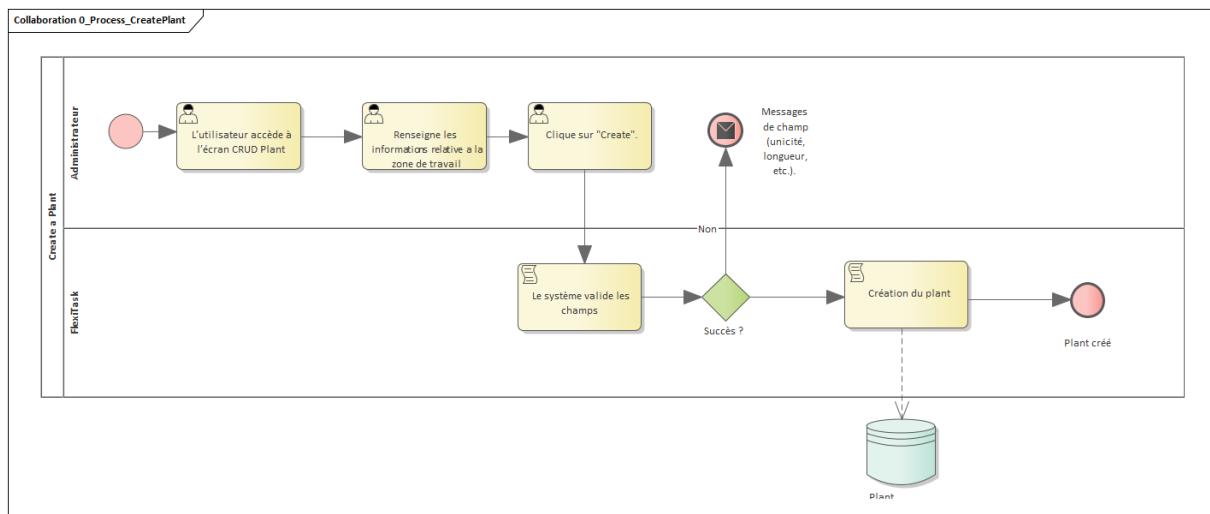
F01_UC02 : Password recovery



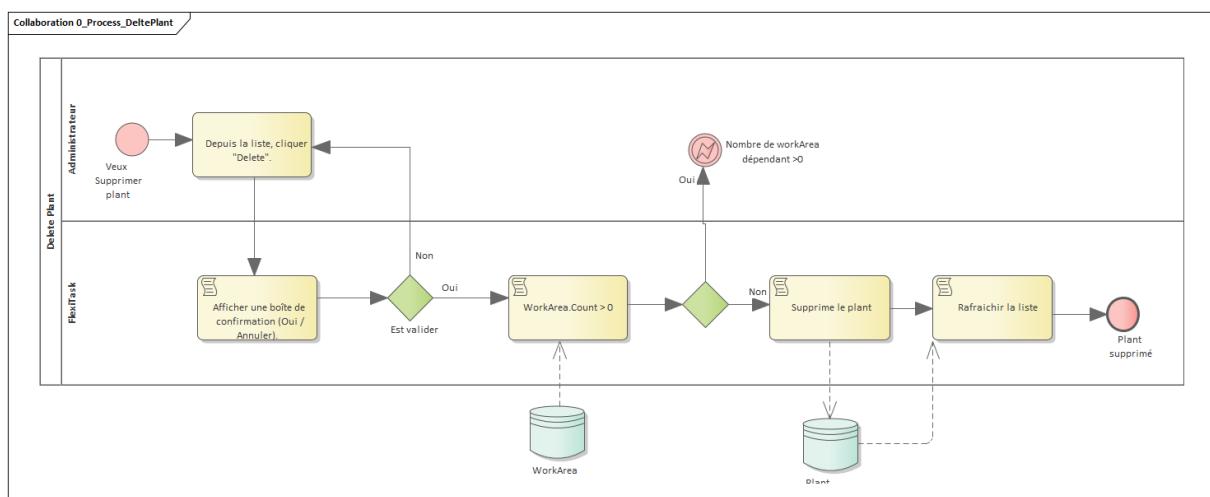
F02 Gestion des zones de travail et plant (Work Area)

Cette fonctionnalité permet gérer l'ensemble des différentes entités qui représente informatiquement les emplacements physiques au sein d'une usine càd plant et zone de travail

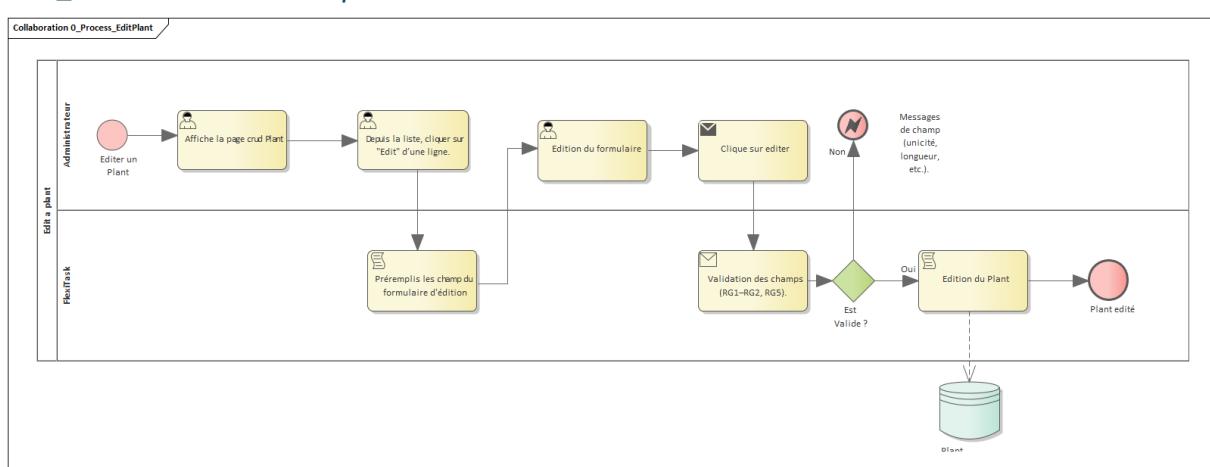
F02_UC01 : Création d'un plant



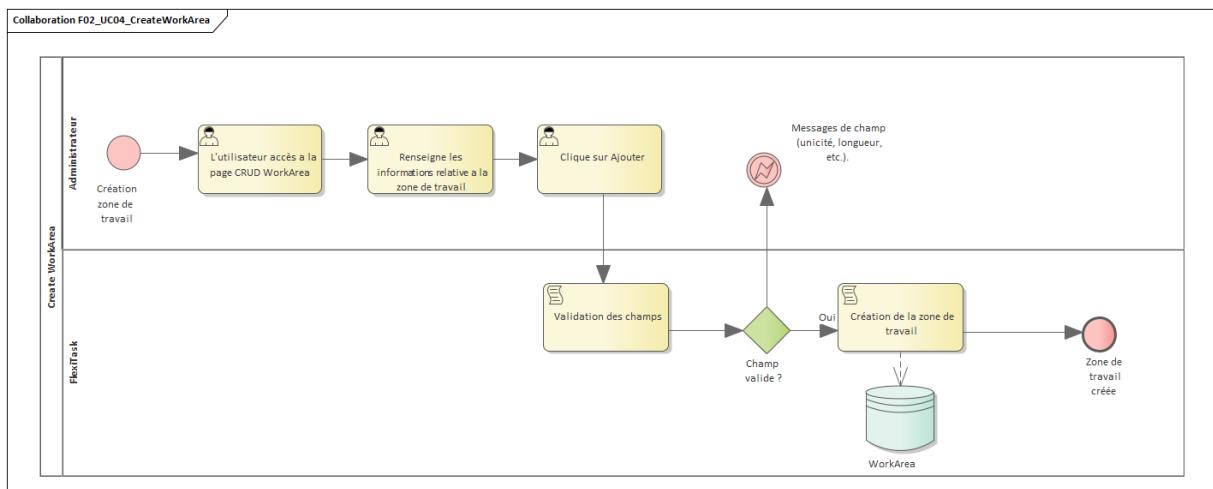
F02_UC02 : Suppression d'un plant



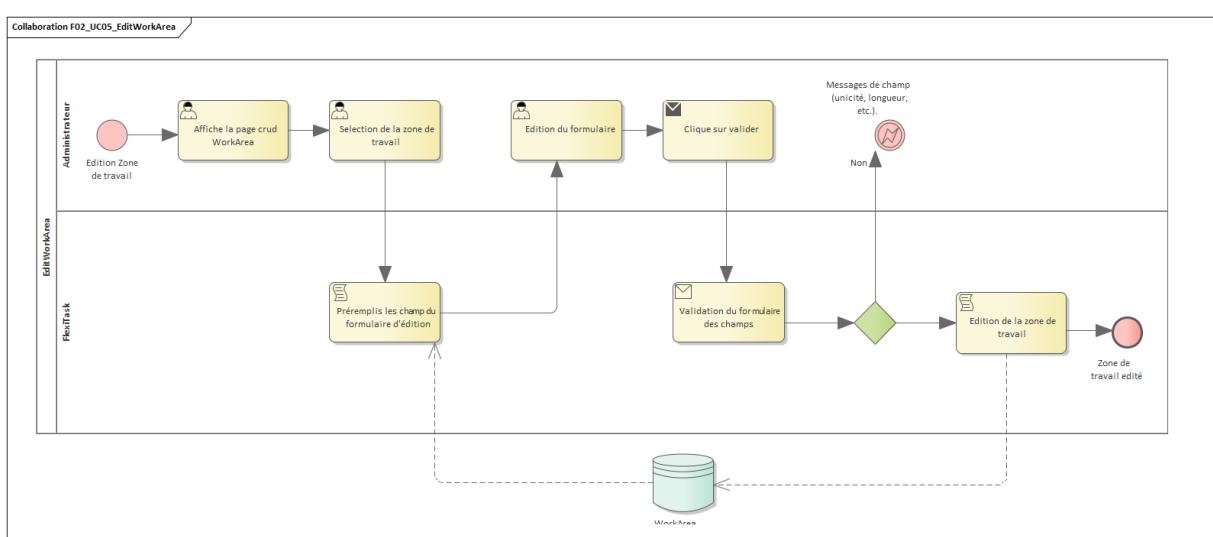
F02_UC03 : édition d'un plant



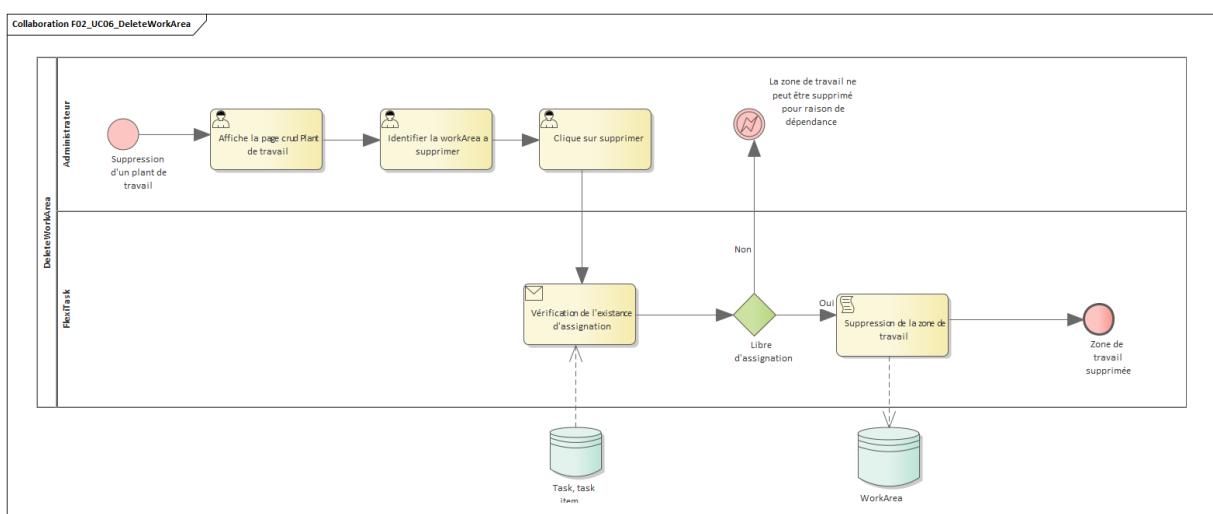
F02_UC04 : Création d'une zone de travail



F02_UC05 : Edition d'une zone de travail



F02_UC06 : Suppression d'une zone de travail

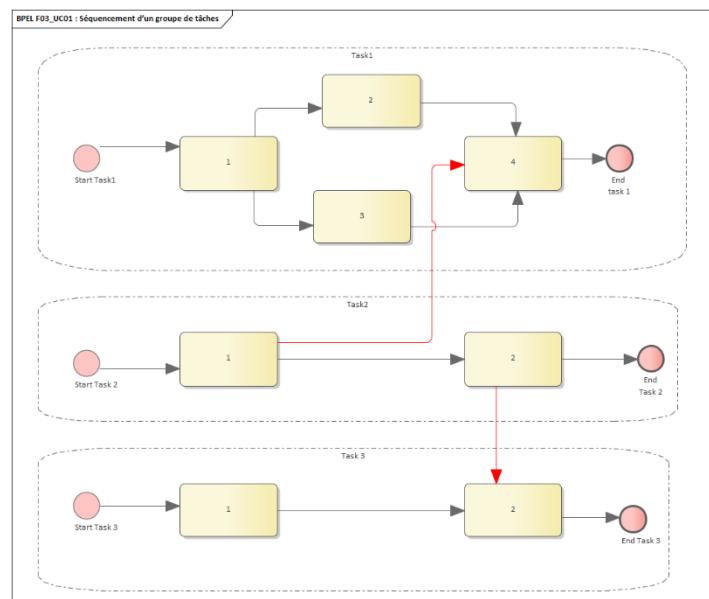


F03 Gestionnaire de tâches

La plupart des tâches exécutées dans l'usine reposent sur des modèles de données hétérogènes, provenant de sources multiples et spécifiques à chaque type d'activité. Afin de simplifier leur orchestration, l'objectif est d'uniformiser la manière dont les tâches sont traitées en les considérant comme des porteurs d'information génériques. Chaque tâche devient ainsi un conteneur standardisé, quel que soit son modèle de données d'origine, et peut être exécutée selon une timeline définie, incluant des séquences, des parallélismes et des dépendances. Cette approche permet de rendre l'exécution des tâches plus flexible, évolutive.

F03_UC01 : Séquencement d'un groupe de tâches

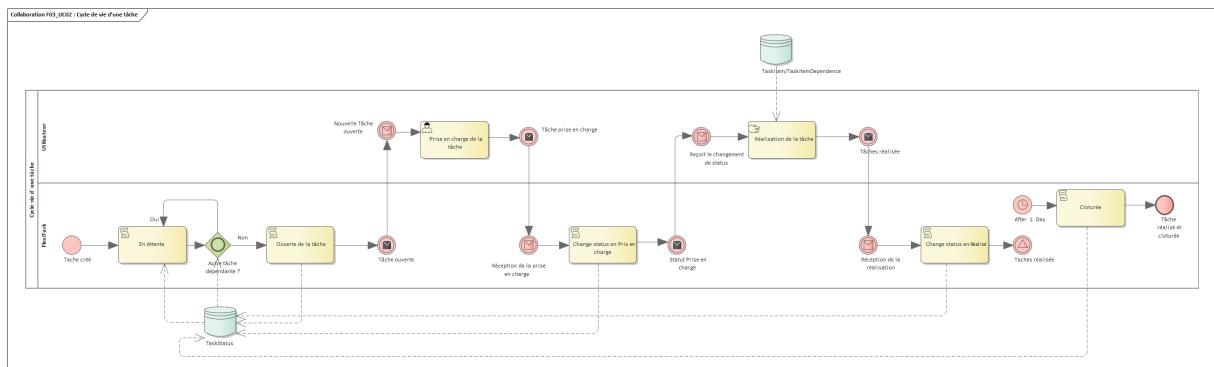
Dans un entrepôt industriel, les activités sont réparties entre plusieurs acteurs et doivent s'enchaîner selon une logique précise. Certaines tâches doivent être réalisées en série, tandis que d'autres peuvent s'exécuter en parallèle. De plus, des dépendances critiques existent entre tâches : aucune tâche ne peut démarrer tant que la ou les précédentes dont elle dépend n'ont pas été entièrement exécutées et validées. (faire le lien avec analyse technique et choix)



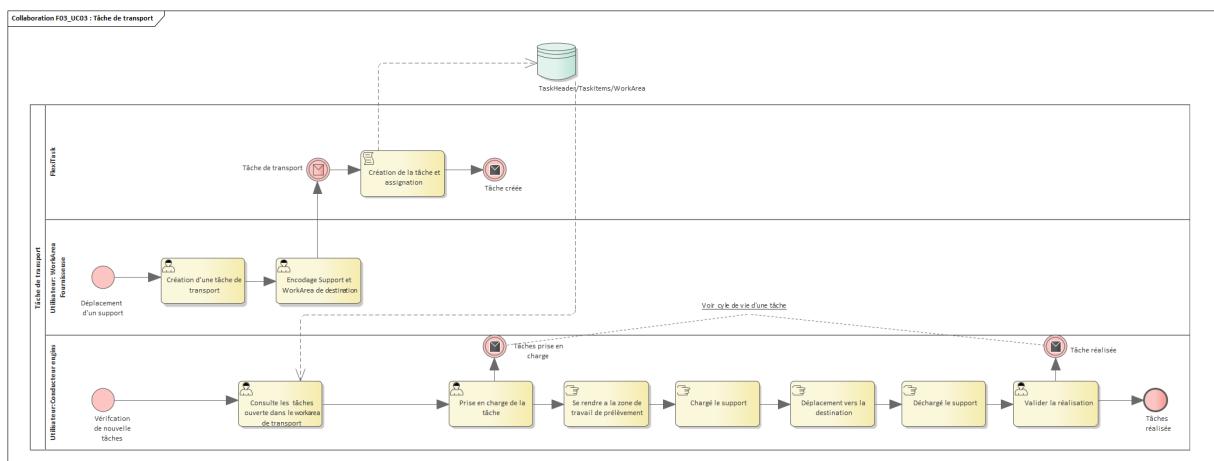
F03_UC02 : Cycle de vie d'une tâche

Chaque tâche passe par des états principaux comme :

Statut	Description
En attente	La tâche est liée à une dépendance elle ne peut pas être prise en charge immédiatement
Ouverte	Elle peut être exécuté
Prise en charge	La tâche est en cours d'exécution
Réalisée	La tâche est réalisée mais peut encore être modifiée
Clôturée	La tâche est clôturée et archivée



F03_UC03 : Tâche de transport



Modèle entité-association

Schéma physique de la base de données

Le schéma physique de la base de données complété par la description des éventuelles contraintes d'intégrité non exprimables sur le schéma. Ce schéma peut être automatiquement généré à partir d'une première version de la base de données physique.

Structure des autres systèmes de persistance de l'information (si applicable)

Une présentation de la structure des autres systèmes de persistance de l'information éventuellement mis en œuvre, comme par exemple la structure des index Elastic Search (format JSON) avec les commentaires explicatifs.

Prototypage Les prototypes des principaux écrans de l'application.

Ces écrans seront complétés par des explications utiles à leur compréhension, en lien avec les processus métiers et les entités de données. Pour créer ces prototypes d'écrans, plusieurs outils sont disponibles. Par exemple, Pencil (<https://pencil.evolus.vn/>) ou encore Balsamiq Mockups, disponible gratuitement pour quelques jours (<http://www.balsamiq.com/products/mockups>). Il est également possible d'utiliser simplement Powerpoint.

Choix techniques

F01 Login Authentication

Le choix technique ce porte sur une analyse technique disponible dans [ce document](#), durant cette analyse technique j'ai mis en vis-à-vis plusieurs technologies :

- Le système d'identité asp.net
- Un système décentralisé
- Un développement maison

Et défini les différentes exigences techniques requise par l'application sur différent axes :

- **Sécurité** : Authentification avec gestion des rôles, politique de mot de passe, verrouillage après plusieurs échecs
- **Performance** : Authentification rapide
- **Maintenabilité** : utilisation d'un système standard permettant d'évoluer dans un environnement .net
- **Intégration** : être capable d'intervenir avec différentes technologies
- **Évolutivité** : être capable de la faire évoluer vers une utilisation centralisée (AD, Azure, ...)
- **Support / communauté** : une communauté disponible et une documentation complète

Solution retenue

ASP.NET Core Identity

Justification

Cette solution concilie sécurité, intégration native à .NET, maintenabilité et flexibilité, tout en restant peu coûteuse à implémenter et capable de répondre aux exigences techniques.

F02 Gestion des zones de travail et plant

Solution retenue

Justification

F03 Gestionnaire des tâches

Solution retenue

Justification

De manière générale l'ensemble des tâches seront exécuté de manière serial mais également de manière parallèle avec des moments d'attente, j'ai choisi de me focalisé sur l'exécution des tâches de manière générale et générique, chaque tâches étant différentes et devant englober des Template de donnée différentes par exemple :

Une tâches de transport devra intégrer des informations (area src , area dest , support déplacé)

Une tâches de mise en stock devra intégrer des informations (support ,area dest , localisation au sein de l'area)

La mixité de ses informations risques d'être longue et compliqué a maintenir dans la représentation systématique par une entité étendu , c'est pour cela que j'ai investigué sur des solutions afin de pouvoir stocker des informations spécifique suivant un Template défini au niveau applicatif qui sera intégré dans un champ json

Ce champ JSON permet :

- de garder une **structure flexible**
- de supporter **n'importe quel type de tâche**
- d'évoluer facilement **sans devoir modifier le schéma SQL**
- d'adapter chaque task à son Template de données

Difficulté : Difficulté de requêtage avancé , ? Filtrer les tâches du type *Transport* où areaSrc = "MOU1" ?

? Filtrer les tâches de *Stockage* où location LIKE 'RACK-A%' ?

? Dans du JSON, ça reste possible, mais :

- plus verbeux en SQL
- souvent moins performant

- dépendant des fonctions JSON du SGBD (et de leurs limites)

Reporting plus compliqué

Perte de typage fort côté code

Validation plus complexe

Modèle **polymorphique** (tables par type de tâche)

Pattern classique :

- Table Task (générique)
 - TaskID, Type, Status, CreatedAt, AssignedTo, ...
- Table TaskTransport
 - TaskID (FK), AreaSrc, AreaDest, Support
- Table TaskStock
 - TaskID (FK), AreaDest, Location, Support

Avantages :

- Requêtes SQL propres, rapides, indexables.
- Modèle clair pour tout le monde.
- Typage fort côté code (TransportTask, StockTask, etc.).

Inconvénients :

- Plus de tables / de configuration EF.
- Un peu plus verbeux à maintenir quand tu ajoutes un nouveau type de tâche.

 C'est la solution la plus "propre" côté relationnel.

Solution B – Modèle EAV / Propriétés dynamiques (Key/Value)

Pattern :

- Table Task
- Table TaskProperty
 - TaskID
 - Key
 - Value

Exemple :

TaskID	Key	Value
42	areaSrc	MOU1
42	areaDest	OUTBOUND
42	support	STL123

 Avantages :

- Pas besoin de nouvelles colonnes/tables à chaque type de tâche.
- SQL interrogeable, même si moins confortable.

 Inconvénients :

- Très vite illisible.
- Difficile pour les contraintes (NOT NULL, types, etc.).
- Ça finit souvent en “anti-pattern” si très utilisé.

Recommandation réaliste pour ton projet (et ton TFE)

Vu ce que tu m’as déjà expliqué sur ton Task Management System, je ferais :

1. **Task générique** (table Task)

- TaskID, Type, Status, Priority, AreaSrc, AreaDest, Support, CreatedAt, PlannedAt, ...

2. **Colonne JSON Details**

- Pour stocker les champs très spécifiques au scénario.

3. **Côté code (.NET)**

- Une hiérarchie :
 - Task + TaskDetailsBase
 - TransportTaskDetails, StockTaskDetails, etc.
- Sérialisation/désérialisation **fortement typée** (pas du dynamic sauvage).
- Validation avec FluentValidation ou similaire.

4. **Pour ton analyse fonctionnelle**

- Expliquer que le JSON est là pour **gérer l’hétérogénéité**, MAIS que les champs critiques sont en colonnes pour :
 - performance

- reporting
- filtrage opérationnel

Gestion de l'exécution et modélisation des tâches

De manière générale, l'exécution des tâches au sein du système peut se faire aussi bien de façon sérielle que parallèle, selon les besoins opérationnels et les éventuels temps d'attente. L'approche retenue vise à proposer une exécution générique et flexible des différentes tâches, tout en tenant compte de leur diversité intrinsèque : chaque tâche présente en effet des spécificités et nécessite l'intégration de modèles de données adaptés à son contexte.

Exemples de spécificités selon le type de tâche

- Une tâche de transport doit intégrer des informations telles que la zone source (area src), la zone de destination (area dest) et le support déplacé.
- Une tâche de mise en stock nécessite l'indication du support, de la zone de destination (area dest) et de la localisation précise au sein de la zone.

La coexistence de ces différentes structures de données rend la maintenance d'une représentation systématique par entité étendue difficile et coûteuse. Pour y remédier, une solution a été étudiée afin de stocker les informations spécifiques à chaque scénario de tâche, selon un modèle (template) défini au niveau applicatif, dans un champ de type JSON.

Utilisation d'un champ JSON

Le recours à un champ JSON permet de :

- Conserver une structure de données flexible.
- Supporter tout type de tâche, quelle que soit sa spécificité.
- Faire évoluer le modèle sans modifier le schéma SQL sous-jacent.
- Adapter chaque tâche à son propre modèle de données.

Néanmoins, cette approche présente certaines difficultés :

- Complexité accrue pour les requêtes avancées, par exemple :
- Filtrer les tâches de type Transport où areaSrc = "MOU1".
- Filtrer les tâches de Stockage où location commence par 'RACK-A'.

Les requêtes SQL sont plus verbeuses et souvent moins performantes.

La solution dépend fortement des fonctionnalités JSON du SGBD et de leurs limites.

Le reporting devient plus complexe.

La perte du typage fort côté code complique la validation des données.

Alternative : Modèle polymorphe avec tables spécialisées

Un autre modèle classique consiste à utiliser une structure polymorphe, avec une table générique pour les tâches et des tables spécialisées pour chaque type de tâche :

- Table Task (générique) : TaskID, Type, Status, CreatedAt, AssignedTo, etc.
- Table TaskTransport : TaskID (FK), AreaSrc, AreaDest, Support.
- Table TaskStock : TaskID (FK), AreaDest, Location, Support.

Avantages :

- Requêtes SQL claires, rapides et facilement indexables.
- Modèle compréhensible par tous les intervenants.
- Typage fort côté code, avec des entités distinctes pour chaque type de tâche.

Inconvénients :

- Nécessite plus de tables et une configuration Entity Framework plus importante.
- Maintenance plus fastidieuse lors de l'ajout de nouveaux types de tâches.

Cette solution reste néanmoins la plus “propre” d'un point de vue relationnel.

Alternative : Modèle EAV / Propriétés dynamiques (Key/Value)

Cette approche s'appuie sur une table de propriétés dynamiques liée à la table des tâches :

- Table Task
- Table TaskProperty : TaskID, Key, Value

Exemple :

TaskID	Key	Value
42	areaSrc	MOU1
42	areaDest	OUTBOUND
42	support	STL123

Avantages : Pas besoin d'ajouter de nouvelles colonnes ou tables pour chaque type de tâche ; le SQL reste interrogeable, bien que moins confortable.

Inconvénients : Structure rapidement illisible, contraintes et typage difficiles à gérer, et risque élevé d'aboutir à un anti-pattern si l'approche est trop généralisée.

Recommandation adaptée

Pour le contexte du Task Management System, la solution recommandée consiste à maintenir une table générique (Task) avec les champs critiques (Type, Status, Priority, AreaSrc, AreaDest, Support, CreatedAt, PlannedAt, etc.) en colonnes classiques, et à utiliser une colonne JSON (Details) pour stocker les champs très spécifiques à chaque scénario.

Côté code (.NET), cela implique la mise en place d'une hiérarchie d'objets : une entité Task associée à une base TaskDetailsBase, avec des classes dérivées telles que TransportTaskDetails, StockTaskDetails, etc. La sérialisation et la désérialisation se font de manière fortement typée, avec une validation renforcée (par exemple, via FluentValidation), afin d'éviter les problèmes liés à l'utilisation de types dynamiques.

Pour l'analyse fonctionnelle, il convient de préciser que le recours au JSON vise à gérer l'hétérogénéité des tâches, tandis que le stockage des champs critiques en colonnes vise à garantir les performances, la facilité de reporting et l'efficacité du filtrage opérationnel.

Architecture de l'application

Une explication de l'architecture générale de l'application illustrée par un diagramme de classes organisé en couches (en fonction du type d'architecture). Un parallèle sera fait avec la structure physique des sources qui sera reprise sous forme d'une copie d'écran accompagnée des explications nécessaires.

Conclusion de la phase d'analyse technique

Annexes

Figure 1: F01_UC00:BPMN	19
Figure 2:F01_ UC01: Log off.....	20
Figure 3 : F01_ UC02 : Register	21
Figure 4: F01_ UC02 : Password recovery.....	22
Figure 5 : F02_UC01 : Création d'un plant.....	23
Figure 6 : F02_UC02 : Suppression d'un plant	24
Figure 7 : F02_UC03 : édition d'un plant	25
Figure 8: F02_UC04 : Création d'une zone de travail	26
Figure 9: F02_UC05 : Edition d'une zone de travail.....	27
Figure 10 : F02_UC06 : Suppression d'une zone de travail.....	28

Figure 1: F01_UC00:BPBMN

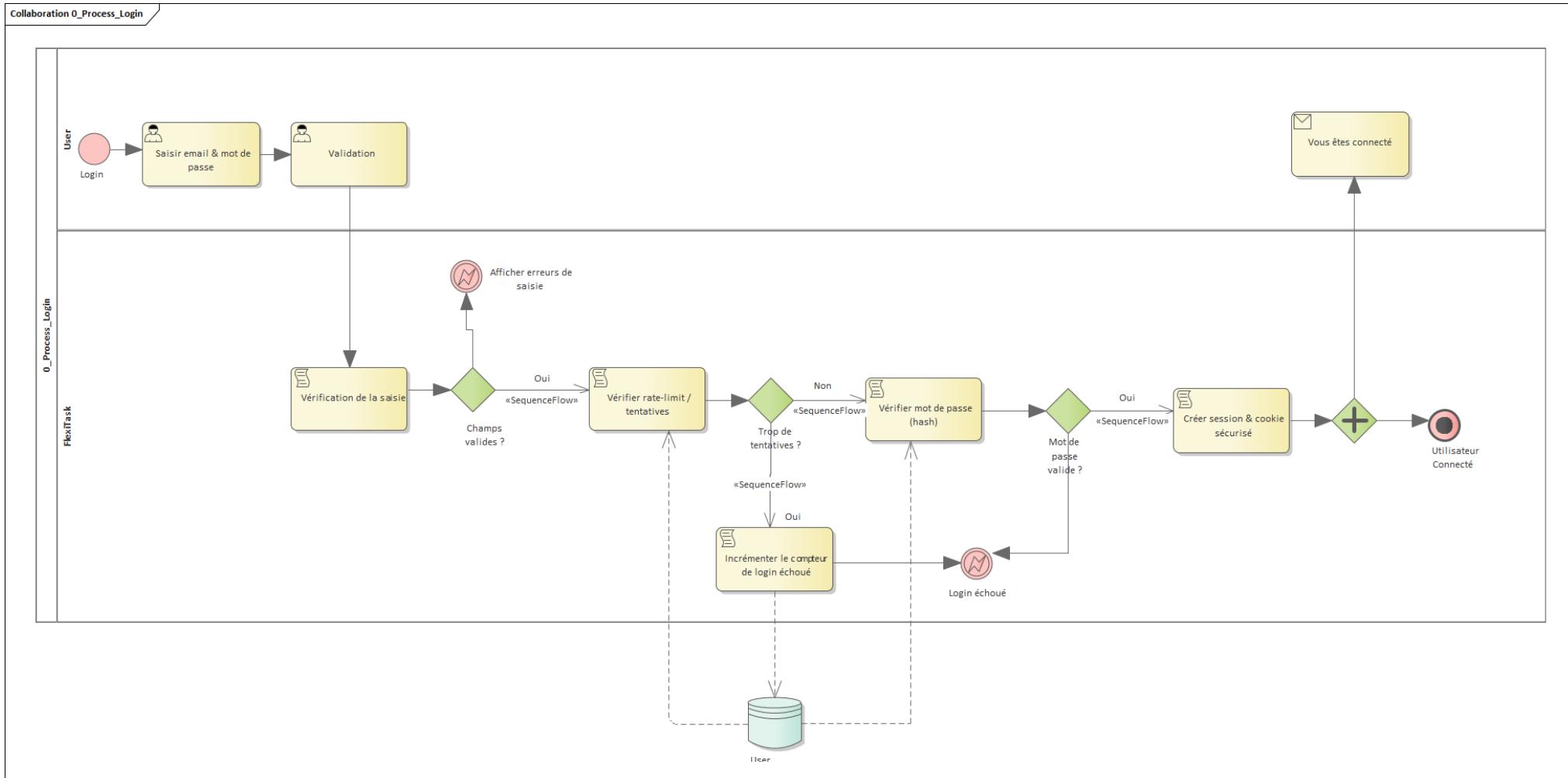


Figure 2:F01_UC01: Log off

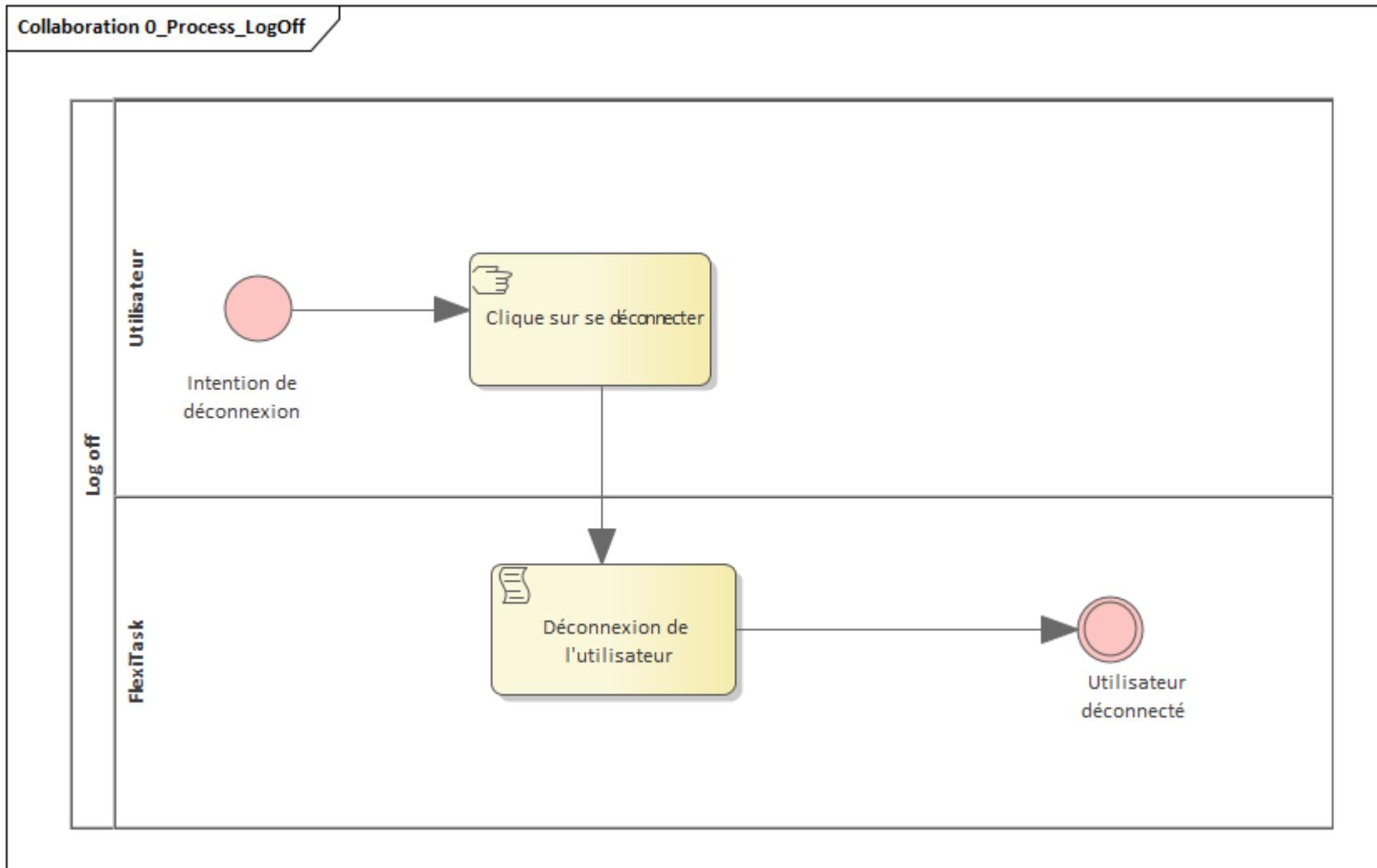


Figure 3 : F01_UC02 : Register

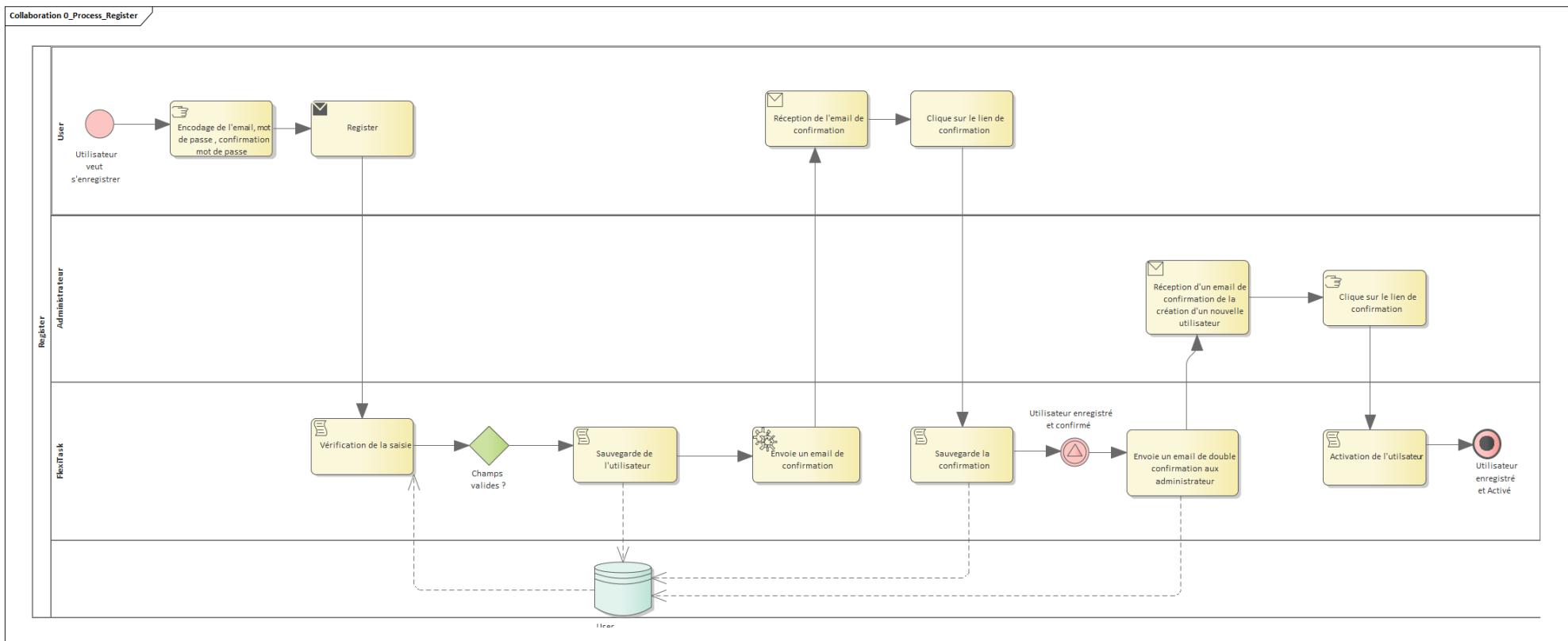


Figure 4: F01_UC02 : Password recovery

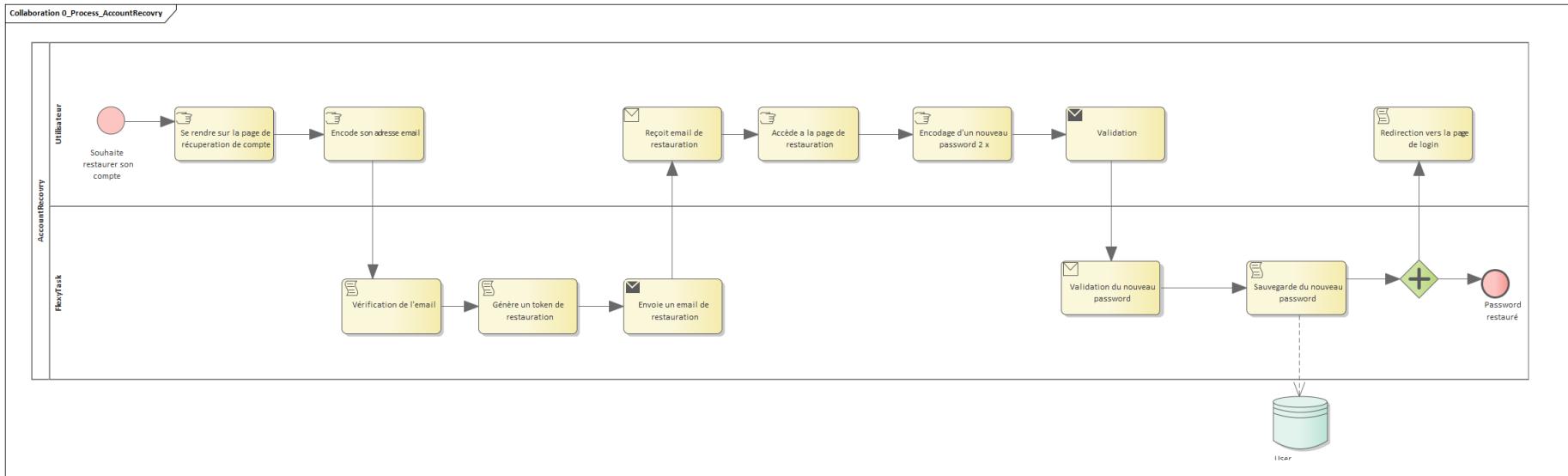


Figure 5 : F02_UC01 : Crédation d'un plant

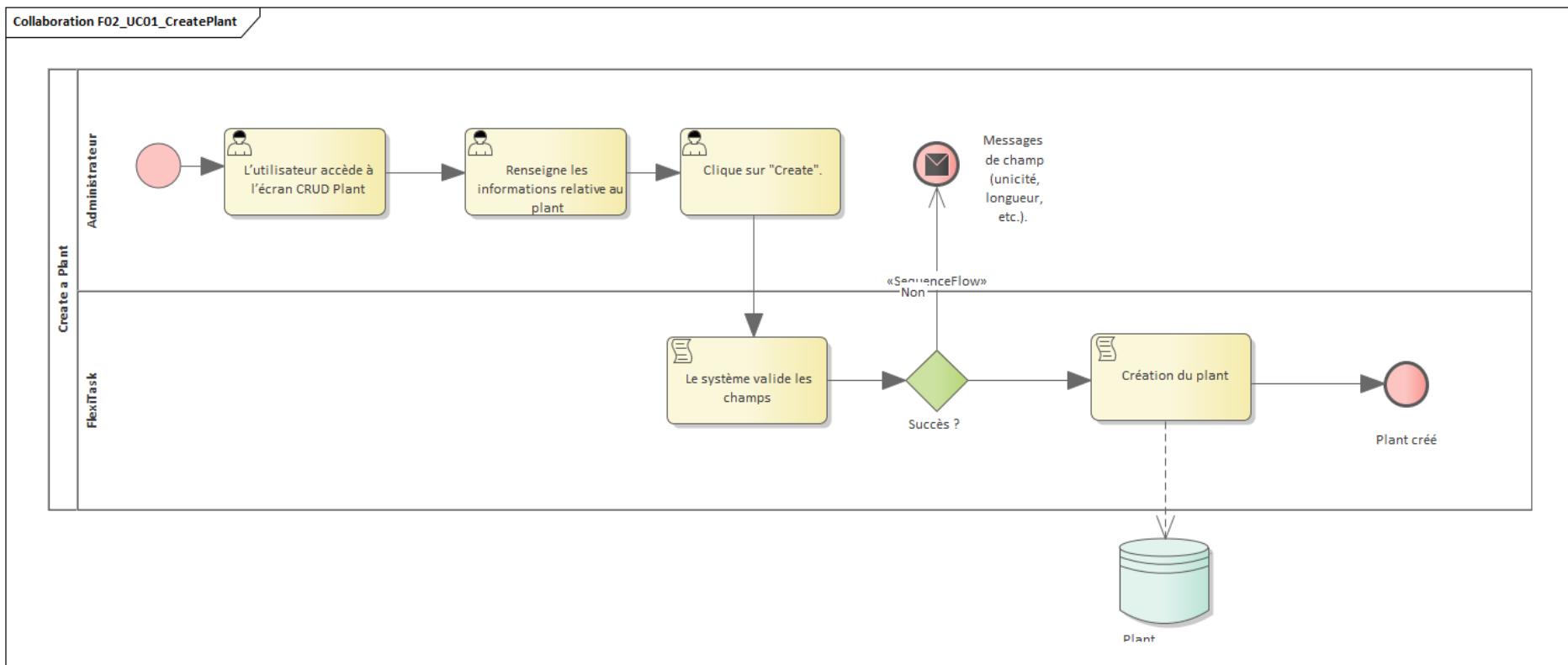


Figure 6 : F02_UC02 : Suppression d'un plant

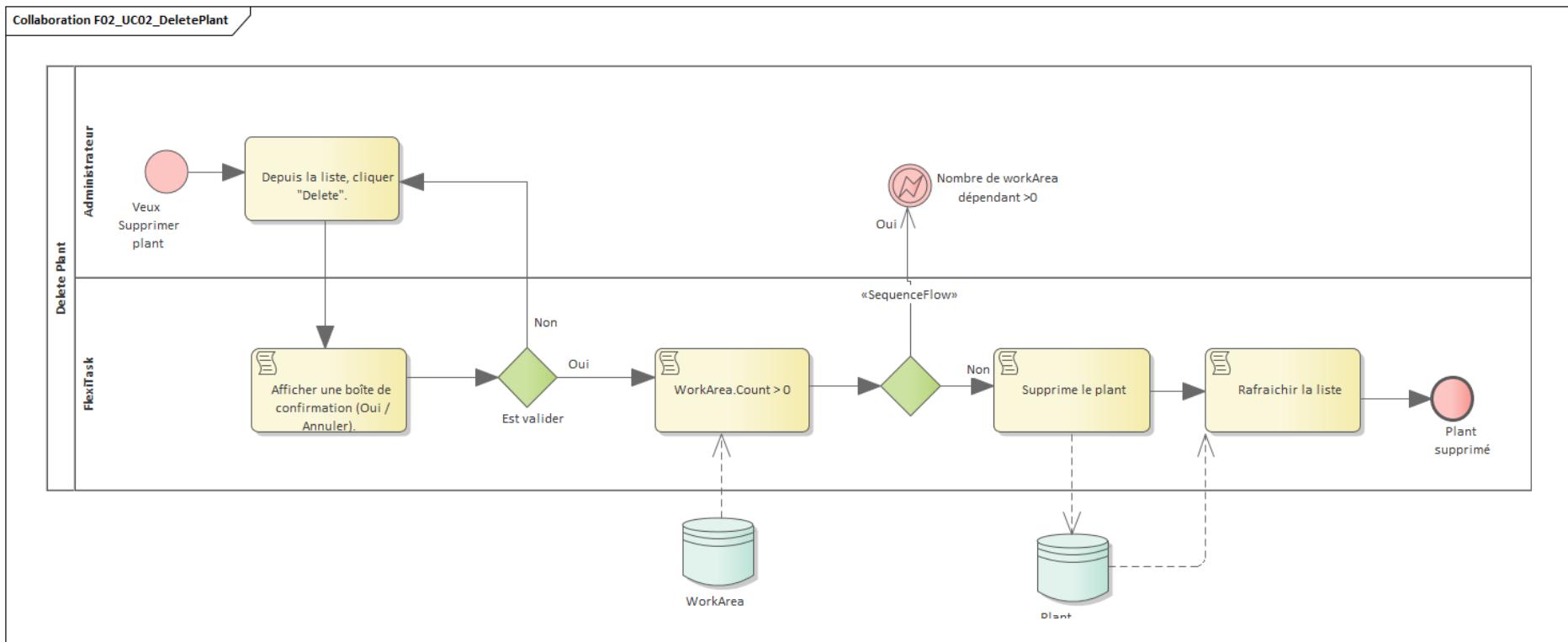


Figure 7 : F02_UC03 : édition d'un plant

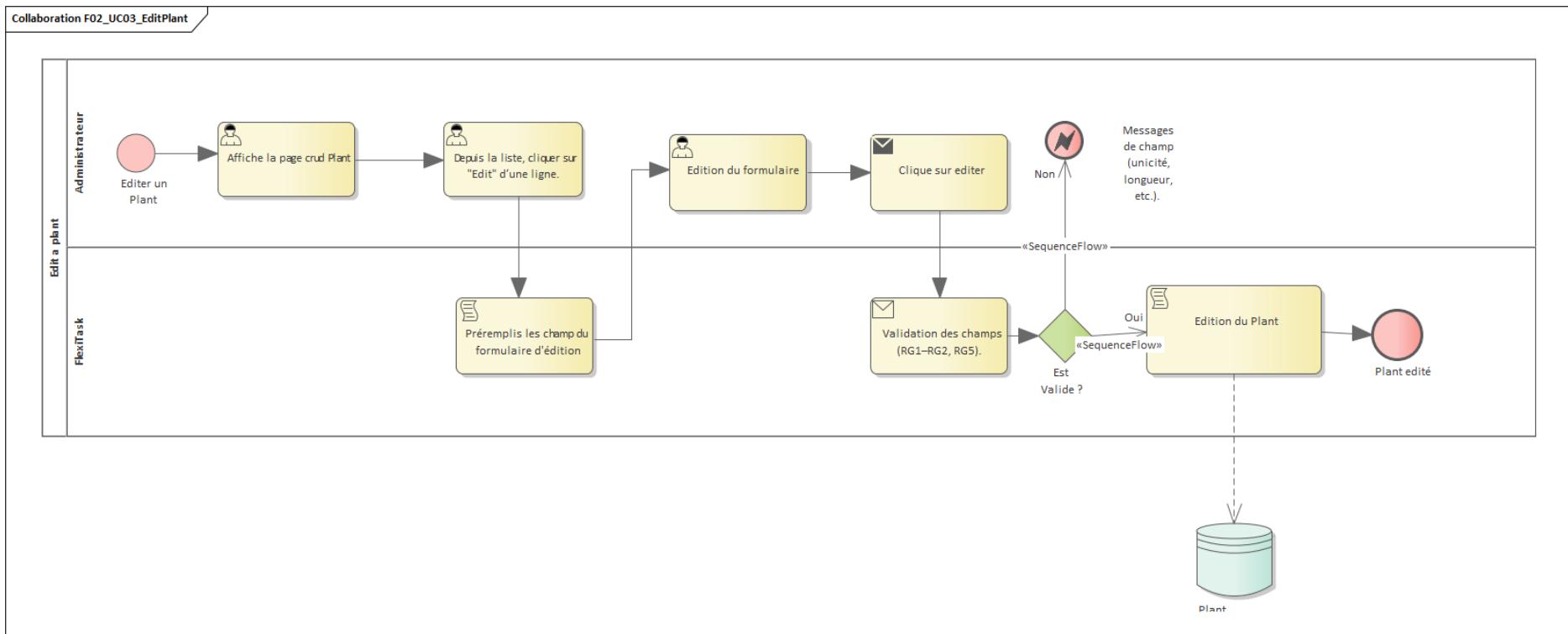


Figure 8: F02_UC04 : Crédation d'une zone de travail

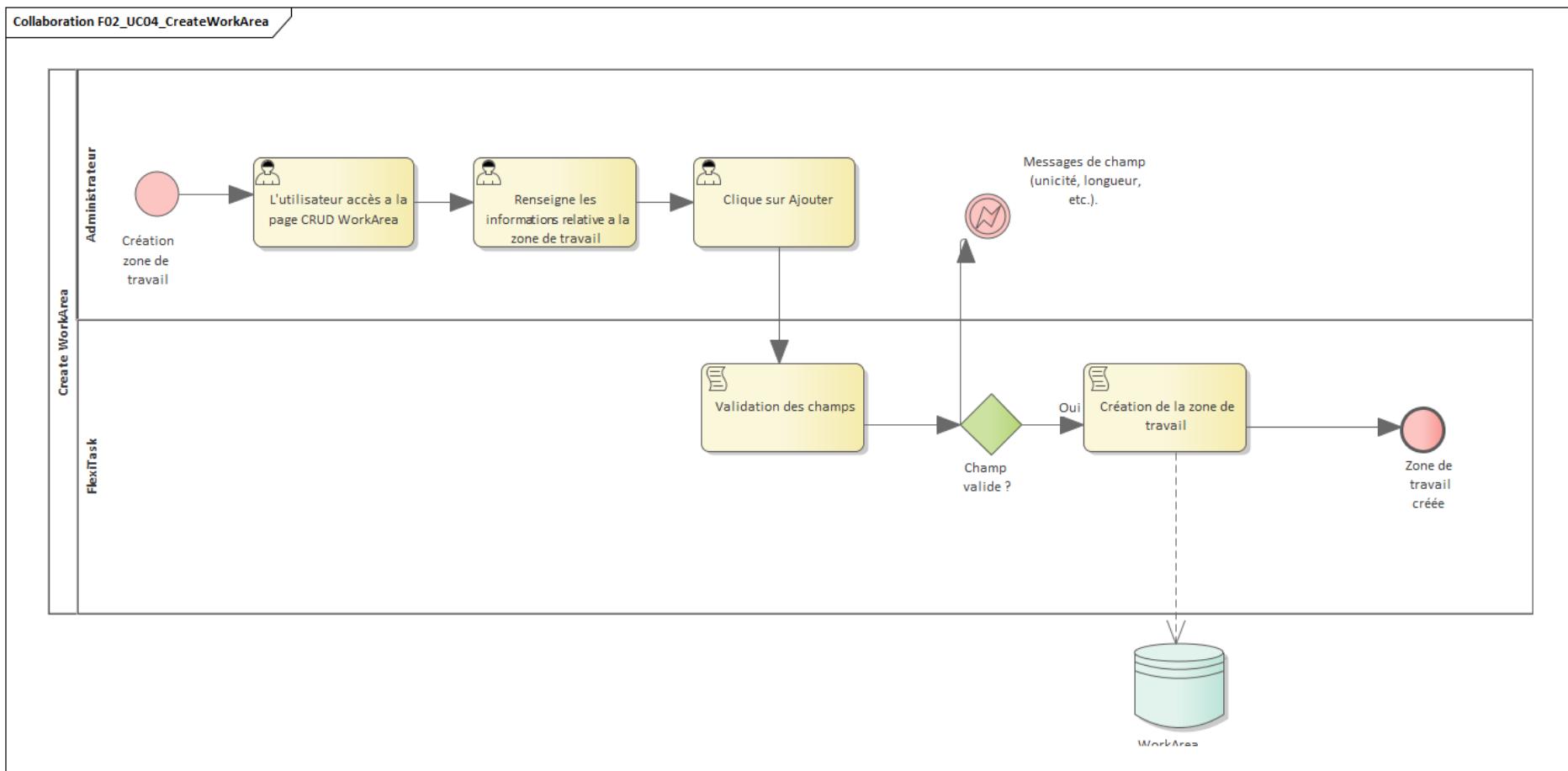


Figure 9: F02_UC05 : Edition d'une zone de travail

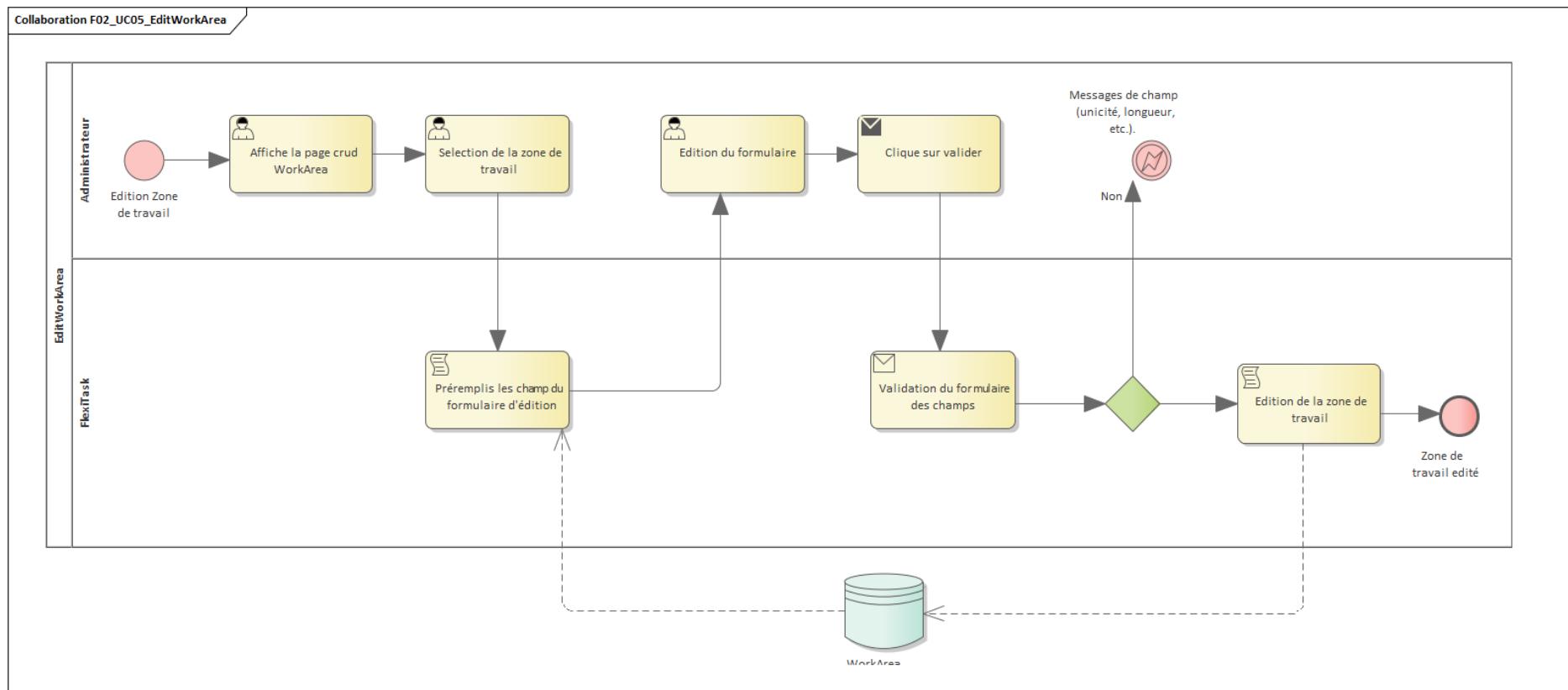


Figure 10 : F02_UC06 : Suppression d'une zone de travail

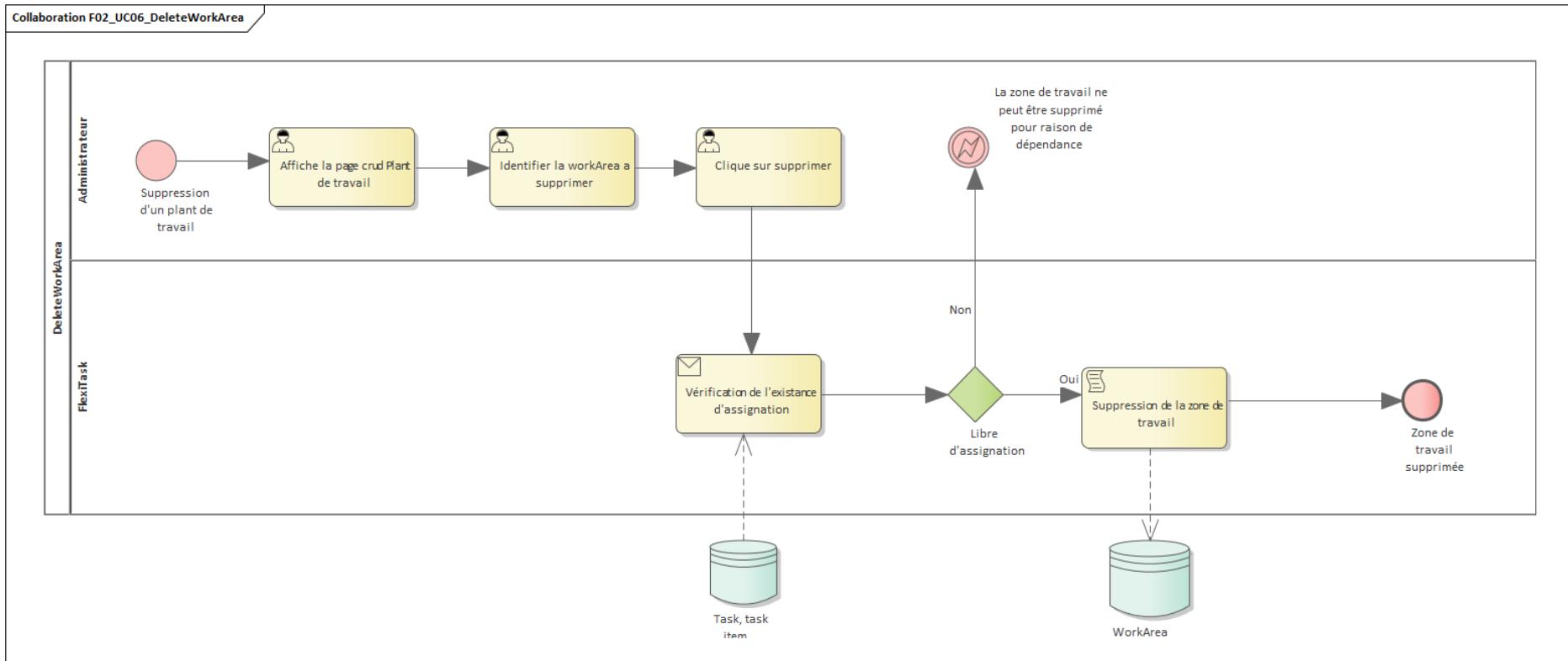


Figure 11 : F03_UC01 : Séquencement d'un groupe de tâches

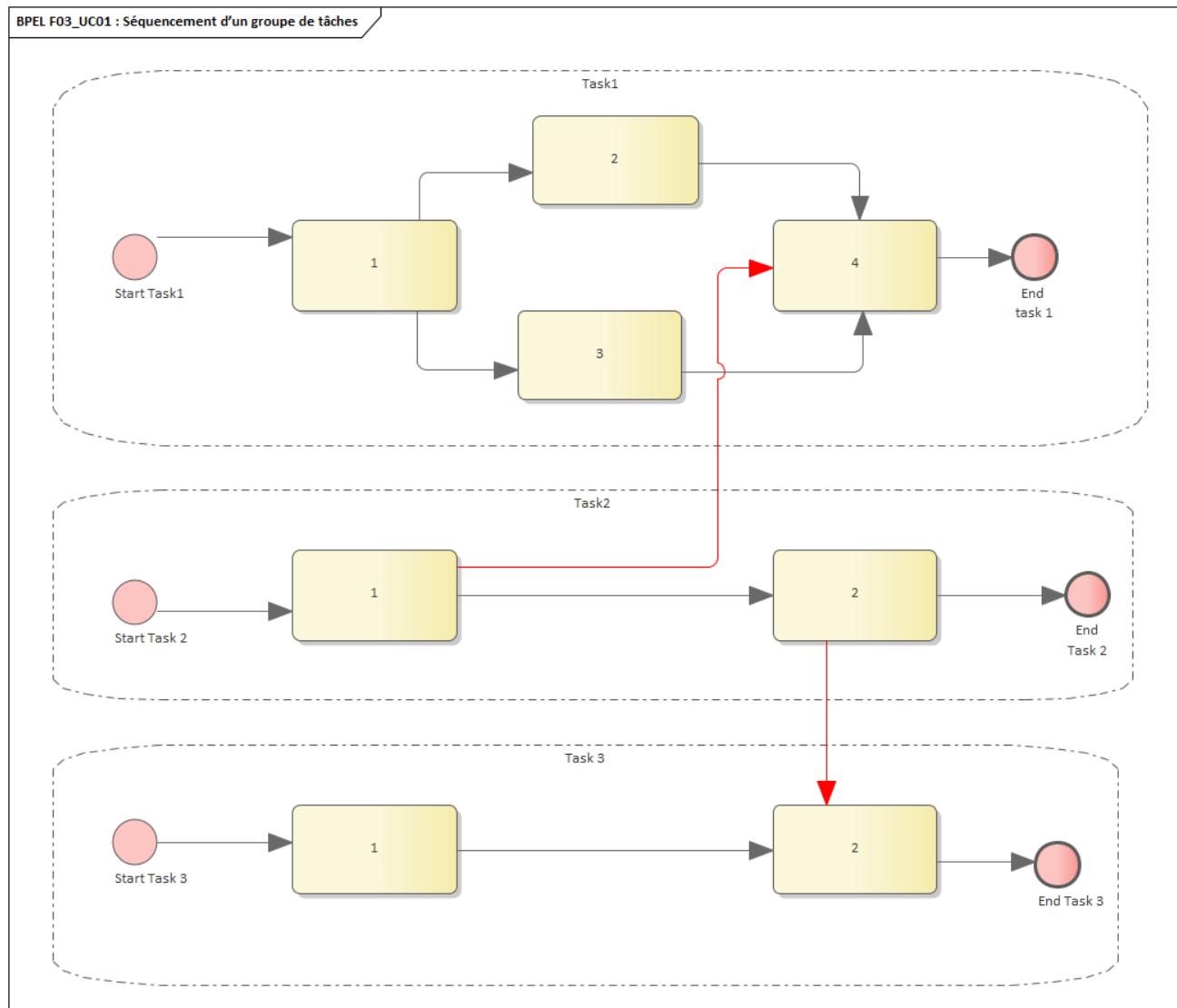


Figure 12: F03_UC02 : Cycle de vie d'une tâche

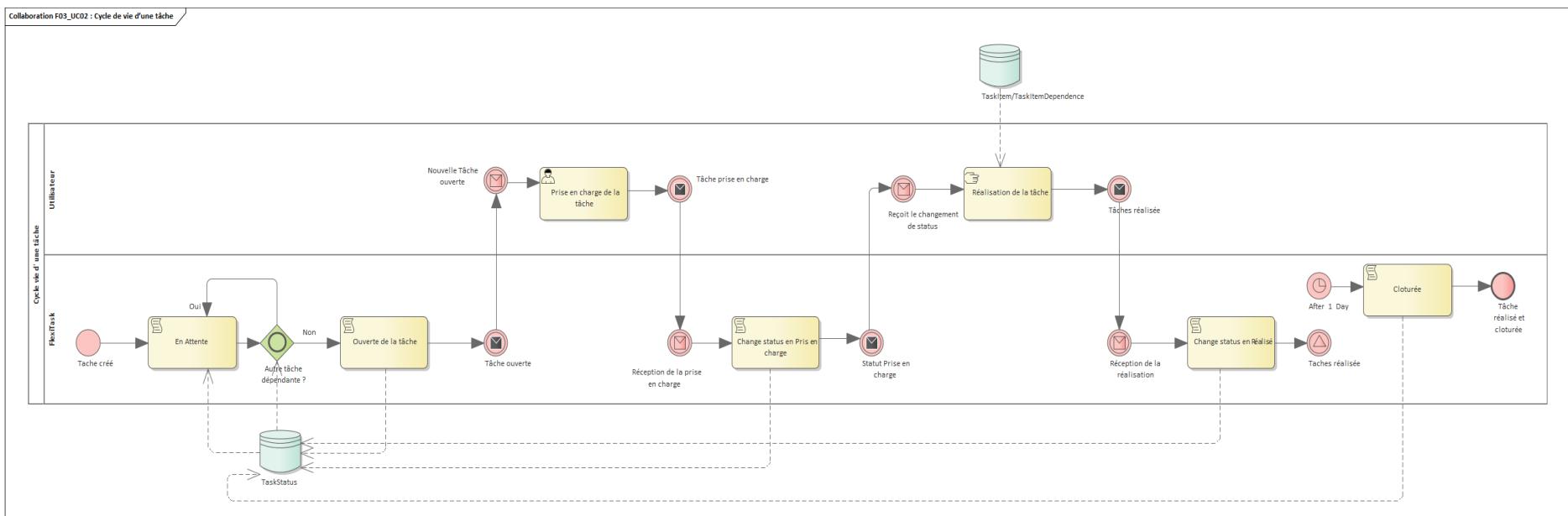


Figure 13 : F03_UC03 : Tâche de transport

