Agence ou Service : NTIC

Projet : Outil de Changement de Repère

DOSSIER DE CONCEPTION DE L'OUTIL CHANGEMENT DE REPERE TREPS

Rédigé par : Benjamin Renard	Diffusé à : CNES / IRAP 
Approuvé par : Chef de projet AKKA – N. Lormant Responsable projet CNES – N. Dufourg	

LISTE DES MODIFICATIONS DU DOCUMENT

Vers.	Date	Paragraphe	Description de la modification
01.0	05/12/13		Création du document
01.01	12/03/14		Conception détaillée

SOMMAIRE

1 INTRODUCTION

L'objectif de ce document est de présenter la conception de l'Outil de Changement de Repère, qui sera désigné par son acronyme TREPS dans la suite de ce document.

TREPS est un outil Web « léger » permettant à un utilisateur de définir, à travers une IHM ergonomique et intuitive, une opération de changement de repère sur des données qu'il fournit en entrée, dans le but d'obtenir des données de sortie correspondantes au résultat de sa transformation.

Le calcul à proprement parlé d'une transformation de changement de repère est assuré par l'appel au Web Service de l'outil CDPP/3DView.

2 CONCEPTION PRELIMINAIRE

2.1 DESCRIPTION GENERALE

L'outil TREPS est un outil Web « léger », dont l'interface graphique est accessible par un utilisateur via un navigateur, permettant de définir et d'exécuter des opérations de changement de repère.

Cet outil est hébergé sur un serveur dont le système d'exploitation est CentOS 6.3 en version 64 bits, mutualisant plusieurs autres applications du CDPP.

L'architecture de l'outil TREPS, ainsi que les langages de programmation utilisés pour le concevoir, s'inspirent très largement des travaux réalisés, ou en cours de réalisation, sur la nouvelle version de l'outil CDPP/AMDA (cf. [R1] e [R2]). Cela permet de conserver une certaine cohérence dans les développements initiés par le CDPP, tout en sachant que les mécanismes à développer pour l'outil TREPS sont similaires à ces deux outils.

2.2 ARCHITECTURE DU SYSTEME

Du point de vue architecture générale, l'outil TREPS est constitué de trois sous-applications :

- L'IHM de l'application (désignée par TREPS-IHM) : il s'agit d'une page Web, s'affichant dans le navigateur de l'utilisateur, développée en Javascript avec le framework ExtJS, permettant à l'utilisateur de définir ses opérations de changement de repère, ainsi que de visualiser et exporter les résultats,
- Le noyau de l'application (désigné par TREPS-Kernel) : il s'agit d'une application standalone, développée en C++, et s'exécutant sur le serveur afin de réaliser l'opération de changement de repère telle que définit par l'utilisateur dans l'IHM,
- Une couche logicielle de communication (désignée par TREPS-Com) : il s'agit d'un ensemble de scripts PHP, dont l'exécution est gérée par un serveur Apache, lui-même exécuté sur le serveur, et dont la responsabilité est de faire communiquer les sous-applications TREPS-IHM et TREPS-Kernel entre elles.

La figure suivante présente une schématisation très simplifiée de l'architecture globale du système, montrant les trois sous-applications :

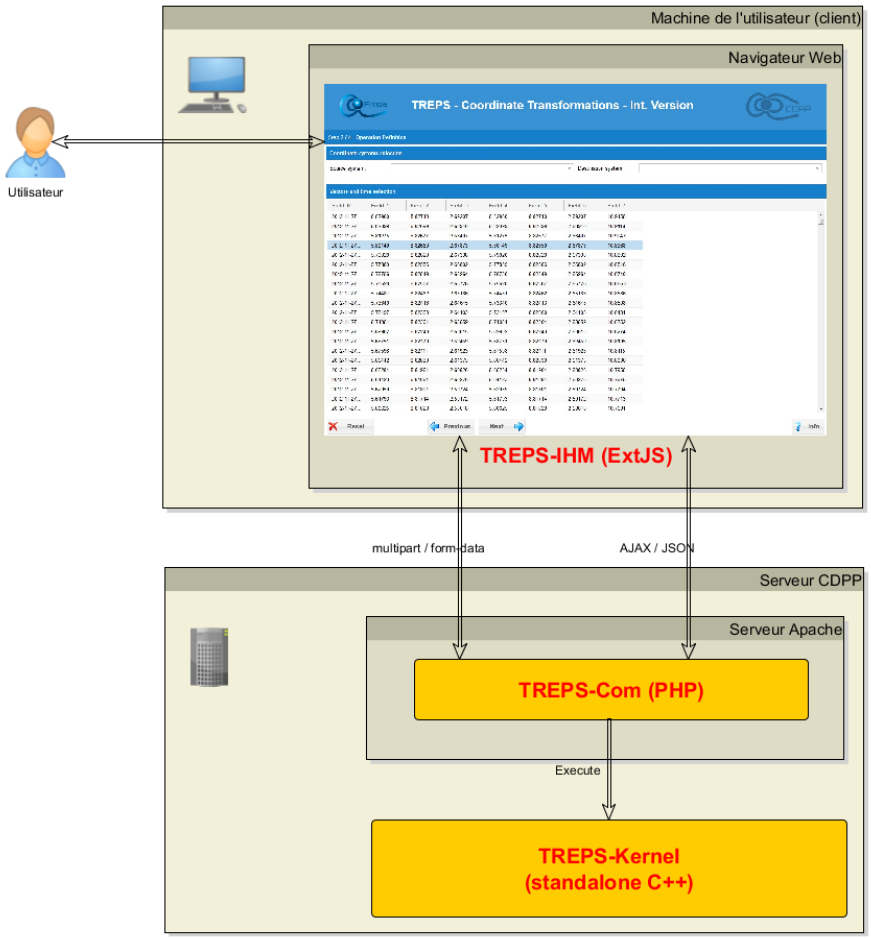


Figure 1 - Architecture générale de l'outil TREPS

2.2.1 Architecture de la sous-application TREPS-IHM

L'architecture de TREPS-IHM s'articule autour d'un **modèle MVC**, permettant une structuration claire de l'application, et facilitant les éventuelles opérations de maintenance.

Le framework ExtJS 4 introduit un moteur MVC permettant d'utiliser une telle architecture côté client, dont voici une description :

- **Model** : collection de champs et de leurs données associées. Cette collection est définie dans une classe de type « Ext.data.Model ». Les données d'un Model sont présentées et gérées par des « Ext.data.Store »,
- **View** : ensemble de composants visuels permettant de présenter les données du Model, et recevant les actions de l'utilisateur pour les envoyer vers un Controller,
- **Controller** : ensemble de traitements métier : traitement des actions, affichage des vues, instanciation des modèles, et toutes autres logiques nécessaires à l'application.

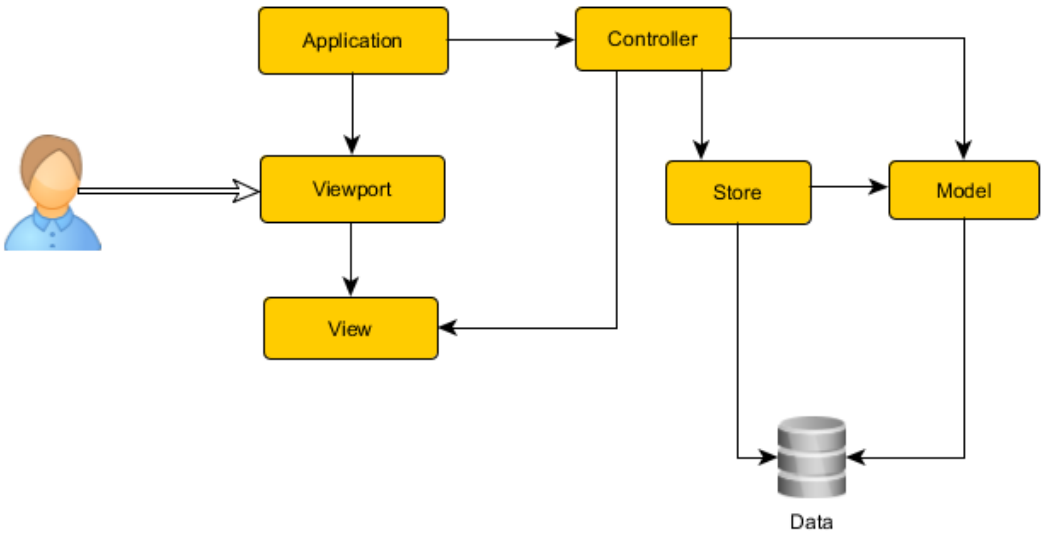


Figure 2 - Architecture générale de l'application TREPS-IHM

Dans la Figure 2, nous retrouvons les éléments du modèle MVC décrits précédemment, avec en plus :

- **Application** : Elle contient l'ensemble des configurations globales de l'application, des références à l'ensemble des modèles, vues et contrôleurs utilisé par l'application.
- **Viewport** : Il s'agit d'un conteneur se redimensionnant automatiquement à la taille de la fenêtre du navigateur de l'utilisateur, dans lequel sont affichées les autres vues de l'application.

Cette figure présente de manière très schématique l'application TREPS-IHM.

Elle est en vérité décomposée en différents modules, qui peuvent, chacun d'entre eux, contenir des vues, des contrôleurs et des modèles :

- Un module « Application », décrit ci-dessus,
- Un module de « Gestion des étapes », qui gère les différentes étapes de définition d'une opération de changement de repère,
- Un module « SAMP », qui gère tout ce qui est en lien avec l'utilisation du protocole SAMP,
- Un module « Etape de sélection des données sources »,
- Un module « Etape de visualisation des données source et de définition de la transformation »,
- Un module « Etape de visualisation des données résultat de l'opération et de définition de l'export »,
- Un module « Etape de mise à disposition du fichier exporté »,
- Un module « Visualisation des données sous forme de grille »,
- Un module « Visualisation des données sous forme de plot ».

2.2.2 Architecture de la sous-application TREPS-Kernel

L'application TREPS-Kernel s'articule autour d'une **architecture orientée service**, elle est composée de trois couches :

- **La couche « Service »** est une entité de traitement qui accepte des requêtes et qui renvoie des réponses au travers d'interfaces standards bien définies (dans notre cas, des fichiers XML dont la structure est définie). Elle est le point d'entrée de notre application.
- **La couche « Métier »** est la partie fonctionnelle de l'application. Chaque action que le service peut réaliser est divisée en opérations gérées par cette couche, en utilisant un ensemble « d'objets métier ».
- **La couche « Accès aux données »** est la partie gérant l'accès aux données du système. Elle permet notamment de fournir une abstraction de l'accès aux données pour la couche métier.

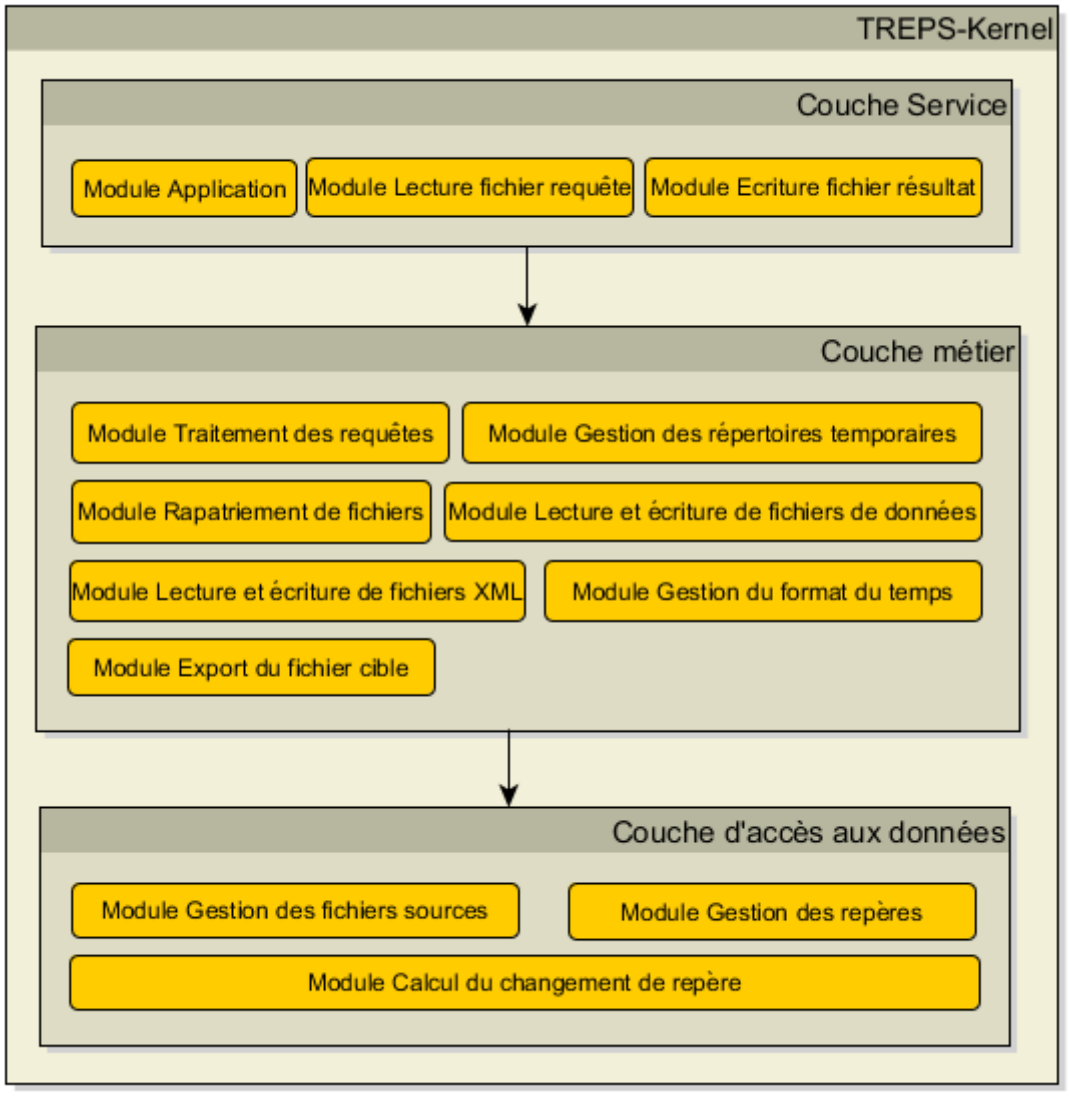


Figure 3 - Architecture générale de l'application TREPS-Kernel

Un **modèle de programmation orienté objet** est utilisé pour l'implémentation des différents éléments constituants l'application TREPS-Kernel.

2.2.3 Architecture de la sous-application TREPS-Com

L'application TREPS-Com est constituée d'un ensemble de scripts PHP, constituant une couche de communication entre les applications TREPS-IHM et TREPS-Kernel.

Cette sous-application est limitée « a-minima », les fonctionnalités étant à intégrer autant que possible au niveau de TREPS-Kernel.

En raison de la dimension limitée de cette application, aucune architecture particulière n'est adoptée, hormis un **modèle de programmation orienté objet**. Cela étant dit, cette application est divisée en 3 modules distincts :

- Un module « Ext.direct », qui va réceptionner les requêtes, hormis celles du upload, de TREPS-IHM,
- Un module « Upload », qui va réceptionner les requêtes, de type upload uniquement, de TREPS-IHM,
- Un module « Download », qui va traiter les requêtes les requêtes de type download de TREPS-IHM.

2.3 DESCRIPTION DE CHAQUE MODULE

2.3.1 Modules de la sous-application TREPS-IHM

Dans la suite de cette section, la notion de « requête » fait systématiquement référence à un appel à une fonction de l'API Ext.direct de l'application, c'est-à-dire une requête de type AJAX/JSON traitée par la sous-application TREPS-Com (qui elle-même interroge la sous-application TREPS-Kernel).

2.3.1.1 Module Application

Ce module se présente sous forme de singleton dont le rôle est l'initialisation de l'application TREPS-IHM, notamment en :

- Chargeant un ensemble de constantes globales à l'application
- Chargeant les API « Ext.direct » disponibles
- Chargeant le « Viewport »
- Chargeant les différents « Controller » nécessaires à l'initialisation de l'application (controllers de gestion des étapes, de gestion de l'aide, de gestion des notifications SAMP)
- Initialisant d'un modèle singleton de « Session »
- Initialisant un singleton gérant le masquage de l'application lors des différentes opérations,
- Initialisant un singleton gérant les boites d'informations ou d'erreurs de l'application

Ces différents éléments chargés par ce module sont accessibles et réutilisables par les autres modules de l'application.

Ce module contient également un contrôleur général lié à l'application, ainsi qu'une vue contenant les composants principaux de l'IHM (header, partie centrale et footer).

2.3.1.2 Module Viewport

Ce module est une vue qui se présente sous forme d'un conteneur se redimensionnement automatiquement à la taille de la fenêtre du navigateur de l'utilisateur.

Elle est directement rattachée au « Module Application » et constitue la zone dans laquelle la vue du module Application est présentée.

2.3.1.3 Module « Gestion des étapes »

Le module StepsManager est un contrôleur central au fonctionnement de l'application, dont la responsabilité est de gérer les différentes étapes de définition d'une opération de changement de repère.

Ce module gère notamment le passage à l'étape suivante de définition de l'opération :

- En sollicitant la validation des entrées utilisateur de l'étape courante.
- Puis en exécutant la requête correspondante à l'étape courante.
- Enfin, et seulement en cas de succès de la validation et de la requête, en commandant l'affichage de la vue correspondante à l'étape suivante.

Ce module gère également :

- La réinitialisation de l'application, dans l'éventualité où l'utilisateur souhaite débuter une nouvelle opération de changement de repère.
- Le retour à l'étape précédente.

2.3.1.4 Module « SAMP »

Le module SAMP est un contrôleur gérant :

- La connexion/déconnexion au hub SAMP
- La réception de notification de chargement de données sources
- L'envoi d'une notification de chargement de données résultats à un autre client du hub SAMP

Ce module contient également une vue permettant à l'utilisateur de :

- Commander la connexion/déconnexion au hub SAMP
- Visualiser la liste des clients actuellement connectés au hub SAMP

2.3.1.5 Module « Etape de sélection des données sources »

Ce module est constituée d'une vue, dans laquelle l'utilisateur peut sélectionner des données sources :

- Par sélection d'un fichier local
- Par drag & drop d'un fichier local sur une zone définie
- Par sélection d'un fichier web

- Par édition manuelle de données dans une grille

Ce module est également constitué d'un contrôleur gérant la validation et l'exécution de la requête de l'étape.

En cas de succès de cette étape, la variable de session « id », faisant référence à l'opération en cours, est conservée au niveau de l'application.

2.3.1.6 Module « Etape de visualisation des données source et de définition de la transformation »

Ce module est constitué d'une vue, dans laquelle l'utilisateur peut sélectionner :

- Les vecteurs sur lesquels appliquer l'opération de changement de repère
- Le champ temps, ainsi que son format
- Le repère d'origine, et le repère cible de la transformation

La vue contient également une grille de visualisation des données sources sélectionnées à l'étape précédente. Ces données sont récupérées par le biais d'une requête sollicité par un store « SourceDataStore » lui-même relié à un modèle de type « SourceDataModel » (ce store et ce modèle sont instanciés dynamiquement en fonction de la structure du fichier source).

Ce module est également constitué d'un contrôleur gérant la validation et l'exécution de la requête de l'étape.

2.3.1.7 Module « Etape de visualisation des données résultat et de définition de l'export »

Ce module est constitué d'une vue, dans laquelle l'utilisateur peut définir :

- Le format du fichier à exporter
- La structure du fichier à exporter
- Le format du temps à exporter

La vue contient également une grille de visualisation des données résultats de la transformation, qui peut être basculée en mode de visualisation sous forme de plot (cf. Module « Visualisation des données sous forme de plot »). Ces données sont récupérées par le biais d'une requête sollicité par un store « ResultDataStore » lui-même relié à un modèle de type « ResultDataModel » (ce store et ce modèle sont instanciés dynamiquement en fonction de la structure des données résultat).

Ce module est également constitué d'un contrôleur gérant la validation et l'exécution de la requête de l'étape.

2.3.1.8 Module « Etape de mise à disposition du fichier exporté »

Ce module est constituée d'une vue, dans laquelle l'utilisateur peut :

- Télécharger le fichier résultat
- Solliciter l'envoie du résultat vers un client SAMP

2.3.1.9 Module « Visualisation des données sous forme de plot »

Ce module contient une vue permettant la visualisation sous forme de plot des données résultats de la transformation (lorsqu'elles existent).

2.3.1.10 Module « Visualisation des données sous forme de grille »

Ce module contient une vue permettant la visualisation sous forme de grille des données sources et / ou des données résultats de la transformation

2.3.2 Modules de la sous-application TREPS-Kernel

2.3.2.1 Module « Application »

Le module « Application » fait partie de la couche « Service » de TREPS-Kernel.

Il s'agit d'un singleton, faisant office de point d'entrée de l'application, qui :

- instancie le logger,
- lit le fichier de configuration,
- récupère le chemin du fichier XML requête fournit en argument,
- lit le fichier requête,
- instancie le module « Traitement des requêtes », lance l'exécution du traitement de la requête et commande l'écriture du fichier XML résultat.
- Retourne le chemin vers le fichier XML résultat, ou renvoie un code d'erreur, en fonction de la réussite ou nom de l'exécution de la requête.

2.3.2.2 Module « Lecture fichier requête »

Le module « Lecture fichier requête » fait partie de la couche « service » de TREPS-Kernel.

Il lit le fichier XML de la requête, et met à disposition aux autres modules les paramètres de la requête.

2.3.2.3 Module « Ecriture fichier résultat »

Le module « Ecriture fichier requête » fait partie de la couche « service » de TREPS-Kernel.

Il écrit, sous forme de fichier XML, la réponse à la requête.

2.3.2.4 Module « Traitement des requêtes »

Le module « Traitement des requêtes » fait partie de la couche « métier » de TREPS-Kernel.

Ses rôles sont de :

- En fonction du type de requête, instancier la classe « Requête » correspondante (une classe « Requête » hérite d'une classe abstraite de base),
- Exécuter les différentes opérations liées au type de la requête,

2.3.2.5 Module « Gestion des répertoires temporaires »

Le module « Gestion des répertoires temporaires » fait partie de la couche « métier » de TREPS-Kernel.

Ses responsabilités sont de :

- Créer un espace de travail pour débuter une nouvelle opération,
- Fournir le chemin vers l'espace de travail correspondant à une opération donnée,
- Supprimer un espace de travail correspondant à une opération donnée.

2.3.2.6 Module « Rapatriement de fichiers »

Le module « Rapatriement de fichiers » fait partie de la couche « métier » de TREPS-Kernel.

Son rôle est de copier un fichier, dans l'espace de travail, à partir d'un chemin local, ou d'une URL.

2.3.2.7 Module « Lecture et écriture de fichiers de données »

Le module « Lecture et écriture de fichiers de données » fait partie de la couche « métier » de TREPS-Kernel.

Il fournit une classe abstraite de lecture et d'écriture de fichiers de données, se dérivant en différentes classes pour chaque type de format géré.

2.3.2.8 Module « Lecture écriture de fichiers XML »

Le module « Lecture et écriture de fichiers XML » fait partie de la couche « métier » de TREPS-Kernel.

Il fournit une classe de lecture et d'écriture de fichiers au format XML, ainsi qu'une fonction de validation à partir d'un fichier schéma XML.

2.3.2.9 Module « Gestion du format du temps »

Le module « Gestion du format du temps » fait partie de la couche « métier » de TREPS-Kernel.

Il fournit un ensemble d'outils de conversion du format du temps.

2.3.2.10 Module « Export du fichier cible »

Le module « Export du fichier cible » fait partie de la couche « métier » de TREPS-Kernel.

Sa responsabilité est de créer le fichier résultat dans un format et avec une structure donnés.

2.3.2.11 Module « Gestion des fichiers sources

Le module « Gestion des fichiers sources » fait partie de la couche « accès aux données » de TREPS-Kernel.

Il offre un accès unifié aux informations et aux données d'un fichier source.

2.3.2.12 Module « Gestion des repères »

Le module « Gestion des repères » fait partie de la couche « accès aux données » de TREPS-Kernel.

Il permet un accès unifié à la liste des repères disponibles dans l'outil, ainsi que certaines informations relatives à chacun des repères.

2.3.2.13 Module « Calcul du changement de repère »

Le module « Calcul du changement de repère » fait partie de la couche « accès aux données » de TREPS-Kernel.

Il offre un accès unifié aux fonctionnalités réalisant la transformation de repère.

2.3.3 Modules de la sous-application TREPS-Com

2.3.3.1 Module « Ext.direct »

Ce module est une implémentation, côté serveur, de la technologie de communication mise à disposition dans le framework ExtJS, et qui dirige les requêtes du client TREPS-IHM vers les méthodes associées côté serveur.

2.3.3.2 Module « Upload »

Les requêtes d'upload de fichiers de TREPS-IHM sont réceptionnées par ce module.

2.3.3.3 Module « Traitement des requêtes »

Les responsabilités de ce module sont :

- Traduire, sous forme de fichier XML compréhensible par TREPS-Kernel, une requête reçue depuis TREPS-IHM,
- Exécuter l'application TREPS-Kernel, avec pour argument d'entrée le fichier XML requête précédemment créé,
- Traduire, sous forme de « array » compréhensible par TREPS-IHM, la réponse à la requête de TREPS-Kernel, reçue sous forme de fichier XML.

3 CONCEPTION DETAILLEE

3.1 CONCEPTION DETAILLEE DE LA SOUS-APPLICATION TREPS-IHM

3.1.1 Diagramme de paquetage

La sous-application TREPS-IHM est constituée :

- De librairies (extjs et sampjs) contenues dans le paquetage « lib »,
- D'un ensemble de pages d'aide contenu dans le paquetage « help »,
- Du code javascript de l'application à proprement parlé, contenu dans le paquetage « app »,
- D'un ensemble de ressources (css et images), contenu dans le paquetage « resources »

Le diagramme suivant montre les différents paquetages constituant la sous-application TREPS-IHM dans sa globalité.

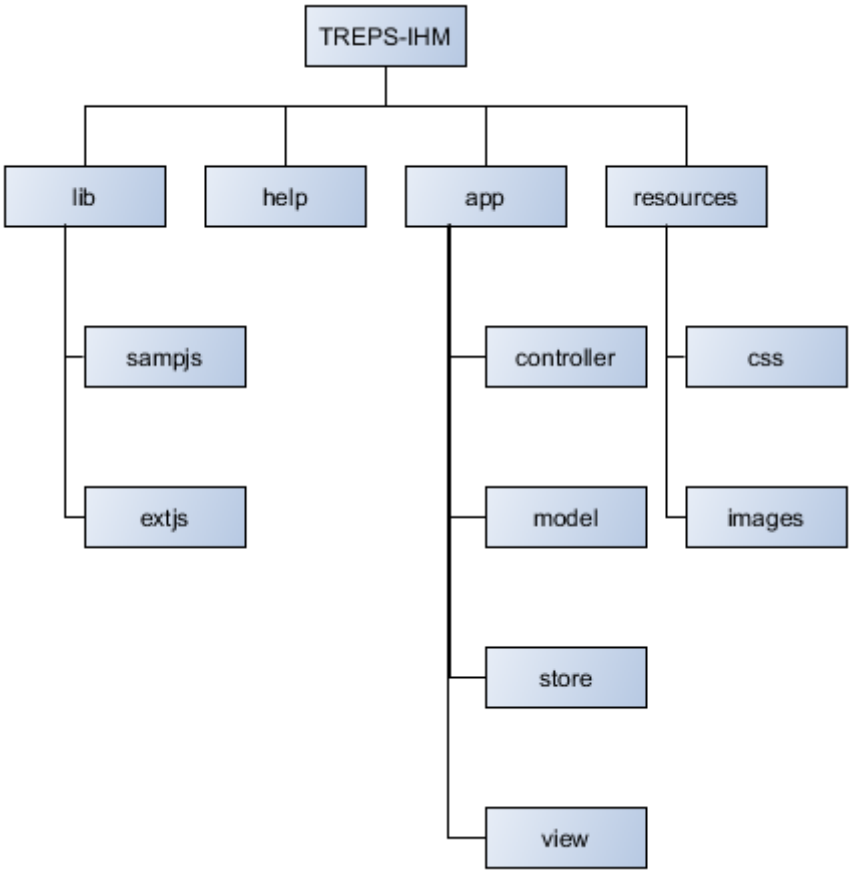


Figure 4 - Diagramme de paquetage de la sous-application TREPS-IHM

3.1.2 Diagramme d'états-transitions relatif à une « étape »

Le mécanisme principal de l'application TREPS-IHM est porté par la classe StepsManager dont la responsabilité est de gérer le passage d'une étape de l'opération de changement de repère vers une autre.

En effet, une opération de changement de repère se décompose en 4 étapes successives, dépendantes les unes des autres.

Le mécanisme en question est présenté dans le diagramme d'états-transitions suivant, en considérant que l'application est sur le point d'entrer sur l'étape n :

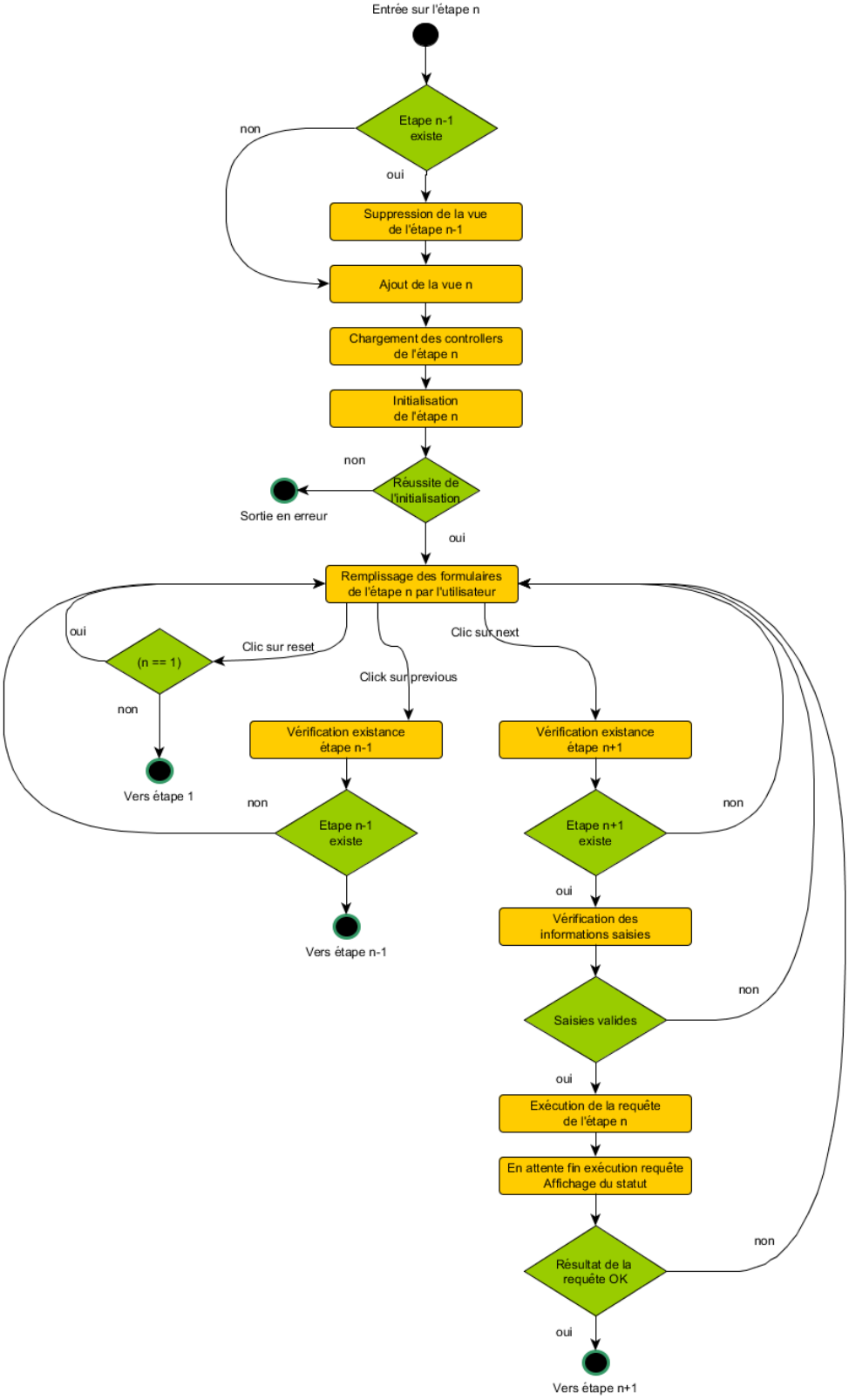


Figure 5 - Diagramme d'états-transitions relatif à l'étape n dans la sous-application TREPS-IHM

3.2 CONCEPTION DETAILLEE DE LA SOUS-APPLICATION TREPS-KERNEL

3.2.1 Diagramme de paquetage

La sous-application TREPS-Kernel est constituée :

- De tests FitNesse et Sonar, contenus dans le paquetage « test »,
- D'un client SOAP pour le Web Service de CDPP/3DView, généré par gSOAP et contenu dans le paquetage « CDPP3DViewSOAPClient »,
- D'un ensemble d'outils utilisés durant l'installation de l'application, contenu dans le paquetage « scripts »,
- De fichiers de configuration de l'application, contenu dans le paquetage « config »,
- De fichiers de données (liste des formats de temps, liste des repères, etc…), contenus dans le paquetage « data »,
- De fichiers XML Schema, utilisés par l'application pour valider les différents fichiers XML lus, contenus dans le paquetage « xsd ».

Le diagramme suivant montre les différents paquetages constituant la sous-application TREPS-Kernel dans sa globalité.

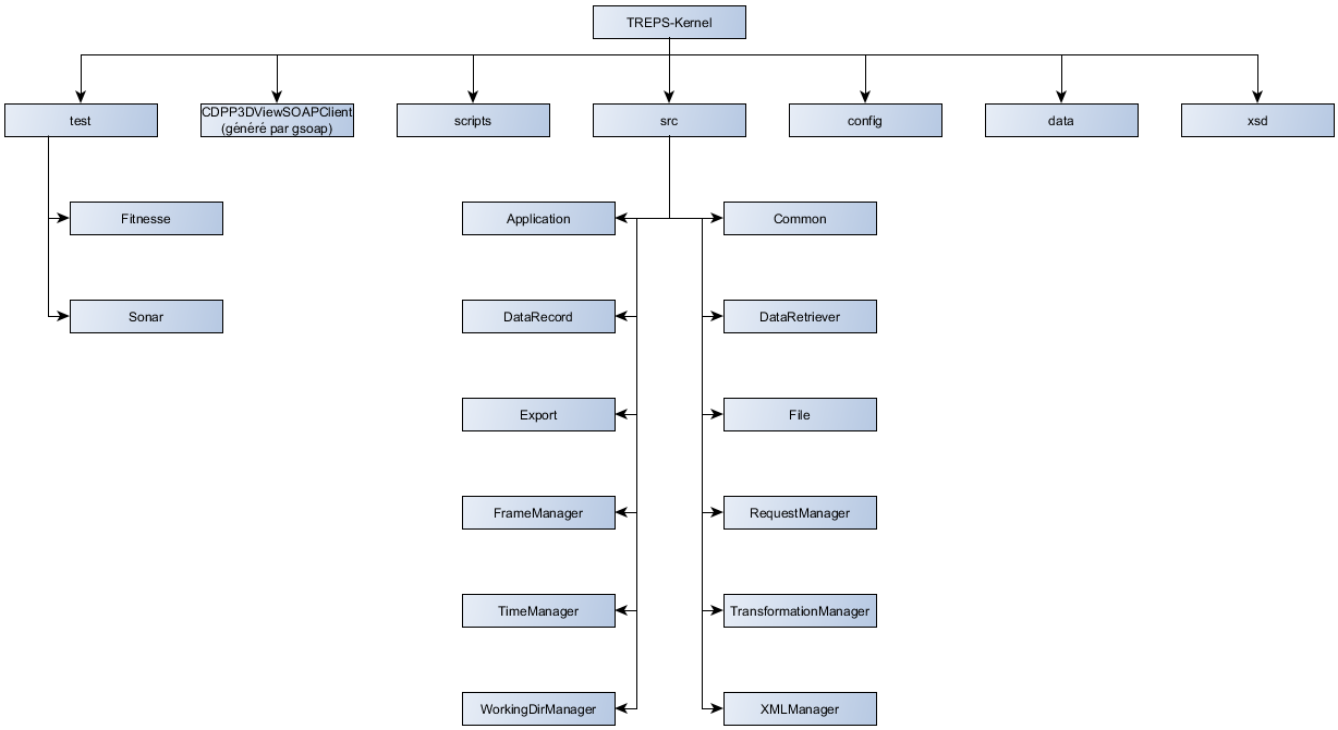


Figure 6 - Diagramme de paquetage de la sous application TREPS-Kernel

3.2.2 Traitement d'une requête par la sous-application TREPS-Kernel

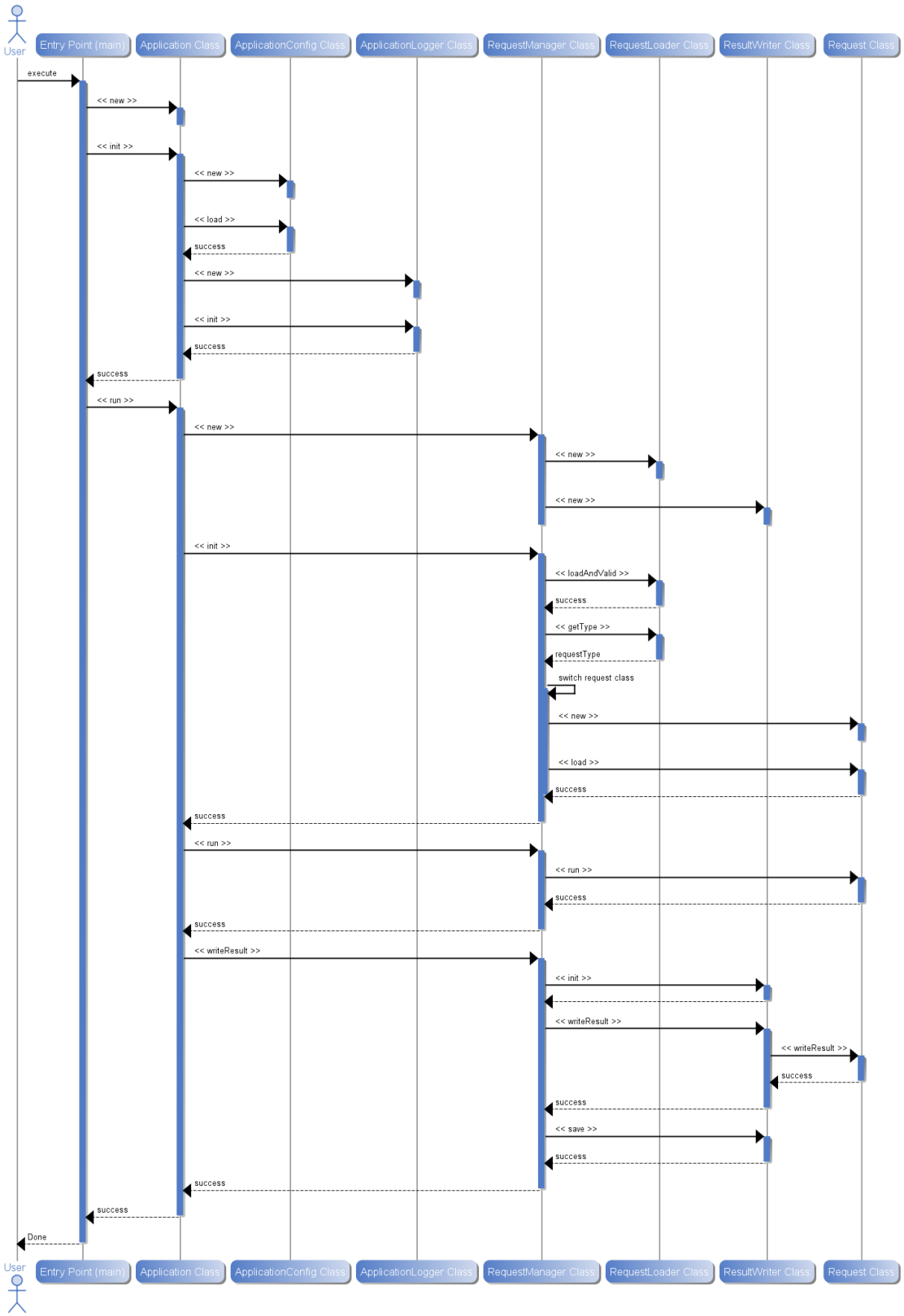


Figure 7 - Diagramme de séquence de la gestion d'une requête

La figure précédente représente le diagramme de séquence relatif à l'exécution de toutes les requêtes par la sous-application TREPS-Kernel.

Le type de requête est détecté par l'appel à la fonction « getType » du « RequestLoader » par le « RequestManager ».

Une fois le type de requête détecté, le « RequestManager » instancie la classe « Request » lui correspondant (représenté par « switch request class » dans le diagramme) afin d'exécuter la requête demandée.

Toutes les classes « Request » héritent d'une classe commune abstraite « RequestAbstract ».

3.2.3 Liste des différentes requêtes de l'application TREPS-Kernel

Dans cette section, les différentes requêtes sont désignées par le nom de leurs classes respectives.

Pour chaque requête nous précisons :

- Les paramètres d'entrée qui sont lus par la fonction « load » de la requête en utilisant la classe « RequestLoader »
- Le processus d'exécution de la requête, sous forme de diagramme de séquence simplifié
- Le type de sortie de la requête, et la liste des paramètres écrits

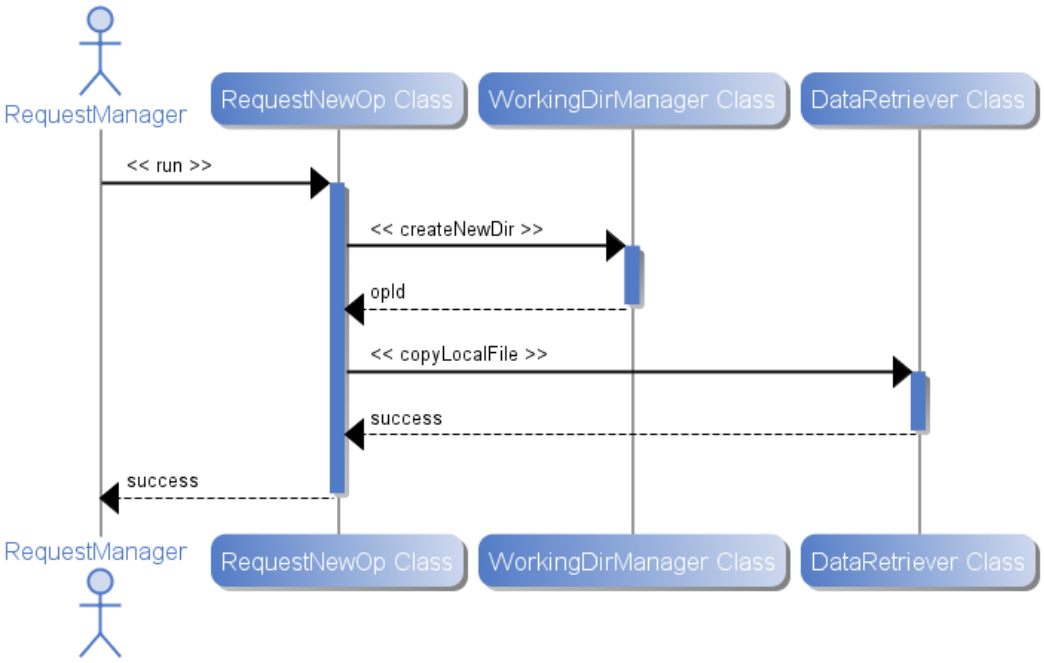
3.2.3.1 RequestNewOp

Cette requête a pour rôle de créer d'une nouvelle opération de changement de repère.

3.2.3.1.1 Paramètres d'entrée :

- testdir (facultatif) :
  - true : pour créer une nouvelle opération qui utilise le répertoire de test « testdir » comme répertoire de travail
  - false : pour créer une nouvelle opération qui génère elle-même son répertoire de travail unique
- type :
  - local : pour récupérer des données source depuis un fichier présent sur la machine locale
  - url : pour récupérer des données source depuis un fichier du Web
- location : correspond au chemin du fichier si « type == local », et à une URL si « type == url »

3.2.3.1.2 Exécution :



3.2.3.1.3 Sortie :

La sortie est de type « OUTPUT_RESULTFILE » (cf. 3.2.4), les paramètres écrits sont :

- id : l'identifiant de l'opération créée

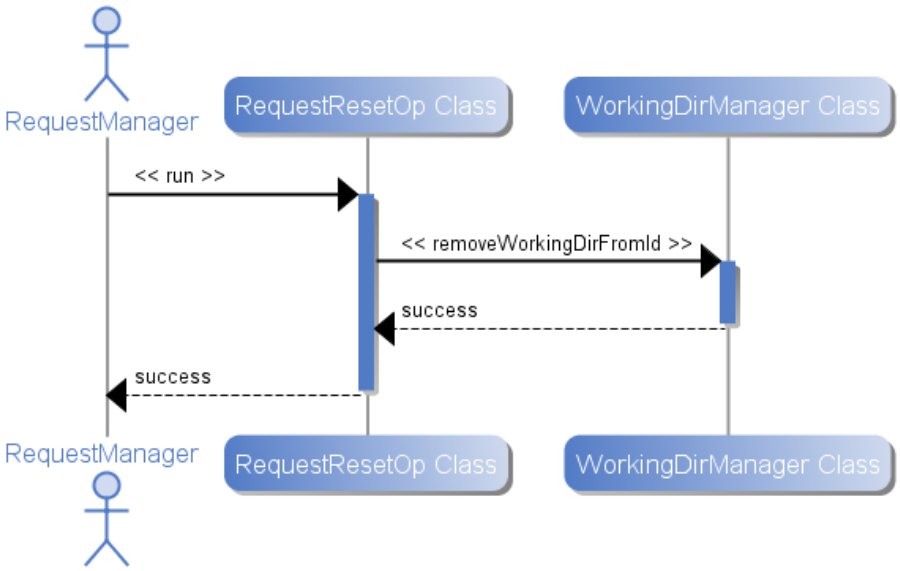
3.2.3.2 RequestResetOp

Cette opération a pour rôle de supprimer une opération de changement de repère.

3.2.3.2.1 Paramètres d'entrée :

- id : l'identifiant de l'opération

3.2.3.2.2 Exécution :



3.2.3.2.3 Sortie :

La sortie est de type « OUTPUT_NONE » (cf. 3.2.4).

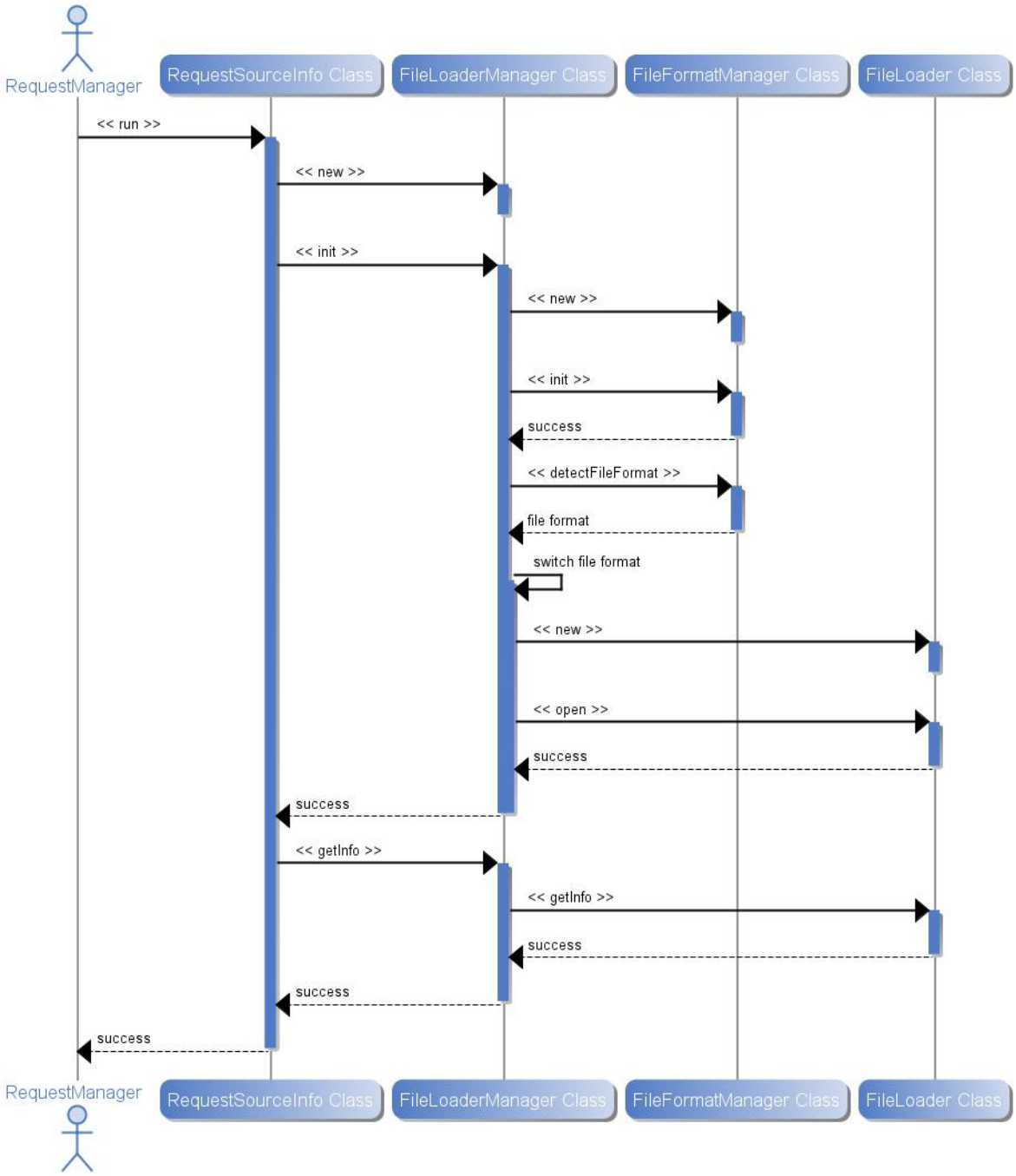
3.2.3.3 RequestSourceInfo

Cette opération a pour rôle de récupérer les informations sur la structure du fichier source.

3.2.3.3.1 Paramètres d'entrée :

- id : l'identifiant de l'opération

3.2.3.3.2 Exécution :



Dans ce diagramme, « FileLoader Class » correspond en réalité « FileLoaderASCII », ou « FileLoaderVOTable », ou « FileLoaderCDF », ou « FileLoaderNetCDF » en fonction du format de fichier détecté lors de l'appel à la fonction « detatectFileFormat » de la classe « FileFormatManager ».

3.2.3.3.3 Sortie :

La sortie est de type « OUTPUT_RESULTFILE » (cf. 3.2.4), les paramètres écrits sont :

- frames : une liste de repère détecté dans le fichier

- fields : une liste des champs détectés dans le fichier
- vectors : une liste de vecteurs détectés dans le fichier
- attributes : une liste d'attributs détectés dans le fichier

3.2.3.4 RequestResultInfo

Cette opération a pour rôle de récupérer les informations sur la structure du fichier résultat.

Cette requête est identique à la requête « RequestSourceInfo » (cf.3.2.3.3), seul le chemin vers le fichier ciblé change.

3.2.3.5 RequestSourceGet

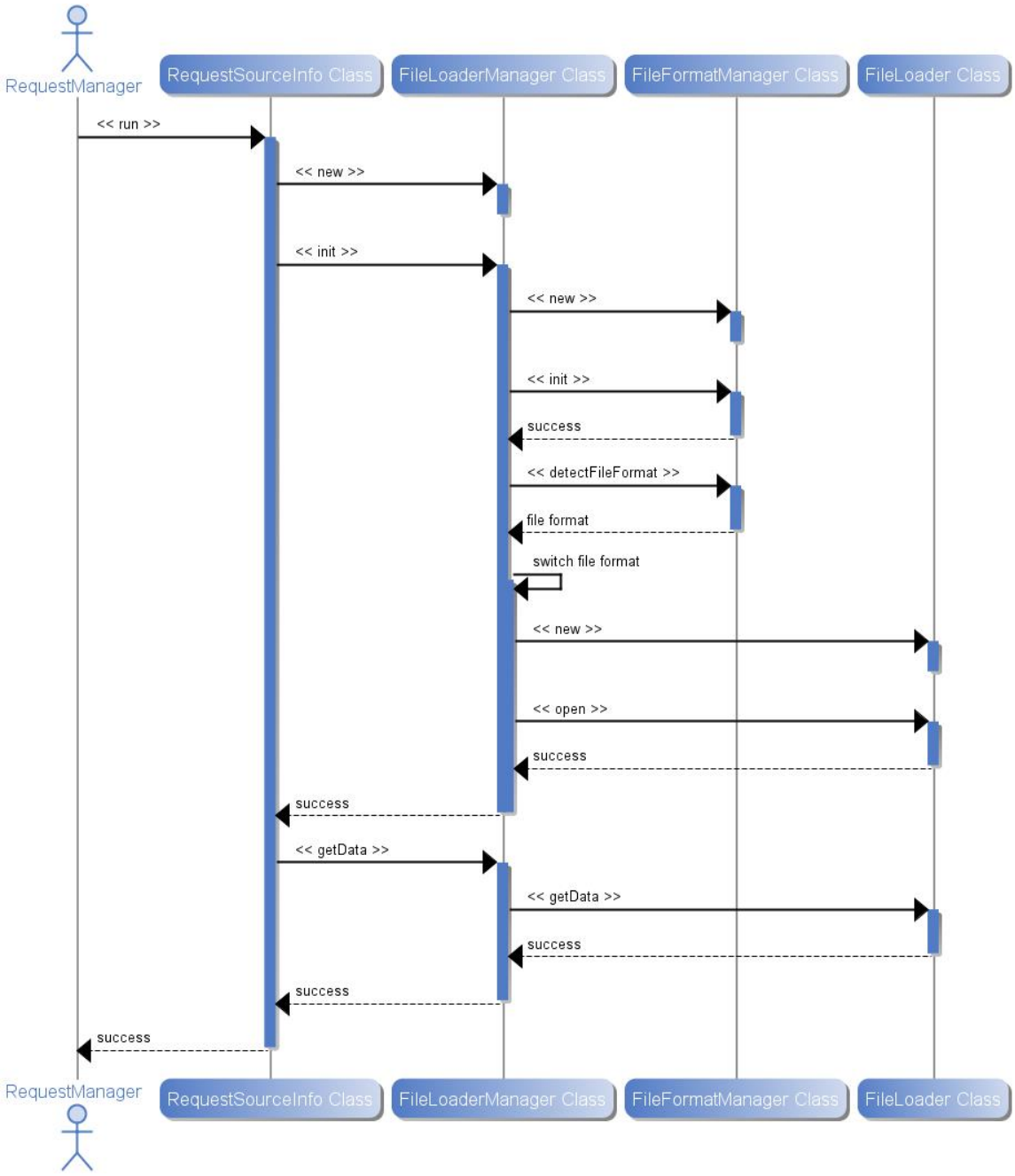
Cette opération a pour rôle de récupérer les données du fichier source.

3.2.3.5.1 Paramètres d'entrée :

- id : l'identifiant de l'opération
- start : index du premier enregistrement à récupérer
- limit : nombre maximal d'enregistrements à récupérer

Remarque : si « start == 0 » et « limit == 0 », tous les enregistrements du fichier sont récupérés

3.2.3.5.2 Exécution :



Dans ce diagramme, « FileLoader Class » correspond en réalité « FileLoaderASCII », ou « FileLoaderVOTable », ou « FileLoaderCDF », ou « FileLoaderNetCDF » en fonction du format de fichier détecté lors de l'appel à la fonction « detatectFileFormat » de la classe « FileFormatManager ».

3.2.3.5.3 Sortie :

La sortie est de type « OUTPUT_RESULTFILE » (cf. 3.2.4), les paramètres écrits sont :

- start : l'index du premier enregistrement renvoyé

- limit : le nombre d'enregistrements renvoyé
- total : le nombre total d'enregistrements détecté dans le fichier
- data : la liste des données récupérées

3.2.3.6 RequestResultGet

Cette opération a pour rôle de récupérer les données du fichier résultat.

Cette requête est identique à la requête « RequestSourceGet » (cf. 3.2.3.5), seul le chemin vers le fichier ciblé change.

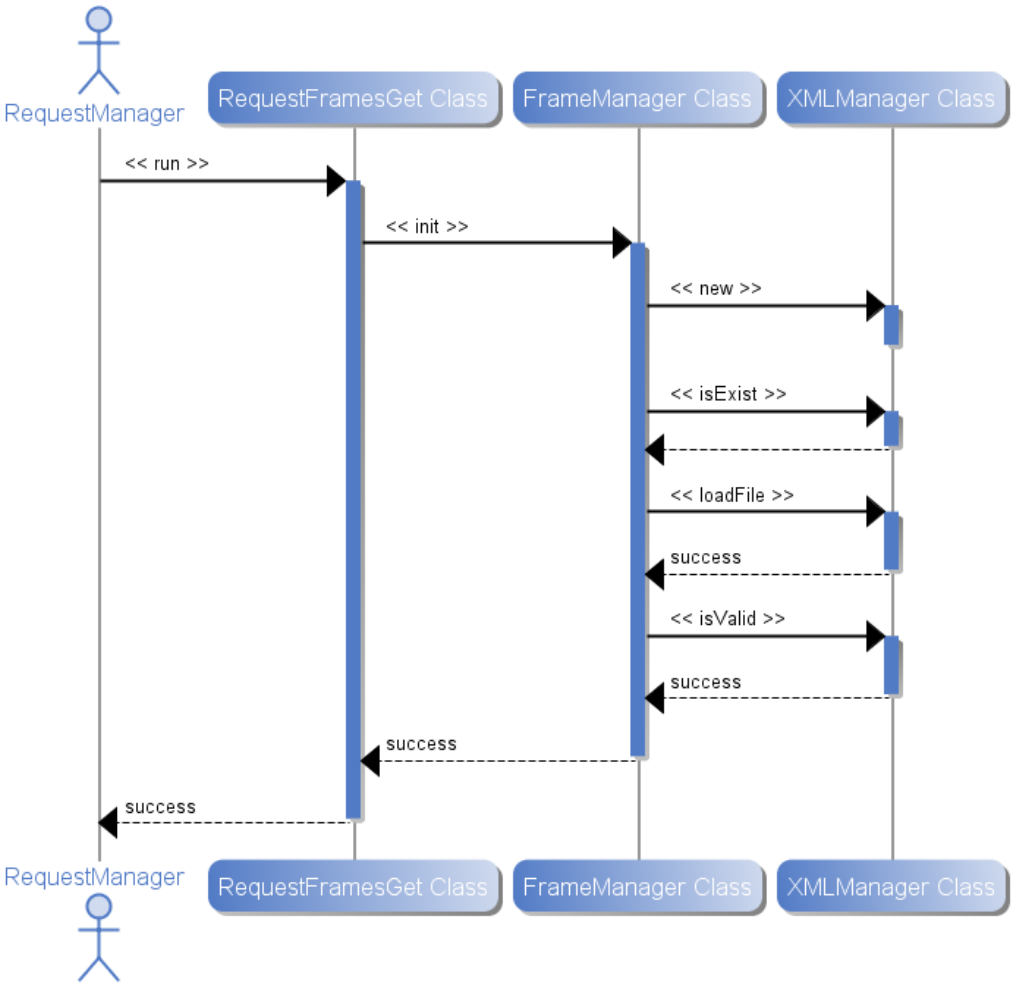
3.2.3.7 RequestFramesGet

Cette opération a pour rôle de récupérer le chemin vers le fichier XML de définition des repères, tout en contrôlant la validité de ce fichier.

3.2.3.7.1 Paramètres d'entrée :

Aucun paramètre d'entrée n'est requis.

3.2.3.7.2 Exécution :



3.2.3.7.3 Sortie :

La sortie est de type « OUTPUT_XMLFILE» (cf. 3.2.4).

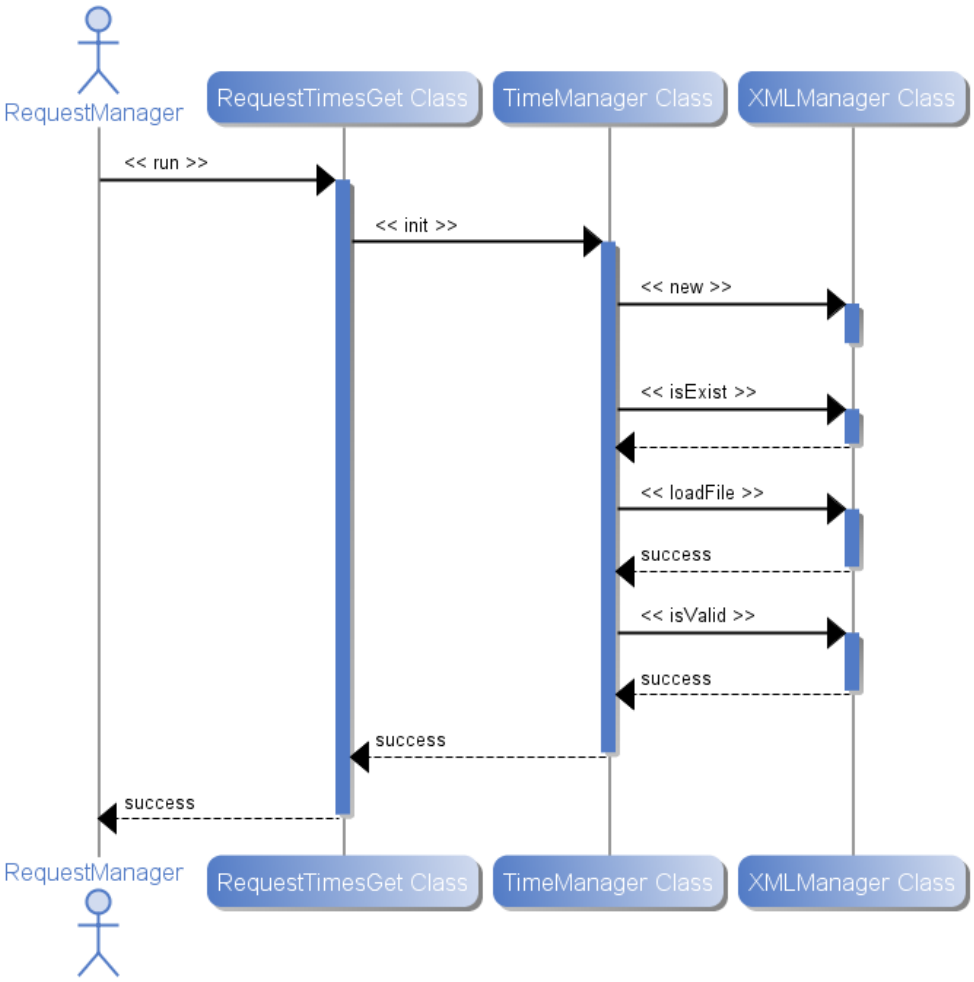
3.2.3.8 RequestTimesGet

Cette opération a pour rôle de récupérer le chemin vers le fichier XML de définition des repères, tout en contrôlant la validité de ce fichier.

3.2.3.8.1 Paramètres d'entrée :

Aucun paramètre d'entrée n'est requis.

3.2.3.8.2 Exécution



3.2.3.8.3 Sortie :

La sortie est de type « OUTPUT_XMLFILE» (cf. 3.2.4).

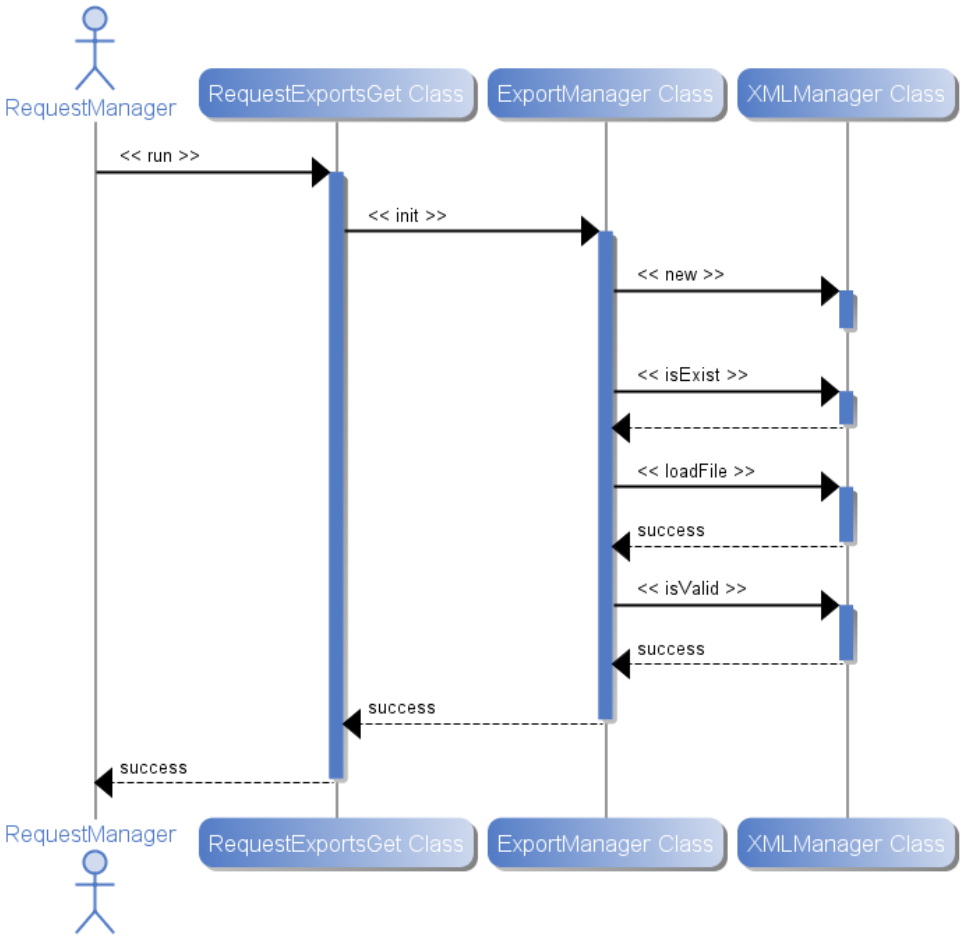
3.2.3.9 RequestExportsGet

Cette opération a pour rôle de récupérer le chemin vers le fichier XML de définition des structures d'exportation, tout en contrôlant la validité de ce fichier.

3.2.3.9.1 Paramètres d'entrée :

Aucun paramètre d'entrée n'est requis.

3.2.3.9.2 Exécution



3.2.3.9.3 Sortie :

La sortie est de type « OUTPUT_XMLFILE» (cf. 3.2.4).

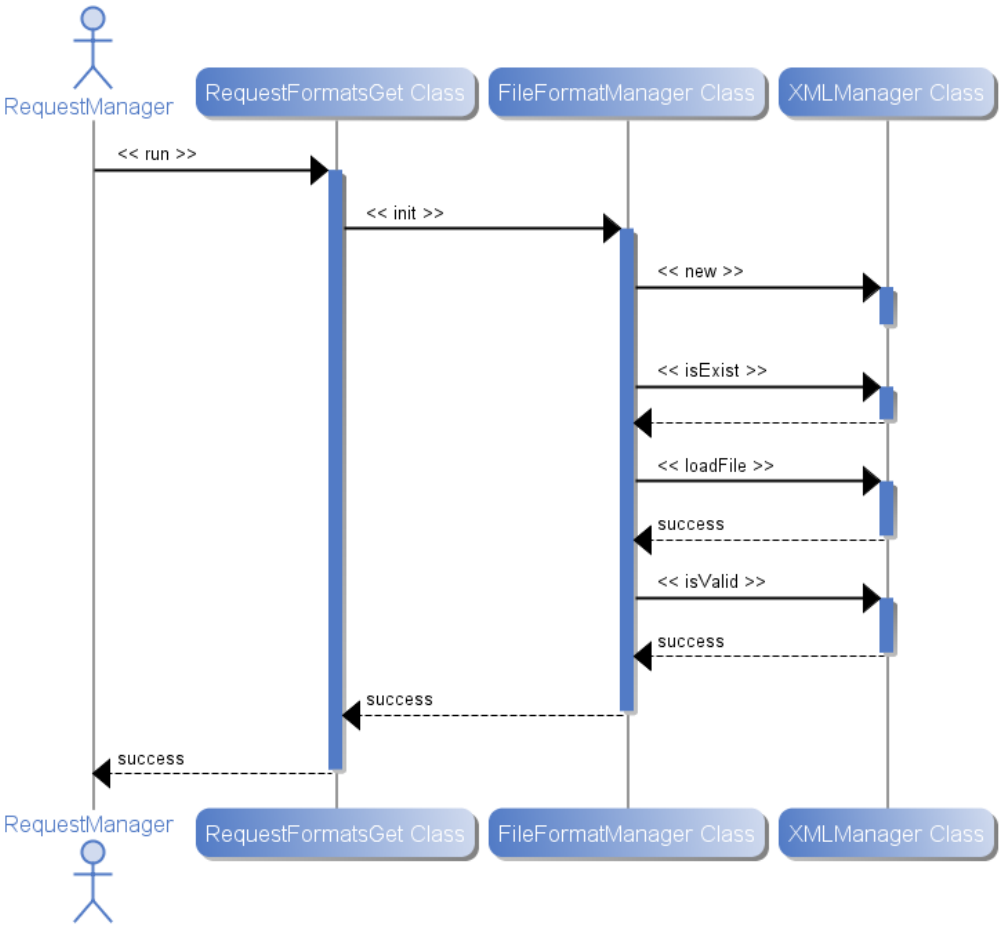
3.2.3.10 RequestFormatsGet

Cette opération a pour rôle de récupérer le chemin vers le fichier XML de définition des formats de fichiers disponibles, tout en contrôlant la validité de ce fichier.

3.2.3.10.1 Paramètres d'entrée :

Aucun paramètre d'entrée n'est requis.

3.2.3.10.2 Exécution :



3.2.3.10.3 Sortie :

La sortie est de type « OUTPUT_XMLFILE» (cf. 3.2.4).

3.2.3.11 RequestRunOp

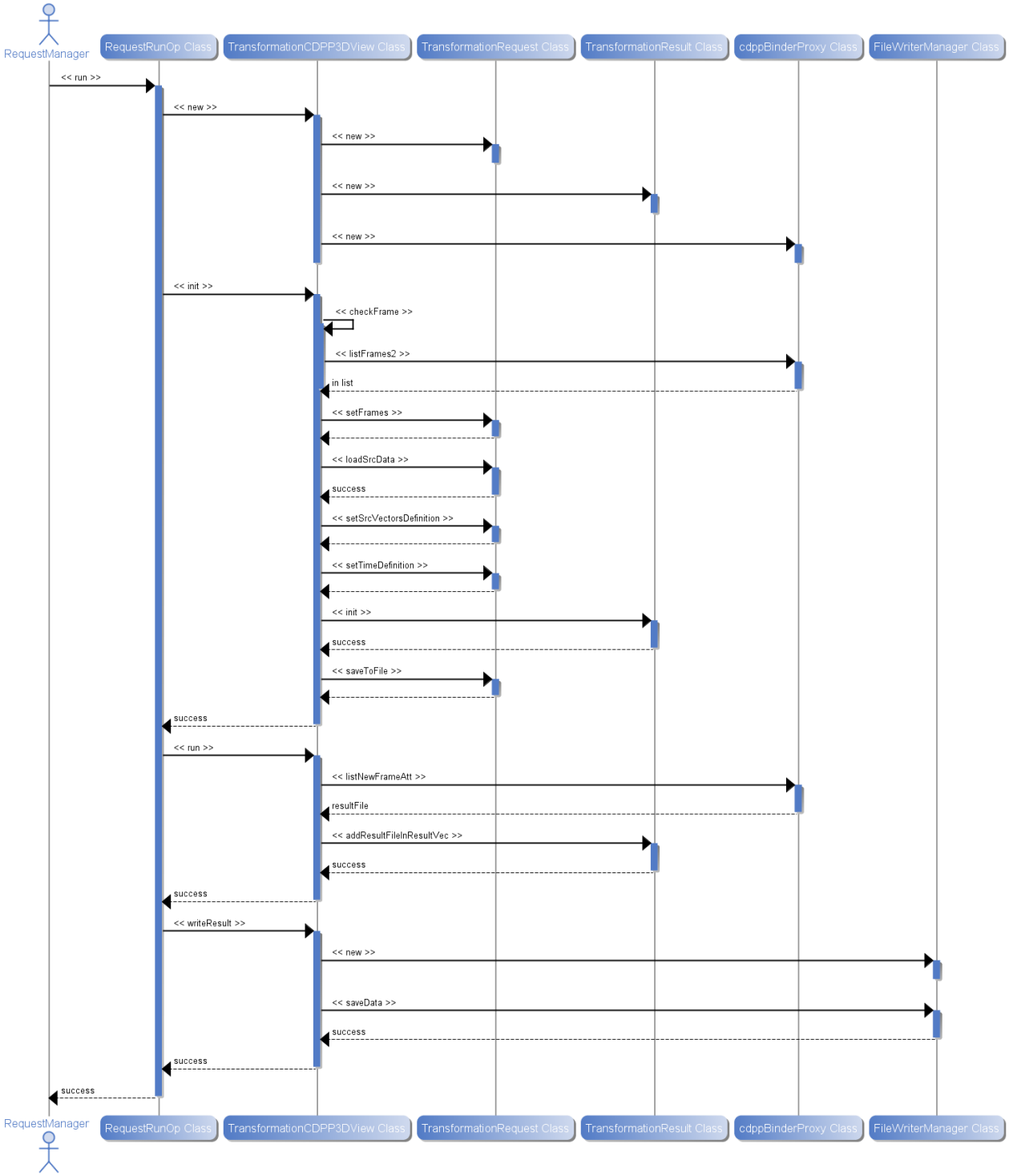
Cette opération a pour rôle d'exécuter l'opération de changement de repère.

3.2.3.11.1 Paramètres d'entrée :

- id : l'identifiant de l'opération
- srcframe : le nom du repère source
- dstframe : le nom du repère de destination
- vectors : la liste des vecteurs sur lesquels appliquer la transformation
- timefieldid (facultatif): l'identifiant du champ représentant le temps
- timeformatid (requis uniquement si timefieldid est défini) : le format du temps du champ désigné par timefieldid

- timepattern (requis suivant le type de format de temps) : le pattern du champ temps

3.2.3.11.2 Exécution :



3.2.3.11.3 Sortie :

La sortie est de type « OUTPUT_RESULTFILE » (cf. 3.2.4).

Aucun paramètre de sortie n'est écrit.

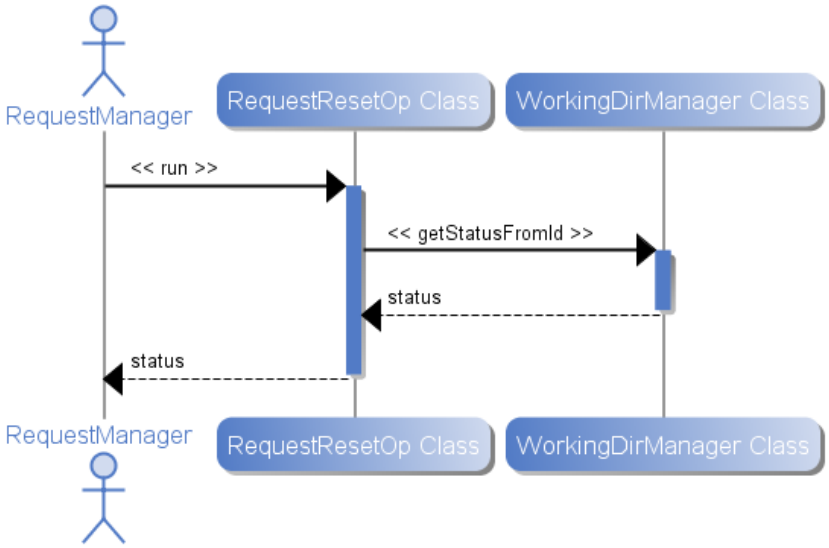
3.2.3.12 RequestStatusGet

Cette opération a pour rôle de récupérer le statut d'exécution d'une requête RequestRunOp (cf. 3.2.3.11).

3.2.3.12.1 Paramètres d'entrée :

- id : l'identifiant de l'opération

3.2.3.12.2 Exécution :



3.2.3.12.3 Sortie :

La sortie est de type « OUTPUT_STRING» (cf. 3.2.4).

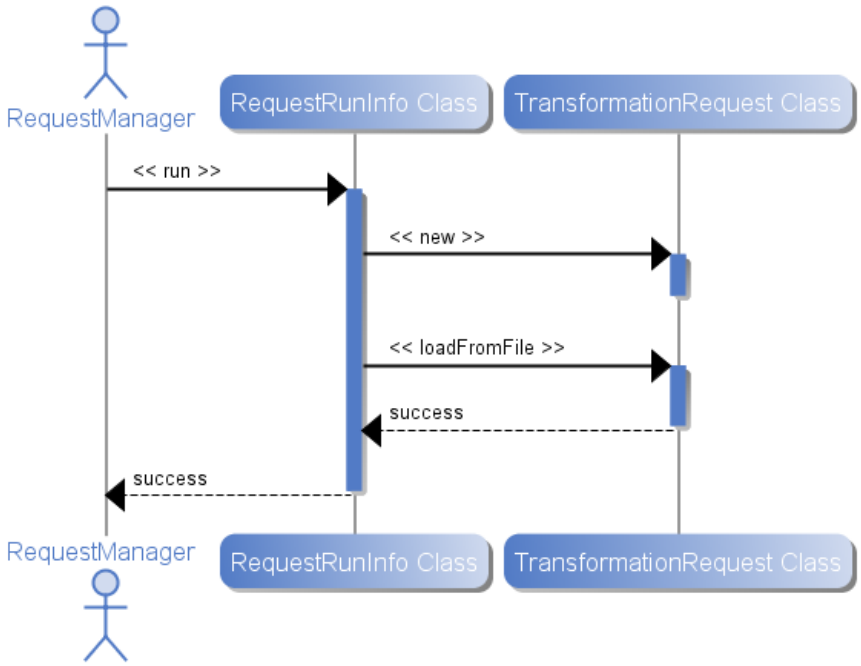
3.2.3.13 RequestRunInfo

Cette opération a pour rôle de récupérer les paramètres d'entrée de la dernière opération de changement de repère exécutée par une requête RequestRunOp (cf. 3.2.3.11).

3.2.3.13.1 Paramètres d'entrée :

- id : l'identifiant de l'opération

3.2.3.13.2 Exécution :



3.2.3.13.3 Sortie :

La sortie est de type « OUTPUT_RESULTFILE » (cf. 3.2.4), les paramètres écrits sont :

- frames : le repère source « src » et le repère de destination « dst » de l'opération de changement de repère
- time : l'identifiant du champ temps « fieldId » utilisé, le format du temps « formatId » et le pattern du temps « pattern »
- vectors : la liste des vecteurs source

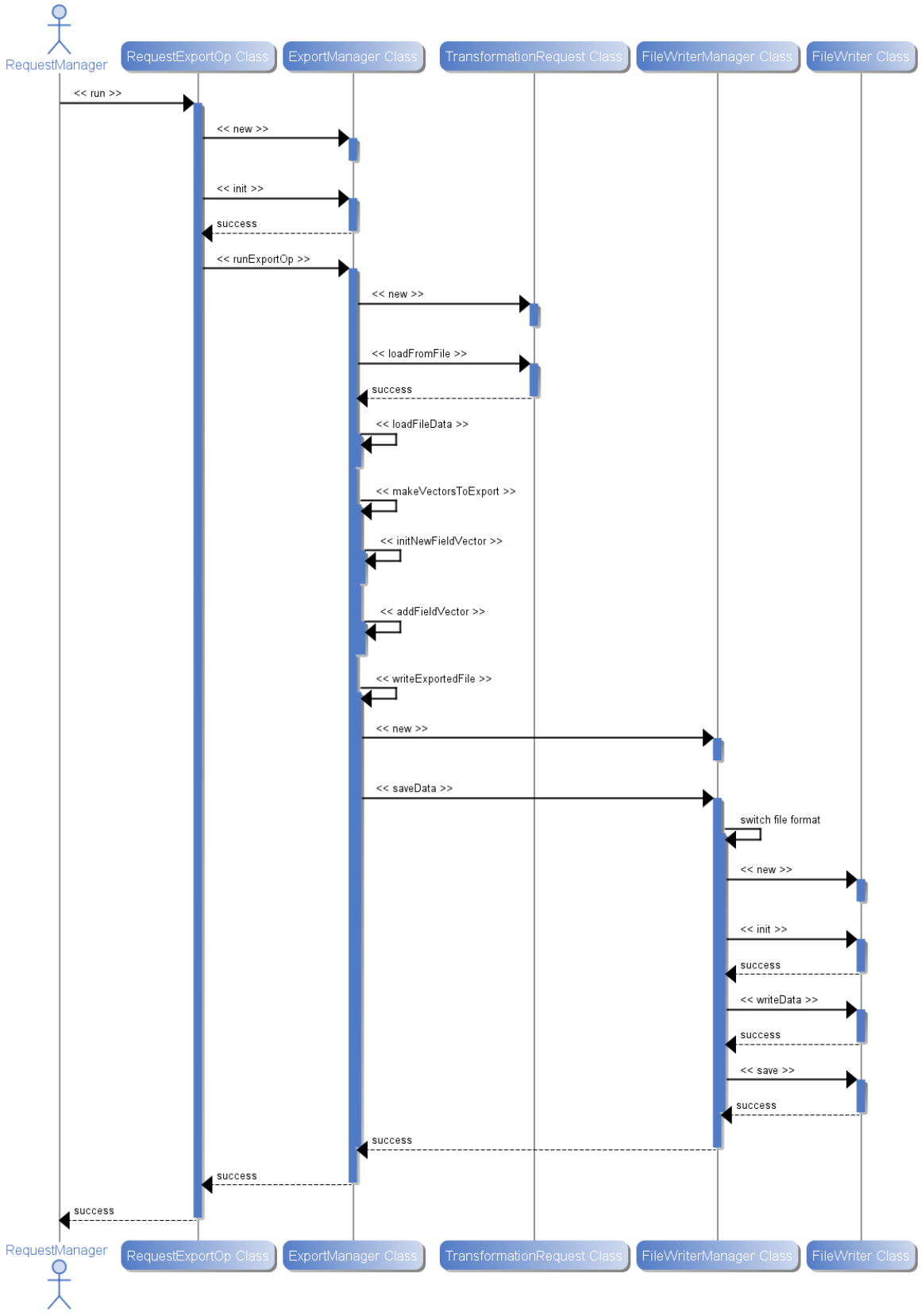
3.2.3.14 RequestExportOp

Cette opération a pour rôle d'exporter le résultat de l'opération de changement de repère.

3.2.3.14.1 Paramètres d'entrée :

- id : l'identifiant de l'opération
- format : le format du fichier à exporter
- structure : la structure du fichier à exporter
- timeformat : le format du temps du fichier à exporter
- timepattern : le pattern du temps du fichier à exporter

3.2.3.14.2 Exécution :



3.2.3.14.3 Sortie :

La sortie est de type « OUTPUT_RESULTFILE » (cf. 3.2.4).

Aucun paramètre de sortie n'est écrit.

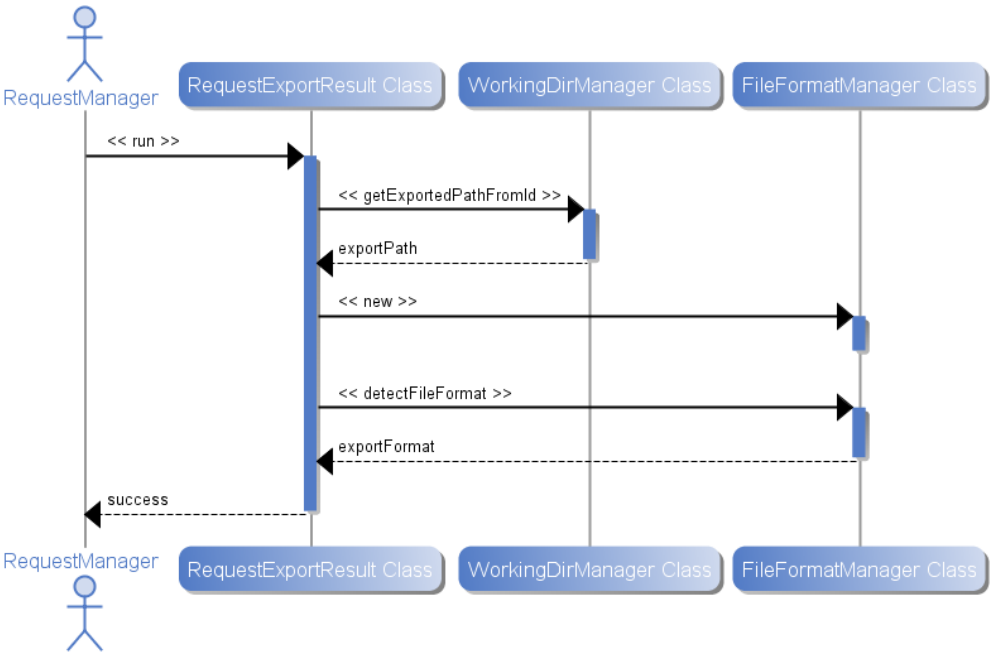
3.2.3.15 RequestExportResult

Cette opération a pour rôle de récupérer les informations (chemin et format) du fichier exporté.

3.2.3.15.1 Paramètres d'entrée :

- id : l'identifiant de l'opération

3.2.3.15.2 Exécution :



3.2.3.15.3 Sortie :

La sortie est de type « OUTPUT_RESULTFILE » (cf. 3.2.4), les paramètres écrits sont :

- path : le chemin vers le fichier exporté
- format : le format du fichier exporté

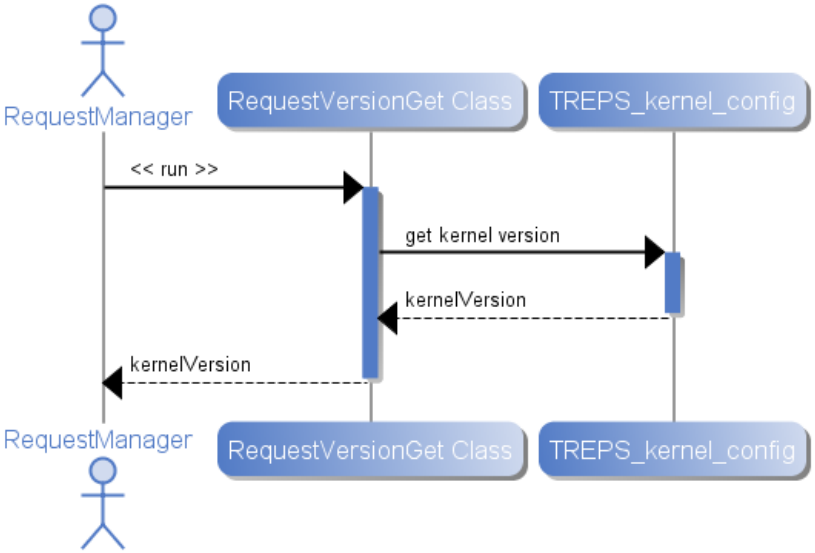
3.2.3.16 RequestVersionGet

Cette opération a pour rôle de récupérer la version de la sous-application TREPS-Kernel courante.

3.2.3.16.1 Paramètres d'entrée :

Aucun paramètre d'entrée n'est requis.

3.2.3.16.2 Exécution :



3.2.3.16.3 Sortie :

La sortie est de type « OUTPUT_STRING» (cf. 3.2.4).

3.2.4 Types de sortie d'une requête

Chaque requête est associée à un type de sortie.

Les types de sortie disponibles sont :

- OUTPUT_NONE :
  - une chaine de caractère vide si une erreur est détectée lors de l'exécution de la requête
  - la chaine de caractère « OK » sinon.
- OUTPUT_XMLFILE : retourne le chemin vers un fichier XML de données (cf. 3.2.5) s'il est accessible et valide, une chaine de caractère vide sinon.
- OUTPUT_STRING : retourne une chaine de caractère (utilisé pour retourner un statut, ou la version)
- OUTPUT_RESULTFILE : retourne le chemin vers le fichier XML résultat de la requête (cf. 4.1.5)

3.2.5 Description des données

Les données de l'application TREPS-Kernel se présentent sous forme de fichiers XML disponibles dans le paquetage « data » (cf. 3.2.1) :

- exports.xml : définition des différentes structures disponibles pour le fichier exporté
- formats.xml : définition des formats de fichier disponibles
- frames.xml : définition des repères disponibles
- times.xml : définition des formats de temps disponibles

A l'ensemble de ces fichiers est associés un schéma XML de validation, présent dans le paquetage « xsd » (cf. 3.2.1).

3.2.6 Description des messages d'erreur

Une journalisation basée sur l'utilisation de Log4Cxx est utilisée.

La stratégie de journalisation est basée sur 5 niveaux :

- FATAL : trace les erreurs qui obligent un arrêt de l'exécutable. A utiliser que dans les cas extrêmes, sur des problèmes system ne permettant pas de remettre le programme dans un état stable.
- ERROR : trace les erreurs fonctionnels et systèmes qui engendrent un retour d'erreur à l'utilisateur. Le code Erreur ainsi que la raison indiquée en anglais doivent être affichés.
- WARN : trace les warnings fonctionnels. Un warning étant une erreur qui dégrade la fonction mais n'empêche pas d'obtenir un résultat.
- INFO : trace des messages d'information relatifs à l'exécution de l'exécutable.
- DEBUG : est utilisé à discrétion du développeur.

La journalisation indique la date, la couche logicielle et composante ayant effectuée la trace et un message explicite en anglais.

3.3 CONCEPTION DETAILLEE DE LA SOUS-APPLICATION TREPS-COM

La sous-application TREPS-Com est composée :

- D'un ensemble de composants implémentant un serveur Ext.Direct :
  - Configuration : pour spécifier les composants à exposer à TREPS-IHM
  - API : pour, à partir de la Configuration, produire un descripteur utilisé côté TREPS-IHM.
  - Router : pour diriger les requêtes vers la fonction de la classe « TREPSAction » associée, et renvoyer le résultat JSON
  - TREPSAction : une classe implémentant les différentes méthodes accessibles via Ext.Direct.
- D'une classe « RequestManager », dont le rôle est de :
  - Produire un fichier d'entrée destiné à TREPS-Kernel
  - Exécuter TREPS-Kernel avec le fichier d'entrée
  - Récupérer le résultat de l'exécution de TREPS-Kernel et produire un résultat sous forme d' « array » exploitable ensuite par le router Ext.Direct
- D'un script « UploadFile » commandant l'upload d'un fichier source destiné à TREPS-Kernel
- D'un script « DownloadExport » commandant le download d'un fichier exporté par TREPS-Kernel

4 ANNEXES

4.1 INTERFACES EXTERNES

Cette section concerne les interfaces externes de la sous-application TREPS-Kernel.

L'exécutable TREPS-Kernel est utilisable en ligne de commande, de la manière suivante :

- treps_kernel ARG

ARG peut prendre les valeurs suivantes :

- unittest : afin d'exécuter les tests unitaires internes
- un chemin vers le fichier XML de définition de la requête

4.1.1 Fichier de configuration de l'application

Le fichier de configuration de l'application TREPS-Kernel doit impérativement être présent au chemin suivant : « <EXEC_DIR>/config/app.config », avec <EXEC_DIR> le répertoire de l'exécutable.

C'est un fichier ASCII de type clef=valeur suivit d'un retour chariot.

Le caractère # en première colonne du fichier indique que la ligne est un commentaire.

Les différentes options de configuration sont présentées dans le manuel d'installation [R3].

4.1.2 Fichier de configuration de log4Cxx

Le fichier de configuration log4Cxx doit impérativement être dans le paquetage « data » (cf. 3.2.1).

Le nom du fichier de configuration est paramétrable dans la configuration de l'application (cf. 4.1.1).

Ce fichier de configuration respecte la syntaxe des fichiers de « properties » de log4j. Une documentation est disponible sur la page suivante:

http://pic.dhe.ibm.com/infocenter/p8docs/v4r5m1/index.jsp?topic=%2Fcom.ibm.p8.doc%2Fdeveloper_help%2Fcontent_engine_api%2Fguide%2Flogging_procedures.htm

4.1.3 Fichier de définition de la requête

Le fichier de définition de la requête est un fichier XML, dont le squelette doit impérativement respecter le schéma « request.xsd » présent dans le paquetage « xsd » de TREPS-Kernel (cf. 3.2.1) :

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="treps">
    <xs:complexType>
```

```
      <xs:sequence>
        <xs:element name="request">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="arg" maxOccurs="unbounded" minOccurs="0">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:string" name="name" use="required"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute type="xs:string" name="type" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

En cas d'échec de la validation du fichier requête avec ce fichier xsd, l'application sortira en erreur.

Les différents arguments nécessaires à l'exécution d'une requête d'un type donné sont présentés dans 3.2.3.

Remarque : des exemples de fichiers requête sont disponibles depuis le paquetage « test » (cf. 3.2.1), dans le répertoire « Fitnesse/requests/ »

Voici la correspondance entre la définition du type de requête dans le fichier de définition de la requête, et la classe Requête associée (cf. 3.2.3) :

Type	Classe	Type	Classe	Type	Classe
new_op	RequestNewOp	frames_get	RequestFramesGet	status_get	RequestStatusGet
reset_op	RequestResetOp	run_op	RequestRunOp	times_get	RequestTimesGet
source_info	RequestSourceInfo	result_info	RequestResultInfo	exports_get	RequestExportsGet
source_get	RequestSourceGet	result_get	RequestResultGet	formats_get	RequestFormatsGet
export_op	RequestExportOp	export_result	RequestExportResult	run_info	RequestRunInfo
version_get	RequestVersionGet				

4.1.4 Interface Web Service CDPP/3DView

Le calcul d'un changement de repère dans TREPS-Kernel est effectué par le biais d'appels au Web Service CDPP/3DView (cf. [A1]).

Les méthodes utilisées sont :

- listFrames2 : afin de récupérer la liste des repères disponibles
- listNewFrameOrb : afin de procéder à un changement de repère sur un vecteur représentant une position
- listNewFrameAtt : afin de procéder à un changement de repère sur un vecteur ne représentant pas une position

4.1.5 Retours de l'application

En cas d'un dysfonctionnement de l'application, le code retour 1 est renvoyé.

En cas de fonctionnement nominal, le code retour 0 est renvoyé et la sortie d'une requête (cf. 3.2.4) est retournée par l'application sur une ligne de la forme : « TREPS_RESULT OUTPUT ».

Dans le cas d'une requête dont la sortie est de type OUTPUT_RESULTFILE, le fichier résultat est un fichier XML dont la structure est la suivante :

```
<?xml version="1.0"?>
<treps>
  <result success="true">
  </result>
</treps>
```

La liste des paramètres de sortie pour une requête donnée (cf. 3.2.3) est incluse dans le nœud « result ».

En cas d'erreur, « success » prend la valeur false, et l'argument « message » contenant le message de l'erreur est ajouté.

4.2 LOGICIELS REUTILISES

COTS	Version	Fonctions	License	Page web	Commentaires
TREPS-IHM					
ExtJS	4.2.1	Framework JavaScript pour application RIA	GPL v3	http://www.sencha.com/products/extjs/	
sampjs	N/A	Librairie JavaScript permettant de faire communiquer une page web avec le protocole SAMP	Libre	https://github.com/astrojs/sampjs/	Implémentation du Web Profile (spécification 1,3 du protocole)
TREPS-Kernel					
libxml2	2.9.1	Parseur et toolkit en C pour les fichiers XML	MIT License	http://xmlsoft.org/	
CDF software	3.5.0	Outils et librairies de manipulation des fichiers CDF	Libre	http://cdf.gsfc.nasa.gov/	
netCDF C++ librairies	4.2.1	Librairie de manipulation des fichiers netCDF	Libre	http://www.unidata.ucar.edu/software/netcdf/	
gSOAP toolkit	2.8.17	Toolkit en C++ pour les Web Services SOAP	gSOAP public license	http://www.cs.fsu.edu/~engelen/soap.html	Utilisé pour générer un client SOAP pour le Web Service CDPP/3DView
file	5.15	Implémentation de la commande Unix File	Libre	http://www.darwinsys.com/file/	Utilisé pour déterminer le type MIME d'un fichier
TREPS-Com					
Ext.Direct PHP	4.2.1	Implémentation PHP de la partie serveur d'Ext.Direct	GPL v3	http://www.sencha.com/products/extjs/	Exemple fourni avec le package d'ExtJS
Traces, Tests et Validation					
log4cxx	0.10.0	Framework de traces pour les applications C++	Apache License v2	http://logging.apache.org/log4cxx/	
Apache Portable Runtime	1.5.0	Librairie Apache assurant la portabilité	Apache License v2	https://apr.apache.org/	Utilisé par log4cxx
Apache Portable Runtime Utility	1.5.3	Librairie Apache assurant la portabilité	Apache License v2	https://apr.apache.org/	Utilisé par log4cxx
CppUnit	1.10.2	Framework de tests unitaires en C++	GNU Lesser General Public License	http://cppunit.sourceforge.net	

FitNesse	20130530	Outil de développement collaboratif axé sur les tests d'acceptation	GPL v3	http://fitnesse.org/	Utilisé pour les tests unitaires de TREPS-Kernel
cppcheck	1.63	Outil d'analyse statique de codes C++	GNU Lesser General Public License	http://cppcheck.sourceforge.net/	Utilisé pour rapport Sonar
Cpputest	1.12.1	Framework de tests unitaires en C++	Libre	http://cpputest.github.io/	Utilisé pour le serveur CSliM (FitNesse)
CSlim	N/A	Implémentation C de FitNesse SliM	Libre	https://github.com/dougbradbury/cslim	Utilisé pour le serveur CSliM (FitNesse)
Sonar-cxx-plugin	0.9	Plugin Sonar pour la prise en charge du langage C++	Libre	https://github.com/wenns/sonar-cxx	Utilisé par Sonar
SonarQube	4.0	Plateforme de gestion de qualité de code	GPL v3	http://www.sonarqube.org/	
Sonar-runner	2.3	Analyseur de projet pour SonarQube	GPL v3	https://github.com/Sonarsource/sonar-runner	Utilisé pour générer des rapports Sonar
Valgrind	3.8.1	Outil de vérification de la gestion de la mémoire en C++	GPL v2	http://valgrind.org/	Utilisé lors de la génération des rapports Sonar

5 DOCUMENTS APPLICABLES ET DE REFERENCE (A/R)

A/R	Référence	Titre
[R1]	CDPP-CD-32500-436-SI	Dossier de conception du noyau AMDA-NG
[R2]	CDPP-AR-32500-382-SI	Dossier d'architecture du noyau d'AMDA-NG
[R3]	CDPP-MI-32100-483-SIL	Manuel d'installation de l'outil de Changement de Repère TREPS
[A1]	IMPEx-IF-1-2-GFI	Interfaces Specifications IMPEx

6 ABREVIATIONS

Abréviation	Nom détaillé
CDPP	Centre de Données de la Physique des Plasmas
CNES	Centre National d'Etudes Spatiales
IRAP	Institut de Recherche en Astrophysique et Planétologie
TREPS	Transformation de REpère pour la Physique Spatiales
SAMP	Simple Application Messaging Protocol