VITAM - Modèle de workflow

Version 0.20.0

VITAM

juil. 21, 2017

Table des matières

CHAPITRE 1

Introduction

1.1 Avertissement

Cette documentation est un travail en cours. Elle est susceptible de changer dans les prochaines releases.

1.2 Objectif du document

Ce document a pour objectif de présenter les différents processus employés par la solution logicielle Vitam. Il est destiné aux administrateurs aussi bien techniques que fonctionnels, aux archivistes souhaitant une connaissance plus avancée du logiciel ainsi qu'aux développeurs.

Il explicite chaque processus (appellés également "workflow"), et pour chacun leurs tâches et traitements.

Ce document comprend également du matériel additionnel pour faciliter la compréhension des processus comme des fiches récaputilatives et des schémas. Il explique également la manière dont est formée la structure des fichiers de workflow.

1.3 Description d'un processus

Un workflow est un processus composé d'étapes (macro-workflow), elles-mêmes composées d'une liste d'actions à exécuter de manière séquentielle, une seule fois ou répétées sur une liste d'éléments (micro-workflow).

Pour chacun de ces éléments, le document décrit :

- La règle générale qui s'applique à cet élément
- Les statuts de sortie possibles (OK,KO...), avec les raisons de ces sorties et les clés associées
- Des informations complémentaires, selon le type d'élément traité

Chaque étape, chaque action peuvent avoir les statuts suivants :

- OK : le traitement associé s'est passé correctement. Le workflow continue.
- Warning : le traitement associé a généré un avertissement (Par exemple le format de l'objet est mal déclaré dans le bordereau, aussi appellé "manifeste"). Le workflow continue.
- KO : le traitement associé a généré une erreur métier. Le workflow s'arrête si le modèle d'execution est bloquant (cf. ci-dessous).
- FATAL : le traitement associé a généré une erreur technique. Le workflow s'arrête.

Chaque action peut avoir les modèles d'éxécutions suivants (toutes les étapes sont par défaut bloquantes) :

- Bloquant

    - Si une action est identifiée en erreur, l'étape en cours est alors arrêtée et le workflow passe à un nouvel état. Dans certains cas, il est directement terminé en erreur alors que dans d'autres, ils passent à une étape de finalisation. Ces comportements spécifiques sont décrits dans chaque workflow.
- Non bloquant
    - Si une action est identifiée en erreur, le reste des actions de l'étape est exécuté avant que le statut de l'étape passe à « erreur ». Le workflow passe alors à un nouvel état. Dans certains cas, il est directement terminé en erreur alors que dans d'autres, ils passent à une étape de finalisation. Ces comportements spécifiques sont décrits dans chaque workflow.

1.4 Structure d'un fichier Properties du Worflow

Le fichier Properties permet de définir la structure du Workflow pour les étapes et actions réalisées dans le module d'Ingest Interne, en excluant les étapes et actions réalisées dans le module d'Ingest externe.

La structure du fichier est la suivante :

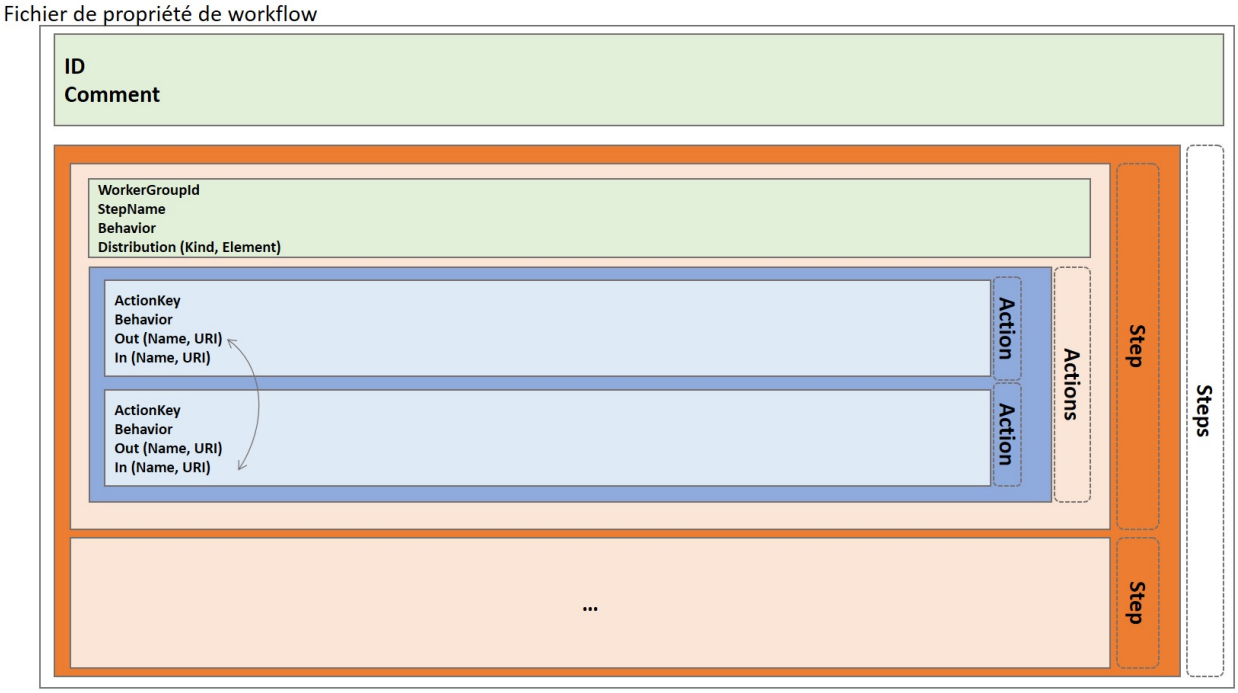


Fig. 1.1 – Structure du fichier de définition du workflow, un exemple est donné dans la figure suivante

Un Workflow est défini en JSON avec la structure suivante :

- un bloc en-tête contenant :
    - `ID` : identifiant unique du workflow,
    - `Comment` : description du workflow ou toutes autres informations utiles concernant le workflow
- une liste d'étapes dont la structure est la suivante :
    - `workerGroupId` : identifiant de famille de Workers,
    - `stepName` : nom de l'étape, servant de clé pour identifier l'étape,
    - `Behavior` : modèle d'exécution pouvant avoir les types suivants :

    - BLOCKING : le traitement est bloqué en cas d'erreur, il est nécessaire de recommencer le workflow. Les étapes FINALLY (définition ci-dessous) sont tout de même exécutées
    - NOBLOCKING : le traitement peut continuer malgré les erreurs ou avertissements,
    - FINALLY : le traitement correspondant est toujours exécuté, même si les étapes précédentes se sont terminées en échec
- `Distribution` : modèle de distribution, décrit comme suit :
    - `Kind` : un type pouvant être REF (i.e. élément unique) ou LIST (i.e. liste d'éléments)
    - `Element` : l'élément de distribution indiquant l'élément unique sous forme d'URI (REF) ou la liste d'éléments en pointant vers un dossier (LIST).
- une liste d'Actions :
    - `ActionKey` : nom de l'action
    - `Behavior` : modèle d'exécution pouvant avoir les types suivants :
        - BLOCKING : l'action est bloquante en cas d'erreur. Les actions suivantes (de la même étape) ne seront pas éxécutées.
        - NOBLOCKING : l'action peut continuer malgré les erreurs ou avertissements.
    - `In` : liste de paramètres d'entrées :
        - `Name` : nom utilisé pour référencer cet élément entre différents handlers d'une même étape,
        - `URI` : cible comportant un schema (WORKSPACE, MEMORY, VALUE) et un path où chaque handler peut accéder à ces valeurs via le handlerIO :
            - WORKSPACE : path indiquant le chemin relatif sur le workspace (implicitement un File),
            - MEMORY : path indiquant le nom de la clef de valeur (implicitement un objet mémoire déjà alloué par un Handler précédent),
            - VALUE : path indiquant la valeur statique en entrée (implicitement une valeur String).
    - `Out` : liste de paramètres de sorties :
        - `Name` : nom utilisé pour référencer cet élément entre différents handlers d'une même étape,
        - `URI` : cible comportant un schema (WORKSPACE, MEMORY) et un path où chaque handler peut stocker les valeurs finales via le handlerIO :
            - WORKSPACE : path indique le chemin relatif sur le workspace (implicitement un File local),
            - MEMORY : path indique le nom de la clef de valeur (implicitement un objet mémoire).


{
"id": "DefaultIngestWorkflow",
"comment": "Default Ingest Workflow V6",
Bloc d'entête
"steps": [
{
"workerGroupId": "DefaultWorker",
"stepName": "STP_INGEST_CONTROL_SIP",
"behavior": "BLOCKING",
Liste d'étapes
"distribution": {
"kind": "REF",
"element": "SIP/manifest.xml"
},
"actions": [
{
"action": {
"actionKey": "CHECK_SEDA",
"behavior": "BLOCKING"
}
},
Liste d'actions de l'étape
{
"action": {
"actionKey": "CHECK_MANIFEST",
"behavior": "BLOCKING",
"out": [
{
"name": "mapsBDOtoOG.file",
"uri": "WORKSPACE:Maps/OG_TO_ARCHIVE_ID_MAP.json"
}
]
}
},
{
"action": {
"actionKey": "CHECK_CONSISTENCY",
"behavior": "NOBLOCKING",
"in": [
{
"name": "mapsBDOtoOG.file",
"uri": "WORKSPACE:Maps/OG_TO_ARCHIVE_ID_MAP.json"
}
]
}
}
]
}

CHAPITRE 2

INGEST

2.1 Workflow d'entrée

2.1.1 Introduction

Cette section décrit le processus (workflow) d'entrée, utilisé lors du transfert d'un Submission Information Package (SIP) dans la solution logicielle Vitam. Ce workflow se décompose en deux grosses catégories : le processus d'entrée externe "ingest externe" et le processus d'entrée interne "ingest interne". Le premier prend en charge le SIP et effectue des contrôles techniques préalables, tandis que le second débute dès le premier traitement métier.

Toutes les étapes et actions sont journalisées dans le journal des opérations. Les étapes et actions associées ci-dessous décrivent le processus d'entrée (clé et description de la clé associée dans le journal des opérations) tel qu'implémenté dans la version actuelle de la solution logicielle Vitam.

Le processus d'entrée externe comprend l'étape : STP_SANITY_CHECK_SIP. Les autres étapes font partie du processus d'entrée interne.

2.1.2 Le cas du processus d'entrée "test à blanc"

Il est possible de procéder à un versement "à blanc", pour tester la conformité du SIP par rapport à la forme attendue par la solution logicielle Vitam sans pour autant le prendre en charge. Dans ce cas, le processus d'entrée à blanc diffère du processus d'entrée "classique" en ignorant un certain nombre d'étapes.

Les étapes non exécutées dans le processus d'entrée à blanc sont les suivantes :

- Rangement des objets (STP_OG_STORING)
- Rangement des unites archivistiques (STP_UNIT_STORING)
- Registre des fonds (STP_ACCESSION_REGISTRATION)

Les tâches relatives à toutes ces étapes sont donc également ignorées.

2.1.3 Contrôles préalables à l'entrée (STP_SANITY_CHECK_SIP)

2.1.3.1 Contrôle sanitaire (SANITY_CHECK_SIP)

- **Règle** : vérification de l'absence de virus dans le SIP.
- **Type** : bloquant.
- **Statuts** :

- OK : aucun virus n'est détecté dans le SIP (SANITY_CHECK_SIP.OK=Succès du contrôle sanitaire : aucun virus détecté)
- KO : un ou plusieurs virus ont été detectés dans le SIP (SANITY_CHECK_SIP.KO=Échec du contrôle sanitaire du SIP : fichier détecté comme infecté)
- FATAL : la vérification de la présence de virus dans le SIP n'a pas pu être faite suite à une erreur technique (SANITY_CHECK_SIP.FATAL=Erreur fatale lors du contrôle sanitaire du SIP)

2.1.3.2 Contrôle du format du conteneur du SIP (CHECK_CONTAINER)

- **Règle** : vérification du format du SIP via un outil d'identification de format, qui lui même se base sur le référientiel des formats qu'il embarque
- **Formats acceptés** : .zip, .tar, .tar.gz, .tar.bz2
- **Type** : bloquant.
- **Statuts** :
  - OK : le conteneur du SIP est au bon format (CHECK_CONTAINER.OK=Succès du contrôle de format du conteneur du SIP)
  - KO : le conteneur du SIP n'est pas au bon format (CHECK_CONTAINER.KO=Échec du contrôle de format du conteneur du SIP)
  - FATAL : la vérification du format du conteneur du SIP n'a pas pu être faite suite à une erreur technique liée à l'outil d'identification des formats (CHECK_CONTAINER.FATAL=Erreur fatale lors du processus du contrôle de format du conteneur du SIP)

2.1.4 Réception dans vitam (STP_UPLOAD_SIP) : Etape de réception du SIP dans Vitam

- **Règle** : vérification de la bonne réception du SIP dans l'espace de travail interne ("workspace")
- **Type** : bloquant.
- **Statuts** :
  - OK : le SIP a été réceptionné dans l'espace de travail interne (STP_UPLOAD_SIP.OK=Succès du processus de téléchargement du SIP)
  - KO : le SIP n'a pas réceptionné dans l'espace de travail interne (STP_UPLOAD_SIP.KO=Échec du processus de téléchargement du SIP)
  - FATAL : la réception du SIP dans la solution logicelle Vitam n'a pas été possible suite à une erreur technique, par exemple un indisponibilité du serveur (STP_UPLOAD_SIP.FATAL=Erreur Fatale lors du processus de téléchargement du SIP)

2.1.5 Contrôle du SIP (STP_INGEST_CONTROL_SIP)

2.1.5.1 Vérification globale du SIP (CHECK_SEDA) : Vérification de la cohérence physique du SIP

- **Règle** : vérification du SIP reçu par rapport au type de SIP accepté
- **Type de SIP accepté** : le manifeste, obligatoire dans le SIP, doit être nommé manifest.xml, doit être conforme au schéma xsd par défaut fourni avec le standard SEDA v. 2.0, doit satisfaire les exigences du document "Spécification des SIP" pour la solution logicielle Vitam et doit posséder un unique répertoire nommé "Content"
- **Type** : bloquant.
- **Statuts** :

- OK : le SIP est présent, nommé manifest.xml et conforme au schéma xsd par défaut fourni avec le standard SEDA v.2.0. (CHECK_SEDA.OK=Succès de la vérification globale du SIP)
- KO : - Cas 1 : le manifeste est introuvable dans le SIP ou n'a pas d'extension .xml (CHECK_SEDA.NO_FILE.KO=Échec de la vérification globale du SIP : le manifeste est introuvable dans le SIP) - Cas 2 : le manifeste n'est pas au format XML (CHECK_SEDA.NOT_XML_FILE.KO=Échec de la vérification globale du SIP : le manifeste est au mauvais format) - Cas 3 : le manifeste ne respecte pas le schéma par défaut fourni avec le standard SEDA 2.0 (CHECK_SEDA.NOT_XSD_VALID.KO=Échec de la vérification globale du SIP : manifeste non conforme au schéma SEDA 2.0) - Cas 4 : le SIP contient plus d'un seul dossier "Content" (CHECK_SEDA.CONTAINER_FORMAT.DIRECTORY.KO=Le SIP contient plus d'un dossier ou un dossier dont le nommage est invalide) - Cas 5 : le SIP contient plus d'un seul fichier à la racine (CHECK_SEDA.CONTAINER_FORMAT.FILE.KO=Le SIP contient plus d'un fichier à sa racine)
- FATAL : le SIP n'a pas pu être contrôlé suite à une erreur technique (CHECK_SEDA.FATAL=Erreur fatale lors de la vérification globale du SIP)

2.1.5.2 Vérification de l'en-tête du manieste (CHECK_HEADER)

- **Règles** : vérification des informations du manifest.xml (nommées "header") et de l'existence du service producteur (OriginatingAgencyIdentifier)
- **Type** : bloquant.
- **Statuts** :
  - OK : les informations du manifeste sont conformes et le service producteur est déclaré. (CHECK_HEADER.OK=Succès de la vérification générale du bordereau)
  - KO : les informations du manifeste ne sont pas conformes ou le service producteur n'est pas déclaré (CHECK_HEADER.KO=Échec de la vérification générale du bordereau)
  - FATAL : une erreur technique est survenue lors des contrôles sur les informations générales du manifeste (CHECK_HEADER.FATAL=Erreur fatale lors de la vérification générale du bordereau)

2.1.5.2.1 La tâche contient les traitements suivants

- Vérification de la relation entre le contrat et le profil SEDA (CHECK_IC_AP_RELATION)
  - **Règle** : le profil SEDA déclaré dans le contrat d'entrée du SIP doit être le même que celui déclaré dans son manifeste. Si aucun profil SEDA ne s'applique au SIP, ce traitement est ignoré.
  - **Statuts** :
    - OK : le profil SEDA déclaré dans le contrat d'entrée et celui déclaré dans le manifeste sont bien les mêmes (CHECK_HEADER.CHECK_IC_AP_RELATION.OK=Succès de la vérification de la relation entre le contrat et le profil SEDA)
    - KO : le profil déclaré dans le contrat d'entrée et celui déclaré dans le manifeste ne sont pas les mêmes (CHECK_HEADER.CHECK_IC_AP_RELATION.OK=Succès de la vérification de la relation entre le contrat et le profil SEDA)
    - FATAL : une erreur technique est survenue lors de la vérification de la relation (CHECK_HEADER.CHECK_IC_AP_RELATION.FATAL=Erreur fatale lors de la vérification de la relation entre le contrat et le profil SEDA)
- Vérification de la présence et contrôle du contrat d'entrée (CHECK_CONTRACT_INGEST)
  - **Règle** : vérification du contrat d'entrée déclaré dans le SIP par rapport au référentiel des contrats d'entrée présent dans le système.
  - **Statuts** :

    - OK : le contrat déclaré dans le SIP est valide (contrat trouvé dans le référentiel des contrats et contrat trouvé au statuf actif)
    - KO : le contrat déclaré dans le SIP est invalide (contrat non trouvé dans la référentiel de contrat OU contrat trouvé mais en statut inactif
    - FATAL : une erreur technique est survenue lors de la vérification de la présence et du contrôle du contrat d'entrée
- Vérification de la conformité du manifeste par le profil SEDA (CHECK_ARCHIVEPROFILE)
    - **Règle** : le manifeste du SIP doit être conforme aux exigences du profil SEDA. Si aucun profil SEDA ne s'applique au SIP, ce traitement est ignoré.
    - **Statuts** :
        - OK : le manifeste est conforme aux exigences du profil SEDA (CHECK_ARCHIVEPROFILE.OK=Succès de la vérification de la conformité au profil SEDA)
        - KO : le manifeste n'est pas conforme aux exigences du profil SEDA (CHECK_ARCHIVEPROFILE.KO=Echec de la vérification de la conformité au profil SEDA)
        - FATAL : une erreur technique est survenue lors de la vérification du manifeste par le profil SEDA (CHECK_ARCHIVEPROFILE.FATAL=Erreur fatale lors de la vérification de la conformité au profil SEDA)

2.1.5.3 Vérification du contenu du bordereau (CHECK_DATAOBJECTPACKAGE)

- **Type** : bloquant.

Cette tâche contient plusieurs traitements, chacun ayant une finalité et des points de sorties spécifiques

- Vérification des usages des groupes d'objets (CHECK_MANIFEST_DATAOBJECT_VERSION)
    - **Règle** : tous les objets décrits dans le manifeste du SIP doivent déclarer un usage conforme à la liste des usages acceptés dans la solution logicielle Vitam ainsi qu'un numéro de version respectant la norme de ce champ.
    - **Types d'usages acceptés** : original papier (PhysicalMaster), original numérique (BinaryMaster), diffusion (Dissemination), vignette (Thumbnail), contenu brut (TextContent). Pour les numéros de version, il s'agit d'un entier positif ou nul (0, 1, 2...). La grammaire est : "usage_version"
    - **Statuts** :
        - OK : les objets contenus dans le SIP déclarent tous dans le manifeste un usage cohérent avec ceux acceptés, et optionnellement un numéro de version respectant la norme de ce champ usage, par exemple "BinaryMaster_2" (CHECK_MANIFEST_DATAOBJECT_VERSION.OK=Succès de la vérification des usages des groupes d'objets)
        - KO : un ou plusieurs objets contenus dans le SIP déclarent dans le manifeste un usage ou un numéro de version incohérent avec ceux acceptés (CHECK_MANIFEST_DATAOBJECT_VERSION.KO=Échec de la vérification des usages des groupes d'objets)
        - FATAL : les usages déclarés dans le manifeste pour les objets contenus dans le SIP n'ont pas pu être contrôlés suite à une erreur technique (CHECK_MANIFEST_DATAOBJECT_VERSION.FATAL=Erreur fatale lors de la vérification des usages des groupes d'objets)
- Vérification du nombre d'objets (CHECK_MANIFEST_OBJECTNUMBER)
    - **Règle** : le nombre d'objets binaires reçus dans la solution logicielle Vitam doit être strictement égal au nombre d'objets binaires déclaré dans le manifeste du SIP
    - **Statuts** :

        - OK : le nombre d'objets reçus dans la solution logicielle Vitam est strictement égal au nombre d'objets déclaré dans le manifeste du SIP (CHECK_MANIFEST_OBJECTNUMBER.OK=Succès de la vérification du nombre d'objets)
        - KO : le nombre d'objets reçus dans la solution logicielle Vitam est inférieur ou supérieur au nombre d'objets déclaré dans le manifeste du SIP (CHECK_MANIFEST_OBJECTNUMBER.KO=Échec de la vérification du nombre d'objets)
        - FATAL : une erreur technique est survenue lors de la vérification du nombre d'objets (CHECK_DATAOBJECTPACKAGE.CHECK_MANIFEST_OBJECTNUMBER.FATAL=Erreur fatale lors de la vérification du nombre dobjets)
- Vérification de la cohérence du bordereau (CHECK_MANIFEST)
    - **Règle** : création des journaux du cycle de vie des unités archivistiques et des groupes d'objets, extraction des unités archivistiques, objets binaires et objets physiques, vérification de la présence de cycles dans les arborescences des unités archivistiques et création de l'arbre d'ordre d'indexation, extraction des métadonnées contenues dans la balise ManagementMetadata du manifeste pour le calcul des règles de gestion, vérification de la validité du rattachement des unités du SIP aux unités présentes dans le système si demandé, détection des problèmes d'encodage dans le manifeste et vérification que les objets ne font pas référence directement à des unités si ces objets possèdent des groupes d'objets.
    - **Statuts** :
        - OK : les journaux du cycle de vie des unités archivistiques et des groupes d'objets ont été créés avec succès, aucune récursivité n'a été detectée dans l'arborescence des unités archivistiques, la structure de rattachement déclarée existe (par exemple, un SIP peut être rattaché à un plan de classement, mais pas l'inverse), le type de structure de rattachement est autorisé, aucun problème d'encodage détecté et les objets avec groupe d'objets ne référencent pas directement les unités (CHECK_MANIFEST.OK=Contrôle du bordereau réalisé avec succès)
        - KO : Une récursivité a été détectée dans l'arborescence des unités archivistiques, la structure de rattachement déclarée est inexistante, le type de structure de rattachement est interdit, il y a un problème d'encodage ou des objets avec groupe d'objets référencent directement des unités (CHECK_MANIFEST.KO=Échec de contrôle du bordereau)
        - FATAL : la vérification de la cohérence du bordereau n'a pas pu être réalisée suite à une erreur système, e.g. les journaux du cycle de vie n'ont pas pu être créés (CHECK_MANIFEST.FATAL=Erreur fatale lors de contrôle du bordereau)
- Vérification de la cohérence entre objets, groupes d'objets et unités archivistiques (CHECK_CONSISTENCY)
    - **Règle** : vérification que chaque objet ou groupe d'objets est référencé par une unité archivistique, rattachement à un groupe d'objet pour les objets sans groupe d'objet mais référencés par une unité archivistique, création de la table de concordance (MAP) pour les identifiants des objets et des unités du SIP et génération de leurs identifiants Vitam (GUID)
    - **Statuts** :
        - OK : Aucun objet ou groupe d'objet n'est orphelin (i.e. non référencé par une unité archivistique) et tous les objets sont rattachés à un groupe d'objets (CHECK_CONSISTENCY.OK=Succès de la vérification de la cohérence entre objets, groupes d'objets et unités archivistiques)
        - KO : Au moins un objet ou groupe d'objet est orphelin (i.e. non référencé par une unité archivistique) (CHECK_CONSISTENCY.KO=Échec de la vérification de la cohérence entre objets, groupes d'objets et unités archivistiques)
        - FATAL : la vérification de la cohérence entre objets, groupes d'objets et unités archivistiques n'a pas pu être réalisée suite à une erreur système (CHECK_CONSISTENCY.FATAL=Erreur fatale lors de la vérification de la cohérence entre objets, groupes d'objets et unités archivistiques)

2.1.6 Contrôle et traitements des objets (STP_OG_CHECK_AND_PROCESS)

2.1.6.1 Vérification de l'intégrité des objets (CHECK_DIGEST)

- **Règle** : vérification de la cohérence entre l'empreinte de l'objet binaire calculée par la solution logicielle Vitam et celle déclarée dans le manifeste. Si l'empreinte déclarée dans le manifeste n'a pas été calculée avec l'algorithme SHA-512, alors le système recalcule une empreinte avec cet algorithme. C'est celle-ci qui sera enregistrée dans le système.
- **Algorithmes autorisés en entrée** : MD5, SHA-1, SHA-256, SHA-512
- **Type** : bloquant.
- **Statuts** :
  - OK : tous les objets binaires reçus sont identiques aux objets binaires attendus. Tous les objets binaires disposent désormais d'une empreinte calculée avec l'algorithme SHA-512 (CHECK_DIGEST.OK=Succès de la vérification de l'intégrité des objets binaires)
  - KO : au moins un objet reçu n'est pas identique aux objets attendus (CHECK_DIGEST.KO=Échec de la vérification de l'intégrité des objets binaires)
  - FATAL : la vérification de l'intégrité des objets binaires n'a pas pu être réalisée suite à une erreur système, par exemple lorsque l'algorithme inconnu (CHECK_DIGEST.FATAL=Erreur fatale lors de la vérification des objets)

2.1.6.2 Identification des formats (OG_OBJECTS_FORMAT_CHECK)

- **Règle** : identification des formats de chaque objet binaire présent dans le SIP, afin de garantir une information homogène et objective. Cette action met en œuvre un outil d'identification prenant l'objet en entrée et fournissant des informations de format en sortie. Ces informations sont comparées avec les formats enregistrés dans le référentiel des formats interne au système et avec celles déclarées dans le manifeste. En cas d'incohérence entre la déclaration dans le SIP et le format identifié par le système, le SIP sera tout de même accepté, générant un avertissement. La solution logicielle Vitam se servira alors des informations qu'elle a elle même identifiées et non de celles fournies dans le SIP
- **Type** : bloquant.
- **Statuts** :
  - OK : l'identification s'est bien passée, les formats identifiés sont référencés dans le référentiel interne et les informations sont cohérentes avec celles déclarées dans le manifeste (OG_OBJECTS_FORMAT_CHECK.OK=Succès de la vérification des formats)
  - KO : le format identifié n'est pas référencé dans le référentiel interne, ou aucun format n'a été trouvé pour un objet (OG_OBJECTS_FORMAT_CHECK.KO=Échec de la vérification des formats)
  - FATAL : l'identification des formats n'a pas été réalisée suite à une erreur technique (OG_OBJECTS_FORMAT_CHECK.FATAL=Erreur fatale lors de la vérification des formats)
  - WARNING : l'identification s'est bien passée, les formats identifiés sont référencés dans le référentiel interne mais les informations ne sont pas cohérentes avec celles déclarées dans le manifeste (OG_OBJECTS_FORMAT_CHECK.WARNING=Avertissement lors de la vérification des formats)

2.1.7 Contrôle et traitements des unités archivistiques (STP_UNIT_CHECK_AND_TRANSFORME)

2.1.7.1 Vérification globale de l'unité archivistique (CHECK_UNIT_SCHEMA)

- **Règle** : contrôle additionnel sur la validité des champs de l'unité archivistique par rapport au schéma prédéfini dans la solution logicielle Vitam. Par exemple, les champs obligatoires, comme les titres des unités archivistiques, ne doivent pas être vides.

- **Type** : bloquant.
- **Statuts** :
  - OK : tous les champs de l'unité archivistique sont conformes à ce qui est attendu (CHECK_UNIT_SCHEMA.OK=Succès du contrôle additionnel sur la validité des champs de l'unité archivistique).
  - KO : au moins un champ de l'unité archivistique n'est pas conforme à ce qui est attendu (titre vide, date incorrecte...) (CHECK_UNIT_SCHEMA.KO=Échec lors du contrôle additionnel sur la validité des champs de l'unité archivistique).
  - FATAL : la vérification de l'unité archivistique n'a pu être effectuée suite à une erreur technique (CHECK_UNIT_SCHEMA.FATAL=Erreur fatale du contrôle additionnel sur la validité des champs de l'unité archivistique).

2.1.7.2 Application des règles de gestion et calcul des dates d'échéances (UNITS_RULES_COMPUTE)

- **Règle** : calcul des dates d'échéances des unités archivistiques du SIP. Pour les unités racines, c'est à dire les unités déclarées dans le SIP et n'ayant aucun parent dans l'arborescence, la solution logicielle Vitam utilise les règles de gestions incluses dans le bloc Management de chacune de ces unités ainsi que celles présentes dans le bloc ManagementMetadata. La solution logicielle Vitam effectue également ce calcul pour les autres unités archivistiques du SIP possédant des règles de gestion déclarées dans leurs balises Management, sans prendre en compte le ManagementMetadata. Le référentiel utilisé pour ces calculs est le référentiel des règles de gestion interne au système.
- **Type** : bloquant.
- **Statuts** :
  - OK : les règles de gestion sont référencées dans le référentiel interne et ont été appliquées avec succès (UNITS_RULES_COMPUTE.OK=Succès du calcul des dates d'échéance)
  - KO : au moins une règle de gestion déclarée dans le manifeste n'est pas référencée dans le référentiel interne (UNITS_RULES_COMPUTE.KO=Échec du calcul des dates d'échéance)
  - FATAL : une erreur technique est survenue lors du calcul des dates d'échéances (UNITS_RULES_COMPUTE.FATAL=Erreur fatale lors du calcul des dates d'échéance)

2.1.8 Préparation de la prise en charge (STP_STORAGE_AVAILABILITY_CHECK)

2.1.8.1 Vérification de la disponibilité de l'offre de stockage (STORAGE_AVAILABILITY_CHECK)

- **Règle** : Vérification de la disponibilité des offres de stockage et de l'espace disponible pour y stocker le contenu du SIP compte tenu de la taille des objets à stocker
- **Type** : bloquant.
- **Statuts** :
  - OK : les offres de stockage sont accessibles et disposent d'assez d'espace pour stocker le contenu du SIP (STORAGE_AVAILABILITY_CHECK.OK=Succès de la vérification de la disponibilité de l'offre de stockage)
  - KO : les offres de stockage ne sont pas disponibles ou ne disposent pas d'assez d'espace pour stocker le contenu du SIP (STORAGE_AVAILABILITY_CHECK.KO=Échec de la vérification de la disponibilité de l'offre de stockage)
  - FATAL : la vérification de la disponibilité de l'offre de stockage n'a pas pu être réalisée suite à une erreur technique (STORAGE_AVAILABILITY_CHECK.FATAL=Erreur fatale lors de la vérification de la disponibilité de l'offre de stockage)

2.1.9 Rangement des objets (STP_OG_STORING)

2.1.9.1 Écriture des objets binaires sur les offres de stockage (OG_STORAGE)

- **Règle** : écriture des objets contenus dans le SIP sur les offres de stockage en fonction de la stratégie de stockage applicable
- **Type** : Bloquant.
- **Statuts** :
  - OK : tous les objets binaires contenus dans le SIP ont été écrits sur les offres de stockage (OG_STORAGE.OK=Succès du rangement des objets et groupes d'objets)
  - KO : au moins un des objets binaires contenus dans le SIP n'ont pas pu être écrits sur les offres de stockage (OG_STORAGE.KO=Échec du rangement des objets et groupes d'objets)
  - WARNING : le SIP ne contient pas d'objet (OBJECTS_LIST_EMPTY.WARNING=Avertissement : le SIP ne contient pas dobjet)
  - FATAL : l'écriture des objets binaires sur les offres de stockage n'a pas pu être réalisés suite à une erreur technique (OG_STORAGE.FATAL=Erreur fatale lors du rangement des objets et groupes d'objets)

2.1.9.2 Indexation des métadonnées des groupes d'objets (OG_METADATA_INDEXATION)

- **Règle** : indexation des métadonnées liées aux groupes d'objets, commela taille des objets, les métadonnées liées aux formats (Type MIME, PUID, etc.), l'empreinte des objets, etc.
- **Type** : bloquant.
- **Statuts** :
  - OK : les métadonnées des groupes d'objets ont été indexées avec succès (OG_METADATA_INDEXATION.OK=Succès de l'indexation des métadonnées des objets et groupes d'objets)
  - KO : les métadonnées des groupes d'objets n'ont pas été indexées (OG_METADATA_INDEXATION.KO=Échec de l'indexation des métadonnées des objets et groupes d'objets)
  - FATAL : l'indexation des métadonnées des groupes d'objets n'a pas pu être réalisée suite à une erreur technique (OG_METADATA_INDEXATION.FATAL=Erreur fatale lors de l'indexation des métadonnées des objets et groupes d'objets)

2.1.9.3 Sauvegarde des métadonnées des groupes d'objets (OG_METADATA_STORAGE)

- **Règle** : sauvegarde des métadonnées liées aux groupes d'objets sur les offres de stockage en fonction de la stratégie de stockage
- **Type** : bloquant.
- **Statuts** :
  - OK : les métadonnées des groupes d'objets ont été sauvegardées avec succès (OG_METADATA_STORAGE.OK=Succès de l'enregistrement des métadonnées des groupes d'objets)
  - KO : les métadonnées des groupes d'objets n'ont pas été sauvegardées (OG_METADATA_STORAGE.KO=Échec de l'enregistrement des métadonnées des objets et groupes d'objets)

2.1.9.4 Sécurisation des journaux des cycles de vie des groupes d'objets (COMMIT_LIFE_CYCLE_OBJECT_GROUP)

- **Règle** : sécurisation en base des journaux du cycle de vie des groupes d'objets (avant cette étape, les journaux du cycle de vie des groupes d'objets sont dans une collection temporaire afin de garder une cohérence entre les métadonnées indexées et les journaux lors d'une entrée en succès ou en échec)
- **Type** : bloquant.
- **Statuts** :
  - OK : La sécurisation des journaux du cycle de vie s'est correctement déroulée (COMMIT_LIFE_CYCLE_OBJECT_GROUP.OK=Succès de la sécurisation des journaux du cycle de vie des groupes d'objets)
  - FATAL : La sécurisation du journal du cycle de vie n'a pas pu être réalisée suite à une erreur technique (COMMIT_LIFE_CYCLE_OBJECT_GROUP.FATAL=Erreur fatale lors de la sécurisation des journaux du cycle de vie des groupes d'objets)

2.1.10 Rangement des unités archivistiques (STP_UNIT_STORING)

2.1.10.1 Indexation des métadonnées des unités archivistiques (UNIT_METADATA_INDEXATION)

- **Règle** : indexation des métadonnées liées aux unités archivistiques, c'est à dire le titre des unités, leurs descriptions, leurs dates extrêmes, etc.
- **Type** : bloquant.
- **Statuts** :
  - OK : les métadonnées des unités archivistiques ont été indexées avec succès (UNIT_METADATA_INDEXATION.OK=Succès de l'indexation des métadonnées des unités archivistiques)
  - KO : les métadonnées des unités archivistiques n'ont pas été indexées (UNIT_METADATA_INDEXATION.KO=Échec de l'indexation des métadonnées des unités archivistiques)
  - FATAL : l'indexation des métadonnées des unités archivistiques n'a pas pu être réalisée suite à une erreur technique (UNIT_METADATA_INDEXATION.FATAL=Erreur fatale lors de l'indexation des métadonnées des unités archivistiques)

2.1.10.2 Sauvegarde des métadonnées des unités archivistiques (UNIT_METADATA_STORAGE)

- **Règle** : sauvegarde des métadonnées liées aux unités archivistiques sur les offres de stockage en fonction de la stratégie de stockage
- **Type** : bloquant.
- **Statuts** :
  - OK : les métadonnées des unités archivistiques ont été sauvegardées avec succès (UNIT_METADATA_STORAGE.OK=Succès de l'enregistrement des métadonnées des unités archivistiques)
  - KO : les métadonnées des unités archivistiques n'ont pas pu être sauvegardées (UNIT_METADATA_STORAGE.KO=Échec de l'enregistrement des métadonnées des unités archivistiques)

2.1.10.3 Sécurisation du journal des cycles de vie des unités archivistiques (COMMIT_LIFE_CYCLE_UNIT)

- **Règle** : sécurisation en base des journaux du cycle de vie des unités archivistiques (avant cette étape, les journaux du cycle de vie des unités archivistiques sont dans une collection temporaire afin de garder une cohérence entre les métadonnées indexées et les journaux lors d'une entrée en succès ou en échec)
- **Type** : bloquant.
- **Statuts** :
  - OK : La sécurisation des journaux du cycle de vie s'est correctement déroulée (COMMIT_LIFE_CYCLE_UNIT.OK=Succès de la sécurisation des journaux du cycle de vie des unités archivistiques)
  - FATAL : La sécurisation des journaux du cycle de vie n'a pas pu être réalisée suite à une erreur système (COMMIT_LIFE_CYCLE_UNIT.FATAL=Erreur fatale lors de la sécurisation des journaux du cycle de vie des unités archivistiques)

2.1.11 Registre des fonds (STP_ACCESSION_REGISTRATION)

2.1.11.1 Alimentation du registre des fonds (ACCESSION_REGISTRATION)

- **Règle** : enregistrement dans le registre des fonds des informations concernant la nouvelle entrée (nombre d'objets, volumétrie). Ces informations viennent s'ajouter aux informations existantes pour un même service producteur. Si le service producteur n'existait pas dans le système et qu'il effectue sa première entrée, cette entrée est enregistrée et ce producteur est créé dans la solution logicielle Vitam.
- **Type** : bloquant.
- **Statuts** :
  - OK : le registre des fonds est correctement alimenté (ACCESSION_REGISTRATION.OK=Succès de l'alimentation du registre des fonds)
  - KO : le registre des fonds n'a pas pu être alimenté (ACCESSION_REGISTRATION.KO=Échec de l'alimentation du registre des fonds)
  - FATAL : l'alimentation du registre des fonds n'a pas pu être réalisée suite à une erreur système (ACCESSION_REGISTRATION.FATAL=Erreur fatale lors de l'alimentation du registre des fonds)

2.1.12 Finalisation de l'entrée (STP_INGEST_FINALISATION)

2.1.12.1 Notification de la fin de l'opération d'entrée (ATR_NOTIFICATION)

- **Règle** : génération de la notification de réponse (ArchiveTransferReply ou ATR) une fois toutes les étapes passées avec succès ou lorsqu'une étape est en échec, puis enregistrement de cette notification dans l'offre de stockage et envoi au service versant.
- **Type** : non bloquant.
- **Statuts** :
  - OK : Le message de réponse a été correctement généré, écrit sur l'offre de stockage et envoyé au service versant (ATR_NOTIFICATION.OK=Succès de la notification à l'opérateur de versement)
  - KO : Le message de réponse n'a pas été correctement généré, écrit sur l'offre de stockage ou reçu par le service versant (ATR_NOTIFICATION.KO=Échec de la notification à l'opérateur de versement)
  - FATAL : la notification de la fin de l'opération n'a pas pu être réalisée suite à une erreur technique (ATR_NOTIFICATION.FATAL=Erreur fatale lors de la notification à l'opérateur de versement)

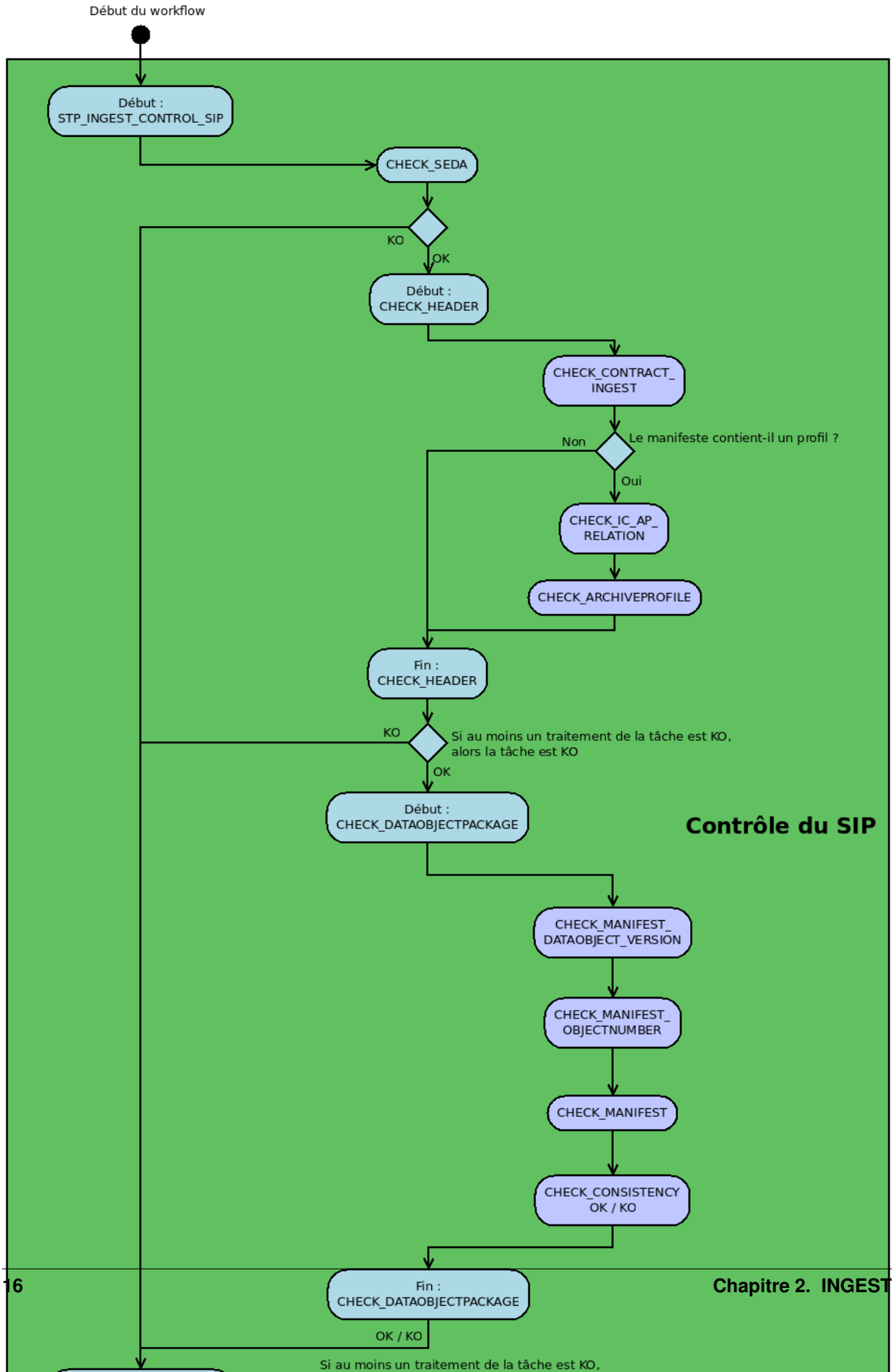
2.1.12.2 Mise en cohérence des journaux du cycle de vie (ROLL_BACK)

- **Règle** : purge des collections temporaires des journaux du cycle de vie.
- **Type** : bloquant.
- **Statuts** :
  - OK : La purge s'est correctement déroulée (ROLL_BACK.OK=Succès de la mise en cohérence des journaux du cycle de vie)
  - FATAL : la purge n'a pas pu être réalisée suite à une erreur technique (ROLL_BACK.FATAL=Erreur fatale lors la mise en cohérence des journaux du cycle de vie)

2.1.13 Structure du Workflow

Le workflow actuel mis en place dans la solution logicielle Vitam est défini dans l'unique fichier "DefaultIngestWorkflow.json". Ce fichier est disponible dans /sources/processing/processing-management/src/main/resources/. Il décrit le processus d'entrée (hors Ingest externe) pour entrer un SIP, indexer les métadonnées et stocker les objets contenus dans le SIP.

D'une façon synthétique, le workflow est décrit de cette façon :


Début du workflow
Début : STP_INGEST_CONTROL_SIP
CHECK_SEDA
KO
OK
Début : CHECK_HEADER
CHECK_CONTRACT_ INGEST
Non
Le manifeste contient-il un profil ?
Oui
CHECK_IC_AP_ RELATION
CHECK_ARCHIVEPROFILE
Fin : CHECK_HEADER
KO
Si au moins un traitement de la tâche est KO, alors la tâche est KO
OK
Début : CHECK_DATAOBJECTPACKAGE
Contrôle du SIP
CHECK_MANIFEST_ DATAOBJECT_VERSION
CHECK_MANIFEST_ OBJECTNUMBER
CHECK_MANIFEST
CHECK_CONSISTENCY OK / KO
Fin : CHECK_DATAOBJECTPACKAGE
OK / KO
Si au moins un traitement de la tâche est KO,

- **Step 1** - STP_INGEST_CONTROL_SIP : Check SIP / distribution sur REF GUID/SIP/manifest.xml
  - CHECK_SEDA (CheckSedaActionHandler.java) :
    - Test de l'existence du manifest.xml
    - Validation XSD du manifeste
    - Validation de la structure du manifeste par rapport au schema par défaut fourni avec le standard SEDA v. 2.0.
    - Test de l'existence d'un fichier unique à la racine du SIP
    - Test de l'existence d'un dossier unique à la racine, nommé "Content" (insensible à la casse)
  - CHECK_HEADER (CheckHeaderActionHandler.java)
    - Test de l'existence du service producteur dans le bordereau
    - Contient CHECK_CONTRACT_INGEST (CheckIngestContractActionHandler.java) :
      - Recherche le nom de contrat d'entrée dans le SIP,
      - Vérification de la validité de contrat par rapport au référentiel de contrats importée dans le système
    - Contient CHECK_IC_AP_RELATION, exécuté si un profil SEDA s'applique pour le SIP (CheckArchiveProfileRelationActionHandler.java) :
      - Vérification que le profil SEDA déclaré dans le contrat d'entrée et celui déclaré dans le SIP est bien le même
    - Contient CHECK_ARCHIVEPROFILE, exécuté si un profil SEDA s'applique pour le SIP (CheckArchiveProfileActionHandler.java) :
      - Vérification de la validité du manifeste par rapport au profil SEDA
  - CHECK_DATAOBJECTPACKAGE (CheckDataObjectPackageActionHandler.java)
    - Contient CHECK_MANIFEST_DATAOBJECT_VERSION (CheckVersionActionHandler.java) :
      - Vérification des usages et numéros de version des objets.
    - Contient CHECK_MANIFEST_OBJECTNUMBER (CheckObjectsNumberActionHandler.java) :
      - Comptage des objets (BinaryDataObject) dans le manifest.xml en s'assurant de l'absence de doublon, que le nombre d'objets reçus est strictement égal au nombre d'objets attendus
      - Création de la liste des objets dans le workspace GUID/SIP/content/,
      - Comparaison du nombre des objets contenus dans le SIP avec ceux définis dans le manifeste.
    - Contient CHECK_MANIFEST (ExtractSedaActionHandler.java) :
      - Extraction des unités archivistiques, des BinaryDataObject, des PhysicalDataObject,
      - Création des journaux du cycle de vie des unités archivistiques et des groupes d'objets,
      - Vérification de la présence de cycles dans les arboresences des Units,
      - Création de l'arbre d'ordre d'indexation,
      - Extraction des métadonnées contenues dans le bloc ManagementMetadata du manifeste pour le calcul des règles de gestion,
      - Vérification du GUID de la structure de rattachement
      - Vérification de la cohérence entre l'unité archivistique rattachée et l'unité archivistique de rattachement.
      - Vérification des problèmes d'encodage dans le manifeste
      - Vérification que les objets ayant un groupe d'objets ne référencent pas directement les unités archivistiques
    - Contient CHECK_CONSISTENCY (CheckObjectUnitConsistencyActionHandler.java) :
      - Extraction des métadonnées des BinaryDataObject et PhysicalDataObject du manifest.xml et création de la MAP (table de concordance) des Id BinaryDataObject ou PhysicalDataObject / Génération GUID (de ces mêmes BinaryDataObject),

        - Extraction des unités archivistiques du manifest.xml et création de la MAP des id d'unités / Génération GUID (de ces mêmes unités archivistiques),
        - Contrôle des références dans les unités archivistiques des Id BinaryDataObject et PhysicalDataObject,
        - Vérification de la cohérence objet/unité archivistique,
        - Stockage dans le Workspace des BinaryDataObject, PhysicalDataObject et des unités archivistiques.
- **Step 2** - STP_OG_CHECK_AND_TRANSFORME : Contrôle et traitements des objets / distribution sur LIST GUID/BinaryDataObject
    - CHECK_DIGEST (CheckConformityActionPlugin.java) :
        - Contrôle de l'objet binaire correspondant : la taille et l'empreinte du BinaryDataObject.
        - Calcul d'une empreinte avec l'algorithme SHA-512 si l'empreinte du manifeste n'a pas été calculée avec cet algorithme
    - OG_OBJECTS_FORMAT_CHECK (FormatIdentificationActionPlugin.java) :
        - Identification du format des BinaryDataObject,
        - Vérification de l'existence du format identifié dans le référentiel des formats
        - Consolidation de l'information du format dans le groupe d'objet correspondant si nécessaire.
- **Step 3** - STP_UNIT_CHECK_AND_PROCESS : Contrôle et traitements des units / distribution sur LIST GUID
    - CHECK_UNIT_SCHEMA (CheckArchiveUnitSchemaActionPlugin.java) :
        - contrôle de validité des champs des unités archivistiques
    - UNITS_RULES_COMPUTE (UnitsRulesComputePlugin.java) :
        - vérification de l'existence de la règle dans le référentiel des règles de gestion
        - calcul des échéances associées à chaque unité archivistique.
- **Step 4** - STP_STORAGE_AVAILABILITY_CHECK : Préparation de la prise en charge / distribution REF GUID/SIP/manifest.xml
    - STORAGE_AVAILABILITY_CHECK (CheckStorageAvailabilityActionHandler.java) :
        - Calcul de la taille totale des objets à stocker,
        - Contrôle de la taille totale des objets à stocker par rapport à la capacité des offres de stockage pour une stratégie et un tenant donnés.
- **Step 5** - STP_OG_STORING : Rangement des objets
    - OG_STORAGE (StoreObjectGroupActionPlugin.java) :
        - Écriture sur l'offre de stockage des objets binaires et des groupes d'objets.
    - OG_METADATA_INDEXATION (IndexObjectGroupActionPlugin.java) :
        - Indexation des métadonnées des groupes d'objets.
    - OG_METADATA_STORAGE (StoreMetaDataObjectGroupActionPlugin.java) :
        - Sauvegarde sur les offres de stockage des métadonnées des groupes d'objets.
    - COMMIT_LIFE_CYCLE_OBJECT_GROUP (CommitLifeCycleObjectGroupActionHandler.java)
        - Sécurisation en base des journaux du cycle de vie des groupes d'objets
- **Step 6** - STP_UNIT_STORING : Rangement des unités archivistique / distribution sur LIST GUID/Units
    - UNIT_METADATA_INDEXATION (IndexUnitActionPlugin.java) :
        - Transformation sous la forme Json des unités archivistiques et intégration du GUID Unit et du GUID des groupes d'objets
    - UNIT_METADATA_STORAGE (StoreMetaDataUnitActionPlugin.java.java) :
        - Sauvegarde sur les offres de stockage des métadonnées des unités archivistiques.

    - COMMIT_LIFE_CYCLE_UNIT (CommitLifeCycleUnitActionHandler.java)
        - Sécurisation en base des journaux du cycle de vie des unités archivistiques
- **Step 7** - STP_ACCESSION_REGISTRATION : Alimentation du registre des fonds
    - ACCESSION_REGISTRATION (AccessionRegisterActionHandler.java) :
        - Création/Mise à jour et enregistrement des collections AccessionRegisterDetail et AccessionRegisterSummary concernant les archives prises en compte, par service producteur.
- **Step 8 et finale** - STP_INGEST_FINALISATION : Finalisation de l'entrée. Cette étape est obligatoire et sera toujours exécutée, en dernière position.
    - ATR_NOTIFICATION (TransferNotificationActionHandler.java) :
        - Génération de l'ArchiveTransferReply.xml (peu importe le statut du processus d'entrée, l'ArchiveTransferReply est obligatoirement généré),
        - Écriture de l'ArchiveTransferReply sur les offres de stockage.
    - ROLL_BACK (RollBackActionHandler.java)
        - Purge des collections temporaires des journaux du cycle de vie.

2.2 Workflow d'entrée d'un plan de classement

2.2.1 Introduction

Cette section décrit le processus (workflow-plan) d'entrée d'un plan de classement dans la solution logicielle Vitam. La structure d'un plan de classement diffère de celle d'un SIP par l'absence d'objet et de vérification par rapport à un profil SEDA. Il s'agit plus simplement d'une arborescence représentée par des unités archivistiques. Ce processus partage donc certaines étapes avec celui du transfert d'un SIP classique, en ignore certaines et rajoute des tâches additionnelles.

Le workflow actuel mis en place dans la solution logicielle Vitam est défini dans le fichier "DefaultFilingSchemeWorkflow.json". Ce fichier est disponible dans : sources/processing/processing-management/src/main/resources/

2.2.2 Processus d'entrée d'un plan de classement (vision métier)

Le processus d'entrée d'un plan est identique au workflow d'entrée d'un SIP. Il débute lors du lancement du téléchargement d'un plan de classement dans la solution logicielle Vitam. Toutes les étapes et actions sont journalisées dans le journal des opérations.

Les étapes et actions associées ci-dessous décrivent le processus d'entrée (clé et description de la clé associée dans le journal des opérations), non encore abordées dans la description de l'entrée d'un SIP.

2.2.2.1 Traitement additionnel dans la tâche CHECK_DATAOBJECTPACKAGE

- Vérification de la non existence d'objets (CHECK_NO_OBJECT)
    - **Règle** : vérification qu'il n'y a pas d'objet numérique dans le manifest.xml du plan
    - **Statuts** :
        - OK : aucun objet numérique n'est présent dans le manifeste (CHECK_DATAOBJECTPACKAGE.CHECK_NO_OBJECT.OK=Succès de la vérification de l'absence d'objet)
        - KO : des objets numériques sont présent dans le manifeste (CHECK_DATAOBJECTPACKAGE.CHECK_NO_OBJECT.KO= de la vérification de l'absence d'objet : objet(s) trouvé(s))

- FATAL : une erreur technique est survenue lors de la vérification de la non existence d'objet numérique (CHECK_DATAOBJECTPACKAGE.CHECK_NO_OBJECT.FATAL=Erreur fatale lors de la vérification de l'absence d'objet)

D'une façon synthétique, le workflow est décrit de cette façon :

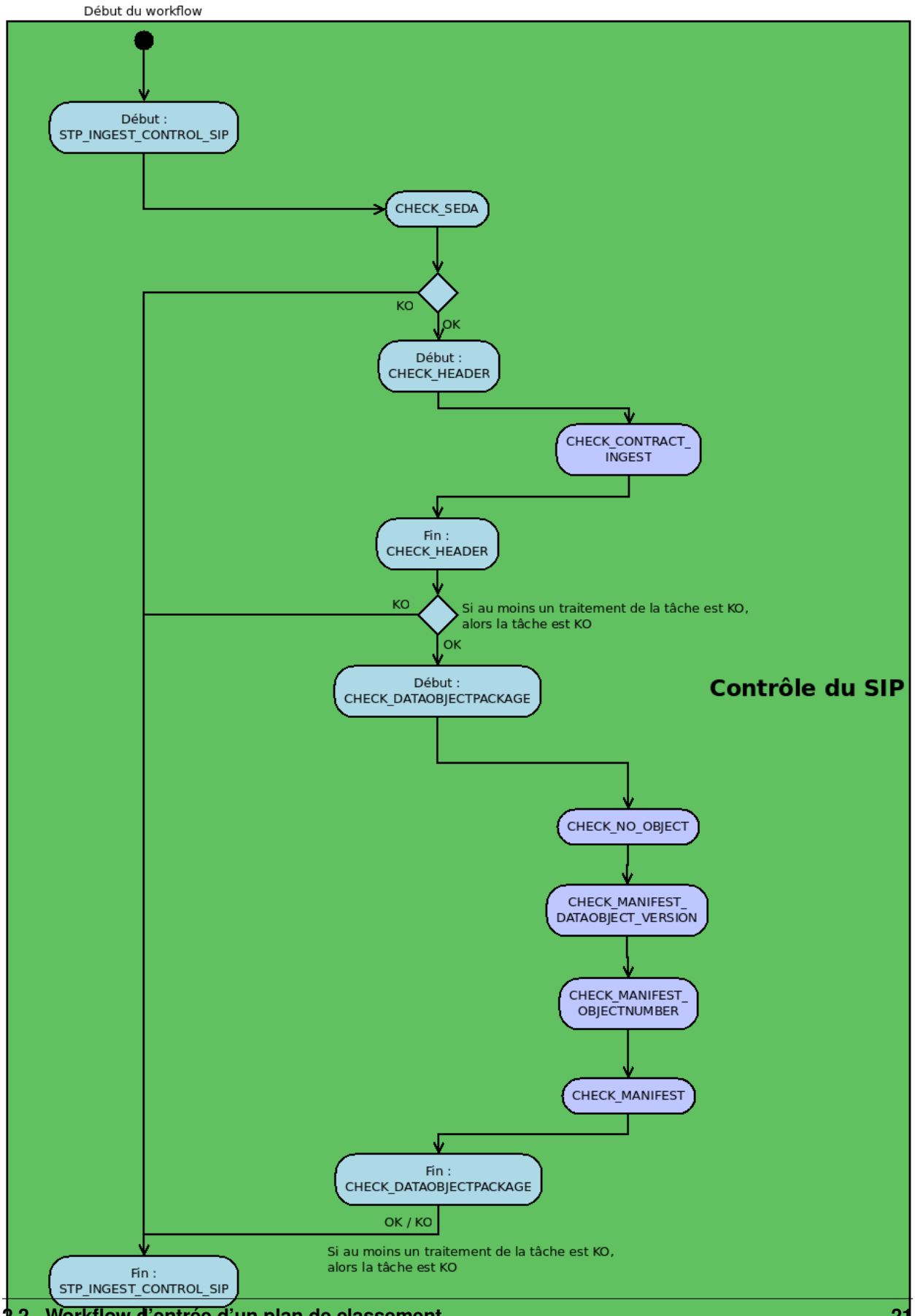

Début du workflow
Début : STP_INGEST_CONTROL_SIP
CHECK_SEDA
KO
OK
Début : CHECK_HEADER
CHECK_CONTRACT_ INGEST
Fin : CHECK_HEADER
KO
Si au moins un traitement de la tâche est KO, alors la tâche est KO
OK
Début : CHECK_DATAOBJECTPACKAGE
Contrôle du SIP
CHECK_NO_OBJECT
CHECK_MANIFEST_ DATAOBJECT_VERSION
CHECK_MANIFEST_ OBJECTNUMBER
CHECK_MANIFEST
Fin : CHECK_DATAOBJECTPACKAGE
OK / KO
Si au moins un traitement de la tâche est KO, alors la tâche est KO
Fin : STP_INGEST_CONTROL_SIP
Si au moins une action de l'étape est KO, alors l'étape est KO

Diagramme d'activité du workflow du plan de classement

- **Step 1** - STP_INGEST_CONTROL_SIP : Check SIP / distribution sur REF GUID/SIP/manifest.xml
  - CHECK_SEDA (CheckSedaActionHandler.java) :
    - Test de l'existence du manifest.xml,
    - Test de l'existence d'un fichier unique à la racine du SIP
    - Test de l'existence d'un dossier unique à la racine, nommé "Content" (insensible à la casse)
    - Validation XSD du manifeste,
    - Validation de la structure du manifeste par rapport au schema par défaut fourni avec le standard SEDA v. 2.0.
  - CHECK_HEADER (CheckHeaderActionHandler.java)
    - Test de l'existence du service producteur dans le bordereau
    - Contient CHECK_CONTRACT_INGEST (CheckIngestContractActionHandler.java) :
      - Recherche le nom de contrat d'entrée dans le SIP,
      - Vérification de la validité de contrat par rapport au référentiel de contrats importée dans le système
  - CHECK_DATAOBJECTPACKAGE (CheckDataObjectPackageActionHandler.java)
    - Contient CHECK_NO_OBJECT
      - Vérification de l'absence d'objets
    - Contient CHECK_MANIFEST_OBJECTNUMBER (CheckObjectsNumberActionHandler.java) :
      - Comptage des objets (BinaryDataObject) dans le manifest.xml en s'assurant de l'absence de doublon, que le nombre d'objets binaires reçus est strictement égal au nombre d'objets attendus
      - Création de la liste des objets binaires dans le workspace GUID/SIP/content/,
      - Comparaison du nombre et des URI des objets binaires contenus dans le SIP avec ceux définis dans le manifeste.
  - Contient CHECK_MANIFEST (ExtractSedaActionHandler.java) :
    - Extraction des unités archivistiques, des BinaryDataObject, des PhysicalDataObject,
    - Création des journaux du cycle de vie des unités archivistiques et des groupes d'objets,
    - Vérification de la présence de cycles dans les arboresences des Units,
    - Création de l'arbre d'ordre d'indexation,
    - Extraction des métadonnées contenues dans le bloc ManagementMetadata du manifeste pour le calcul des règles de gestion,
    - Vérification du GUID de la structure de rattachement
    - Vérification de la cohérence entre l'unité archivistique rattachée et l'unité archivistique de rattachement.
    - Vérification des problèmes d'encodage dans le manifeste
    - Vérification que les objets ayant un groupe d'objets ne référencent pas directement les unités archivistiques
- **Step 2** - STP_UNIT_CHECK_AND_PROCESS : Contrôle et traitements des units / distribution sur LIST GUID
  - UNITS_RULES_COMPUTE (UnitsRulesComputePlugin.java) :
    - vérification de l'existence de la règle dans le référentiel des règles de gestion
    - calcul des échéances associées à chaque unité archivistique.
- **Step 3** - STP_UNIT_STORING : Rangement des unités archivistique / distribution sur LIST GUID/Units
  - UNIT_METADATA_INDEXATION (IndexUnitActionPlugin.java) :
    - Transformation sous la forme Json des unités archivistiques et intégration du GUID Unit et du GUID des groupes d'objets

    - UNIT_METADATA_STORAGE (StoreMetaDataUnitActionPlugin.java.java) :
        - Sauvegarde sur les offres de stockage des métadonnées des unités archivistiques.
    - COMMIT_LIFE_CYCLE_UNIT (CommitLifeCycleUnitActionHandler.java)
        - Sécurisation en base des journaux du cycle de vie des unités archivistiques
- **Step 4** - STP_ACCESSION_REGISTRATION : Alimentation du registre des fonds
    - ACCESSION_REGISTRATION (AccessionRegisterActionHandler.java) :
        - Création/Mise à jour et enregistrement des collections AccessionRegisterDetail et AccessionRegisterSummary concernant les archives prises en compte, par service producteur.
- **Step 5 et finale** - STP_INGEST_FINALISATION : Finalisation de l'entrée. Cette étape est obligatoire et sera toujours exécutée, en dernière position.
    - ATR_NOTIFICATION (TransferNotificationActionHandler.java) :
        - Génération de l'ArchiveTransferReply.xml (peu importe le statut du processus d'entrée, l'ArchiveTransferReply est obligatoirement généré),
        - Écriture de l'ArchiveTransferReply sur les offres de stockage.

CHAPITRE 3

MASTERDATA

3.1 Workflow d'import d'un arbre de positionnement

3.1.1 Introduction

Cette section décrit le processus (workflow-tree) permettant d'importer un arbre de positionnement dans la solution logicielle Vitam. La structure d'un arbre de positionnement diffère de celle d'un SIP en plusieurs points.

Un arbre ne doit pas avoir d'objet, ni de service producteur, ni de contrat. Il s'agit plus simplement d'une arborescence représentée par des unités archivistiques. Ce processus partage donc certaines étapes avec celui du transfert d'un SIP classique, en ignore certaines et rajoute des tâches additionnelles.

Le workflow mis en place dans la solution logicielle Vitam est défini dans le fichier "DefaultHoldingSchemeWorkflow.json". Ce fichier est disponible à sources/processing/processing-management/src/main/resources/

3.1.2 Processus d'import d'un arbre (vision métier)

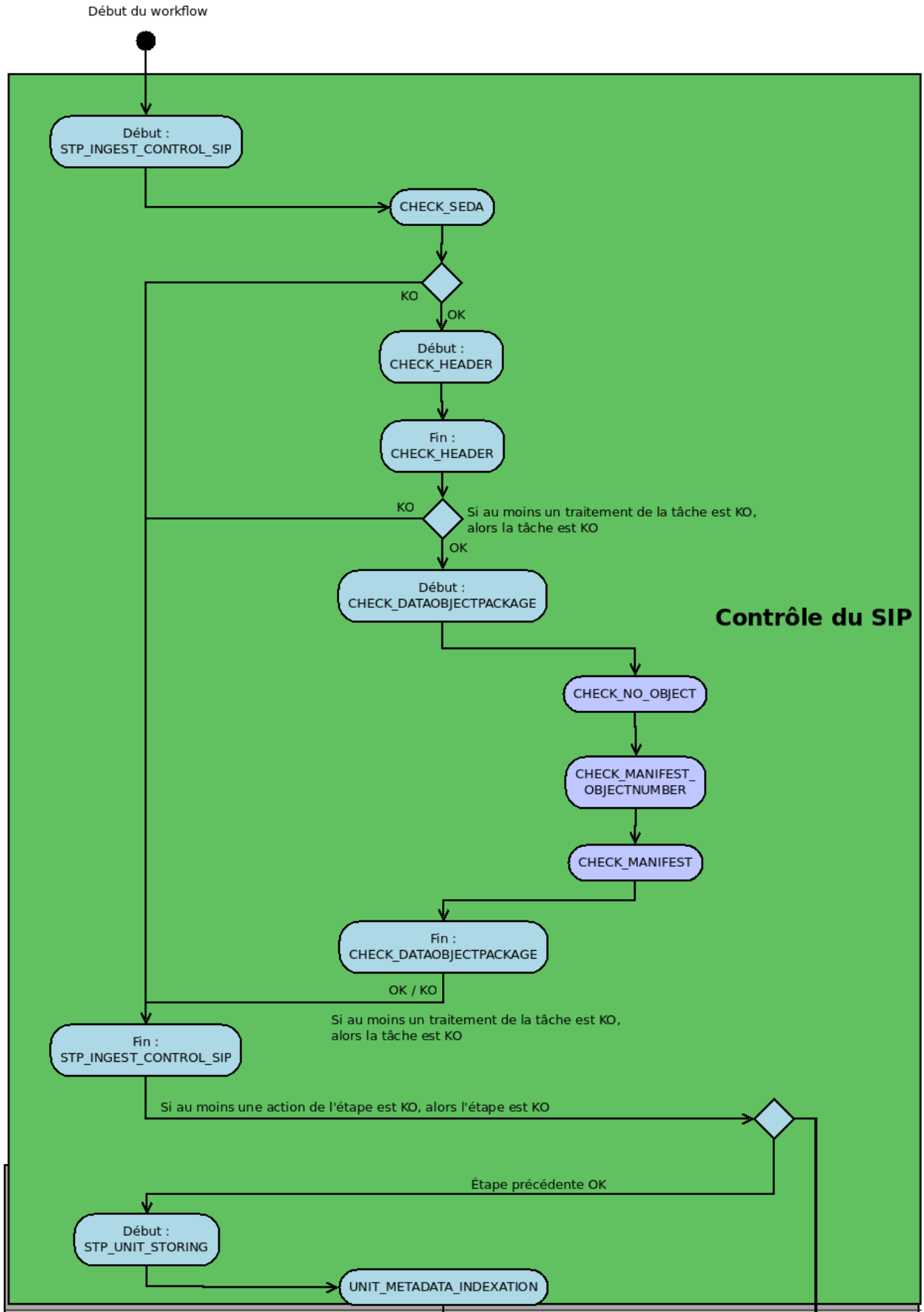
Le processus d'import d'un arbre est identique au workflow d'entrée d'un SIP. Il débute lors du lancement du téléchargement de l'arbre dans la solution logicielle Vitam. De plus, toutes les étapes et actions sont journalisées dans le journal des opérations.

Les étapes et actions associées ci-dessous décrivent le processus d'import (clé et description de la clé associée dans le journal des opérations), non encore abordées dans la description de l'entrée d'un SIP.

3.1.2.1 Traitement additionnel dans la tâche CHECK_DATAOBJECTPACKAGE

- Vérification de la non existence d'objets (CHECK_NO_OBJECT)
  - **Règle** : vérification qu'il n'y a pas d'objet numérique dans le manifest.xml du plan
  - **Statuts** :
    - OK : aucun objet numérique n'est présent dans le manifeste (CHECK_DATAOBJECTPACKAGE.CHECK_NO_OBJECT.OK=Succès de la vérification de l'absence d'objet)
    - KO : des objets numériques sont présent dans le manifeste (CHECK_DATAOBJECTPACKAGE.CHECK_NO_OBJECT.KO=Échec de la vérification de l'absence d''objet : objet(s) trouvé(s))
    - FATAL : une erreur technique est survenue lors de la vérification de la non existence d'objet numérique (CHECK_DATAOBJECTPACKAGE.CHECK_NO_OBJECT.FATAL=Erreur fatale lors de la vérification de l'absence d''objet)

D'une façon synthétique, le workflow est décrit de cette façon :


Début du workflow
Début : STP_INGEST_CONTROL_SIP
CHECK_SEDA
KO
OK
Début : CHECK_HEADER
Fin : CHECK_HEADER
KO
Si au moins un traitement de la tâche est KO, alors la tâche est KO
OK
Début : CHECK_DATAOBJECTPACKAGE
Contrôle du SIP
CHECK_NO_OBJECT
CHECK_MANIFEST_ OBJECTNUMBER
CHECK_MANIFEST
Fin : CHECK_DATAOBJECTPACKAGE
OK / KO
Si au moins un traitement de la tâche est KO, alors la tâche est KO
Fin : STP_INGEST_CONTROL_SIP
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente OK
Début : STP_UNIT_STORING
UNIT_METADATA_INDEXATION
KO
OK

- **Step 1** - STP_INGEST_CONTROL_SIP : Check SIP / distribution sur REF GUID/SIP/manifest.xml
  - CHECK_SEDA (CheckSedaActionHandler.java) :
    - Test de l'existence du manifest.xml,
    - Test de l'existence d'un fichier unique à la racine du SIP
    - Test de l'existence d'un dossier unique à la racine, nommé "Content" (insensible à la casse)
    - Validation XSD du manifeste,
    - Validation de la structure du manifeste par rapport au schema par défaut fourni avec le standard SEDA v. 2.0.
  - CHECK_HEADER (CheckHeaderActionHandler.java)
    - Test de l'existence du service producteur dans le bordereau
  - CHECK_DATAOBJECTPACKAGE (CheckDataObjectPackageActionHandler.java)
    - Contient CHECK_NO_OBJECT (CheckNoObjectsActionHandler.java)
      - Vérification de l'absence d'objets
    - Contient CHECK_MANIFEST_OBJECTNUMBER (CheckObjectsNumberActionHandler.java) :
      - Comptage des objets (BinaryDataObject) dans le manifest.xml en s'assurant de l'absence de doublon, que le nombre d'objets binaires reçus est strictement égal au nombre d'objets attendus
      - Création de la liste des objets binaires dans le workspace GUID/SIP/content/,
      - Comparaison du nombre et des URI des objets binaires contenus dans le SIP avec ceux définis dans le manifeste.
    - Contient CHECK_MANIFEST (ExtractSedaActionHandler.java) :
      - Extraction des unités archivistiques, des BinaryDataObject, des PhysicalDataObject,
      - Création des journaux du cycle de vie des unités archivistiques et des groupes d'objets,
      - Vérification de la présence de cycles dans les arboresences des Units,
      - Création de l'arbre d'ordre d'indexation,
      - Extraction des métadonnées contenues dans le bloc ManagementMetadata du manifeste pour le calcul des règles de gestion,
      - Vérification du GUID de la structure de rattachement
      - Vérification de la cohérence entre l'unité archivistique rattachée et l'unité archivistique de rattachement.
      - Vérification des problèmes d'encodage dans le manifeste
      - Vérification que les objets ayant un groupe d'objets ne référencent pas directement les unités archivistiques
- **Step 2** - STP_UNIT_STORING : Rangement des unités archivistique / distribution sur LIST GUID/Units
  - UNIT_METADATA_INDEXATION (IndexUnitActionPlugin.java) :
    - Transformation sous la forme Json des unités archivistiques et intégration du GUID Unit et du GUID des groupes d'objets
  - UNIT_METADATA_STORAGE (StoreMetaDataUnitActionPlugin.java.java) :
    - Sauvegarde sur les offres de stockage des métadonnées des unités archivistiques.
  - COMMIT_LIFE_CYCLE_UNIT (CommitLifeCycleUnitActionHandler.java)
- **Step 3 et finale** - STP_INGEST_FINALISATION : Finalisation de l'entrée. Cette étape est obligatoire et sera toujours exécutée, en dernière position.
  - ATR_NOTIFICATION (TransferNotificationActionHandler.java) :
    - Génération de l'ArchiveTransferReply.xml (peu importe le statut du processus d'entrée, l'ArchiveTransferReply est obligatoirement généré),
    - Écriture de l'ArchiveTransferReply sur les offres de stockage.

CHAPITRE 4

AUDIT

4.1 Workflow de contrôle d'intégrité d'un journal sécurisé

4.1.1 Introduction

Cette section décrit le processus (workflow) de contrôle d'intégrité d'un journal sécurisé mis en place dans la solution logicielle Vitam.

Celui-ci est défini dans le fichier "DefaultCheckTraceability.json" (situé ici : sources/processing/processing-management/src/main/resources/)

4.1.2 Processus de contrôle d'intégrité d'un journal sécurisé (vision métier)

Le processus de contrôle d'intégrité débute lorsqu'un identifiant d'opération de sécurisation des journaux d'opération est soumis au service de contrôle d'intégrité des journaux sécurisés. Le service permet de récupérer le journal sécurisé, d'en extraire son contenu et de valider que son contenu n'a pas été altéré.

Pour cela, il calcule un arbre de Merkle à partir des journaux d'opérations que contient le journal sécurisé, puis en calcule un second à partir des journaux correspondants disponibles dans la solution logicielle Vitam. Une comparaison est ensuite effectuée entre ces deux arbres et celui contenu dans les métadonnées du journal sécurisé.

Ensuite, dans une dernière étape, le tampon d'horodatage est vérifié et validé.

4.1.3 Préparation du processus de vérification des journaux sécurisés (STP_PREPARE_TRACEABILITY_CHECK)

4.1.3.1 PREPARE_TRACEABILITY_CHECK (PrepareTraceabilityCheckProcessActionHandler.java)

- Règle : vérification que l'opération donnée en entrée est de type TRACEABILITY. Récupération du zip associé à cette opération et extraction de son contenu.
- Type : bloquant
- **Statuts :**
    - OK : l'opération donnée en entrée est une opération de type TRACEABILITY, le zip a été trouvé et son contenu extrait (PREPARE_TRACEABILITY_CHECK.OK=Succès de la préparation du processus de la vérification des journaux sécurisés)
    - KO : l'opération donnée en entrée n'est pas une opération de type TRACEABILITY (PREPARE_TRACEABILITY_CHECK.KO=Échec de la préparation du processus de vérification des journaux sécurisés)

- FATAL : Une erreur technique est survenue lors de la préparation du processus de vérification (PREPARE_TRACEABILITY_CHECK.FATAL=Erreur fatale lors de la préparation du processus de vérification des journaux sécurisés)

4.1.4 Vérification de l'arbre de Merkle (STP_MERKLE_TREE)

4.1.4.1 CHECK_MERKLE_TREE (VerifyMerkleTreeActionHandler.java)

- Règle : recalcul de l'arbre de Merkle des journaux contenus dans le journal sécurisé, calcul d'un autre arbre à partir des journaux indexés correspondants et vérification que tous deux correspondent à celui stocké dans les métadonnées du journal sécurisé
- Type : bloquant
- **Statuts :**
  - OK : les arbres de Merkle correspondent (CHECK_MERKLE_TREE.OK=Succès de la vérification de l'arbre de MERKLE)
  - KO : les arbres de Merkle ne correspondent pas (CHECK_Merkle_TREE.KO=Échec de la vérification de l'arbre de MERKLE)
  - FATAL : erreur technique lors de la vérification des arbres de Merkle (CHECK_MERKLE_TREE.FATAL=Erreur lors de la vérification de l'arbre de MERKLE)

La tâche contient les traitements suivants

- **Comparaison de l'arbre de MERKLE avec le Hash enregistré**
  - Règle : vérification que l'arbre de Merkle calculé à partir des journaux contenus dans le journal sécurisé est identique à celui stocké dans les métadonnées du journal sécurisé
  - Type : bloquant
  - **Statuts :**
    - OK : l'arbre de Merkle des journaux contenus dans le journal sécurisé correspond à celui stocké dans les métadonnées du journal sécurisé (CHECK_MERKLE_TREE.COMPARE_MERKLE_HASH_WITH_SAVED_HASH.OK=Succès de la comparaison de l'arbre de MERKLE avec le Hash enregistré)
    - KO : l'arbre de Merkle des journaux contenus dans le journal sécurisé ne correspond pas à celui stocké dans les métadonnées du journal sécurisé (CHECK_MERKLE_TREE.COMPARE_MERKLE_HASH_WITH_SAVED_HASH.KO=Échec de la comparaison de l'arbre de MERKLE avec le Hash enregistré)
- **Comparaison de l'arbre de MERKLE avec le Hash indexé**
  - Règle : vérification que l'arbre de Merkle calculé à partir des journaux indexés est identique à celui stocké dans les métadonnées du journal sécurisé
  - Type : bloquant
  - Statuts :
  - **OK** [l'arbre de Merkle des journaux indexés correspond à celui stocké dans les métadonnées du journal sécurisé (CHECK_MERKLE_TREE.COMPARE_MERKLE_HASH_WITH_INDEXED_HASH.OK=Succès de la comparaison de l'arbre de MERKLE avec le Hash indexé)]
    - KO : l'arbre de Merkle des journaux indexés ne correspond pas à celui stocké dans les métadonnées du journal sécurisé (CHECK_MERKLE_TREE.COMPARE_MERKLE_HASH_WITH_INDEXED_HASH.KO=Échec de la comparaison de l'arbre de MERKLE avec le Hash indexé)

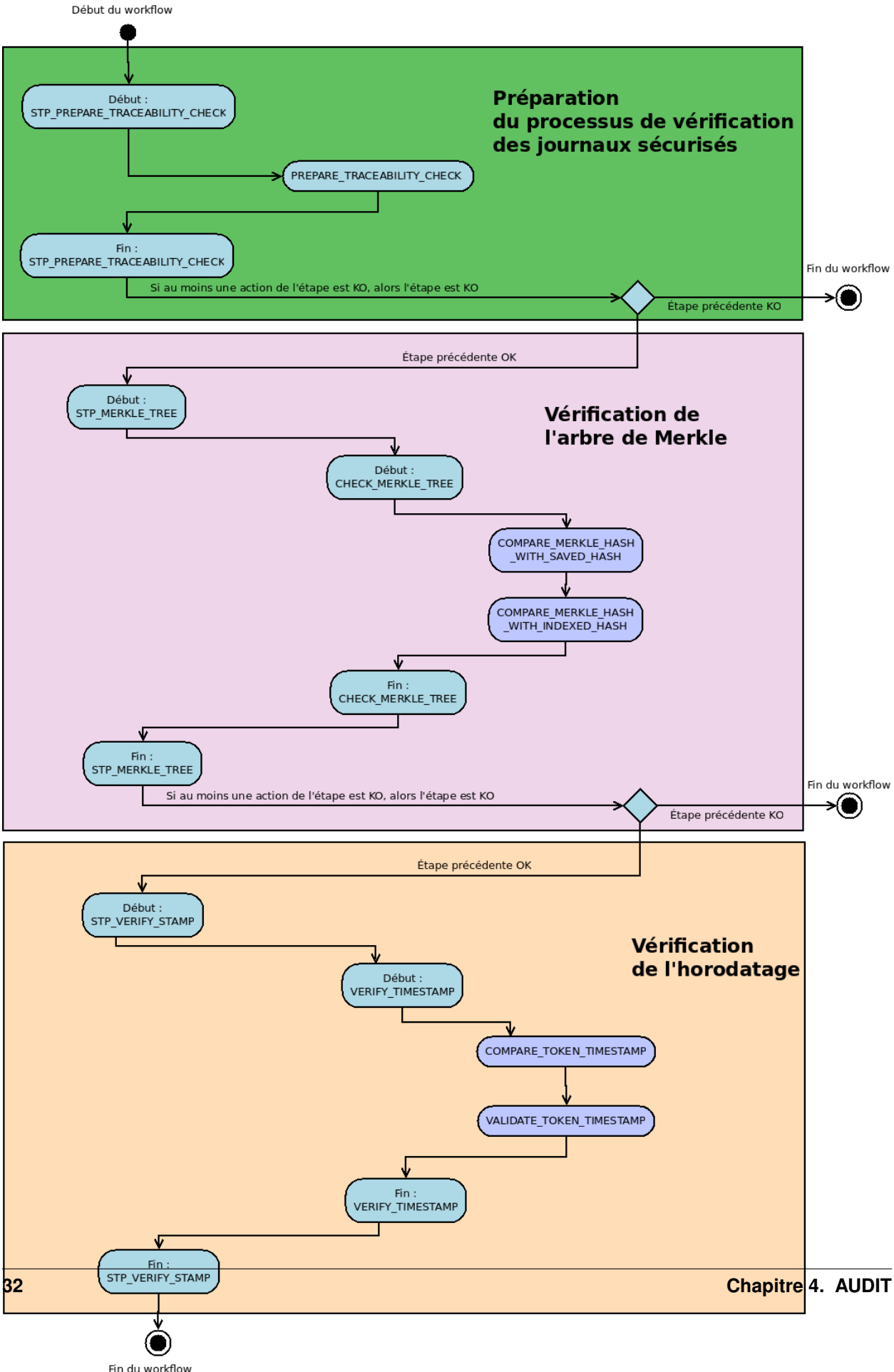
4.1.5 Vérification de l'horodatage (STP_VERIFY_STAMP)

4.1.5.1 VERIFY_TIMESTAMP (VerifyTimeStampActionHandler.java)

- Règle : vérification et validation du tampon d'horodatage.
- Type : bloquant
- **Statuts :**
    - OK : le tampon d'horadatage est correct (VERIFY_TIMESTAMP.OK=Succès de la vérification de l'horodatage)
    - KO : le tampon d'horadatage est incorrect (VERIFY_TIMESTAMP.KO=Échec de la vérification de l'horodatage)
    - FATAL : erreur technique lors de la vérification du tampon d'horodatage (VERIFY_TIMESTAMP.FATAL=Erreur lors de la vérification de l'horodatage)

La tâche contient les traitements suivants

- **Comparaison du tampon du fichier (token.tsp) par rapport au tampon enregistré dans le logbook (COMPARE_TOKEN_**
    - Règle : vérification que le tampon enregistré dans la collection logbookOperation est le même que celui présent dans le fichier zip généré
    - Type : bloquant
    - **Status :**
        - OK : les tampons sont identiques (VERIFY_TIMESTAMP.COMPARE_TOKEN_TIMESTAMP.OK=Succès de la comparaison des tampons d'horodatage)
        - KO : les tampons sont différents (VERIFY_TIMESTAMP.COMPARE_TOKEN_TIMESTAMP.KO=Échec de la comparaison des tampons d'horodatage)
- **Validation du tampon d'horodatage (VALIDATE_TOKEN_TIMESTAMP)**
    - Règle : vérification cryptographique du tampon et vérification de la chaîne de certification
    - Type : bloquant
    - **Status :**
        - OK : le tampon est validé (VERIFY_TIMESTAMP.VALIDATE_TOKEN_TIMESTAMP.OK=Succès de la validation du tampon d'horodatage)
        - KO : le tampon est invalidé (VERIFY_TIMESTAMP.VALIDATE_TOKEN_TIMESTAMP.KO=Échec de la validation du tampon d'horodatage)


Début du workflow
Début :
STP_PREPARE_TRACEABILITY_CHECK
Préparation
du processus de vérification
des journaux sécurisés
PREPARE_TRACEABILITY_CHECK
Fin :
STP_PREPARE_TRACEABILITY_CHECK
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Début :
STP_MERKLE_TREE
Vérification de
l'arbre de Merkle
Début :
CHECK_MERKLE_TREE
COMPARE_MERKLE_HASH
_WITH_SAVED_HASH
COMPARE_MERKLE_HASH
_WITH_INDEXED_HASH
Fin :
CHECK_MERKLE_TREE
Fin :
STP_MERKLE_TREE
Si au moins une action de l'étape est KO, alors l'étape est KO
Étape précédente KO
Fin du workflow
Étape précédente OK
Début :
STP_VERIFY_STAMP
Vérification
de l'horodatage
Début :
VERIFY_TIMESTAMP
COMPARE_TOKEN_TIMESTAMP
VALIDATE_TOKEN_TIMESTAMP
Fin :
VERIFY_TIMESTAMP
Fin :
STP_VERIFY_STAMP
Fin du workflow

CHAPITRE 5

TRACEABILITY

5.1 Workflow de création d'un journal sécurisé

5.1.1 Introduction

Cette section décrit le processus (workflow) de sécurisation des journaux mis en place dans la solution logicielle Vitam.

Celui-ci est défini dans le fichier "LogbookAdministration.java" (situé ici : sources/logbook/logbook-administration/src/main/java/fr/gouv/vitam/logbook/administration/core/)

5.1.2 Processus de sécurisation des journaux (vision métier)

Le processus de sécurisation des journaux consiste en la création d'un fichier .zip contenant l'ensemble des journaux d'opérations à sécuriser, ainsi que le tampon d'horodatage calculé à partir de l'arbre de Merkle de la liste de ces mêmes journaux.

Ce fichier zip est ensuite enregistré sur les offres de stockage, en fonction de la stratégie de stockage.

5.1.3 Sécurisation des journaux (STP_OP_SECURISATION)

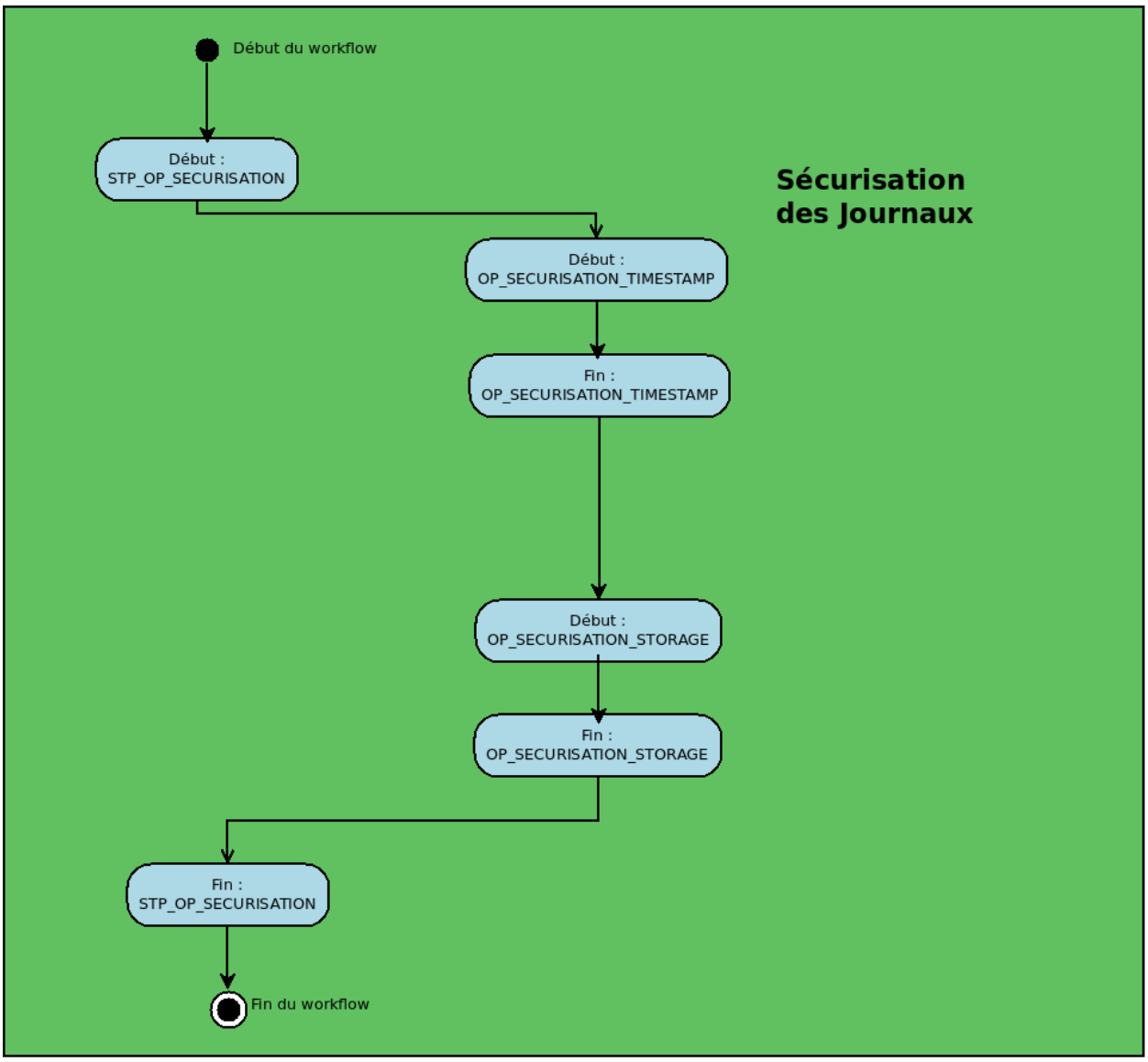
5.1.3.1 OP_SECURISATION_TIMESTAMP (LogbookAdministration.java)

- Règle : calcul du tampon d'horodatage à partir de la racine de l'arbre de merkle consitué de la liste des journaux qui sont en train d'être sécurisés.
- Type : bloquant
- **Status :**
    - OK : le tampon d'horodatage est calculé (OP_SECURISATION_TIMESTAMP.OK=Succès de création du tampon d'horodatage de l'ensemble des journaux)
    - FATAL : l'horodatage n'a pas été calculé suite à une erreur technique (OP_SECURISATION_TIMESTAMP.FATAL=Erreur fatale lors de création du tampon d'horodatage de l'ensemble des journaux)

5.1.3.2 OP_SECURISATION_STORAGE (LogbookAdministration.java)

- Règle : écriture des journaux sécurisés sur les offres de stockage, en fonction de la stratégie de stockage.
- Type : bloquant

- **Status :**
  - OK : le journal sécurisé est écrit sur les offres de stockage (OP_SECURISATION_STORAGE.OK=Succès du stockage des journaux)
  - FATAL : l'écriture du journal sécurisé n'a pas été réalisé suite à une erreur technique (OP_SECURISATION_STORAGE.FATAL=Erreur fatale du stockage des journaux)



CHAPITRE 6

Annexes

Table des figures

Liste des tableaux