MATISSE 3D

Implémentation d'un connecteur FTP

Document	Note technique – Implémentation d'un connecteur FTP / Matisse 3D		
Client	IFREMER		
Référence	M21009-NT01	**Version**	1.1
Date	27/06/2022		
Auteur	B. GADAIX		
Confidentialité	N/A		
Approbation			
Nom			
Qualité			
Date			
Visa			

innosense

Historique des modifications

Date	Version	Motif
23/06/2022	1.0	Version finale du document
27/06/2022	1.1	Précisions sur la gestion de la connexion / déconnexion Précisions sur l'implémentation des méthodes et slots Précisions sur les signaux remontés Précisions sur la remontée d'informations de progression Précisions sur la normalisation des erreurs réseau

SOMMAIRE

1 Introduction

1.1 Contexte

Le logiciel Matisse 3D permet le traitement en temps différé des images prises par les sous-marins d'IFREMER en plongée, pour réaliser une reconstruction 2D ou 3D d'un site d'intérêt.

Matisse 3D est un logiciel client destiné à un poste de travail Windows ou Linux. Selon la volumétrie des jeux de données images utilisés en entrée, les algorithmes de traitement nécessitent des ressources de calcul importantes pour lesquels les postes de travail des utilisateurs ne sont pas dimensionnés.

Une interface de calcul déporté a été réalisée dans le cadre du projet Matisse Datarmor en plusieurs incréments entre 2020 et 2022, en utilisant des librairies Qt open source pour implémenter les protocoles SSH, FTP et SFTP. L'intégration du module de calcul déporté dans l'environnement cible (réseau IFREMER et supercalculateur Datarmor) a fait apparaître certaines incompatibilités protocolaires entre les librairies utilisées et les serveurs Datarmor.

Il est donc nécessaire de pouvoir implémenter rapidement un nouveau connecteur réseau utilisant une librairie compatible avec l'environnement cible.

1.2 Objet du document

La note technique présente l'architecture de la librairie NetworkTools de Matisse3D et décrit la procédure pour rajouter / remplacer un connecteur réseau.

Dans le contexte de la phase d'intégration du projet Matisse Datarmor, la procédure est décrite pour le connecteur FTP.

1.3 Documents de référence

N/A

2 Architecture de NetworkTools

2.1 Interface réseau

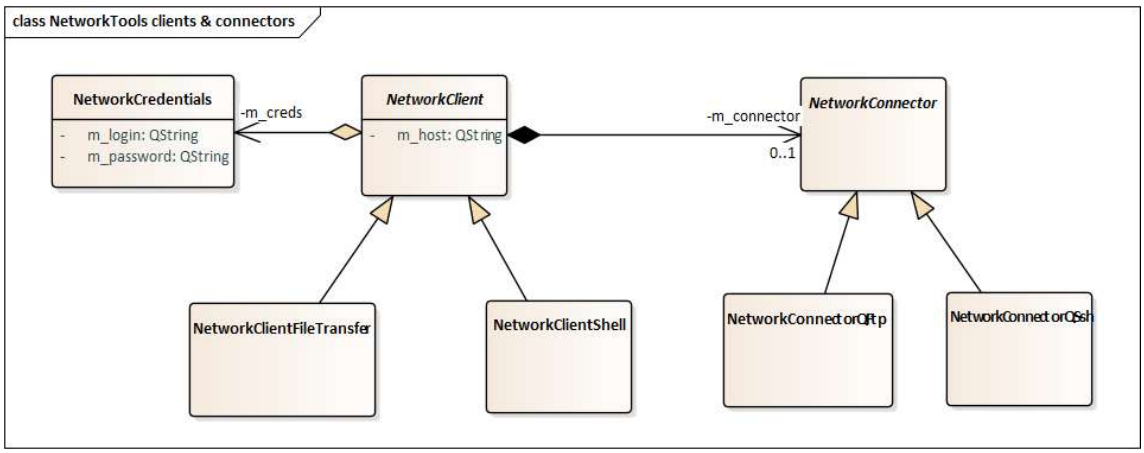
Une interface réseau est contrôlée par un objet de type NetworkClient, avec deux implémentations concrètes pour le transfert de fichier (NetworkClientFileTransfer) et l'interface shell (NetworkClientShell).

Un client (NetworkClient) gère un seul protocole dans tout son cycle de vie. Par exemple, deux objets de type NetworkClientFileTransfer sont nécessaires pour gérer le FTP et le SFTP.

Un client est associé à un serveur hôte et à des identifiants (login, mot de passe), qui peuvent être modifiés au cours de son cycle de vie.

Les classes NetworkClient gèrent le contrôle générique de l'interface réseau avec les spécificités fonctionnelles du type d'interface (transfert de fichier ou shell), mais elles ne gèrent pas dans les spécificités du protocole qui sont dépendantes d'une librairie tierce.

La classe NetworkConnector sert de base à l'implémentation d'un connecteur qui encapsule la librairie tierce utilisée. L'implémentation comprend tout le workflow de contrôle spécifique à une librairie pour la mise en œuvre d'un ou plusieurs protocoles.



Deux implémentations de NetworkConnector sont livrées avec la librairie NetworkTools :

- NetworkConnectorQSsh qui encapsule la librairie QSsh et gère les protocoles SFTP et SSH
- NetworConnectorQFtp qui encapsule la librairie QFtp et gère le protocole FTP

Dans la même logique, une instance de connecteur est nécessaire pour chaque protocole. Il faut donc deux instances de NetworkConnectorQSsh, rattachée chacune à une instance de NetworkClient (NetworkClientFileTransfer et NetworkClientShell respectivement) pour gérer les protocoles SFTP et SSH.

2.2 Contrôle des actions réseau

L'interface réseau (NetworkClient) gère une file d'attente d'actions réseau (NetworkAction). Une action réseau permet de déclencher une séquence d'échanges réseau constituant une opération logique unitaire. Les actions sont créées et ajoutées à la file d'attente par l'utilisateur de la librairie NetworkClient.

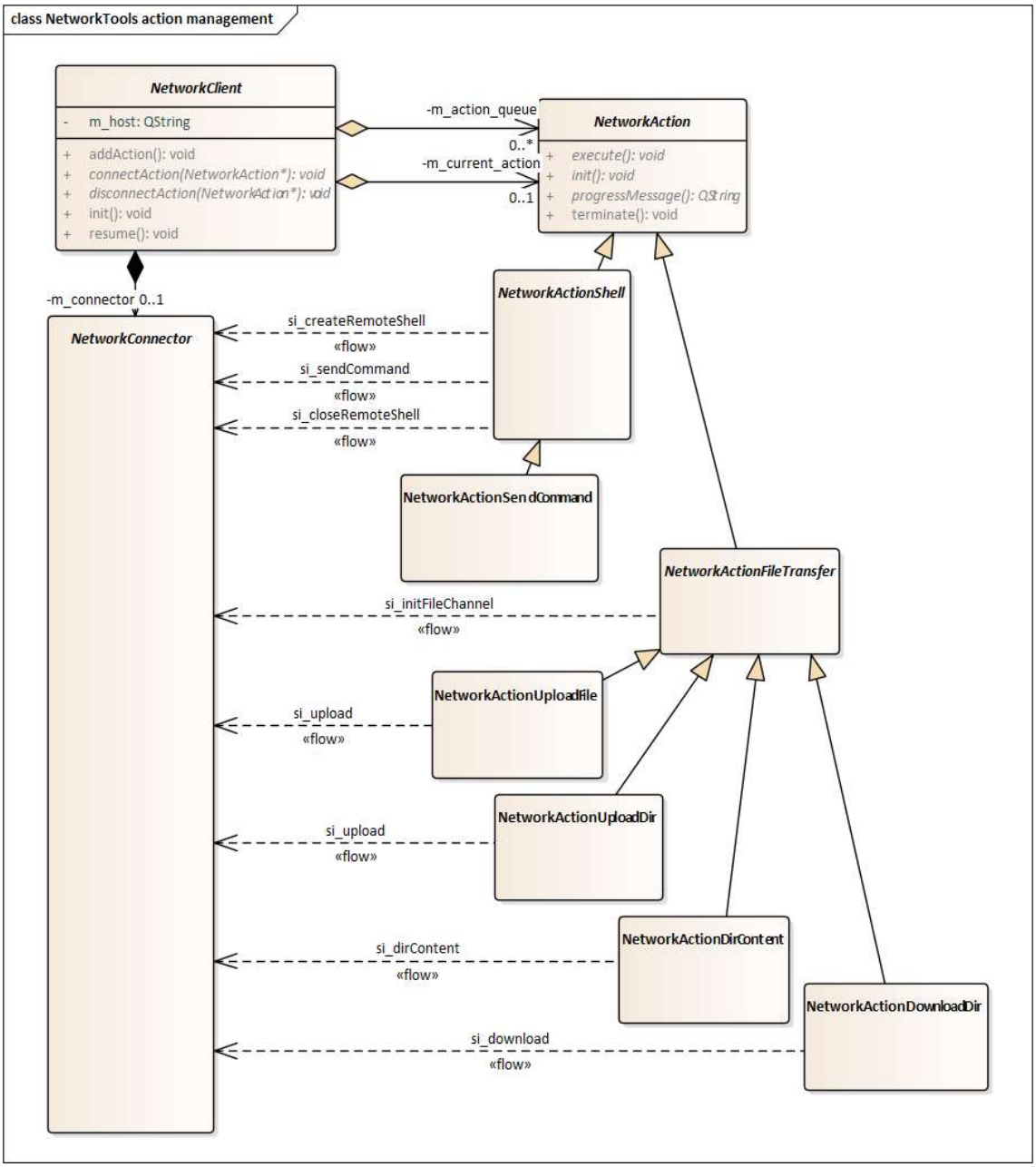
Deux types d'action (NetworkActionFileTransfer et NetworkActionShell) regroupent les actions destinées aux clients NetworkClientFileTransfer et NetworkClientShell respectivement :

- Les actions de transfert de fichier comprennent :
  - Le téléversement d'un fichier (NetworkActionUploadFile),
  - Le téléversement d'un dossier (NetworkActionUploadDir),
  - Le téléchargement d'un dossier (NetworkActionDownloadDir),
  - Le listing d'un dossier (NetworkActionDirContent).
- La seule action destinée au client shell est :
  - L'exécution d'une commande shell (NetworkActionSendCommand).

Chaque client exécute séquentiellement les actions présentes dans sa file d'attente. Toutes les actions suivent le même cycle de vie : 1- init ; 2- execute ; 3- terminate, ce qui permet une gestion polymorphique de la file d'attente. Les méthodes du cycle de vie provoquent la remontée de signaux vers le connecteur réseau (NetworkConnector), déclenchant l'initialisation, l'opération réseau et la finalisation spécifiques de l'action et selon le protocole associé.

Une action est connectée par le client au connecteur pour l'envoi de signaux (méthode NetworkClient::connectAction) lorsqu'elle est extraite de la file d'attente et déconnectée du connecteur (NetworkClient::disconnectAction) lorsque son cycle de vie est terminé.

Le diagramme de classe suivant présente la hiérarchie d'actions et les différents signaux remontés par chaque action au connecteur correspondant.


class NetworkTools action management
NetworkClient
- m_host: QString
+ addAction(): void
+ connectAction(NetworkAction*): void
+ disconnectAction(NetworkAction*): void
+ init(): void
+ resume(): void
-m_action_queue
0..*
-m_current_action
0..1
NetworkAction
+ execute(): void
+ init(): void
+ progressMessage(): QString
+ terminate(): void
-m_connector 0..1
NetworkConnector
NetworkActionShell
si_createRemoteShell
«flow»
si_sendCommand
«flow»
si_closeRemoteShell
«flow»
NetworkActionSendCommand
NetworkActionFileTransfer
si_initFileChannel
«flow»
NetworkActionUploadFile
si_upload
«flow»
NetworkActionUploadDir
si_upload
«flow»
NetworkActionDirContent
si_dirContent
«flow»
NetworkActionDownloadDir
si_download
«flow»

2.3 Gestion de la connexion au serveur distant

C'est le client réseau qui pilote la connexion / déconnexion au serveur distant et c'est le connecteur réseau qui implémente cette connexion / déconnexion.

Le client réseau déclenche l'ouverture d'une connexion lorsqu'il dépile la première action réseau de la file d'attente et déclenche la fermeture lorsque toutes les actions de la file d'attente ont été traitées.

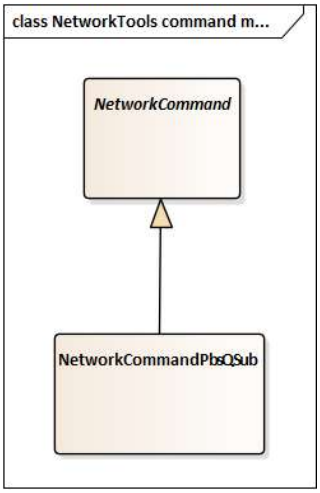
Lorsqu'une erreur est remontée par le connecteur, l'action réseau en cours ainsi que toutes les suivantes présentes dans la file d'attente sont annulées et le client réseau déclenche la fermeture de la connexion.

2.4 Gestion des commandes shell

Un gestionnaire de commande shell (NetworkCommand) permet de générer la chaîne de caractères de la commande une fois paramétrée puis de traiter et d'interpréter la réponse.

Le gestionnaire de commande est décorrélé de la logique d'action réseau, c'est l'utilisateur de la librairie qui doit faire le lien entre la commande et l'action d'envoi de commande.

La librairie est livrée avec une implémentation pour la commande PBS 'qsub' (NetworkCommandPbsQsub).



2.5 Dynamique d'invocation

La dynamique d'invocation est représentée sur le diagramme de séquence suivant avec l'exemple de la fonctionnalité d'upload de dataset, réalisée en SFTP avec la librairie QSsh.

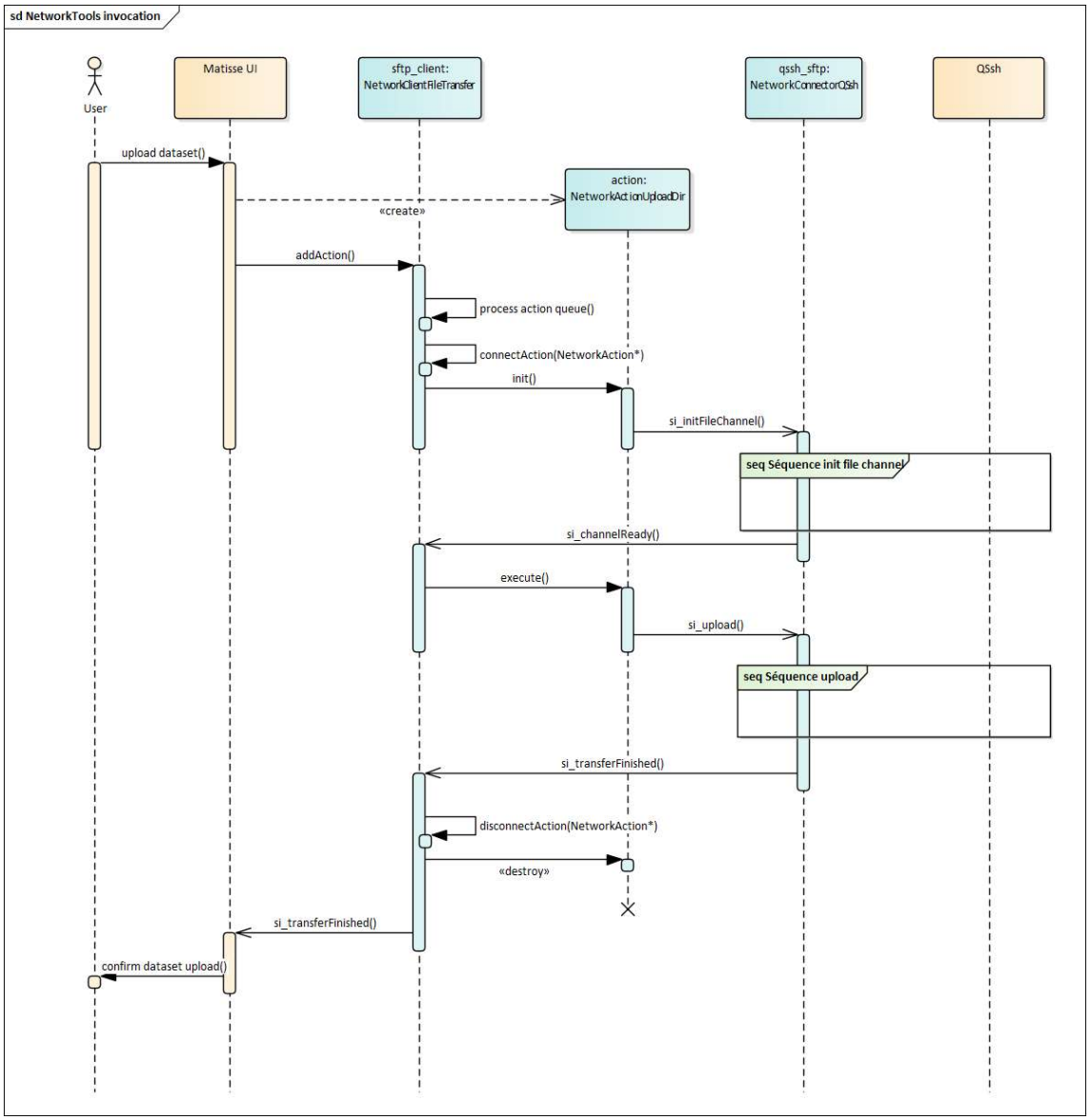
Le bloc « Séquence upload » représente l'ensemble de la séquence d'interaction entre le connecteur et la librairie tierce pour gérer le transfert de fichiers. Cette séquence est spécifique à la librairie tierce utilisée et sa complexité dépendra des possibilités offertes par la librairie tierce :

Dans le cas de la librairie QSsh, cette séquence est relativement simple car la librairie gère la récursivité.

Dans le cas de la librairie QFtp, cette séquence est beaucoup plus complexe car la librairie ne gère pas la récursivité et c'est le connecteur qui balaye l'arborescence de fichiers pour créer les dossiers distants et contrôler l'upload des fichiers.

De la même façon, le bloc « Séquence init file channel » est spécifique à la librairie utilisée.

Tout le reste de la séquence d'interaction est générique pour la fonctionnalité d'upload de dataset et sera le même quel que soit le connecteur utilisé.



3 Utilisation de la librairie dans Matisse 3D

3.1 Contrôle des fonctionnalités de déport de calcul

Les fonctionnalités de déport de calcul sont contrôlées par la couche d'IHM. Elles sont regroupées dans une classe déléguée de la classe d'IHM principale appelée RemoteJobHelper (MatisseGui/RemoteJobUi).

La classe gère par délégation les interactions utilisateur, invoque les méthodes des clients réseau et traite les signaux remontés.

3.2 Serveurs et authentification

Matisse 3D permet d'utiliser deux serveurs différents pour le transfert de fichier et le shell distant. Conformément à l'environnement IFREMER, les mêmes identifiants de connexion sont utilisés pour les deux serveurs.

Le nom d'utilisateur est renseigné dans les préférences. Le mot de passe est saisi à la première action réseau de la session. C'est la classe RemoteJobHelper qui contrôle la saisie du mot de passe.

3.3 Préférences de protocole et clients réseau

Matisse 3D permet le transfert de fichier par FTP ou SFTP. Le choix est effectué par l'utilisateur dans les préférences. Pour permettre le basculement de l'un à l'autre au cours d'une même session en changeant de préférence, une instance de client réseau par protocole est nécessaire.

La classe RemoteJobHelper s'appuie donc sur 3 instances de NetworkClient :

- Une instance pour le transfert de fichier par FTP
- Une instance pour le transfert de fichier par SFTP
- Une instance pour l'envoi de commande shell par SSH

3.4 Cycle de vie des clients réseau

Le cycle de vie des clients réseau correspond à celui de l'application.

Les clients réseau sont réinitialisés si l'authentification est invalidée (changement de username ou de protocole dans les préférences).

3.5 Injection des clients réseau

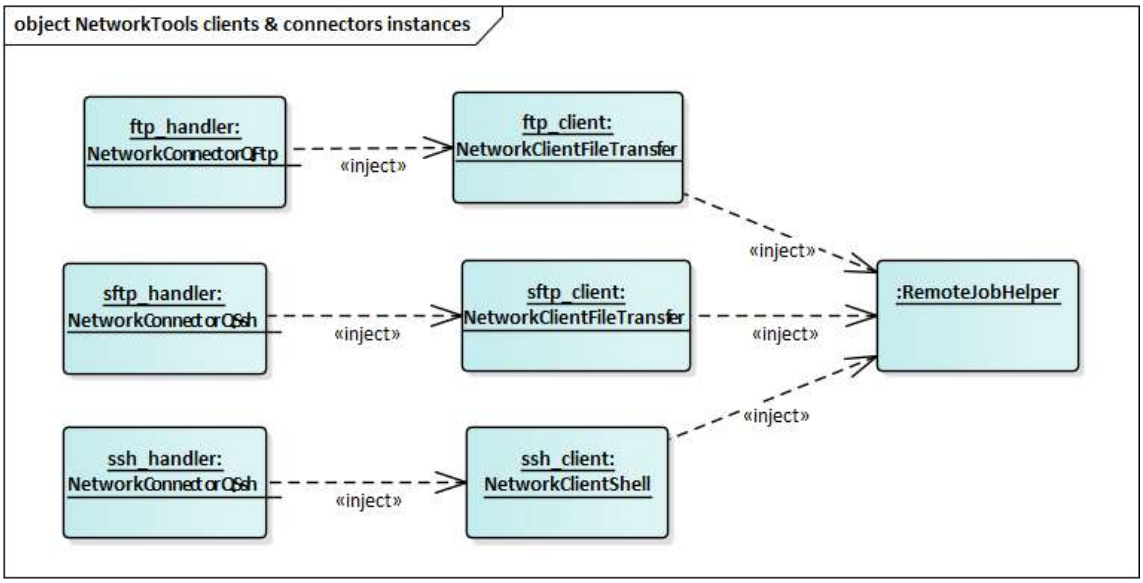
Les clients réseau ne sont pas instanciés directement par RemoteJobHelper, ils sont instanciés au démarrage de l'application dans le main et injectés dans la classe RemoteJobHelper selon le principe d'inversion de contrôle qui facilite leur substitution.

Pour permettre la gestion multi-protocolaire et d'éventuelles évolutions ulterieures, les clients réseau sont indexés par protocole au moment de l'injection (clés d'indexation de type eFileTransferProtocol et eShellProtocol).

3.6 Injection des connecteurs

Selon le même principe, les connecteurs sont instanciés dans le main et chaque connecteur est injecté au client réseau dont il implémente le protocole.

Le diagramme suivant montre le câblage des clients et des connecteurs réseau instanciés par Matisse 3D.



Le code d'instanciation et d'injection extrait du main :

```
NetworkConnector* ssh_handler = new NetworkConnectorQSsh();
NetworkClient* ssh_client = new NetworkClientShell();
ssh_client->setConnector(ssh_handler);

NetworkConnector* ftp_handler = new NetworkConnectorQFtp();
NetworkClient* ftp_client = new NetworkClientFileTransfer();
ftp_client->setConnector(ftp_handler);

NetworkConnector* sftp_handler = new NetworkConnectorQSsh();
NetworkClient* sftp_client = new NetworkClientFileTransfer();
sftp_client->setConnector(sftp_handler);

RemoteJobHelper remote_job_helper;
remote_job_helper.registerNetworkFileClient(eFileTransferProtocol::FTP, ftp_client);
remote_job_helper.registerNetworkFileClient(eFileTransferProtocol::SFTP, sftp_client);
remote_job_helper.registerNetworkShellClient(eShellProtocol::SSH, ssh_client);
```

4 Changement d'implémentation de protocole

4.1 Logique d'implémentation

Pour changer l'implémentation d'un protocole avec une nouvelle librairie tierce, en l'occurrence FTP avec la librairie FtpClient-Cpp, il est recommandé de ne pas modifier les classes de connecteur existantes mais de suivre la procédure suivante :

1. Implémenter une nouvelle classe de connecteur réseau.
2. Remplacer le code d'instanciation et d'injection dans le main, uniquement pour le client réseau FTP (ftp_client).

Il n'est pas recommandé de remplacer l'implémentation de SFTP qui s'appuie sur la librairie QSsh a priori plus robuste (multithreadée et gérant la récursivité).

Aucune autre modification de code n'est nécessaire a priori, les autres classes de la librairie NetworkTools étant conçues comme génériques.

4.2 Implémentation d'un nouveau connecteur réseau

L'implémentation d'un nouveau connecteur est réalisée dans une nouvelle classe fille de la classe NetworkConnector, en suivant les étapes :

1. Implémenter les méthodes virtuelles pures qui spécialisent la gestion de la connexion au serveur distant.
2. Implémenter les méthodes et slots virtuels purs qui spécialisent le traitement des opérations réseau
3. Implémenter la remontée d'information de progression
4. Implémenter le mapping des codes erreurs

Les méthodes et slots à spécialiser implémentent le déclenchement d'un workflow technique, asynchrone dans la très grande majorité des cas et dont l'aboutissement est notifié par un signal remonté au client réseau.

Les erreurs sont également remontées par un signal dédié.

Le workflow peut être très différent selon la librairie tierce utilisée.

4.2.1 Spécialiser la gestion de la connexion au serveur distant

Implémenter obligatoirement les 5 méthodes virtuelles pures héritées de NetworkConnector :

```
public:
    virtual void resetConnection() = 0;
protected:
    virtual void disableConnection() = 0;
    virtual void freeConnection() = 0;
    virtual void connectToRemoteHost() = 0;
    virtual void disconnectFromHost() = 0;
```

4.2.1.1 connectToRemoteHost

La méthode doit déclencher le processus de connexion au serveur distant :

- Ouverture d'une session réseau
- Appel du serveur distant
- Authentification

Dans tous les cas, le processus est asynchrone et le signal si_connected doit être remonté au client réseau lorsque la connexion est établie et authentifiée.

En cas d'erreur de connexion, elle remonte le signal si_connectionFailed avec le code erreur correspondant.

4.2.1.2 disconnectFromHost

La méthode doit déclencher le processus de déconnexion du serveur distant :

- Interruption de toute opération en cours
- Fermeture de la connexion

Le processus peut être synchrone ou asynchrone selon que la librairie tierce contrôle la réponse du serveur distant.

Dans tous les cas, le processus doit remonter le signal si_clearConnection lorsque la déconnexion est effective.

4.2.1.3 disableConnection

Le processus de clôture d'une session d'échanges réseau est piloté par le client réseau dans la méthode clearConnectionAndActionQueue(). Ici, la sémantique de « connexion » n'est pas identique à la connexion établie avec un serveur qui a ce stade est déjà fermée ou en cours de fermeture. Elle correspond dans ce contexte au manager qui gère le traitement des opérations réseau encapsulé dans le connecteur et qu'il est nécessaire de nettoyer en fin de séquence.

La clôture d'une session d'échanges réseau prévoit plusieurs étapes dont 2 sont spécialisées par le connecteur.

La première étape spécialisée est l'inactivation de la connexion qui consiste à débrancher les slots susceptibles de déclencher une séquence non voulue.

4.2.1.4 freeConnection

La deuxième étape spécialisée est la libération des ressources associées au manager traitant les opérations réseau.

4.2.1.5 resetConnection

Cette méthode publique permet à l'utilisateur de la librairie d'interrompre à tout moment la session d'échanges réseau.

Cette méthode permet un déclenchement externe du processus de déconnexion et de nettoyage.

Le comportement attendu est une remontée synchrone du signal si_clearConnection au client réseau pour le rendre immédiatement disponible. Dans le cas où le processus de déconnexion est asynchrone, le connecteur gère l'obsolescence du manager de traitement et la désallocation propre des ressources une fois déconnecté.

4.2.2 Implémenter le traitement des opérations réseau

Implémenter obligatoirement les 7 slots virtuels purs hérités de NetworkConnector :

```
protected slots:
    virtual void sl_initFileChannel();
    virtual void sl_upload(QString _local_path, QString _remote_path, bool _is_dir_upload, bool
_recurse);
    virtual void sl_download(QString _remote_path, QString _local_path, bool
_is_dir_download);
    virtual void sl_dirContent(QString _remote_dir_path, FileTypeFilters _flags, QStringList
_file_filters, bool _is_for_dir_transfer=false);

    virtual void sl_createRemoteShell(QString& _command);
    virtual void sl_closeRemoteShell();
    virtual void sl_executeShellCommand();
```

Comme décrit précédemment, le connecteur peut implémenter soit des fonctions de transfert de fichier, soit des fonctions shell, soit les 2.

En cas d'implémentation de fonctions shell, les 2 méthodes virtuelles pures suivantes doivent être implémentées :

```
protected:
    virtual QByteArray readShellStandardOutput() = 0;
    virtual QByteArray readShellStandardError() = 0;
```

Les slots et méthodes qui ne sont pas pertinents selon les fonctions implémentées doivent être bouchonnés en loguant un message d'erreur.

Dans le cas de la demande présente où il s'agit d'implémenter le transfert de fichier par FTP, les 3 derniers slots et les 2 méthodes doivent être bouchonnés (cf. NetworkConnectorQFtp).

4.2.2.1 sl_initFileChannel

Ce slot déclenche le processus d'ouverture d'un canal de transfert de fichier.

Le processus peut être synchrone ou asynchrone selon la librairie tierce utilisée, voire inexistant (cf. QFtp).

Dans tous les cas, le workflow de traitement doit remonter le signal si_channelReady lorsque le manager est prêt à transmettre des fichiers.

4.2.2.2 sl_upload

Ce slot déclenche le processus d'upload d'un fichier ou d'upload d'un répertoire, de manière récursive ou non selon les paramètres spécifiés par le signal.

Le workflow est nécessairement asynchrone.

Il doit émettre le signal si_transferFinished une fois l'opération de transfert terminée.

4.2.2.3 si_download

Ce slot déclenche le processus de download d'un fichier ou d'un répertoire, selon les paramètres spécifiés par le signal. La récursivité n'est pas gérée, le download d'un fichier n'est pas implémenté dans les connecteurs livrés.

Le workflow est nécessairement asynchrone.

Il doit émettre le signal si_transferFinished une fois l'opération de transfert terminée.

4.2.2.4 sl_dirContent

Ce slot déclenche le processus de listing non récursif d'un répertoire avec la possibilité de spécifier dans les paramètres du signal le type d'entrée attendu (fichier ou répertoire) et un filtre sur l'extension du fichier.

Le paramètre _is_for_dir_transfer est réservé à un usage interne au connecteur (listing du répertoire avant transfert de fichier).

Le workflow de listing géré par le connecteur doit collecter l'ensemble des entrées du répertoire avant de remonter le résultat du listing en une seule fois avec le signal si_dirContents.

4.2.2.5 Fermeture du canal de transfert de fichier

A la fin de chaque opération de transfert de fichier (sl_upload, sl_download, sl_dirContent), réussie ou non, le connecteur doit gérer la fermeture du canal de transfert de fichier si la librairie fournit cette fonctionnalité.

Dans tous les cas le connecteur doit émettre le signal si_channelClosed une fois le canal de transfert de fichiers fermé.

4.2.2.6 Erreur de transfert

Si une erreur intervient lors d'une opération de transfert de fichier, elle doit être remontée au client réseau avec le signal si_transferFailed avec le code erreur correspondant.

4.2.2.7 sl_createRemoteShell

Le slot déclenche le processus d'ouverture d'un shell sur le serveur distant.

Le processus est nécessairement asynchrone et doit remonter le signal si_shellReady lorsque le shell est prêt à exécuter une commande.

4.2.2.8 sl_executeCommand

Le slot déclenche le processus d'exécution de la commande :

- Ecriture sur le shell de la chaîne de caractères de la commande
- Ecoute du canal de sortie standard et du canal d'erreur standard

A la réception de données sur la sortie standard, le workflow d'exécution doit remonter le signal si_readyReadStandardOutput (sans paramètre).

A la réception de données sur le canal d'erreur standard, le workflow d'exécution doit remonter le signal si_readyReadStandardError (sans paramètre).

C'est le client qui contrôle la lecture du flux disponible sur les 2 canaux. C'est l'utilisateur de la librairie NetworkTools qui contrôle l'interprétation du résultat de la commande à partir des données remontées, en utilisant l'implémentation de la classe NetworkCommand correspondant à la commande appelée.

4.2.2.9 readShellStandardOutput

Cette méthode doit lire les données disponibles sur le canal de sortie standard du shell.

4.2.2.10 readShellStandardError

Cette méthode doit lire les données disponibles sur le canal d'erreur standard du shell.

4.2.2.11 sl_closeRemoteShell

Ce slot déclenche le processus de fermeture du shell distant.

Le processus asynchrone doit remonter le signal si_shellClosed une fois le shell fermé.

4.2.3 Remontée d'informations de progression

Le connecteur doit évaluer au fil de l'eau la progression de l'opération (action réseau) en cours à partir des éventuelles informations remontées par la librairie tierce utilisée.

Pour évaluer la progression dans le cas d'un upload ou download de répertoire, il est nécessaire de garder la trace du nombre d'octets émis / reçus pour chaque fichier et de comparer le cumul à la taille totale des fichiers à transmettre. La classe NetworkConnector définit les champs protégés nécessaires, notamment la matrice de progression m_progress_matrix.

L'algorithme de progression doit veiller à ne pas signaler plusieurs fois le même niveau de progression pour éviter de saturer l'IHM. L'algorithme est sensiblement le même dans les 2 connecteurs livrés.

Les informations de progression sont remontées à l'utilisateur de NetworkTools avec le signal si_progressUpdate qui véhicule une valeur entière entre 0 et 100 correspondant au pourcentage d'avancement de l'opération.

C'est l'utilisateur de NetworkTools qui est responsable de l'affichage de la progression.

4.2.4 Normalisation des codes erreur

Chaque librairie tierce d'implémentation d'un protocole réseau a sa propre nomenclature d'erreurs.

Lorsqu'une erreur est détectée, le connecteur doit convertir le code erreur reçu en code erreur normalisé. La librairie NetworkTools définit 2 nomenclatures d'erreurs (initialement adaptés du modèle QSsh) sous la forme de types énumérés :

- eConnectionError pour les erreurs liées à la gestion de la connexion au serveur distant
- eTransferError pour les erreurs liées au transfert de fichier

Le connecteur définit donc 2 méthodes privées pour normaliser les code erreur de connexion ou de transfert de fichier.

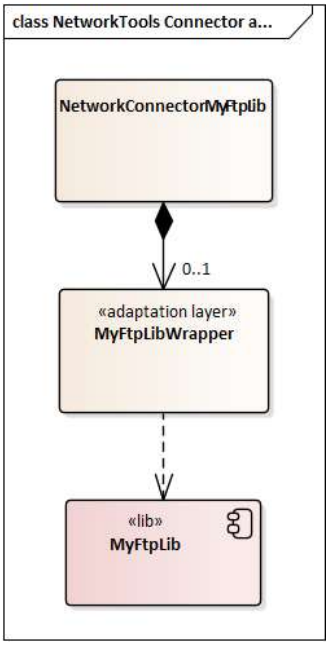
L'appel de ces méthodes permet de remonter le code erreur normalisé via les signaux si_connectionFailed et si_transferFailed respectivement.

Il conviendra d'enrichir ces nomenclatures si de nouveaux cas d'erreur sont identifiés.

Il est à noter que l'utilisateur de NetworkTools, en l'occurrence la classe RemoteJobHelper a la possibilité d'afficher l'erreur remontée à l'utilisateur (cas général) ou de l'intercepter comme c'est le cas avec l'erreur d'authentification qui déclenche un nouveau prompt de mot de passe.

4.3 Couche d'adaptation

La librairie cible FtpClientCpp étant de bas niveau et s'appuyant sur un mécanisme de callback C++ et non sur des événements Qt, il est recommandé de créer une classe intermédiaire d'adaptation pour limiter la complexité du connecteur et rester homogène avec les autres connecteurs en termes d'implémentation. Dans cette logique, le connecteur dépend du wrapper mais ne dépend plus directement de la librairie tierce.



Cette classe qui peut être caractérisée de wrapper d'API aura pour rôles principaux :

- L'invocation de l'API par délégation pour les seuls services utiles à Matisse 3D
- La gestion des callbacks transformés en signaux Qt

Il est également recommandé d'élaborer ce wrapper pour gérer la récursivité en dehors du connecteur. Selon le choix d'implémentation :

Si les performances de transfert de fichier s'avèrent insuffisantes, le wrapper pourra gérer en plus la parallélisation des transferts de fichier.

Par principe, il est recommandé d'injecter le wrapper au connecteur pour rester cohérent avec la logique d'inversion de contrôle.

4.4 Modèle d'implémentation du connecteur

Si la récursivité est implémentée dans le wrapper, l'implémentation du nouveau connecteur pourra suivre le modèle d'implementation de haut niveau de NetworkConnectorQSsh.

Si la récursivité est implémentée dans le nouveau connecteur directement, alors il conviendra de suivre le modèle d'implémentation de plus bas niveau (plus complexe) de NetworkConnectorQFtp.

4.5 Substitution du connecteur existant

La substitution s'effectue simplement en remplaçant la ligne de code d'instanciation du connecteur réseau.

Remplacer :

```
NetworkConnector* ftp_handler = new NetworkConnectorQFtp();
```

Par :

```
NetworkConnector* ftp_handler = new NetworkConnectorMyFtpLib();
```

où NetworkConnectorMyFtpLib correspond à la nouvelle implementation de connecteur réseau.

Le nouveau modèle d'instanciation est le suivant :