CReSTIC

myDICOM

Architecture et fonctionnalités

Oudot Bastien

Présentations des fonctionnalités et de l'architecture de myDICOM, bibliothèque pouvant être intégrée à un projet ou utilisée en stand-alone.

Contenu

myDICOM

Présentation

myDICOM est une bibliothèque dont le but principal est de communiquer avec un PACS en réseau, et de proposer une interface simple pour consulter les données du PACS et télécharger des fichiers DICOM. myDICOM est inspiré du widget ***ctkDICOMAppWidget*** proposé par *The Common ToolKit*.

Architecture

myDICOM est composé de différents modules :

- Un module axé autour du format DICOM et fournissant des méthodes de communication et de gestion de requêtes auprès d'un PACS ;
- un module de paramétrage graphique des paramètres de connexion ;
- un module graphique proposant des widgets commodes ;
- un module graphique offrant une interface dédiée à la réalisation simple et instinctive des opérations de consultation et de téléchargement de données sur un PACS.

Il inclut également un outil autosuffisant permettant de charger un fichier DICOM local et de le transformer en image.

Dépendances

myDICOM s'appuie sur deux bibliothèques, Qt (version 5.3.2) pour les modules graphiques et DCMTK (version 3.6.1) pour le module DICOM&PACS.

myDICOM s'appuie également, sous Windows, sur la bibliothèque ws2_32 (native).

Fonctionnalités

myDICOM propose la liste de fonctionnalités suivante :

- Connexion à un PACS avec réglages des paramètres
- Affichage des données du PACS sous forme hiérarchique
- Filtrage des données du PACS selon différents critères :
    - Nom du patient
    - Nom de l'étude
    - Numéro de la série
    - Numéro de l'image
    - Date de naissance du patient
    - Date de l'étude
    - Date de la série

  - Date de l'image
  - Modalité de l'image
- Affichage des informations d'un fichier DICOM du PACS et prévisualisation des images
- Téléchargement des fichiers du PACS vers une machine
- Conversion de fichiers DICOM locaux en image BMP

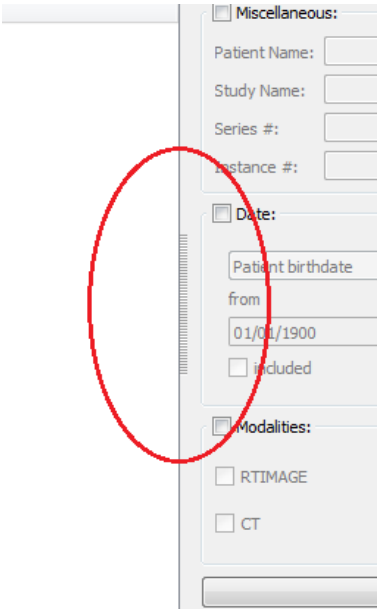
Liste des classes

- Module DICOM&PACS
  - PACSScu
  - StoreScp
- Module paramétrage graphique
  - ConnectionWidget
- Module widgets
  - QvisibleHandleSplitter
  - DateSelector
  - LogWidget
  - LabeledField
- Module IHM
  - MainWidget
  - QTreeWidget[1]
  - FilterWidget
  - InfoWidget
- Utilitaires
  - DcmToBmp

Description des classes

QvisibleHandleSplitter

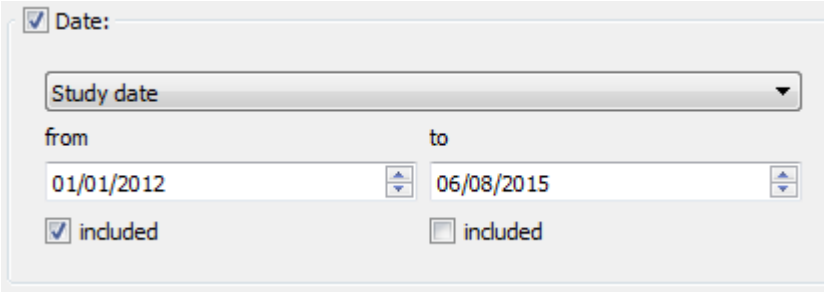
Cette classe est une modification de la classe Qt QSplitter et possède les mêmes fonctionnalités, mais permet de mettre en évidence la zone de redimensionnement (non apparente de base).



[1] Non présent dans les fichiers sources car il s'agit d'une classe de basse de Qt ; cependant, c'est un composant important du logiciel, car le QTreeWidget constitue l'une des trois parties principales du MainWidget (avec l'InfoWidget en bas et le FilterWidget à droite).

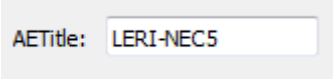
Date Selector

Date Selector est un widget permettant de sélectionner une plage temporelle, définie par deux dates, et spécifique aux champs de dates fournis par la spécification DICOM.



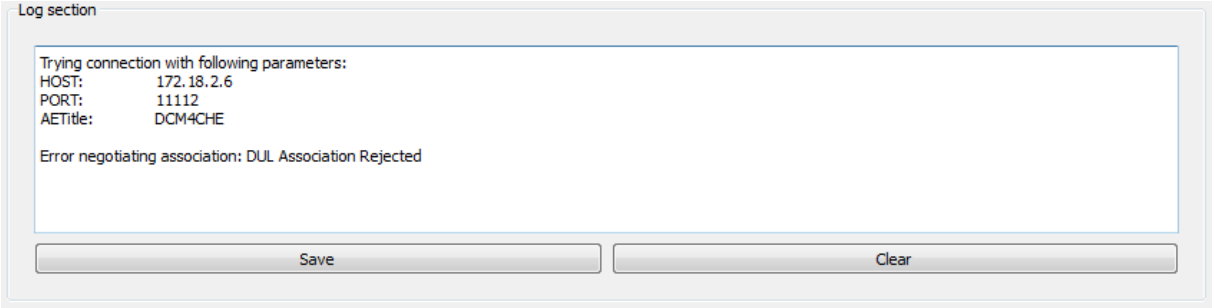
LabeledField

LabeledField est simplement un layout réunissant un Qlabel et un QlineEdit.



LogWidget

Le LogWidget est un widget destiné à afficher une zone de log et permettant d'effacer ou d'enregistrer sur le disque son contenu.

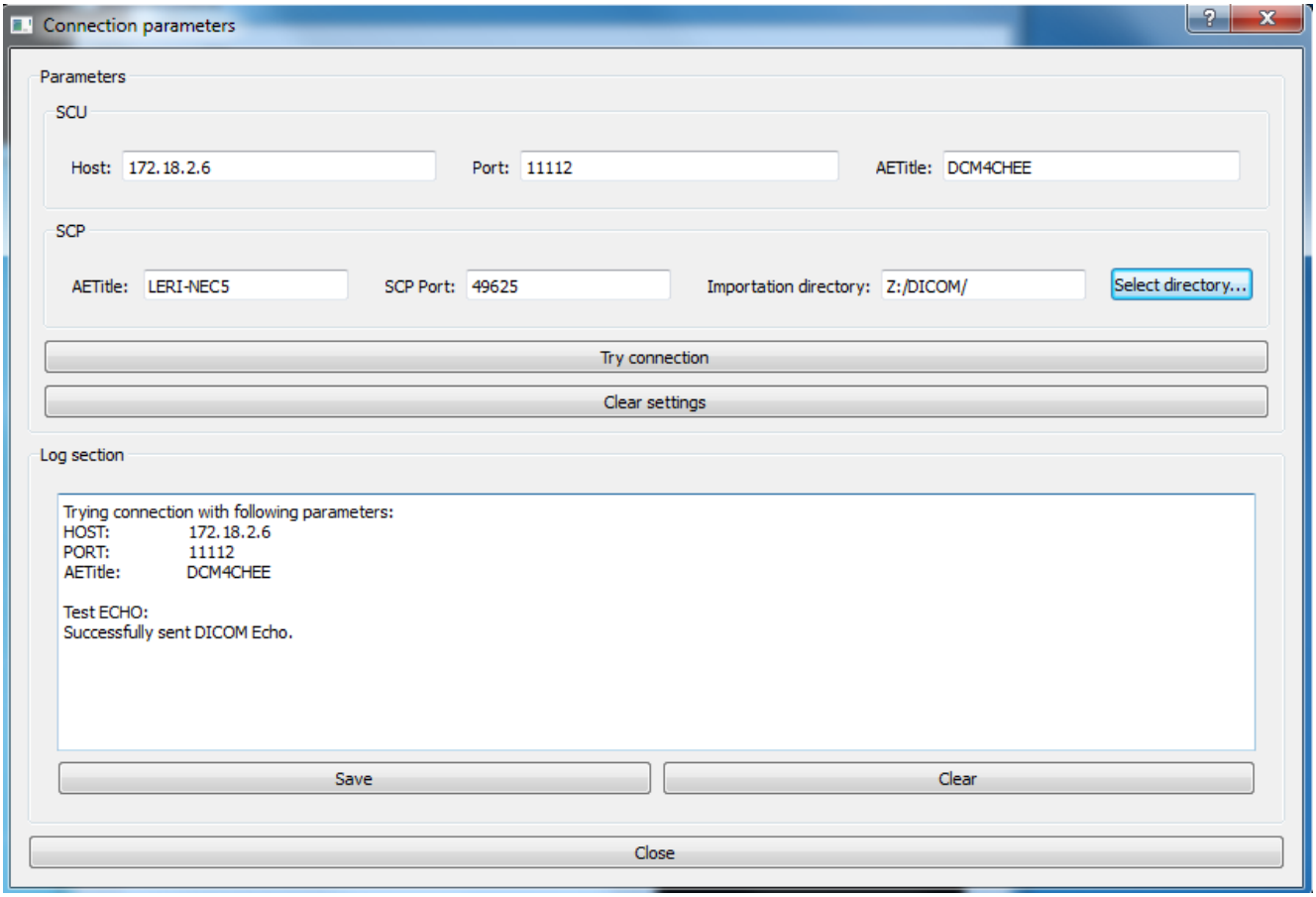


InfoWidget

InfoWidget est un widget regroupant les différents composants primordiaux à l'affichage d'informations d'un fichier DICOM.

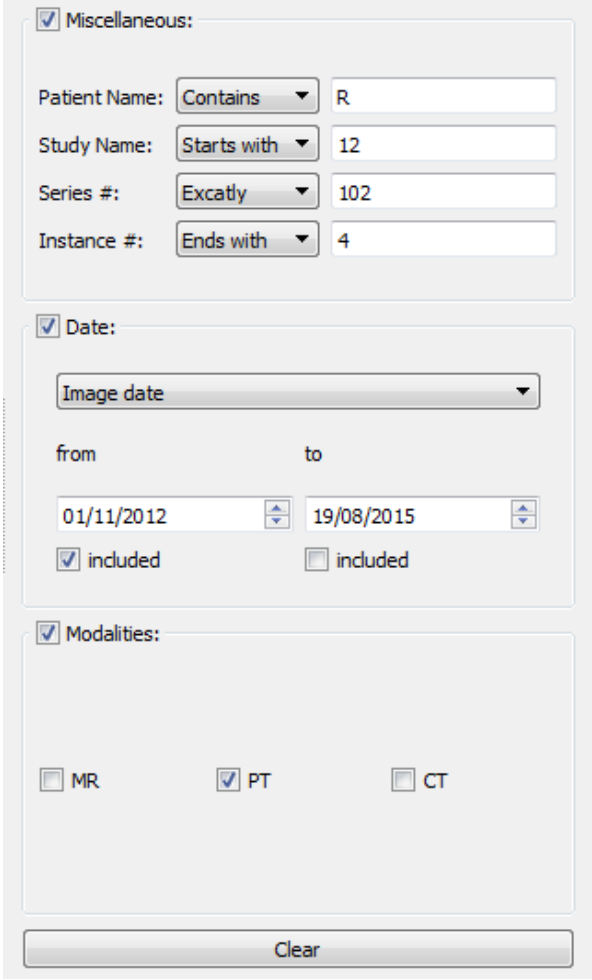
ConnectionWidget

Cette classe regroupe les composants graphiques de la fenêtre de paramétrage de connexion à un PACS.



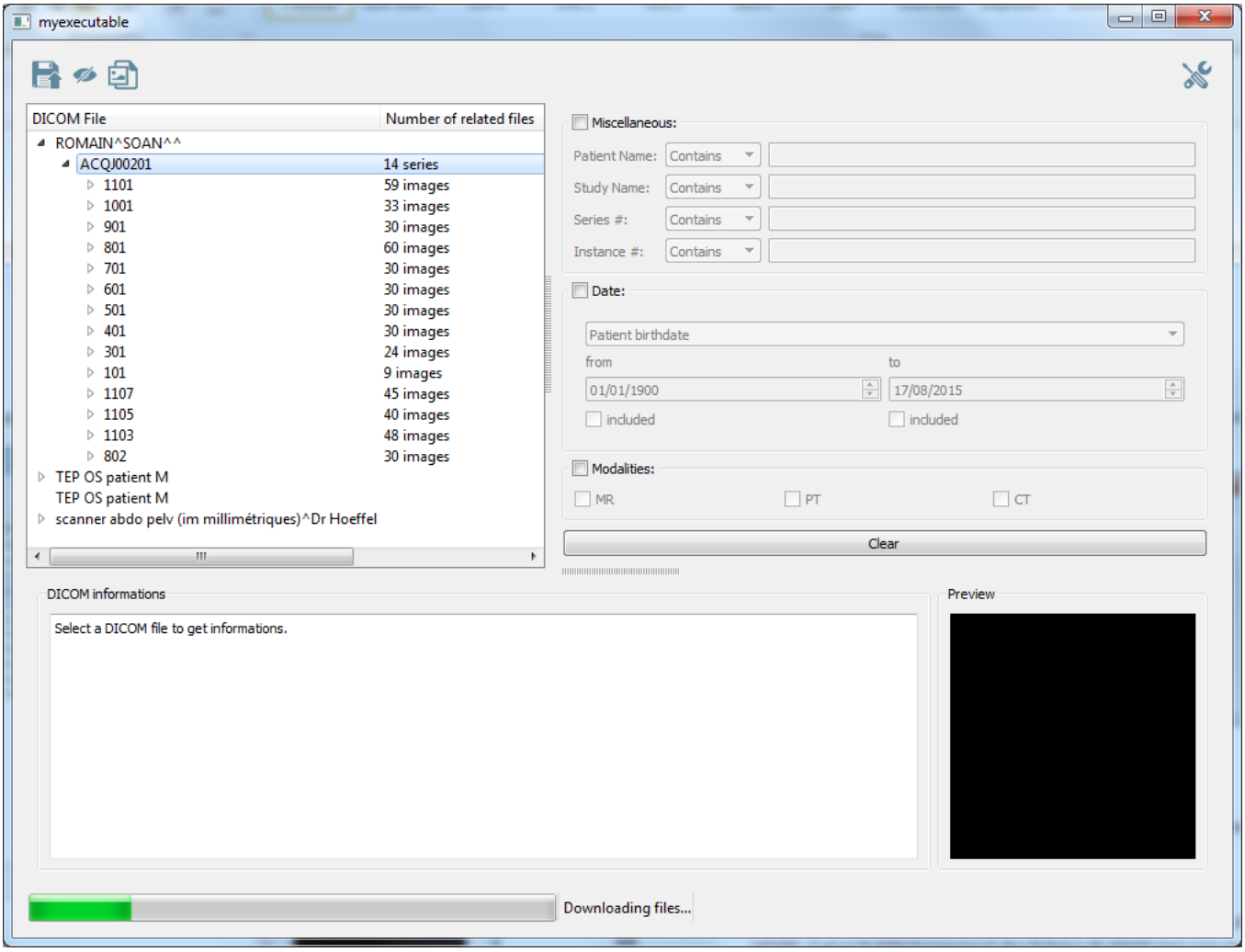
FilterWidget

Cette classe regroupe les composants graphiques de la fenêtre de paramétrage de filtrage des données d'un PACS.



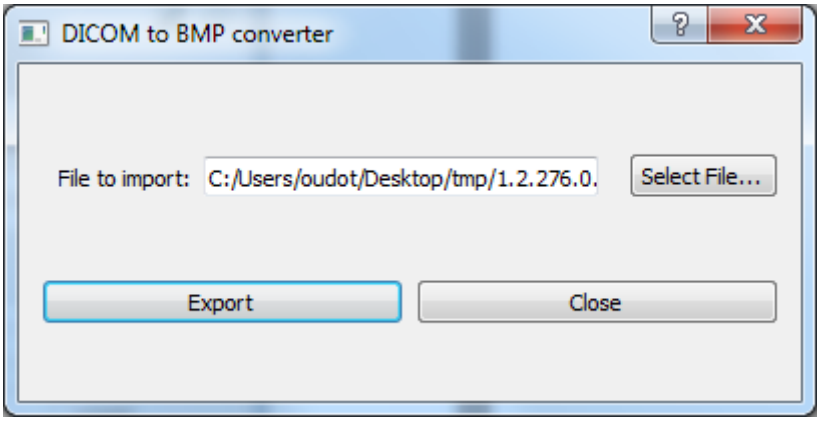
MainWidget

Le MainWidget correspond à la fenêtre principale du programme.



DcmToBmp

Cette classe correspond à un outil utilitaire permettant de charger un fichier DICOM sur le disque et de le convertir en fichier image (bmp).



PACSScu

La classe PACSScu est une implantation d'un scu. Elle est responsable de l'envoi de requêtes vers le PACS.

StoreScp

La classe StoreScp est une implantation d'un scp local, dédiée au traitement de la requête STORE. Elle gère le téléchargement des fichiers du PACS vers une machine.

Fonctionnement

Paramétrage et auto-connexion

La connexion à un PACS requiert de nombreux paramètres parfois fastidieux à fixer (des adresses IP, des identifiants, …). Il est important de pouvoir proposer une interface simple pour les remplir, et également de pouvoir gérer l'auto-connexion de l'application à partir du moment où les paramètres renseignés ont été validés.

Description du fonctionnement

ConnectionWidget offre une fenêtre contenant tous les champs nécessaires à l'indication des paramètres de connexion requis pour la communication au PACS. Une fois ces paramètres fournis et le test de connexion lancé, les paramètres sont mémorisés dans un fichier de configuration sur le disque via un objet QSettings. Lorsque l'on ferme la fenêtre de connexion, les paramètres précédemment enregistrés dans le QSettings sont lus pour paramétrer les objets scu (PACSScu) et scp (StoreScp).

De plus, on enregistre dans le QSettings l'état de la précédente connexion. Si celle-ci a réussi, alors à chaque lancement de l'application, on va automatiquement lancer une connexion avec les précédents paramètres. L'auto-connexion se désactive dès lors qu'une tentative de connexion a échoué (au démarrage de l'application ou lors d'un changement de paramètre à travers le ConnectionWidget).

Signaux, slots et fonctions invoqués

Le ConnectionWidget émet le signal `windowClosed` lorsque sa fenêtre est fermée ; ce signal est récupéré par le MainWidget qui enregistre alors les paramètres de connexion quand le QSettings dans la fonction `setConnectionParameters`.

Thread

Toutes les fonctions, signaux et slots relatifs au paramétrage du scp et du scu se déroulent dans le thread principal de l'application.

Chargement de la base de données

Une des fonctionnalités fondamentales d'un PACS est sa consultation. Certains programmes proposent de télécharger toutes les données et de construire une base de données locale aveccelles-ci, mais cela peut prendre énormément de temps et de place sur le disque. J'ai choisi ici d'interroger la base du PACS en utilisant de simples requêtes FIND et de construire des objets à insérer dans un modèle QTreeWidget à chaque réponse reçue.

Description du fonctionnement

La base de données du PACS est chargée à deux moments :

- Lors du test de connexion effectué au sein du ConnectionWidget ;
- lors du démarrage de l'application si l'auto-connexion est activée.

Dans les deux cas, le chargement est effectué en envoyant une requête FIND au PACS. Cette requête cherche d'abord tous les patients présents, puis pour chaque patient trouvé, génère une requête pour chercher toutes les études de ce patient. Pour chaque étude, une requête pour trouver toutes les séries est envoyée, et pour chaque série, les instances sont récupérées. Toutes les

données ainsi reçues sont ajoutées dans la QTreeView afin de conserver visuellement la structure hiérarchique du PACS.

Signaux, slots et fonctions invoqués

Le chargement de la base de données est lancé par un appel à la fonction `tryConnection` du ConnectionWidget. Le scu prend alors le relai en générant la requête FIND et l'interaction avec le reste de l'application est gérée par trois signaux émis par la classe PACSScu.

- `onFINDRequestStarted - void` : envoyé au moment où la connexion commence. Ce signal est récupéré par le MainWidget qui vide alors le contenu de son QTreeWidget et vide le contenu de la partie « modalité » du FilterWidget. De plus, l'action de masquage/affichage des sections vides du QTreeWidget est désactivée. Enfin, la barre de progression est réglée en mode indéterminé ;
- `onFINDResponse – DcmDataSet*` : émis lorsqu'une réponse FIND positive a été envoyée par le PACS. On récupère alors les données transmises par la réponse qui seront ajoutées en tant que QTreeWidgetItem. De plus, si l'objet reçu est une image, on regarde sa modalité et on ajoute cette option de modalité au FilterWidget (si elle n'y est pas déjà présente). De cette façon, on construit un panneau de modalités qui ne contient que des modalités présentent sur le PACS [2];
- `onFINDEnd – bool` : notifie la fin de la requête. L'action de masquage/affichage des sections vides est réactivée et la barre de progression ramenée à 100%.

Thread

L'envoi de la requête par le scu et son traitement par le scp sont traités dans un thread différent du thread principal de l'application. Cela permet de naviguer dans l'application pendant que les données continuent de charger. Le contraire est susceptible de poser problème dès lors que le PACS interrogé contient une très grande quantité de données qui peuvent de fait mettre beaucoup de temps à se charger entièrement.

Filtrage

Il est important, dès lors qu'un PACS peut contenir plusieurs milliers de fichiers, d'avoir la possibilité de ne montrer que les données obéissant à certains critères. Les fichiers DICOM possédant chacun des centaines de champs, cela fait d'autant plus de critères potentiellement intéressants par rapport auxquels on souhaiterait pouvoir effectuer un filtrage.

Description du fonctionnement

Le filtrage a pour but d'éliminer certains objets présents dans le QTreeWidget ne répondant pas à certains critères.

La requête FIND permet de consulter le PACS et d'obtenir des résultats en fonction de certains critères, il paraissait alors naturel d'utiliser cette méthode pour filter les données. Cependant, cela suscitait beaucoup de complications, pour la plupart liées aux temps d'exécution. Implanter le filtrage de cette façon signifiait exécuter des requêtes FIND à chaque demande de

[2] L'intérêt premier vient du fait qu'il existe une liste de plus de cinquante modalités différentes applicables au format DICOM. Toutes les ajouter au FilterWidget nuirait gravement à la lisibilité du programme (d'autant plus que certaines modalités tendent à être dépréciées).

filtrage, et surtout ré-exécuter le chargement de la base de données à chaque fois qu'on souhaitait annuler le filtrage. Cette opération est certes gérée en programmation concurrentielle pour ne pas bloquer l'utilisateur pendant le chargement, mais reste tout de même longue et complexe pour un simple réaffichage (sur ma machine, charger toutes les données du PACS à ma disposition, à savoir plus de 4600 fichiers, ce qui correspond à un PACS relativement petit, prenait une dizaine de secondes).

J'ai donc opté pour une approche différente. Plutôt que de travailler sur le PACS directement, j'ai décidé de travailler sur les fichiers chargés, qui correspondaient à des QTreeWidgetItem. A chaque objet correspondait un ensemble de 6 chaînes de caractères, sélectionnés parmi tous les champs DICOM en fonction de leur pertinence :

1. DICOM File : le nom du fichier ;
2. Number Of Related File : le nombre de fichiers contenus en tant que fils de cet objet (utilisé pour indiquer combien il y a de séries par étude et combien d'images par série) ;
3. ID : chaque fichier DICOM possède un identifiant présupposé unique ;
4. Date : la date du fichier, dépendant de son type ;
5. Modality : utilisé pour renseigner les modalités de prise de vue des images (IRM, rayons X, …) ;
6. TYPE : permet de décrire le type de données du fichier (patient, étude, série ou image).

DICOM File	Number of related files	ID	Date	Modality	TYPE
ROMAIN^SOAN^^		29817185	12/04/2012		PATIENT
ACQJ00201	14 series	1.2.124.113532.80.22205.22074.20120511.140155.3053709565	11/05/2012		STUDY
1101	59 images	1.3.46.670589.11.17131.5.0.5596.2012051114462681110	11/05/2012		SERIES
1		1.3.46.670589.11.17131.5.0.5596.2012051114492700154	11/05/2012	MR	IMAGE
2		1.3.46.670589.11.17131.5.0.5596.2012051114492715155	11/05/2012	MR	IMAGE

De plus, les objets QTreeWidgetItem sont des objets Qt proposant des fonctionnalités graphiques comme le fait d'être visible ou invisible au sein du QTreeWidget. Il était donc largement moins coûteux, plus simple et plus rapide de rendre invisibles les objets qui ne correspondent pas aux critères de filtrage en comparant les chaînes de caractères avec les paramètres choisis par l'utilisateur. L'annulation du filtrage est alors un simple parcours des objets en les rendant tous visibles.

Signaux, slots et fonctions invoqués

Le FilterWidget possède deux signaux qui lui permettent de communiquer avec le MainWidget :

- `onClear – void` : émis lorsque le filtrage est annulé. Cela a pour effet de rétablir la visibilité des objets du QTreeWidget. Le FilterWidget vide également tous ses champs de filtrage.
- `onFilter – 3*bool` : le FilterWidget indique quelle(s) partie(s) des paramètres de filtrage ont été activées.

Le déclenchement du signal onFilter obéit au concept de recherche progressive (incremental search). Chaque modification du FilterWidget déclenche la fonction `filter` qui initialise et remplit les

structures de données contenant les paramètres, et émet le signal `onFilter` pour un filtrage en temps réel.

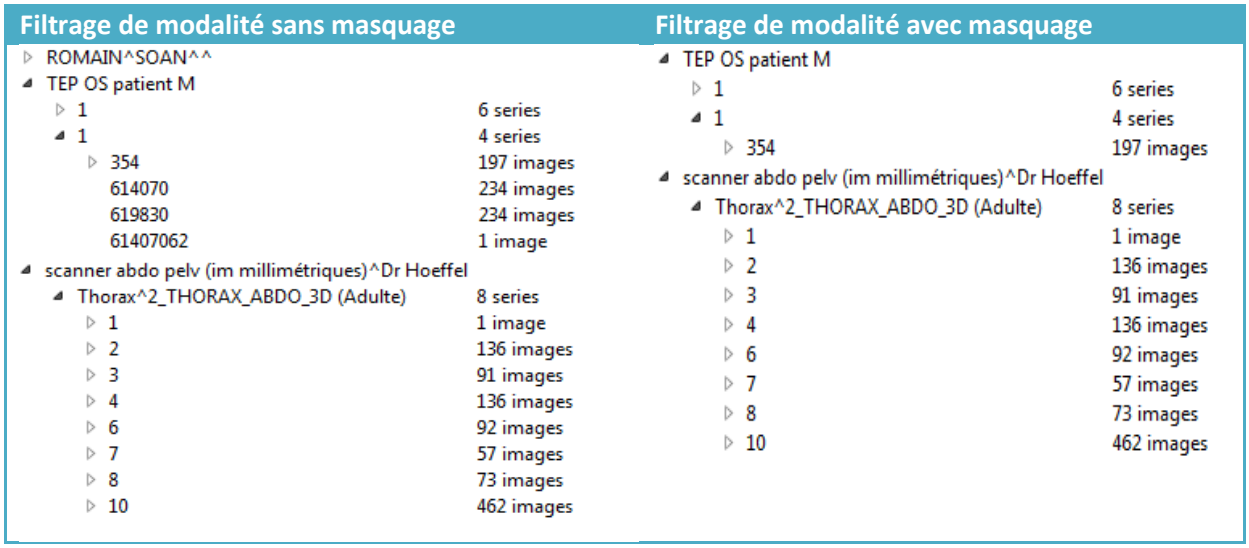
Le signal `onFilter` passe trois arguments au MainWidget qui correspondent à l'état d'activation des trois sous-panneaux du FilterWidget : le panneau divers (nom du patient, nom de l'étude, etc.), le panneau de date (date de naissance, date de capture de l'image, etc.), et le panneau de modalité. Le MainWidget récupère alors les structures préparées par le FilterWidget en fonction de ces états.

Thread

La fonction de filtrage est rapide puisqu'elle consiste en une lecture des objets du QTreeWidget et d'une comparaison pour chaque objet avec les paramètres reçus (fonction de complexité linéaire). Tout fonctionne donc au sein du thread principal de l'application.

Masquage/affichage des sections vides

Après un filtrage, ou selon les données présentes sur un PACS, il peut exister des sections vides (c'est-à-dire des séries qui ne contiennent aucune image (série vide), des études qui ne contiennent que des séries vides (étude vide) ou des patients qui ne contiennent que des études vides (patient vide)). Ces sections vides peuvent gêner la visibilité du programme, notamment dans le cas du filtrage (par exemple, si on filtre les données par rapport à un paramètre propre au type IMAGE, on va cacher les images qui ne correspondent pas à ce paramètre. Si une série ne contient que des images ne correspondant pas au paramètre, elle devient une série vide.), mais nécessitent parfois d'être représentées (dans le cas où un patient de la base ne possèderait aucune étude, on pourrait toutefois souhaiter le fait qu'il soit visible dans la liste des patients). Il est donc important de proposer une action activable / désactivable permettant de masquer ou d'afficher les sections vides.



Description du fonctionnement

Un bouton activable est présent sur la barre d'outils. Son activation déclenche la fonction de masquage.

La fonction de masquage est simplement une fonction récursive qui va masquer une section si cette section n'est pas une image et ne contient pas de sous objet, ou si tous les objets de cette section sont invisibles. L'algorithme remonte alors d'un cran et vérifie que la section qui contient la

section sur laquelle on vient de travailler ne devient pas elle-même une section vide du fait du masquage potentielle de ses objets. Cette fonction est automatiquement lancée après un filtrage si le bouton est activé.

Lorsque le bouton est désactivé, on rend apparent tous les objets du QTreeWidget puis on relance la fonction de filtrage si celui-ci était activé.

Thread

Les fonctions de masquage et d'affichage sont des simples fonctions de lecture linéaire. Elles sont donc rapides, et s'exécutent dans le thread principal de l'application.

Construction des dossiers

Le QTreeWidget présente les différents fichiers du PACS sous une structure hiérarchique (un patient contient des études, qui contiennent des séries, qui contiennent des images). Lorsqu'on télécharge des images, il paraît pertinent de retrouver cette structure à l'emplacement du téléchargement en utilisant des dossiers et sous-dossiers.

Description du fonctionnement

La création des dossiers se fait au moment de l'importation de fichiers (mais pas pour la gestion de la prévisualisation). On regarde quel élément est sélectionné et on construit tous les dossiers et sous-dossiers en rapport avec l'élément sélectionné.

Signaux, slots et fonctions invoqués

Au moment de l'importation, le MainWidget exécute sa fonction `createDataBaseDirectories` qui prend en paramètre l'objet sélectionné dans le QTreeWidget et le répertoire de base où seront téléchargés les fichiers (renseigné dans les paramètres de connexion). Cette fonction crée la hiérarchie de dossier qui correspond à l'élément sélectionné (un dossier patient, les dossiers études, les dossiers séries).

Thread

Cette fonctionnalité s'appuie sur les mécanismes offerts par la classe QDir. Simple parcours des objets du QTreeWidget et création de quelques dossiers, cette fonction rapide s'exécute dans le thread principal de l'application.

Téléchargement des fichiers

Le téléchargement de fichiers depuis un PACS vers un ordinateur est une fonctionnalité essentielle de ce type d'application. Même si les mécanismes de cette action sont parfois complexes, en termes de requêtes et de protocoles, il était important de pouvoir proposer cette possibilité de façon simple et efficace.

Description du fonctionnement[3]

Les fonctions de téléchargement de fichiers rentrent en jeu à deux moments : d'abord lors de la sélection d'une image dans le QTreeWidget (*cf.* Gestion de la prévisualisation), mais aussi lorsque l'utilisateur choisit de télécharger des fichiers, en cliquant sur le bouton ***Import Files*** de la barre d'outils. Au moment de l'importation, on distingue deux cas de figures :

[3] Pour une description théorique du fonctionnement, consulter le rapport d'analyse, aux paragraphes requête MOVE et requête STORE.

1. Le fichier sélectionné est de type IMAGE : on vérifie que cette image n'est pas déjà chargée dans le répertoire temporaire. Si elle y est, on la copie vers le chemin adéquat, sinon, on construit la requête MOVE correspondante.
2. Le fichier sélectionné n'est pas de type IMAGE : on construit la requête MOVE adéquate pour télécharger toutes les images concernées. On calcule également le nombre de fichiers totaux à télécharger (en s'appuyant sur le nombre d'objets images fils de la section que l'on souhaite télécharger) afin d'initialiser la barre de progression.

Lors du téléchargement de fichiers, le bouton de téléchargement est désactivé pour éviter que l'utilisateur ne télécharge plusieurs paquets de fichiers en même temps. Même si cela est techniquement rendu possible par l'architecture de l'application, cela provoque un dysfonctionnement au niveau de la barre de progression, qui n'est pas pensée pour gérer des téléchargements en parallèle.

Signaux, slots et fonctions invoqués

Pour l'importation de fichiers, le MainWidget s'appuie sur les fonctions du scu.

Le scu génère deux threads : un pour générer et envoyer les requêtes MOVE, et un pour lancer l'écoute du scp. Il est important que ces deux fonctions s'exécutent sur deux threads différents car il y a établissement d'une connexion TCP/IP entre les deux agents. Lorsque le scp local reçoit une requête MOVE, il effectue, dans sa fonction de callback, la sauvegarde du fichier demandé sur le disque, puis renvoie une réponse STORE. A ce moment, le scu émet le signal `fileReceived` pour informer le MainWidget qu'un fichier a été téléchargé, permettant de mettre à jour la barre de progression.

Thread

Les fonctions gérant les requêtes MOVE et STORE sont créées dans deux threads différents, de façon à les faire fonctionner parallèlement (obligatoire dans le cas d'une communication entre scp et scu), et ces deux threads sont différents du thread principal de l'application, de façon à ne pas gêner la navigation au sein de l'application pendant un téléchargement.

Gestion de la prévisualisation

Les noms de fichiers, de série ou d'étude ne sont pas toujours explicites (il s'agit la plupart du temps de nombres). Ainsi, il est important de proposer un moyen de pré-visualiser les fichiers sélectionnés, en offrant des informations succinctes et un aperçu de l'image.

Description du fonctionnement

Le chargement de la prévisualisation est effectuée dès lors qu'un fichier de type IMAGE est sélectionné dans le QTreeWidget. Cela provoque le téléchargement du fichier sélectionné, dans le dossier temporaire (la hiérarchie des dossiers n'est alors pas prise en compte). On extrait du fichier DICOM les informations intéressantes, ensuite mises en forme en HTML. On convertit ensuite le fichier DICOM en image BMP, qu'on redimensionne et qu'on place en tant que QPixMap dans le label de prévisualisation.

Signaux, slots et fonctions invoqués

La prévisualisation est décrite par la fonction `setPreview` du MainWidget. Elle s'occupe de télécharger le fichier sélectionné s'il n'est pas déjà présent dans le dossier temporaire. L'application attend ensuite que le fichier soit téléchargé avant d'en extraire les informations importantes, et de les transmettre à l'InfoWidget.

Thread

Comme pour le téléchargement de fichiers, deux threads sont dédiés aux requêtes du scu et du scp. Cependant, l'application doit attendre que le fichier soit trouvé avant de l'exploiter. Il n'est donc pas possible de naviguer au sein de l'application pendant que le chargement se déroule.

Améliorations possibles

Le programme myDICOM est dans une première version stable et fonctionnelle. Cependant, des améliorations sont possibles.

Prévisualisation d'une série

Il pourrait être intéressant d'offrir à l'utilisateur la possibilité d'avoir un aperçu de tout ou partie des images d'une série plutôt que de n'avoir un aperçu que pour un fichier de type IMAGE. Cela demanderait une requête STORE plus longue mais pourrait être fait de façon concurrentielle afin de ne pas bloquer l'application.

Meilleure pertinence des informations de prévisualisation

A l'heure actuelle, seules quelques informations sont affichées dans la prévisualisation. Il serait intéressant de discuter avec les différents médecins afin de connaître leurs opinions sur l'importance des champs à afficher.

Meilleure pertinence des paramètres de filtrage

A l'instar des informations de prévisualisation, il serait important de connaître l'avis des médecins sur les différents critères de filtrage à mettre en place.

Meilleur système de log

Le LogWidget ne permet à l'heure actuelle que d'avoir un retour sur la tentative de connexion et la requête ECHO envoyée au PACS. Il serait pratique de pouvoir avoir un retour sur le déroulement des requêtes FIND et des requêtes MOVE/STORE.

Téléchargements multiples

Au vue du code, il est possible de lancer plusieurs téléchargements de paquets de fichiers simultanément. La fonctionnalité a été désactivée volontairement (l'action ***Import Files*** est grisée pendant le téléchargement) car mal gérée vis-à-vis de la barre de progression.

Sélection des dates

La sélection des dates dans le DateSelector n'est pas toujours très pratique. Il serait judicieux de coupler les QDateEdit avec des QCalendarWidget pour permettre une sélection plus simple.

Cohérence cache/PACS

Les fichiers temporaires chargés pour la prévisualisation ne sont pas effacés après la fermeture de l'application, de façon à pouvoir être directement réutilisée ensuite. Il pourrait être intéressant toutefois de comparer le contenu du dossier avec le contenu du PACS, pour supprimer des fichiers qui ne seraient plus dans le PACS, ou pour éviter des problèmes de doublons si l'on travaille sur plusieurs PACS.

Annexes

Annexe 1

Diagramme de classe