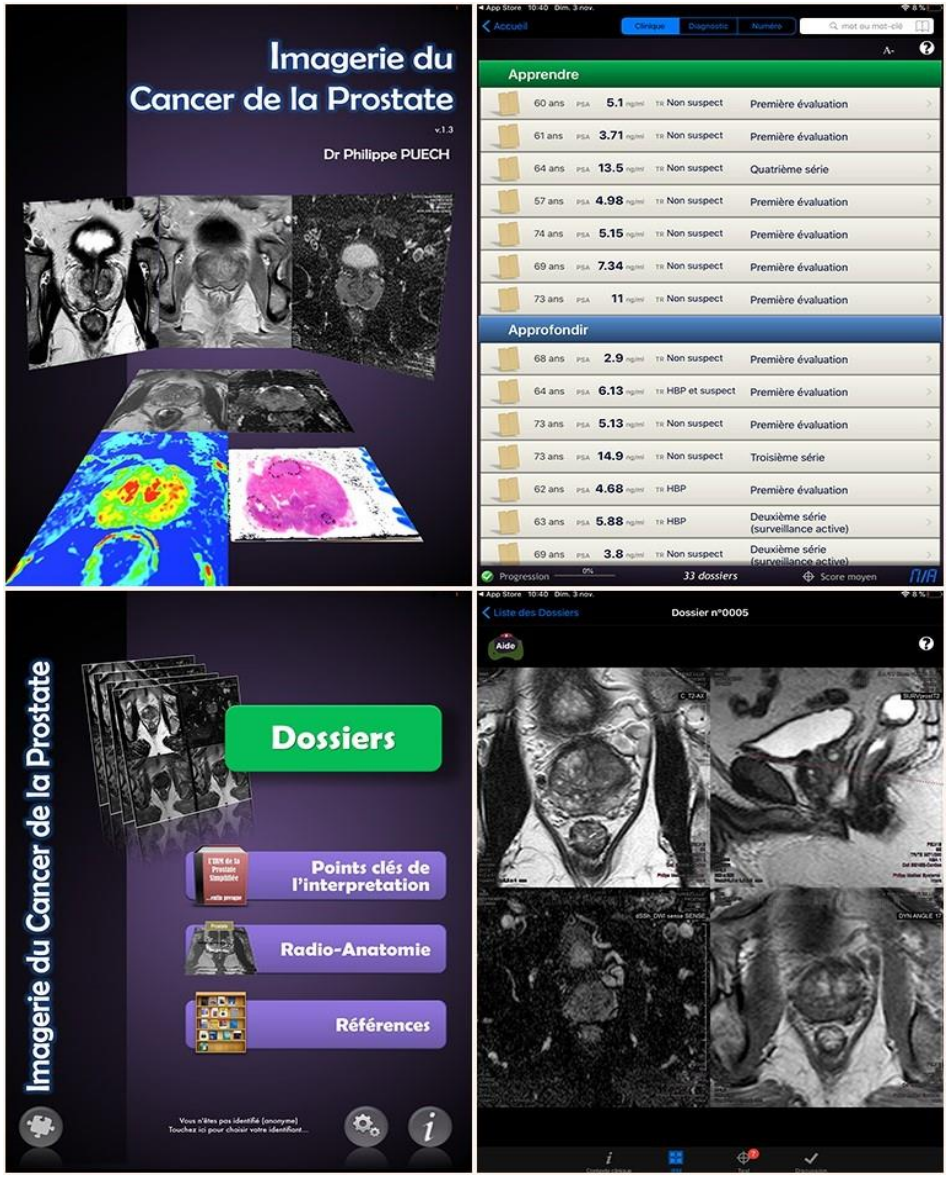
RadioLogic

A case-based learning and self-assessment tool for the Orthanc Ecosystem for Medical Imaging

by Marco Barnig

1. Introduction

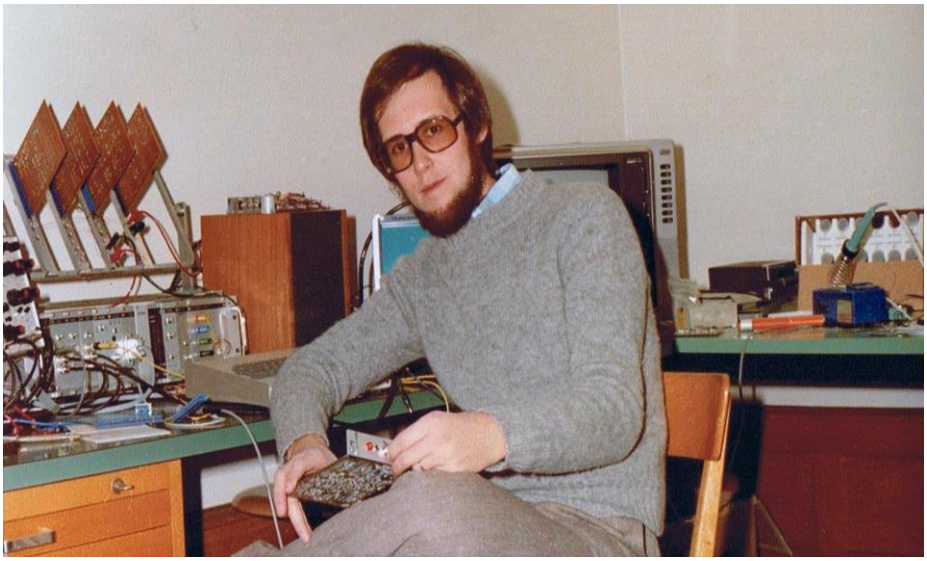
A few years ago, my son in law, Guillaume Bierry, who is professor of radiology at the university hospital in Strasbourg, asked me if I could create a teaching tool for medical students, based on real clinical cases. Of an example he showed me an application about prostate cancer imaging, developped by professor Philippe Puech from Lille.



Case-based learning (CBL) is an efficient method for radiologist education. Most CBL systems available on the web are based on a single image. Guillaume wanted a tool including a real DICOM viewer to visualize whole medical imaging studies from different modalities. After doing some search work on Google concerning the state of art of available medical imaging software, I took up the challenge to start the development of OFUR (outil de formation universitaire radiologique) in march 2015.

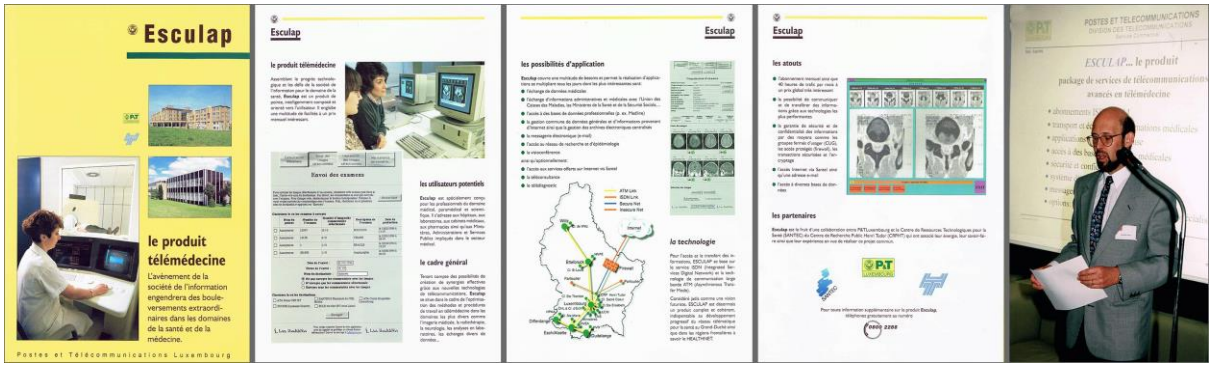
2. My background

There are two types of software developers: the specialist who knows everything about nothing and the generalist who knows nothing about everything. I am close to the second category. I had the privilege to work with the first microprocessor Intel 4004 in the early seventies when I was a research assistant at the Federal Institute of Technology in Zurich.



In 1978 I joined POST Luxembourg, the incumbent telecom operator in Luxembourg. At the beginning I was in charge of new telecommunication services : Alarmis, Euronet, Luxpac, Videotex, ISDN, LUXGSM, Internet. Fifteen years later I became responsible for sales and marketing of all telecommunications services. I changed from bits and bytes to customers and contracts. In my new function I had the pleasure to present the latest new technical service, the first mobile Internet access (WAP), to the Grand-Duke of Luxembourg.

In 1996, POST Luxembourg launched an experimental package for telemedicine, called Esculap.



During my leisure, I continued to follow the progress of the new advanced technologies. I created my first personal websites in the late 90's where I blogged about algorithmic music, speech synthesis and speech recognition, 3D animations, artificial intelligence, digital and conceptual art. When I retired in mid 2013, I had all the assets at my disposal to progress with challenging developments: plenty of time, experience, powerful hardware (Windows, Apple and Linux workstations and tablets) and independence.

3. OFUR (outil de formation universitaire radiologique)

In June 2015 I presented the concept of the planned radiology teaching tool to Professor Bierry.

The main points were:

1. use Orthanc (version 0.8.6) as PACS to serve the clinical case files.
2. use Cornerstone (by Chris Hafey) or DWV (by Yves Martelli) as a framework for the DICOM viewer.
3. optimize the viewer for iPADs.
4. target three groups: radiology students, graduated radiologists (life-long learning), physicians.
5. create three products : OFUR-chu, a server-client platform running on Apple devices; OFUR-lite, a free client HTML5 web application; OFUR-pro, a client web application in App-Store with integrated purchase of clinical cases.
6. use english as interface language.
7. use open source software license.
8. do not request a medical certification, because the tool is only used for education, not for diagnostic purposes.
9. comply with personal data protection.

10. create the project in five steps.
11. foresee a development time of about twelve months.
12. host the project on Amazon Web Services (AWS).
13. provide personal assessment facilities.
14. allow usage in training rooms in the hospital where Internet is not available.
15. set a priority on low costs.

The concept was approved and I started with the development of OFUR-lite, the progressive web application which was the main part of the whole system. After evaluating the Cornerstone and DWV javascript libraries in detail, I took the decision to go on with the (complex) Cornerstone framework because I experienced some severe problems with the (simple) DWV project on the iPAD. Several weeks later a first prototype of the OFUR-lite web viewer was ready.

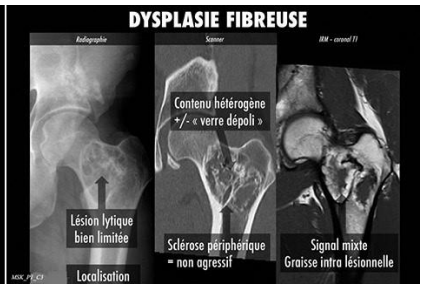
4. Clinical cases

In August 2015 Professor Bierry created the first clinical case to start real-life technical tests. The first tests executed revealed several issues:

- the duration to download the clinical case from AWS or from the Orthanc server to the client was high.
- the image size of CR files was to high to be handled on the iPAD.
- the cornerstone framework supported only ASCII characters, french characters (é,è,à, ..) were not displayed as expected.
- the Javascript same-origin policy did not allow to make requests across domain boundaries without enabling CORS.

While I tried to solve the reported problems, Professor Bierry continued to create additional clinical cases.

A first session included five cases concerning a MSK (Musculoskeletal) pathology. A second session presented three SPINE pathologies. Each clinical case was described with three pictures: one showing the clinical data (observation), a second listing possible diagnoses (questions) and a third presenting the correct diagnosis (answer) with detailed explanations.



The DICOM files for the clinical cases and the integration of the pictures in DICOM files were assembled with Osirix, the most widely used DICOM viewer in the world.

5. DICOM is easy

To solve the mentioned issues, it was necessary to dive deeper into the DICOM standard. I first explored the famous DICOM opensource toolkit DCMTK consisting of libraries and applications provided by OFFIS e.V. from Germany. The second explored toolkit was GDCM, an implementation of the DICOM standard developed by Mathieu Malaterre. You probably know that these tools are also the foundation of the Orthanc server. I used these tools to down-scale large DICOM images, to compress all images lossless in JPEG2000 format and to modify the DICOM tags.

I used other great tools like the DVTk editor, the dicom3tools package created by David A. Clunie, including the DICOM validators dciodvfy and dcentvfy, to get familiar with medical imaging. I published some overviews about DICOM viewers for desktops and mobiles on my blog web3.lu.

Roni Zaharia, CEO of the DICOM company H.R.Z. Software services LTD, located in Tel-Aviv, Israel, maintains a blog DICOM is easy including an outstanding tutorial about DICOM. I my opinion DICOM is not easy, I think that the DICOM standard is a nightmare. Despite the steep learning curve, I gradually got the necessary knowledge to manually process the DICOM files to reduce their size and to embed private tags, to save information about the possible and correct diagnoses. I created a patch to modify Chris
Hafeys DicomParser to support UTF8 characters. One year later I published a modification of the related source code (cornerstoneDicomParserUTF8) on GitHub. To enable CORS I added a reversed proxy (nginx) to the Orthanc server.

It's worth noting that the section Understanding DICOM with Orthanc of Sébastien Jodogne's DICOM book, first published in 2016, is now a great help to start with digital imaging.

6. Name and Logo

As all the blocking issues were solved at the end of 2015, we decided to progress with the project. Time was come to choose a definitive name and to define a logo. Early 2016, Guillaume proposed RadioLogic as final project name and he dressed the following logo, representing and old-fashioned x-ray film viewer.

I converted the logo picture into SVG files and derived icons in different resolutions for use on the web. The next step was to obtain a domain name (radiologic.fr) and to set up a first (hidden) website on AWS to host all the documentation and code about the project.

7. RadioLogicTutor

A stable version of the RadioLogic viewer, called RadioLogicTutor, was ready in mid-2016. This web application is composed of ten modules which can be accessed by the navigation menu at the top of the screen. The different menus are enabled or disabled depending on the user context. For example a user can only view the correct diagnosis when he has submitted his own diagnosis. Before submitting a diagnosis the user must view the DICOM images and before accessing the viewer he must select a session and a related clinical case.

The RadioLogicTutor uses the jQuery Mobile framework to display the content.

7.1. Login / Logout

The first module allows the user to login or logout to the web application. At the first visit the user chooses a username and a password which is saved inside the device browser. The user id's are not saved in a central server. In the case of a common

device by different user, a maximum of ten accounts can be defined and one user acts as administrator to clear the user content, if necessary.

7.2. About

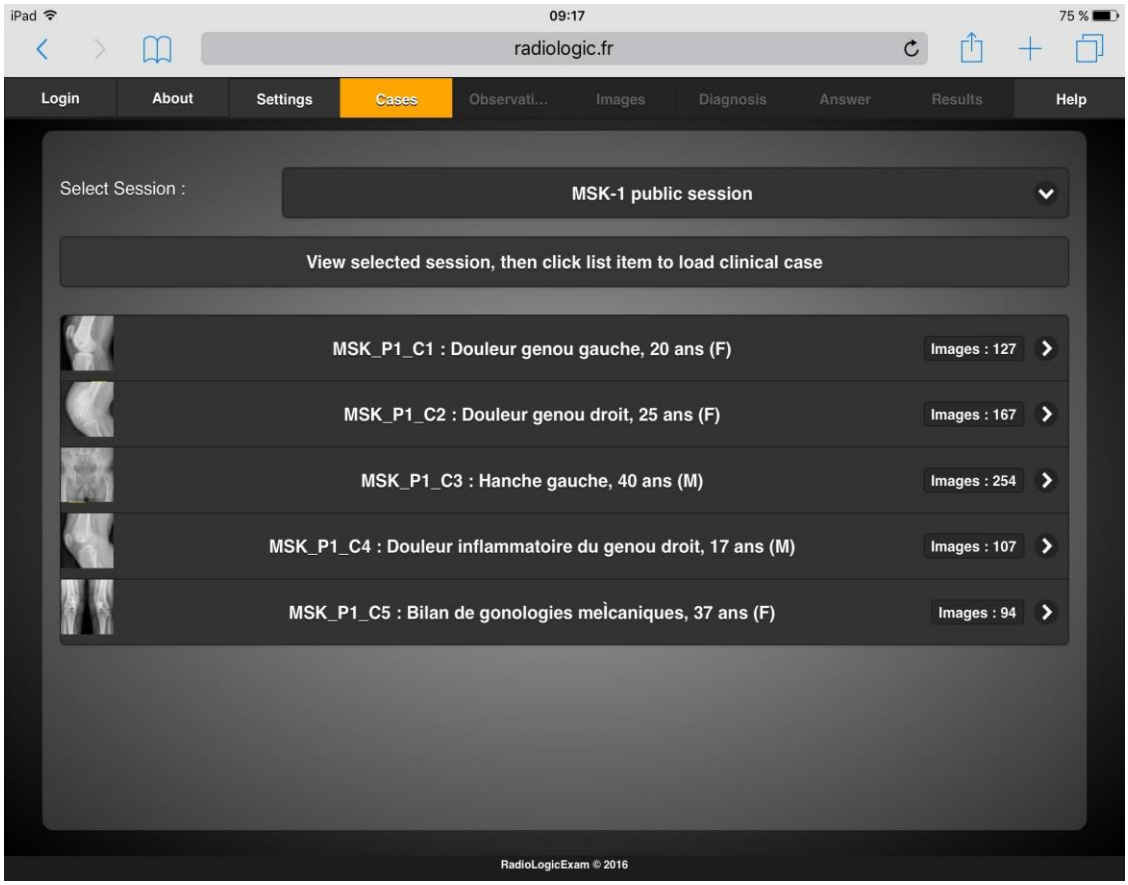
The tab About shows a classic webpage about the project with infos, links and credits about the contributors.

7.3. Settings

The main settings are related to the location of the DICOM archive hosting the clinical cases: public Orthanc server on the Internet, private Orthanc server in the local network or even a ZIP file on an USB stick (see my public GitHub repositories dumpDICOMDIRarchive and dumpZIPwithDICOMfiles). Other settings allow users to change some system preferences.

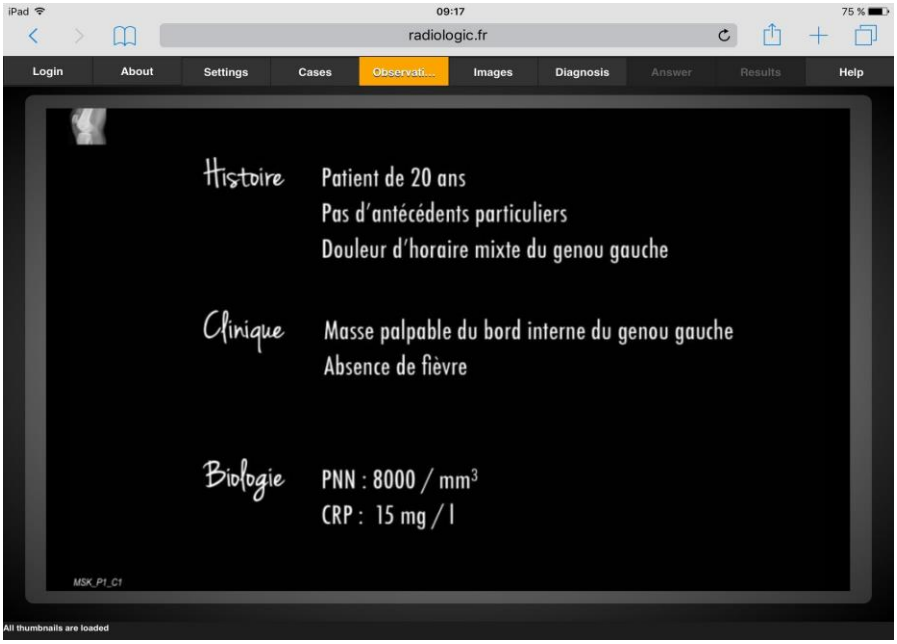
7.4. Cases

The module Cases provides an interface to select an available session, to list the included clinical cases and to download the chosen case from the archive. The thumbnails used to illustrate the clinical cases are extracted as 64 x 64 pixels in the upper left corner of the observation picture.



7.5. Observation

The tab Observation shows clinical data about the patient with free text in a grey or color image (see thumbnail in the upper left corner).



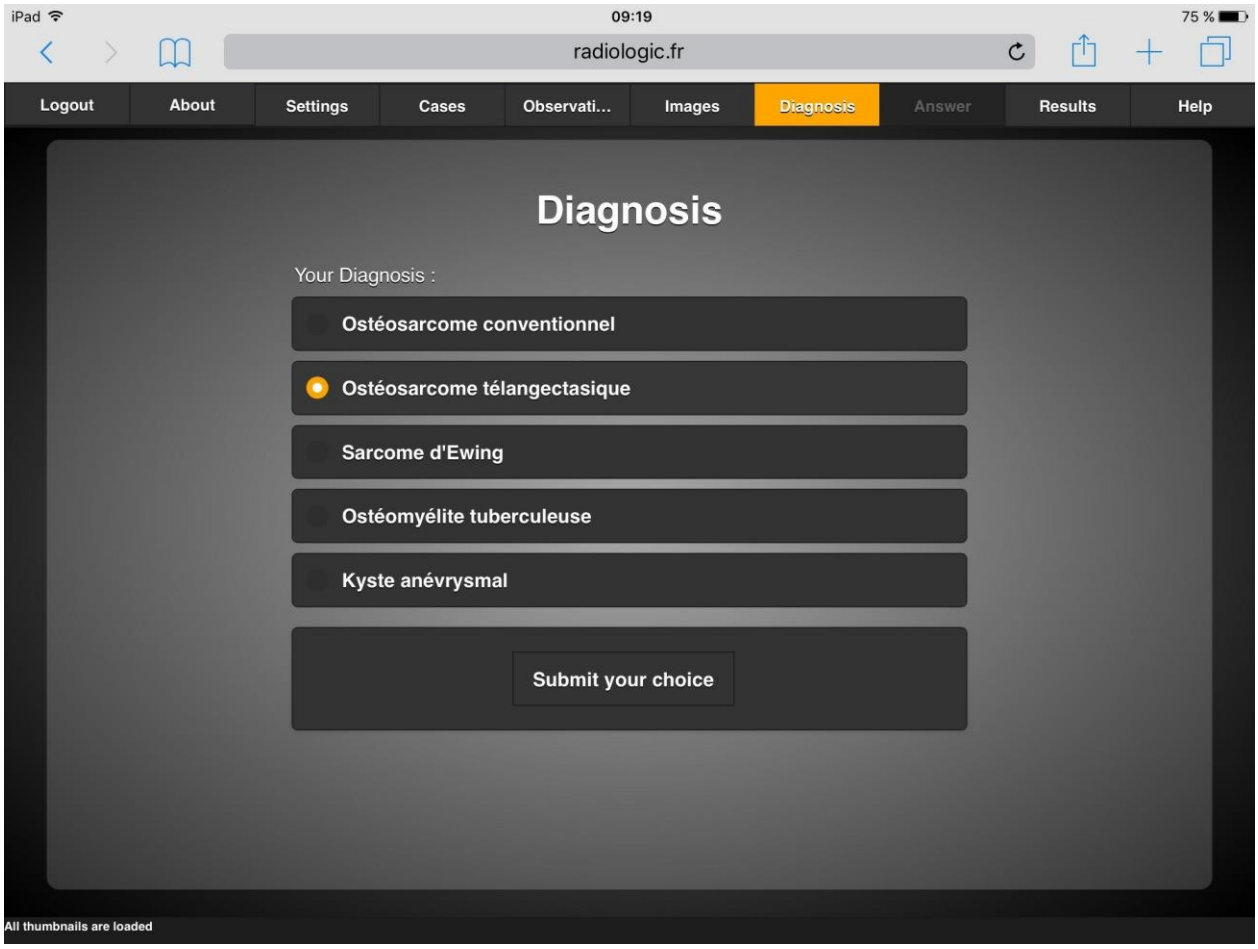
7.6. Images

The image panel is divided in the four Viewports named leftTop, leftBottom, rightTop and rightBottom. A thumbnail bar containing all series is displayed at left when the screen is in landscape orientation. The bar is displayed at the bottom when the screen is in portrait orientation. In both cases the menu bar is displayed at the top of the screen.

The DICOM images of a series are rendered in a viewport by dragging and dropping the related thumbnail from the thumbnail bar to one of the four viewports. A viewport can be enlarged or shrinked with a pan movement in the layout mode (see menu). All the series included in a clinical case are presented in the thumbnail bar with a thumbnail of the first image and the name of each series. The thumbnail bar can be scrolled if it exceeds the size of the screen. In landscape orientation the bar is scrolled vertically, in portrait mode the scrolling is done horizontally. The Image Menu Bar provides the following functions which are self explaining: Return, Layout, Scroll, Zoom, Contrast, Info on/off.

7.7 Diagnosis

The diagnosis module presents the list of possible diagnoses with radio-buttons to let the user submit his own diagnosis. A popup window informs the user whether the submitted diagnosis is correct (Congratulations) or wrong. When the user closes the pop-up window he is redirected to the Answer module.



7.8 Answer

The tab Answer shows a grey or color image explaining the correct diagnoses with free text and pictures.

7.9 Results

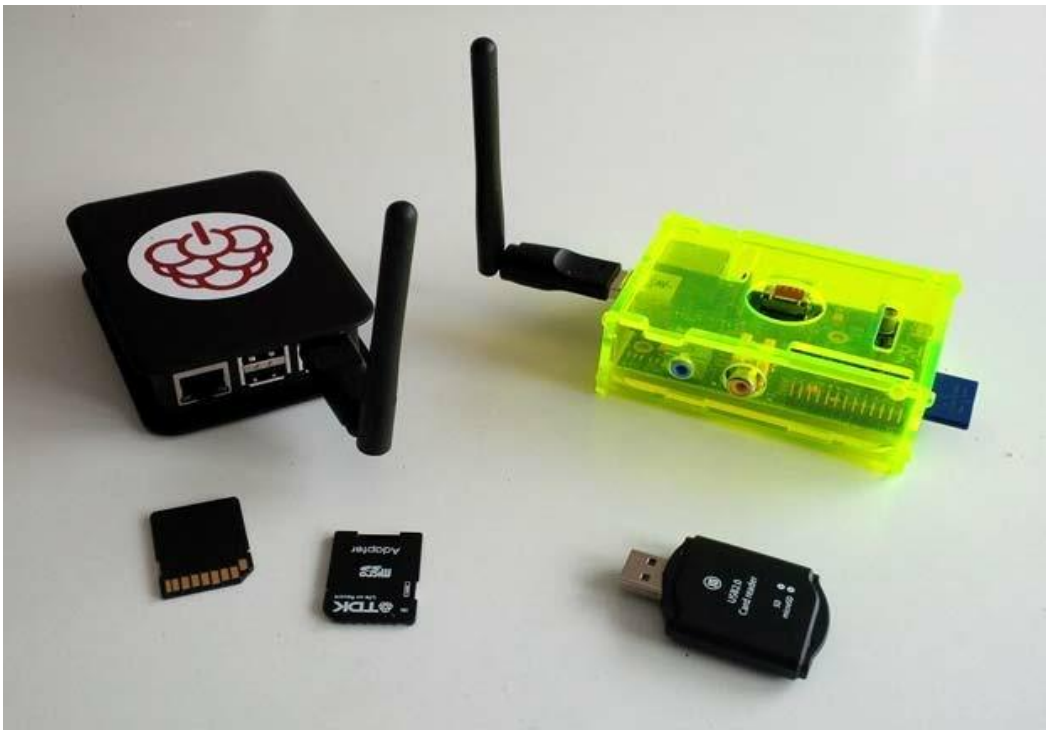
The progress and performance of the user are shown in graphical form. The number of completed, pending, correct and wrong sessions, the number of pending cases and the number of submitted, correct and wrong diagnoses are reported. If available, the user can send his results to an assessment server and he can compare his score with the global results of other (anonymous) users.

7.10 Help

The last module provides some hints and informations how to use the application. Sébastien Jodogne's Orthanc Book served as reference to create a RadioLogicBook for the Help module.

8. OrthancPi and OrthancMac

Early 2016, I ported the Orthanc source code to the Raspberry Pi, a credit-card–sized single-board computer. The software and the data are stored on an SD-card. OrthancPi was tested on the Raspberry models B Pi1, Pi2 and Pi3 with different WiFi adapters.
I preferred the model Pi3 for performance reasons (10 x the speed of model 1) and because a dual-band WiFi chip (Broadcom BCM43438) is integrated on the board. The same chip supports Bluetooth 4.0, which allowed the wireless connection of a keyboard and mouse during the development (see Orthanc Raspberry Pi on my blog web3.lu).

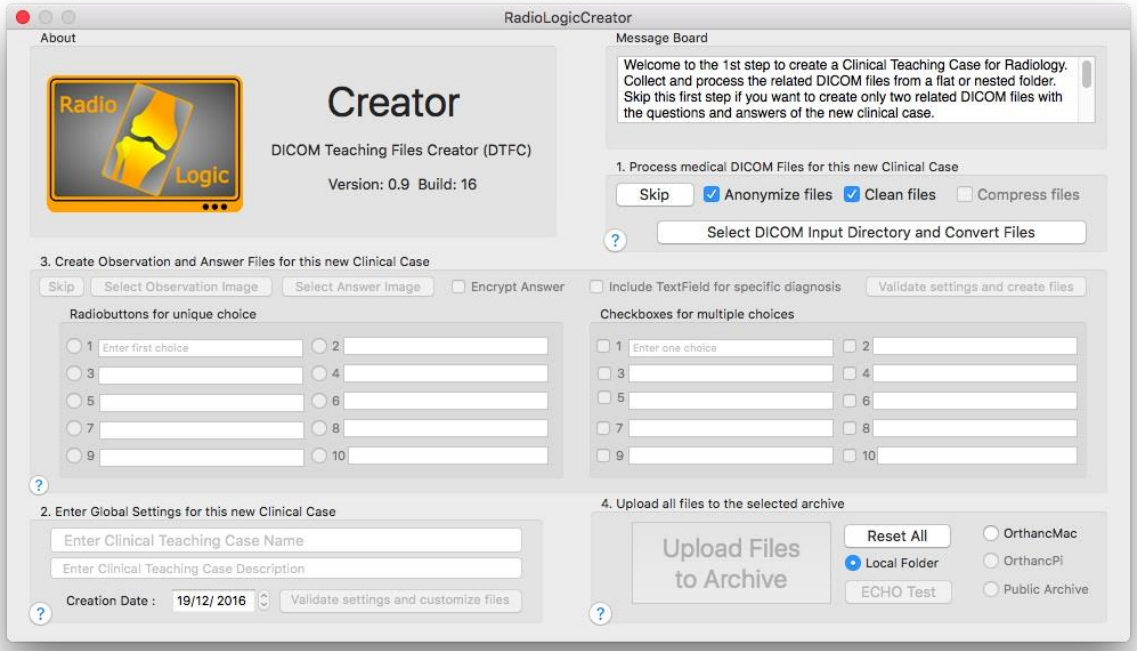


The idea was to use OrthancPi as clinical case archive in training rooms where Internet is not available. It was however not used in practice. I consider this project as a proof of concept. In parallel, I compiled the Orthanc source code on my MacBook to use Apple computers as DICOM archive serving the clinical cases (see OrthancMac OS X El Capitan on my blog web3.lu). Today upto-date Orthanc installers for OS X

are provided courtesy of Osimis, a spin-off from the CHU Liège, founded in September 2015, who acts now as commercial partner for the Orthanc ecosystem.

9. RadioLogicCreator on Mac

Assembling, processing and checking selected DICOM files manually with a half-dozen different tools to create clinical cases was cumbersome and not error-proof. Therefore I decided to develop a specific tool to automate the whole process. To comply to the original OFUR concept, this tool should run on a Apple Mac platform. As programming language I had the choice between C, C++, Objective-C or Swift 3. I never did C-related programming in the past. My first impression was that C is an ugly language. In Swift I found more similarities with Java, J2ME, PHP, Python and Javascript which I used extensively in the past. I was already a registered Apple developer and able to start the development of the RadioLogicCreator tool in mid-2016. A first stable version was available at the end of 2016. I made some code snippets available to other Swift developers (see MKWebView, readabilityHandler, fifoScrollDown, waitForWebAccessToLocalhost and twoTabViews on GitHub).



The tool created a clinical case in four steps. First the collection of all related DICOM file was done recursively from a folder. The files were anonymized, cleaned, compressed and scaled. In the second step the patient name was replaced with the name of the clinical case and a description was added. In the third step the

Observation and Answer pictures were imported, the possible diagnoses were entered and the radio-button for the correct diagnosis was selected. After the validation of all text entries two DICOM files were created, embedding the pictures as DICOM images and the text as private tags. In the last step the modified and new created DICOM files were uploaded to the DICOM archive.

From the beginning the RadioLogicCreator tool running on Mac was a continuous source of frustration for the users and the developer, for the following reasons:

- Due to the severe security restrictions of Apple, the deployment of the RadioLogicCreator application on a Mac was difficult, despite the fact that I was an identified developer and that the app was signed.
- At each change of the OS X, it was necessary to adapt the application to new rules. My OS X was Yosemite when I started with the development, followed by El Capitan (end 2015), Sierra (end 2016), High-Sierra (end 2017), Mojave (end 2018) and now Catalina.
- The processing of DICOM files in four steps was synchronous and it took some waiting time before the creation of a clinical case was finished.
- The main problem was the connection between the MAC computers running the RadioLogicCreator tool and the Orthanc server working as a DICOM archive with an integrated reversed nginx proxy. The computers were not dedicated for this purpose and setting up a temporary connection between them in a local network by non technical users most often failed.

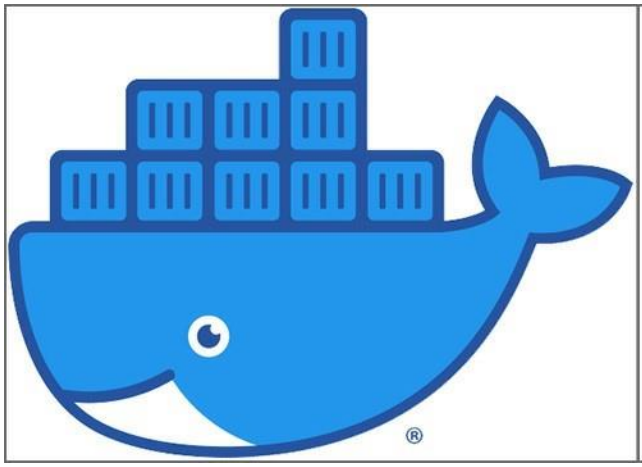
The last issue was not completely new to me. In 2015 I invested already a lot of work to guarantee reliable connections between the Orthanc server and the RadioLogicTutor in a local network operated without router (see my contributions Accessing Local Virtual Hosts and Mac OSX Wireless Networks on my blog web3.lu).

10. Improvements and enhancements : Docker

In 2017, I looked for solutions to tackle the network issues. One way was to improve the download of clinical cases from zipped archives. New HTML5 technologies like indexedDB made it possible to store large files locally in the browser. I forked Chris Hafey's cornerstoneImageLoader to read and decode DICOM files from a standard zipped DICOMDIR archive (see my GitHub repository cornerstoneArchiveImageLoader). A simple way to create such an archive is to download a DICOMDIR file from an Orthanc server.

A second way explored was the installation of Orthanc inside a Docker container. The launch of Docker in 2013 started a revolution in application development by democratizing software containers.

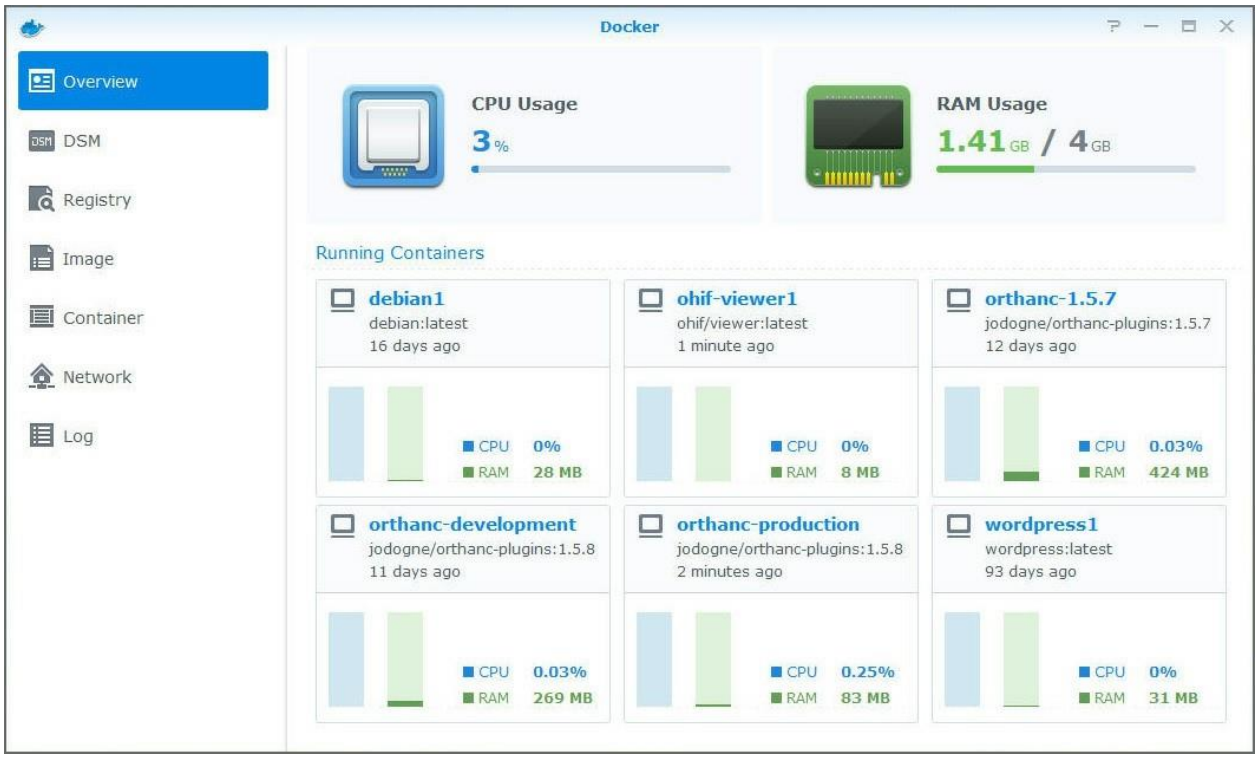
A first release of Docker for Mac was announced in July 2016. I started a first trial in March 2017 with the Docker Community Edition 17.03.0. I was impressed by the advantages of Docker to deploy and run software. In my eyes that was really a revolution for software engineers.



The third way was to replace the allround Mac computer, used from time to time as Orthanc server, by a dedicated hardware. A review of the requirements defined for OFUR in 2015 showed that "low costs" and "usage without Internet connection" were no longer considered as a priority. As I used a Synology diskstation since 2012 for my multimedia projects, I was in favor of using a dedicated NAS (Network-attached storage) as the RadioLogic DICOM archive based on Orthanc. When Synology announced the availablity of a natif Orthanc package with the DMS (Diskstation Manager Synology) version 6.2 beta in October 2017, I was convinced that this was the right way to go. I did some first trials on my existing diskstation (see my report with the first Orthanc package and found a bug in the configuration file which I communicated as an issue to Synology with the reference #1209135.

The first native Docker package (version 1.5.0-0027) was provided by Synology in May 2015. After doing some comparative performance tests in November 2017, I concluded that the optimal solution was to run Orthanc as a DICOM archive for clinical cases in an Orthanc container in a dedicated diskstation. I opted for a Synology DS918+ diskstation, with an Intel CPU and 4 GB

RAM. It was installed early 2018. Since that date, a swarm of Orthanc servers is running on this diskstation: for production (RadioLogicArchive), for development, for tests as a peer and so on. A very reliable and stable platform which works as expected on the Internet and in the local network.



11. The last mile

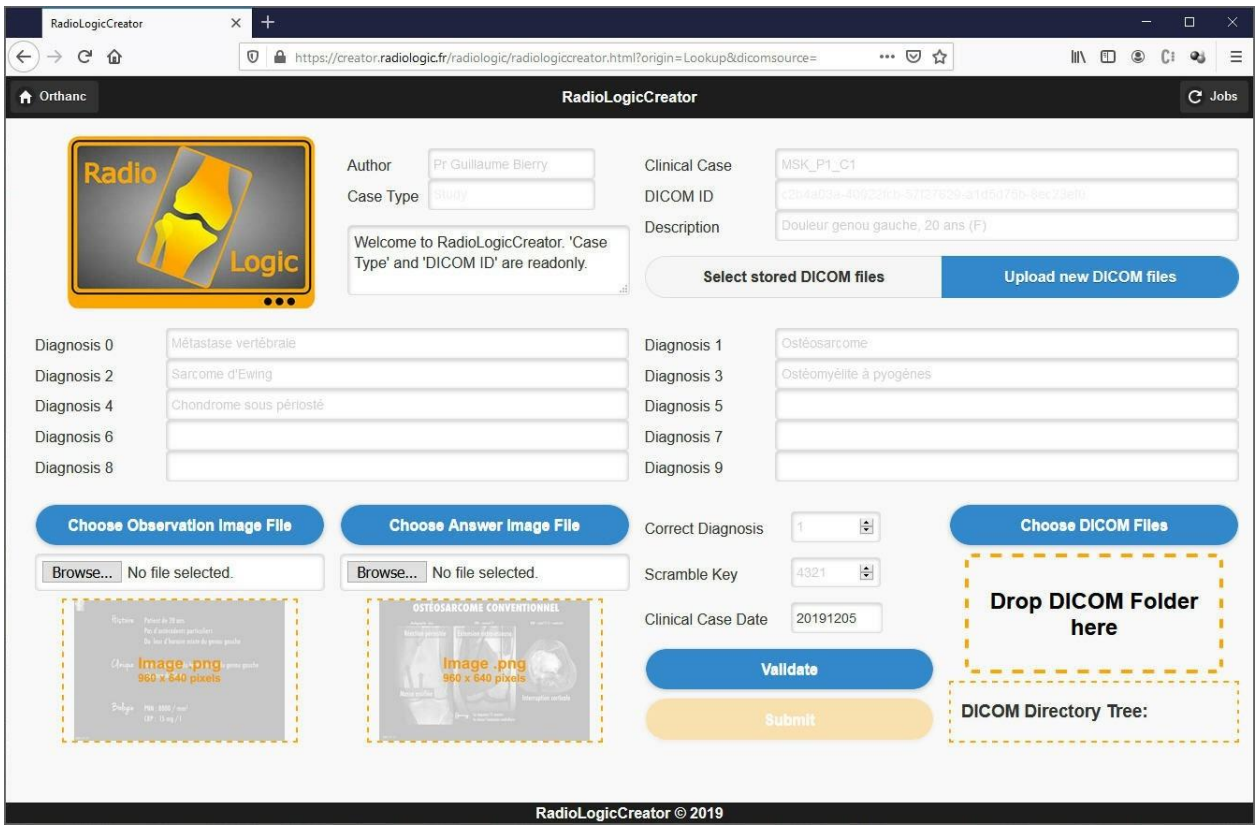
After RadioLogicTutor, RadioLogicArchive was the second component of the RadioLogic system which was ready for a real-life deployment. The last component, RadioLogicCreator, was still waiting for improvement. In July 2018, version 1.4.0 of Orthanc was released which included a new advanced job engine and new metadata options. When I looked at these features I was convinced that a plugin, using asynchronous jobs to create clinical cases, would solve all remaining problems, at least most of them. I was so enthusiastic that I started the same day learning the hated ugly C++ programming language.

To start the plugin development I set up a Docker container, based on a Debian image, im my diskstation. I installed all required tools and libraries to launch some programming tutorials and to start my new career as a C++ developer. My first plugin running in Orthanc was very simple. A callback was registered for a REST API returning a webpage with the message "Hi Orthanc" instead of the usual "Hello World". It took some time before the plugin was able to say "Hi" in concurrent threads. In May 2019 I asked for help concerning the job handling in the Orthanc forum.

Following the suggestion of Sébastien Jodogne to use the C++ wrapper and to derive a class from the Orthanc Job class I was able to finalize my plugin development in September this year. Time has now come to present the last member of the RadioLogic family: RadioLogicCreatorPlugin.

12. RadioLogicCreatorPlugin

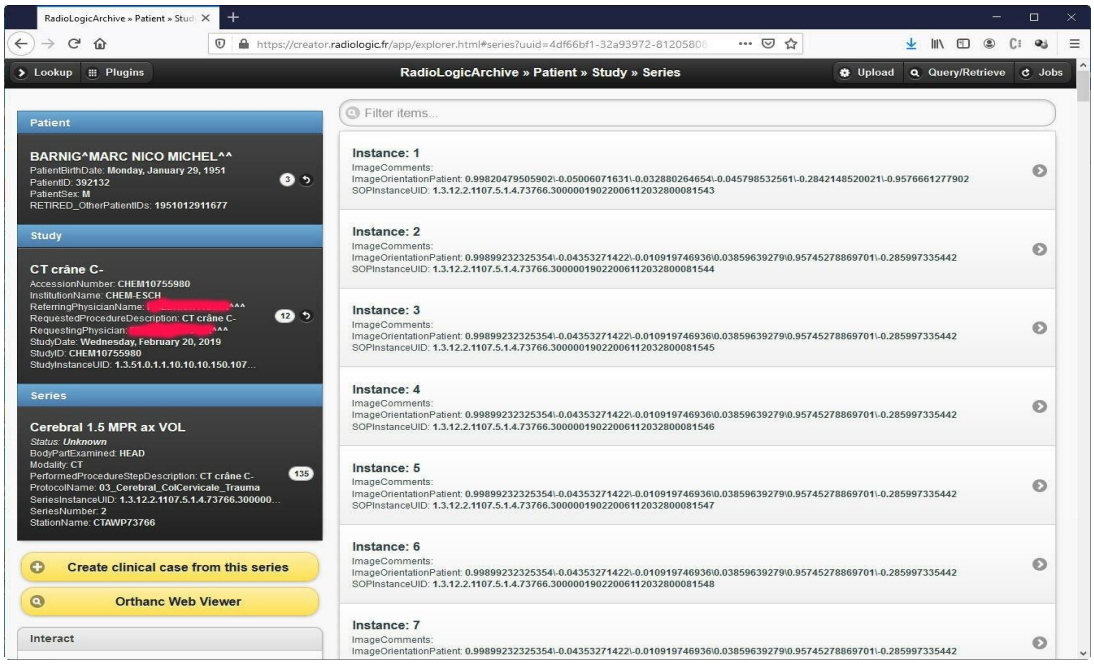
To introduce the plugin lets have a look at the user interface which is a HTML5/Javascript webpage in the colors and style of the Orthanc explorer.



There is a a close similarity with the RadioLogicCreator OS X tool. The main difference is that all data is entered in one step. By clicking the validation button, a quick check is done if all required informations have been provided. For example setting the name for the clinical case is mandatory, whereas the description for the case is optional. The validation function checks also if the entered data are relevant and plausible. When setting the correct diagnosis to 3 while the possible diagnosis field 3 is empty, an error is raised.

When the validation is successful and the enabled Submit button is clicked, the Observation and Answer images are uploaded to the server with the REST API "/tools/create-dicom" and the entered data is uploaded as JSON body to the server with the new initialized REST API "/ralo/startjob".

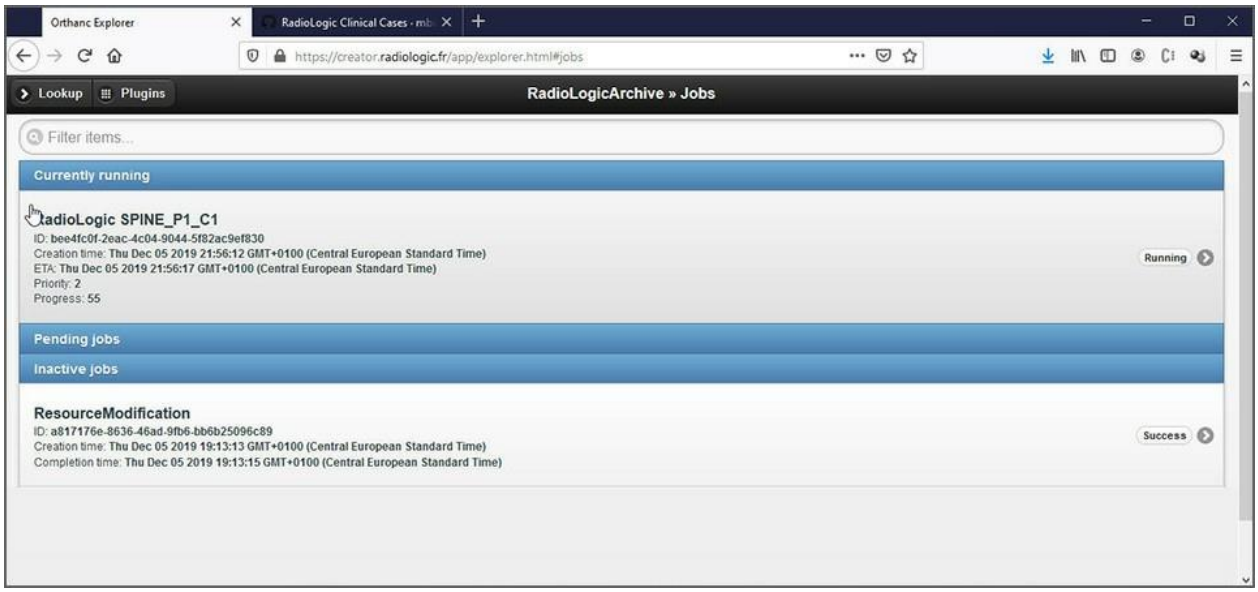
To select the DICOM files to be part of the new clinical case, the recommended procedure is to do it with the standard Orthanc interface. The Orthanc exlorer javascript code is extended to add buttons in the webpages related to the different levels of the DICOM data model. A clinical case can be created with all instances associated to a patient, a study or a series, respectively with a single instance.



A second possibility is to upload the DICOM file from an external source. A recursive upload of files from a nested folder can be done by dragging and dropping the folder to the "drop zone" in the interface webpage.

A second difference between the old and new RadioLogicCreator interface is the removal of all checkboxes and radiobuttons to simplify the user interface. One new radiobutton has been added to switch between the source of the DICOM instances (stored or uploaded files).

When the HTTP POST request to start a job is successful, the user is redirected to the Orthanc jobs webpage to view the running job. The job can be paused or canceled. The user must not wait the completion of the job before creating an additional clinical case or launching another tasks, because jobs are executed asynchronously.



If we view the list of patients, we see that the original DICOM instances are unchanged. A new patient is created with the name specified for the clinical case and with a unique PatientID, prefixed with the term "radiologic-".

13. Plugin functions

The program code of the RadioLogicCreator Orthanc plugin is a mix of the following elements:

- Orthanc SDK function (from the modules Toolbox, Orthanc, Callback and REST)
- members of the Orthanc C++ wrapper class
- specific C++ functions

13.1 Function CallbackStartJob()

The CallbackStartJob() function is called with the registered REST API "/ralo/startjob". The JSON body embedded in the HTTP request is parsed and the included parameters are extracted. A new job is created from the class RadioLogicCreator. The member variable counter of the job is set to 0. The job is initialized with the other received parameters and submitted to the Orthanc job-engine. This engine makes a first call to the function RadioLogicCreator::Step().

13.2 Function RadioLogicCreator::Step()

The Step() function works in the following three states by checking and incrementing the value of the member variable counter:

- counter == 0 : the member variable maxSteps is set to the number of instances to process. The anonymization and customization of the instances is prepared and the function CustomizeClinicalCaseInstances() is called a first time. The job information content is initialized. The counter is incremented and the Step() function returns with "continue".
- counter > 0 and < maxSteps : the function CustomizeClinicalCaseInstances() is called at each step by passing the related instanceId. The job content is regularly updated. If the CustomizeClinicalCaseInstances() returns an error, the Step() function returns with "stop" (failure), otherwise with "continue".
- counter == maxSteps : the function Customize-Clinica-lCase-Instances() is called a last time, followed by calling the functions Customize-Clinical-Case-Observation-Instance() and Customize-Clinical-Case-Answer-Instance(). If everything works fine, the Step() function returns with "success".

13.3 Function CustomizeClinicalCaseInstances()

The CustomizeClinicalCaseInstances() function generates new unique identifiers based on the RadioLogic DICOM UID prefix for the studies, series and SOP UID's for each instance. Then the REST API "/instances/[resourceID]/anonymize" is called to delete, replace and keep specific DICOM tags. The transferSyntax of the file is retrieved to check if the image is compressed. If not, the returned processed DICOM file is saved into a temporary folder and the function CompressAndScaleDicomImage() is called. If yes, the processed file is stored to the Orthanc server with new name, ID's and UID's, by using the SDK function OrthancPluginRestApiPost().

13.4 Function CompressAndScaleDicomImage()

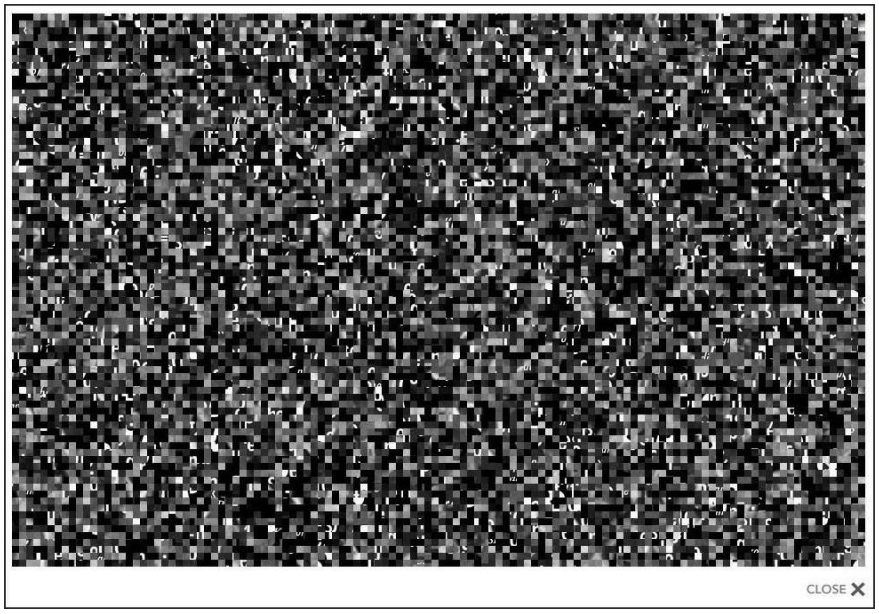
The CompressAndScaleDicomImage() function does what its name says. If the width or height of the image exceeds 1024 pixels, the image is scaled with a system call to the Offis DCMTK tool "dcmscale". Next the image is lossless compressed with a system call to Mathieu Malaterre's GDCM tool "gdcmconv --j2k" (JPEG2000 format). The compressed file is then stored to the Orthanc server and the CompressAndScaleDicomImage() function returns "true" if no error happened.

13.5 Function CustomizeClinicalCaseObservationInstance()

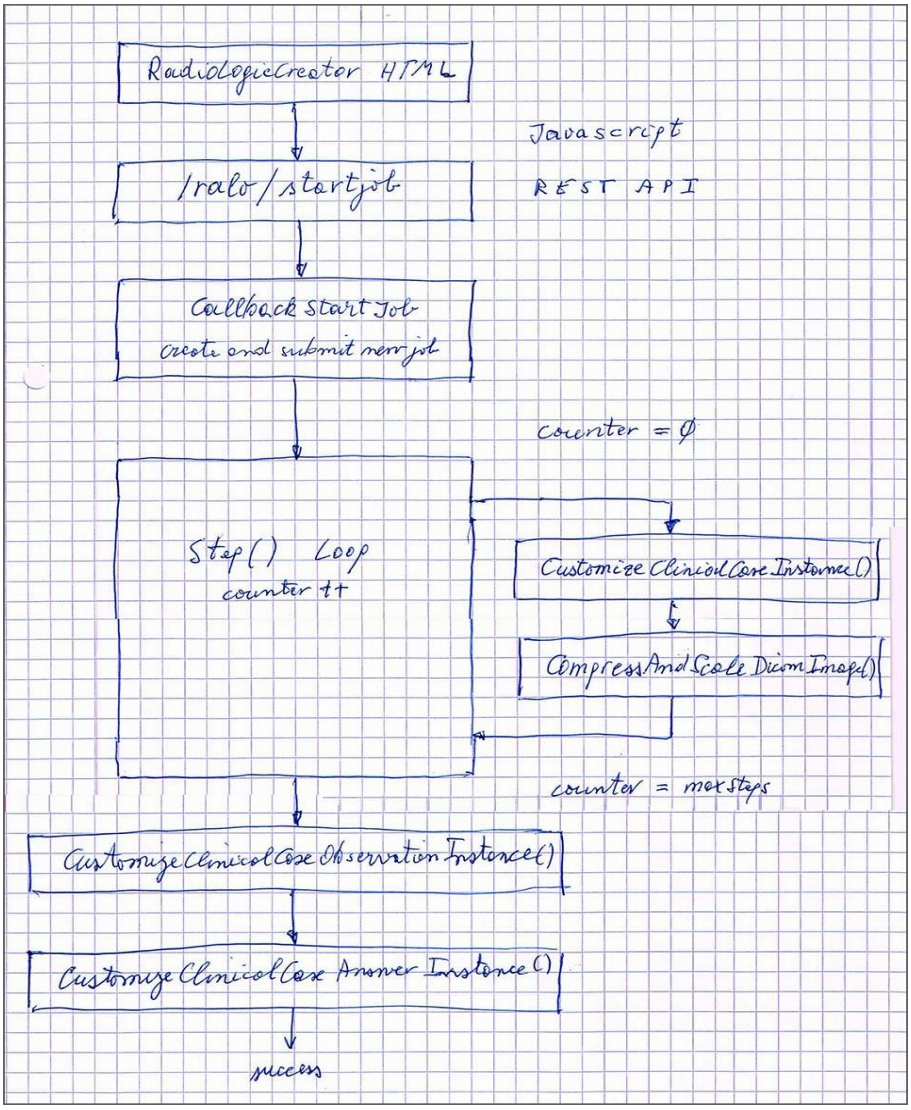
The task of the function CustomizeClinicalCaseObservationInstance() consist in adding the possible diagnoses as private tags into the stored DICOM file including the observation picture. Due to issue 140 with Orthanc core that I created in May 2019, it is not possible to insert or modify private tags inside the Orthanc REST API's "/tools/create-dicom" or "/instances/{resourceID}/modify". As a workaround I use a DICOM template with empty private tags, replace the pixels in the template with the data of the observation-image-file and modify this image with a system call to the Offis DCMTK tool "dcmodify". The process ends by storing the modified template as a new observation DICOM instance to the Orthanc server and deleting the original observation DICOM instance. The whole procedure is rather complex and needs writing and reading files in a temporary folder, which are removed before the function returns.

13.6 Function CustomizeClinicalCaseAnswerInstance()

The function CustomizeClinicalCaseAnswerInstance() is similar to the function described above and has the same complexity due to issue 140. The answer image has been scrambled with javascript before the upload to the server. This prevents that a clever user inspects the image preview on Orthanc to see the results with the correct diagnosis before submitting his own diagnosis in the RadioLogicTutor. The correct diagnosis is also written in scrambled text into a private DICOM tag. Other private tags are used to store the name of the clinical case author and the scramble key which is required by the RadioLogicTutor application to decode the image and the answer text. The whole scrambling process is not hacker proof, but I think that encrypting the answers with public-key cryptography would be design-overkill.



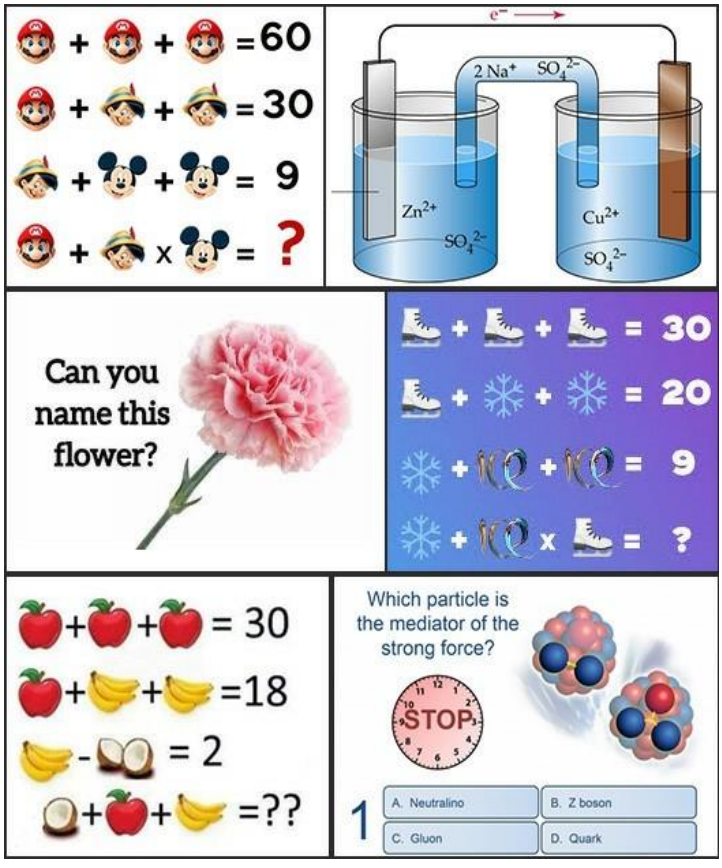
The following scheme shows the simplified workflow in graphical form.



The complete source-code is available in my public GitHub repository RadioLogic. In the Wiki of this repository you will find a short tutorial how to install and operate an Orthanc server in a Docker container and how to start with the development of some simple Orthanc plugins, including plugins using the Job engine.

14. Learning with quizzes

The Orthanc ecosystem and the presented RadioLogic tools are not limited to store, to view and to assess knowledge to interpret medical images. Technically we can use pictures presenting any real-world problem with a related question and use DICOM tags, or Orthanc metadata, to store the correct answer. A simple example is an e-learning quizz. Such quizzes are widespread on the Internet and related to all sort of topics : IQ tests, STEM (science, technology, engineering, mathematics) education, general knowledge, trivia or even fun. The answer types of these quizzes could be multiple-choice, true-false or open-ended.



Whether it is ethical to use medical software for other purposes than health is a justified question. A discussion thread launched early November 2019 in the Orthanc user forum concerning this topic got only a few answers. The general opinion is that derived use of medical software for teaching purposes is a highly ethical use case.

15. Conclusions

I close my presentation with a few personal conclusions:

- A project which evolved from a minimum viable project in 2012 to an outstanding open-source ecosystem and gave birth to a university spin-off, which occupies now more than 12 qualified people, is a great success story. Congratulations to Sébastien Jodogne for creating Orthanc and co-founding Osimis.
- A large project related to advanced technologies takes three times more resources and time than your initial estimation, due to the appearance of non-expected issues.
- The interest in new technologies can turn into addiction