Universidade de Aveiro
2013
Departamento de Electrónica, Telecomunicações e Informática

António Mário Constantino Malta Novo

DicoogleWeb: uma interface Web para repositório de imagem médica

DicoogleWeb: a Web interface for a medical image repository

Universidade de Aveiro
2013
Departamento de Electrónica, Telecomunicações e Informática

António Mário Constantino Malta Novo

DicoogleWeb: uma interface Web para repositório de imagem médica

DicoogleWeb: a Web interface for a medical image repository

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Dr. Carlos Manuel Azevedo Costa, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri

presidente

Prof. Dr. António Manuel Melo de Sousa Pereira
professor Catedrático da Universidade de Aveiro

Prof. Dr. Rui Pedro Sanches de Castro Lopes
professor Coordenador do Dep. Informatica e Comunicações da ESTG do Instituto Politécnico de Bragança

Prof. Dr. Carlos Manuel Azevedo Costa
professor Auxiliar da Universidade de Aveiro

agradecimentos

Gostaria de agradecer à minha família por me apoiar constantemente.
Agradeço também a todo o grupo de bioinformática, em especial ao Luís Bastião, Carlos Ferreira e Frederico Valente por toda a ajuda e suporte.
O meu obrigado segue também para o professor Carlos Costa pela ajuda e paciência que teve para comigo.

acknowledgements

I would like to thank my family for the continuously support me.
I would also like to thank the whole bioinformatics group, especially to Luís Bastião, Carlos Ferreira and Frederico Valente for all the help and support.
A big thank you note goes also to professor Carlos Costa for all the help and patience towards me.

palavras-chave

PACS; DICOM; Telemedicina; Imagem Médica; Web.

resumo

O Dicoogle é uma solução de software, em código fonte aberto, que foi desenhada para suportar o fluxo de informação num laboratório de imagem médica. Além disso, está dotado de um mecanismo de indexação que permite indexar todos os metadados contidos nos ficheiros DICOM do seu repositório. A actual implementação pode ser lançada em modo servidor (por omissão) mas também dispõem de um módulo gráfico cliente que pode conectar-se a qualquer instância servidor através de Java RMI. Isto proporciona um acesso simples a estações de trabalho clientes dentro ou fora de uma organização medica.

Esta dissertação propõe e implementa uma solução que permite migrar o Dicoogle para ambiente Web. Para além de disponibilizar todas as funcionalidades da versão anterior, a versão Web oferece um conjunto de novos serviços e interface de acesso aos dados.

keywords PACS; DICOM; Telemedicine; Medical Imaging; Web

abstract Dicoogle is an open-source software solution designed to manage the information workflow in a PACS as well as the archiving and indexing process of the arriving DICOM files. The current implementation can either be run in server mode (default) or used as a client to connect to another server instance of Dicoogle (via Java RMI). This enables simple access to client workstation within or outside a medical organization.
This work will focus on adapting the current implementation of Dicoogle for the Web environment, allowing, system clients, to view medical images, on the majority of devices with network access.

Table of Contents

1 Introduction

Nowadays, with advancements in medical imagery equipments devices and formats, there is a growing demand for more and more storage and archiving solutions. Not too long ago, patients and doctors had to maintain hard-copies of medical imagery exams, like X-Ray radiograph sheets or an ECG paper log. In this era of digital devices and low expense digital storage space, these imagery exams, as-well as patient information, can be easily stored digitally (soft-copies) with smaller costs than maintaining hard-copies of it. And since this data is stored digitally, there is also the possibility of making it available to any network device with access to the data storage device. This also creates a simpler environment where doctors can access this information faster and with greater ease.

In the Bioinformatics group at IEETA, it was developed a medical imaging achieve with search engine and distributed computing techniques, named Dicoogle [1]. It is an open-source software solution designed to manage the information workflow in a PACS as well as the archiving and indexing process of the arriving DICOM files. The core part of application relies on a client-server architecture, that can run in server mode and also as a client to connect to another server instance of Dicoogle (via Java RMI).

With this dissertation, we plan to provide a web solution for the current Dicoogle implementation. The target implementation must allow web clients, at least within the same medical institution, to be able to retrieve, view and analyze the information stored within a DICOM file, as well as searching the PACS. Effectively removing the need for 3rd party external DICOM viewer installations on the doctors' workstations, and create the possibility to extend the functionality through the web-based components, which may be useful to other Dicoogle developers.

Being a web based scenario, various technologies and implementation options related to the web environment, that will help accomplish the above, will be discussed throughout this paper. Taking into account the current state of the project, there will also be some improvements that can, later, be added to the new solution.

2 State of the Art

In this chapter, some attention will be put towards the current scenarios and technologies currently in use in the medical environment, Dicoogle and the web environment, as well as some new technologies that might aid the development of the new solution.

2.1 Medical Environment

With current medical imaging techniques and records shifting to the digital era, new technologies and standards were developed to aid in the storage and later-on analysis of medical exams. The next point in this paper will focus on these technologies.

2.1.1 PACS

A PACS (Picture Archiving and Communication System) provides a "simple" yet powerful digital storage and access model for medical imaging information, removing the need to manually retrieving and archiving said data [2].

This architecture relies on a secure network for transferring the images and information, and is based on four major components:

- Medical imaging devices: these are used to create images (of parts) of (organs and tissues of) the human body for clinical purposes. Among these are MIR machines, X-Rays machines, and many others;
- A QA (Quality Assurance) workstation or gateway: this component of the network manages the consistency and quality of the (image) data originated from the imaging devices;
- An archive: used for storing the medical information;
- A set of doctors workstations: these workstations will have access to the archive for retrieving the information, and are used by the doctors to analyze the previously mentioned retrieved information;

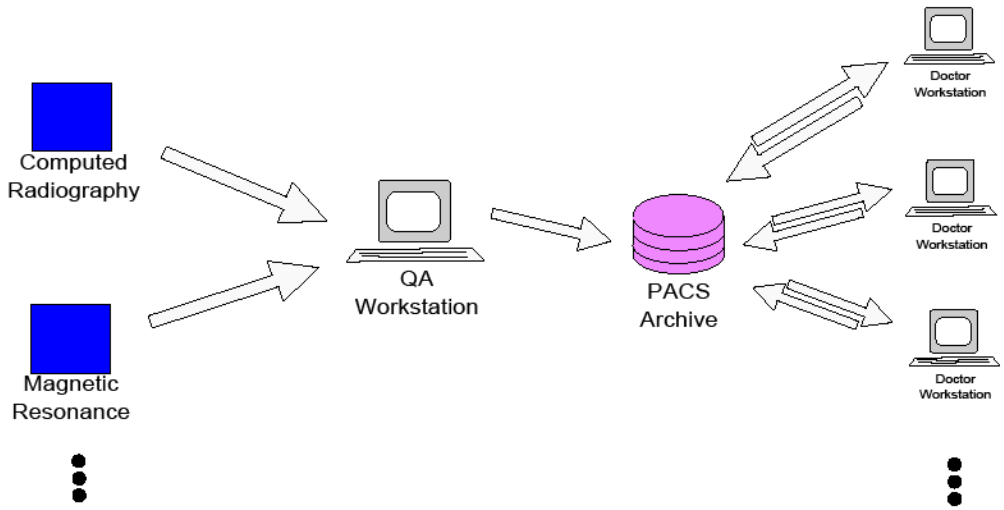


Figure 1: PACS components and set of data transfers between them.

All these components/devices, depicted in figure 1, communicate with one another using a standard protocol called DICOM (Digital Imaging and Communications in Medicine) [3] that dictates the image storage and transfer format. The doctor workstations can use a set of DICOM (Digital Imaging and Communications in Medicine) messages to query, retrieve and also "forward" information from the PACS archive.

All this guarantees that, in very little time and with greater efficiency, the information obtained with the medical imaging devices will be available for medical professionals to retrieve and analyze.

2.1.2 DICOM

The DICOM standard [3] defines how to handle, transmit and store medical imaging information. It defines both a TCP/IP network messaging protocol and a digital file/data format. Firstly released in 1985, by NEMA (National Electrical Manufacturers Association), it is an evolving standard, and its currently in version, 2011, is defined in 20 independent and freely available parts/documents from PS 3.1 to PS 3.20 [3].

Since this file/data format aims to represent real-world data, it represents the data using IODs (information object definitions) with properties and attributes. These IODs are used to store/represent, for example, the patient, with name, sex, and many other attributes. There are several IODs within the same DICOM file, each one representing a different real-world entity:

Tag ID	Tag Name
0008,0080	Institution Name
0010,0010	Patient Name
0010,1010	Patient Age
0010,0030	Patient Birth Date
0010,0040	Patient Sex
0018,5010	Patient Position
0021,1091	Biopsy Position
...	...

Table 1: List of some of the IODs available.

Each IOD contains a list of child properties and values pertaining to the parent IOD. And since there is a multitude of real-world data types (like millimeters, date and others) the DICOM format uses a Data Element encoding scheme with Value Representations accordingly to the data/value the element holds:

Value Representation	Description
AE	Application Entity
AS	Age String
AT	Attribute Tag
CS	Code String
DA	Date
DS	Decimal String
DT	Date/Time
FL	Floating Point Single (4 bytes)
FD	Floating Point Double (8 bytes)
IS	Integer String
LO	Long String
LT	Long Text
OB	Other Byte
OF	Other Float
OW	Other Word
PN	Person Name
SH	Short String
SL	Signed Long
SQ	Sequence of Items
SS	Signed Short
ST	Short Text
TM	Time
UI	Unique Identifier
UL	Unsigned Long
UN	Unknown
US	Unsigned Short
UT	Unlimited Text

Table 2: List if all the DICOM Value Representations and their description, taken from Chapter 6.2 of DICOM document PS 3.5 [3].

For instance, the Series IOD lists the property Modality Type, which indicates the application area (or type of radiology imaging system) that produced the image(s)/data archived within the file.

DICOM files can also contain an IOD with pixel information referring to one or more images.

SOP	SOP Class UID SOP Instance UID
Patient	Patient Name Patient ID Patient Birth Date Patient Gender
Study	Study UID Study Date Study Time Study ID Referring Physician Accession Number
Series	Series UID Series Number Modality Type
Equipment	Manufacturer Institution Name
Image	Acquisition Attributes… Position Attributes… Image Number Image Type Rows Columns Samples-per-Pixel Planar Configuration Pixel Representation Pixel Data

Table 3: This is the structure of an example DICOM file that would contain an image.

The data, depicted in table 3, appears in the same order as it would appear in the DICOM file, meaning that a Patient is subject of a Study, which contains a Series, which used the following Equipment to produce the following Image.

These files can hold very large image sets, and on large medical institutions, the number of DICOM files, need to be sent for storage, can grow very quickly. This creates a problem, since all the information is stored inside the file. When a workstation, or any other PACS client, wants to retrieve that file it has to know the name of the file, that contains the information needed, beforehand. One of the solutions is indexing all the properties, and their values, of a DICOM file along with its file name on a database. This will ensure faster search and retrieval of the archive contents.

2.1.2.1 WADO

The DICOM specifications, at document PS 3.18 [3], already has defined a system for querying and retrieving DICOM information via Web, it is called WADO (Web Access to DICOM Persistent Objects).

It is a standard that dictates how a web enabled PACS server to should process web clients' requests for information and which data and format to return to the client. Based on HTTP, it provides a standard URI format to be used to query the server for specific DICOM data.

The WADO specification is rather minimalistic, since it provides only a standard set of guidelines on how to provide the DICOM information that resides on the PACS to requesting web clients (URI format and response MIME type support). This DICOM information can be accessed by clients that know, beforehand, the identification numbers (StudyUID, SOPInstanceUID, and others) of each requested DICOM file, leaving the searching capability within the PACS out of scope.

2.2 Web

The World Wide Web (WWW, W3 or Web for short), is, without doubt, the most used service across the Internet. By providing easy access to visual, audio and textual information stored within interlinked documents, any Internet user can have access to the billions of public documents stored within the remote servers that build the Internet.

The Web is free to both be accessed or to supply content by anyone, as opposed to its previous predecessor the Gopher protocol [4], which is, since 1993 when the Web was declared free by CERN, deprecated.

The WWW consists in the symbiosis between three systems:

- URI/URL: the Uniform Resource Locator and Uniform Resource Identifier are a set of globally unique identifiers for each resources on the Web;
- HTML: the HyperText Markup Language is the publishing language of most Web documents;
- HTTP: the Hypertext Transfer Protocol is the protocol that allows clients and servers to interact with each other, exchanging information and documents;

Since the access to each server is dictated by which IP address, or translated DNS record, is visited, the usage of URLs/URIs on the Web allows each web client to either point directly to the needed document or to simply follow the set of links on that server documents that lead to the pertaining document.

2.2.1 HTTP

The Hyper Text Transfer Protocol (HTTP) standard was developed by the Internet Engineering Task Force (IETF) in cooperation with the World Wide Web Consortium (W3C). After some improvements and updates to the standard, version 1.1 of HTTP was now defined by RFC 2616 [5], as of in June 1999.

As its name indicates, the main purpose of the HTTP protocol was to easily exchange textual information (in ASCII format), but it can also handle the exchange of any binary data type.

It defines a set of message types and response codes in order to exchange data between client and server, in a web environment. It also relies on an underlying TCP connection for exchanging these messages and resulting data.

Two message types are defined: request and response. The request message, which is sent from the client to the server, carries some headers indicating what resource the

client is requesting from the server. The server after receiving receives the request, checks the existence of the requested resource and sends a response message back to the client.

The request message will carry, besides the URI and parameters of the resource to access/request, but also which action or method should the server perform to said resource. This method can have one of the following values:

Method	**Meaning**
GET	Retrieve whatever information is identified by the Request-URI and returns its information and contents.
HEAD	Identical to GET except that the server MUST NOT return a message-body in the response.
POST	Used to request the origin server to accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI.
PUT	Requests that the enclosed entity be stored under the supplied Request-URI.
DELETE	Requests that the origin server delete the resource identified by the Request-URI.
OPTIONS	Request for information about the communication options available on the request/response chain identified by the Request-URI.
TRACE	Used to invoke a remote, application-layer loop-back of the request message.

Table 4: List of HTTP request methods and their meaning.

The response message, besides carrying the requested resources' data and information, if found, will also carry a response code header indicating the successfulness of the request parsing and response created. Here are the values this response code header can have, and their meaning accordingly to their RFC 2616 documentation:

Group	**Code**	**Title**	**Meaning**
Informational	100	Continue	The client SHOULD continue with its request.
	101	Switching Protocols	Indicates a provisional response, consisting only of the Status-Line and optional headers.
Successful	200	OK	Indicates that the client's request was successfully received, understood, and accepted.
	201	Created	The request has been fulfilled and resulted in a new resource being created.

	202	Accepted	The request has been accepted for processing, but the processing has not been completed.
	203	Non-Authoritative Information	The returned meta-information in the entity-header is not the definitive set as available from the origin server, but is gathered from a local copy.
	204	No Content	The server has fulfilled the request but does not need to return an entity-body.
	205	Reset Content	The server has fulfilled the request and the user agent SHOULD reset the document view which caused the request to be sent.
	206	Partial Content	The server has fulfilled the partial GET request for the resource.
Redirection	300	Multiple Choices	The requested resource corresponds to any one of a set of representations.
	301	Moved Permanently	The requested resource has been assigned a new permanent URI.
	302	Found	The requested resource resides temporarily under a different URI.
	303	See Other	The response to the request can be found under a different URI.
	304	Not Modified	The requested document has not been modified.
	305	Use Proxy	The requested resource must be accessed through the proxy given by the Location field
	307	Temporary Redirect	The requested resource resides temporarily under a different URI.
Client Error	400	Bad Request	The request could not be understood by the server due to malformed syntax.
	401	Unauthorized	Requires user authentication.
	402	Payment Required	Code reserved for future use.
	403	Forbidden	The server understood the request, but is refusing to fulfill it.
	404	Not Found	The server has not found anything matching the Request-URI.
	405	Method Not Allowed	The method specified in the Request-Line is not allowed for the resource identified by the Request-URI.
	406	Not Acceptable	The resource identified by the Request-URI is unable to accept the headers sent in the request.
	407	Proxy Authentication Required	Indicates that the client must first authenticate itself with the proxy.

	408	Request Timeout	The client did not produce a request within the time that the server was prepared to wait.
	409	Conflict	The request could not be completed due to a conflict with the current state of the resource.
	410	Gone	The requested resource is no longer available at the server and no forwarding address is known.
	411	Length Required	The server refuses to accept the request without a defined Content-Length.
	412	Precondition Failed	The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.
	413	Request Entity Too Large	The server is refusing to process a request because the request entity is larger than the server is willing or able to process.
	414	Request-URI Too Long	The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.
	415	Unsupported Media Type	The entity of the request is in a format not supported by the requested resource.
	416	Requested Range Not Satisfiable	The request included a Range request-header field, and none of the range-specifier values in this field overlap the current extent of the selected resource.
	417	Expectation Failed	The expectation given in an Expect request-header field could not be met by this server.
Server Error	500	Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.
	501	Not Implemented	The server does not support the functionality required to fulfill the request.
	502	Bad Gateway	The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.
	503	Service Unavailable	The server is currently unable to handle the request.
	504	Gateway Timeout	The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI.
	505	HTTP Version Not Supported	The server does not support, or refuses to support, the HTTP protocol version that was used in the request message

Table 5: List of HTTP response codes and their meaning.

The response code value is then check by the client, upon receiving the response packet(s), to check if additional requests must be made in order to retrieve the resource (for instance, if the resource has moved to another location a new request will be made to retrieve the resource from said location).

Here is an example communication between a HTTP client and server, after a TCP connection has been open:

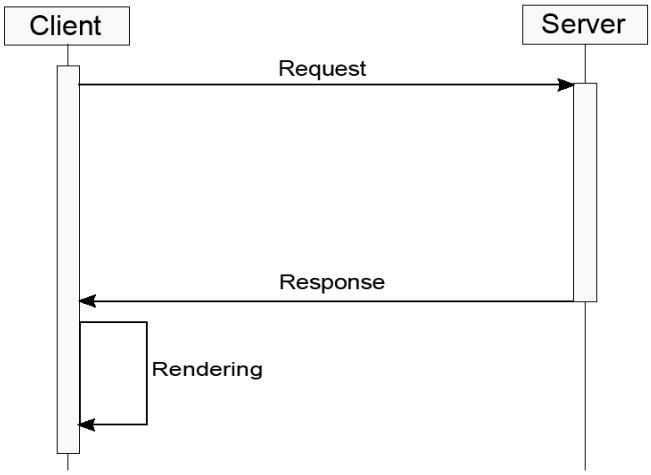


Figure 2: Example communication between a HTTP client and server.

An example of Request and Response packets can be the ones bellow:

Request	Response
GET /index.html HTTP/1.1 Host: www.example.com	HTTP/1.1 200 OK Date: Wed, 05 Jun 2013 00:00:01 GMT Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux) Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT Content-Type: text/html; charset=UTF-8 Content-Length: 131 Connection: close <html> <head> <title>An Example Page</title> </head> <body> Hello World, this is a very simple HTML document. </body> </html>

Table 6: Example request and resulting response HTTP packet data.

Despite being a simple protocol definition, and because it is build on-top of the state-full TCP protocol, it provides an excellent and error free way of transferring data on an otherwise "state-less" network like the internet.

Given the simplicity of the HTTP protocol, new architecture specifications emerged, that aimed to provide more functionality, and with a larger feature set and more semantic information, to the HTTP protocol.

One of these technologies is ReST (Representational State Transfer), and by relying on the underlying HTTP protocol, aims to provides better and easier resource representation and access. It is still client-server based, like HTTP, but uses a set of guidelines in order to guarantee that clients can access server resources and in a more streamlined manner while also being able to perform actions, traditional HTTP methods/actions, over said resources. Solutions that adopt the ReST architecture style can be easily extendable and accessible by other ReST systems. This is mainly due to the URL specification that guarantees easy system interaction and better scalability by adoption a more generalist interface set.

Table 7 presents an example communication between a client and server, where the client requests the number of devices on the server, and then removes the ability to “edit” of the first device:

Request	**Response**
GET URL: /device/	OK Body: /device/1/ /device/2/ /device/2/
GET URL: /device/1/capabilities/	OK Body: /device/1/capabilities/print/ /device/1/capabilities/edit/
DELETE URL: /device/1/capabilities/edit/	OK

Table 7: Example ReST communication between a client and server.

SOAP (Simple Object Access Protocol) is another technology that aims to standardize the exchanging of information on HTTP based systems. By relying less on the URL structure and using the HTTP request and response body to exchange data and actions between server and client, SOAP can provide a broader feature set than ReST or solely

HTTP can, while still maintain consistency among implementing systems. SOAP messages use XML structured documents, on HTTP bodies, that carry a more verbose and data set than other types of message passing. This can be a benefit when large sets of data with many meaning must be transmitted from one system to another (be it client or server). But can also be a drawback due to the extra amount of information that ship with each request.

Table 8 depicts an example communication between client and server where the clients requests the stock price of Google stocks:

Request

```
POST
URL:

        /InStock/

Body:

        <?xml version="1.0"?>
        <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
        soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
        <soap:Body xmlns:m="http://www.example.org/stock">
          <m:GetStockPrice>
            <m:StockName>GOOG</m:StockName>
          </m:GetStockPrice>
        </soap:Body>
        </soap:Envelope>
```

Response

```
OK
Body:

        <?xml version="1.0"?>
        <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
        soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
        <soap:Body xmlns:m="http://www.example.org/stock">
        <m:GetStockPriceResponse>
          <m:Price>915.43</m:Price>
        </m:GetStockPriceResponse>
        </soap:Body>
        </soap:Envelope>
```

Table 8: Example SOAP communication between a client and server.

These technologies will prove useful during the development of the solution presented in this paper, by allowing a standardized set of web-services, completely GUI (Graphical User Interface) agnostic, to be built.

2.2.2 Web Standards

One of the most relevant aspects of a web based solution is its compliance with the open web standards. The compliance with these well defined standards ensures compatibility across a whole multitude of very distinct system architectures that might want to retrieve and consume data from the server.

These royalty free standards are regulated by the World Wide Web Consortium (W3C), and are well defined, in order to guarantee that all platforms can comply and support the standards, while still improving functionality on a web based environment.

Nowadays, one of the main compatibility issues lies on how web browser applications parse and render web pages. Not all browsers follow the standards norm, one of these examples is Microsoft Internet Explorer 6, which uses custom (closed, and non standard) rules for both parsing and rendering web content. Now loosing usage share [6], it used to be, up to a few years ago, the most used internet browser. This forced web developers to target this browser specifically, when developing and deploying web enabled applications.

Newer browsers have as a major goal, the support for open web standards, even Microsofts' new implementations, removing possible implementation compatibility issues from open web standards compliant solutions.

2.2.2.1 HTML5

HyperText Markup Language (HTML) is, as the name implies, a document markup language, and the main document descriptor for web environments. It defines a set of tags, and tag properties, which can be used to create well structured documents that can later on be rendered by a web browser. HTML is also compliant with the SGML (Standard Generalized Markup Language) ISO standard [7], which means that both document and tag validity checks can be performed to each HTML document.

Its initial development started at CERN, in the late 1980's, due to the researchers' necessity of sharing and using documents. It became a standard only in November 24, 1995, with the release of the HTML version 2.0 definitions.

Throughout the years, new versions of HTML, mainly with improvements and fixes to the standard, continued to be launched. However, version 4.01 of HTML, published in December 1999 [8], remained the standard HTML version ever since then.

HTML version 5 is the most recent version of the HTML standard, and as of December 12, 2012, more than a decade since the last version, a W3C Candidate Recommendation. And although not all web-browsers fully support its' directives, the major distributions like Mozilla Firefox, Google Chrome, Microsoft Internet Explorer, Opera and Apple Safari do. Most recent mobile device browsers also support this version, and with continuing mobile device technologies improving, mobile device support will continue to grow.

The new HTML version brings many improvements and new features to the HTML standard. One of the most important features is the addition of the canvas tag element, which allows client side drawing and rendering on a portion of a web page.

2.2.2.2 JavaScript

Since HTML is only capable of providing static content, between HTTP requests, this created the need for a client-side method of adding and modifying HTML content within a web page rendered in a web browser (to avoid multiple server requests).

Originally developed by the Netscape browser team to include in the, now deceased, Netscape browser and allow dynamic user interaction and browser control, JavaScript, which was based on a Java implementation, is, as of June 1998, an ISO standard.

JavaScript is a programming language that is run-time evaluated within the web browser, meaning no target architecture compilation is needed, making it cross-platform. It features a dynamic typing and object-based definition, that provides great functionality and flexibility.

With a syntax and subset of features very similar to regular EMACScript languages, development in JavaScript is no longer tied to the web environment, and the language is now used in regular stand-alone application, games and even operative systems, as in Microsofts' new Windows 8, validating the ease of use and development of JavaScript (code) in any scenario, not just the web target.

2.2.2.3 CSS3

As with most documents, the placement and formatting of the rendered information is important for document layout and proper information flow.

Cascading Style Sheets (CSS) is a document definition that can be integrated with markup languages, like HTML, to help style and format each web pages layout. This approach of separating document content and document presentation, can improve content accessibility while still providing flexibility and control of the presentation characteristics. It also enables multiple pages to share the same layout/formatting, thus reducing complexity and possible repetitions of either content or layout definitions.

Based on priority rule blocks, each rule being composed of element selectors and a declaration set of attributes and values, CSS is capable of modifying each individual element, as-well as a group of elements and their graphical attributes, allowing the browser application to render each HTML element as intended.

Version 3 of this specification adds new features to element selectors, allowing an advanced (almost regular expression based) syntax to be used for element selection, and new attributes to elements. Animations, in the form of multiple/chained attribute changes, can also be achieved with this new specification.

And while version 3 of the specification is not yet a standard, leading to multiple vendor specific implementations, some 3rd party tools allow for simpler CSS implementations. One of these tools is LESS, and allows CSS developers to focus only on the main CSS specification, and afterwards the LESS compiler can create a CSS document with rules for multiple vendor specific implementations.

2.3 Dicoogle

Dicoogle [1] is an open-source software solution, designed and implemented at Universidade de Aveiro, which implements and conforms to the DICOM standard [3]. Bundled in an all-in-one Java solution, it is designed to manage the information workflow in a PACS [9] as well as the archiving and indexing process of the arriving DICOM files. It is in essence an improved, yet fully standard compatible, PACS core.

The current implementation can either be run in server mode (default) or used as a client to connect to another server instance of Dicoogle (via Java RMI). This enables simple access to client workstation within or outside a medical organization.

Doctors can use this interface to search among the indexed files using any parameters they please, either free-text or building a more complex query that will improve the search results. One of the great features of Dicoogle is the fact that it indexes DICOM files as they arrive to be stored, granting in-time file/data search and removing the necessity of scheduling indexing processes. Also, since it is implemented using Java and is an all-in-one bundle (jar solution), it can be easily deployed across different system architectures without problems, requiring only that the standard JRE VM (Java Runtime Environment Virtual Machine) to be installed on the target machine.

Dicoogle has also implemented a simple plug-in system, based on its own in-house SDK, and can load and execute plug-ins that will provide extra functionality to the system. For instance, in the current implementation there is a plug-in that allows users to search through a P2P network of Dicoogle peers that effectively enhances the performance and results of a search.

There is also a set of WADO web services already implemented in the current version of Dicoogle.

2.3.1 Plugin Support

The current Dicoogle version already has a plug-in support system. Based on an SDK and on the JSPF library [10], this system allows the easy development and deploy of Dicoogle plug-ins.

There are three plug-in types that can be instanced with the current Dicoogle SDK:

- GenericPluginInterface
  Plug-ins that implement this interface will have access to the indexing mechanism and also contribute to Dicoogles' indexing and searching engine results. All Dicoogle plug-ins must implement this interface.
- NetworkPluginAdapter
  This interface extends GenericPluginInterface and also provides networking handling tasks abilities to plug-ins.
- GraphicPluginAdapter
  This interface extends GenericPluginInterface and also provides plug-ins the ability to access a dedicated part (for each plug-in) of the user interface.

Dicoogle also features some proven examples of each plug-in instance. The DICOM indexing mechanism, driven by the Lucene engine [11], currently reside on a GenericPluginInterface plug-in. The P2P capabilities of Dicoogle are also available via the JGroups plug-in that implements NetworkPluginAdapter. Both plug-ins also make available their configuration options on the GUI by implementing the GraphicPluginAdapter.

This, rather simple plug-in support system, allows extra functionality to be added to Dicoogle without the need for a new deploy of the Dicoogle server, simply add the plug-in to the Dicoogle plug-in directory, restart the Dicoogle server application and the new functionality is added to the system.

Figure 3 denotes the current Dicoogle plug-in system implementation and interaction:

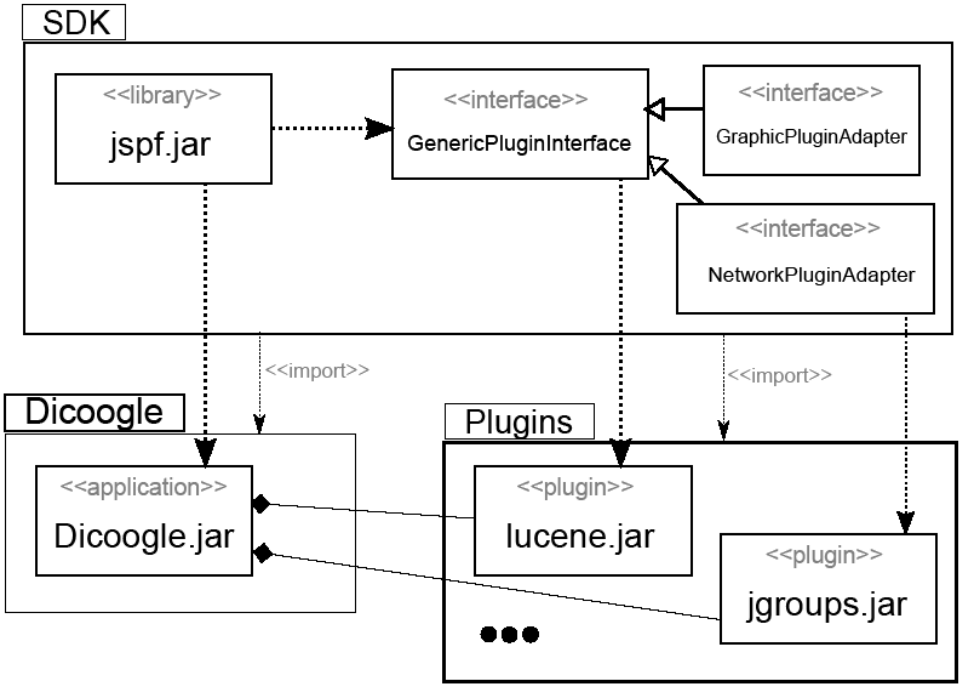


Figure 3: Current Dicoogle plug-in system implementation and interaction.

2.3.2 P2P Architecture

Since Dicoogle only supported searching on one server instance, extra work has been made by the Dicoogle developing team, to implement distributed search architecture.

This allows the several instances of Dicoogle, which are running within a medical institution, to cooperate and distribute the load when performing searches on the indexing engine.

A, custom, peer-to-peer (P2P) architecture was devised and implemented onto Dicoogle [12], that allows peers (Dicoogle clients and servers) to cooperate during a distributed search, enhancing the set of search results, as-well as reducing the time spent until the first search results are seen by the client conducting the search. Given the nature of this search procedure/algorithm, search results will continue to arrive at the requesting client until all peers complete the search.

Figure 4, bellow, is a diagram indicating how the communication between peer is processed, when Peer 1 starts doing a distributed search on a Dicoogle P2P network containing Peer 1 Peer 2 and Peer 3:

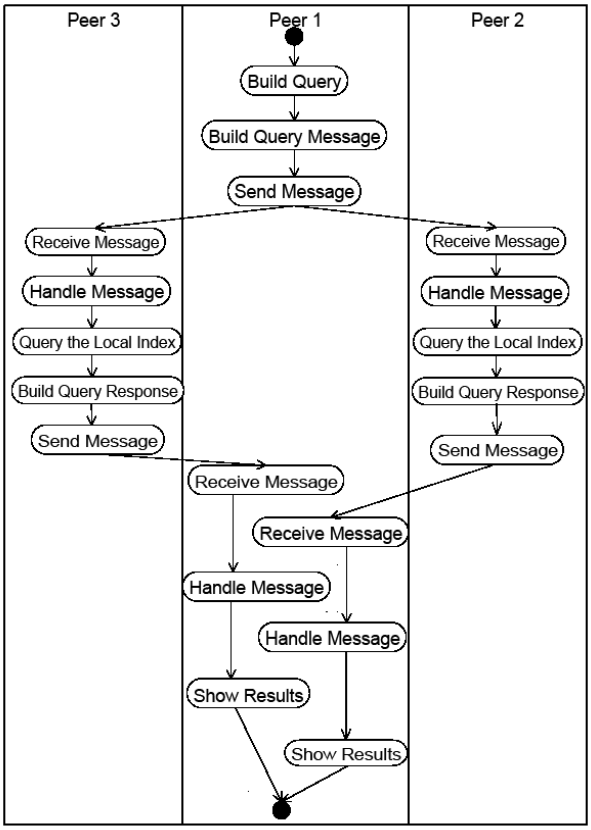


Figure 4: Example Dicoogle P2P search communication diagram [12].

2.3.3 Limitations

The current system has some limitations, and this is part of why a new revamped architecture would greatly benefit both developers and users of Dicoogle.

The current implementation of Dicoogle relies on and underlying Java RMI interface to for the communication between Dicoogle clients and server. Although RMI provides great functionality with small developing effort, it also creates the need of a Dicoogle client application that will need to be deployed and used by the medical professional in order to access and use Dicoogle. And although Java is available to a whole range of distinct platforms, other factors like the Graphical User Interface (GUI) severely limit the ability of extending Dicoogle client use on platforms other than the typical desktop system.

Also, Dicoogle's current plug-in system, mainly due to this RMI interface, requires Dicoogle plug-ins to be transferred to each Dicoogle client, upon client initialization. This approach has the strong point of releasing some of the server load, which would be necessary to accommodate and serve each plug-ins' functionality, to the clients, but also enforces the previous need for a client side instance of Dicoogle.

On the other hand, the RMI implementation is complex, mainly in the communication between the core and GUI part. The migration to the web will also be compliant to the current trends of the new applications, which will better fulfill the requirements of Dicoogle's users.

Finally, despite of the portability of Java, the Dicoogle solution still need to be installed and deployed on the computers, which carries some portability issues.

3 Requirement Analysis

Porting the current Dicoogle platform onto a web enabled environment, imposes, from a developers standpoint, not only some implementation issues, but also architectural issues. In this chapter, it will be introduced the requirements analyses that will lead the developing of new Dicoogle frontend.

3.1 Functional Requirements

Nowadays, the web thrives, and web-based software solutions are a common alternative to stand-alone desktop implementations. Part of the success of these kinds of solutions resides on the fact that it takes a significantly less amount of time and effort for a client to be able to use the system, since the only action the client as to take is open up a web browser and point it to the solutions URL.

Also, with the growing amount of computing devices type and architectures, like mobile communication devices, a person is bound to use several distinct devices throughout their day, be it the nearest desktop computer (commonly x86 based device) or a smart-phone (commonly a ARM device).

In this scenario, if a user wants to have the same functionalities on any of these devices, the applications must first support the target devices architecture and secondly the user (or in charge IT personnel) will need to install and configure the same applications on these multiple devices. This carries some costs, not only on time spent setting-up and configuring the solutions on several devices, but also, most of the times, the cost of multiple software license fees, one for each machine the solution will be installed on.

Providing a software solution as a service over a network and making it accessible to multiple devices via a web-interface reduces the time spent configuring said solution, while also maintaining the same interface and functionality coherence across multiple devices.

One example of a desktop versus a web-based solution is Microsoft Office versus Google Docs. Google Docs might not be as “advanced” nor contain the same functionalities as Microsoft Office, but for the majority of the documents created by users of both systems, Google Docs will provide all the needed functionality without the user

having to buy and install a client-side office solution like Microsoft Office. One other great factor in favor of Google Docs is out-of-the-box multi-platform/architecture support due to being a standard web solution, allowing the solution to be used on multiple devices regardless of architecture or installed software.

The new Dicoogle solution must, also, be able to provide a better overall user experience and easiness of use then its desktop counterpart. By providing all the tools necessary for the target scenario (medical professional searching and analyzing the contents of DICOM files), without the need of installing 3rd party tools, the benefits of the new web solution will be far greater versus of the old desktop solution.

3.1.1 Web Oriented Architecture

The new DicoogleWeb solution must be accessible to any web based client, providing both DICOM search and data results while still maintaining the previous PACS functionality. Since we are adopting the web architecture, new solutions could also, besides providing data to clients, be implemented with new capabilities, removing the need of external analysis tools on the clients' devices for parsing and rendering the DICOM information.

The new usage scenario will allow doctors and other medical personnel within a medical institution, or even outside of said institution given that the new solution can be made available to outside web clients, to access the PACS resources from any web enabled system without any previous configuration of the client device. Moreover, since the new graphical interface will be web based it will be significantly less taxing on the client devices processing and/or rendering abilities, compared to the previous Java AWT + RMI implementation, allowing lower grade (read cheap) smart-phones to still be able to fully use DicoogleWeb.

Integrating a DICOM frame viewer into the solution would maximize the solutions potential compared to the stand-alone desktop implementation, where each client needed a DICOM viewer installed on their client system. This would allow any web enabled client, with proper rendering capabilities, to not only display each medical study and inner DICOM file information but also to display the medical images captured and archived in said DICOM files, making DicoogleWeb a rather complete PACS and analysis system.

All these implementation requirements will greatly improve the medical staff (doctors) usage experience, minimizing the amount of time spend loading distinct software solutions, and thus maximizing diagnosis potentials, while still bringing costs down due to the removal of purchase of external DICOM analysis tools. On the other hand, Dicoogle has also been used by clinical researchers that have deployed the application in several hospitals. Nevertheless, because it was a desktop application, it was complex to disseminate Dicoogle among other researchers, and to have remote access from other devices.

3.1.2 Search

Much like the desktop implementation, the new web-based solution must be able to provide the same searching capabilities within the PACS system.

And given the new web architecture, this can also be provided in a way that would allow other systems/solutions to query DicoogleWeb and use the returning results for other analysis purposes. A REST based web API can then be created to support these requirements. This REST interface will improve DicoogleWebs' functionality, by generalize the web interface, improving compatibility and scalability between component and systems interactions.

Furthermore, providing web clients with a way to obtain and view the full list of tags that are defined within a DICOM file hosted on the system is also needed. And like the search results these can be made available in a way that other systems can request for other analysis purposes.

Allowing users of the system to perform these searches in a interface that is similar to the previous desktop implementation will also allow a smoother and easier, on the users, transition from the old architecture to the new architectures.

3.1.3 DICOM File Transfer

Once again, like the previous desktop implementation, DicoogleWeb must be able to serve the DICOM files hosted within its internal PACS to outside/requesting clients. This will allow doctors to download each DICOM file and then use the external tools of their liking to analyze said DICOM files, to copy the files onto a mobile media so it can be shared with an offline (without access to DicoogleWeb) workstation, or to simply provide a carbon-copy of said files to the patient.

Since file transfers over a web interface are pretty common nowadays this feature can be easily implemented on the new system.

3.1.4 DICOM Frame Export

As mentioned before, if the new solution is to implement a DICOM viewer, a method for retrieving the pixel information stored within an archived DICOM file must be implemented.

The client can either request the complete file using the file transfer, or request just the frames, of that file, that are of interest. Since medical studies can be very long and thus the DICOM files also be very large in size, so fully transfer these files to the client will take a considerable amount of time, even on a fast network. Not to mention that afterwards the client would need external tools to parse and analyze the medical images archived in the downloaded DICOM file.

Since a simple DICOM viewer is to be integrated in the new web solution, providing a standard way for a client to obtain a certain requested frame from a specific DICOM file can be of great value to the viewer implementation. This will also help minimize the load on the client while also removing potential free hard-disk/RAM space or processing power needed to parse the full DICOM file. Also if implemented correctly this can also be used by external, although custom (designed for DicoogleWeb), viewing and analysis tools that can request each DICOM frame as they please. A WADO alike interface can be used for this purpose, maintaining compatibility with standard implementations.

3.1.5 Plugin Support

Dicoogle already has a plug-in development and supportive system. However, to cope with the new web architecture, the internal Dicoogle plug-in system must be revamped, removing RMI requirements and other minor issues with the previous implementation.

Since now plug-ins cannot be transferred to client systems, like they used to be on the previous implementation, extra functionality must be added in order to continue to support the current functionality, while allowing extra functionality to still be added later on.

One of the arising issues is how will each plug-ins' internal settings be configured through the new user interface (web). Without potential access to Java runtime libraries on each client, mainly mobile devices users, the execution of plug-ins client-side is not possible. So maintaining and running the plug-ins on the DicoogleWeb server, only, is necessary, taking into account that their functionalities must still be available to system clients.

Another issue is how will each plug-in access the new web GUI if it desires to display information, this way, directly to the client/user. On the previous Dicoogle solution, each plug-in had access to a dedicated portion of the user interface. While replicating this window-based functionality, on the new solution, would partially solve the issue, it would also create another one: the necessity for server-side event-based user input validation and re-generation of the interface each time a user modifies a field on the interface. This would create multiple AJAX calls that could, depending on the amount of client using the interface at one time, potentially, overwhelm the server and introduce a Denial of Service (DoS) effect.

An extra functionality that can be added to the plug-in system is the ability to provide data directly via external HTTP requests (REST alike). This would allow developers to write plug-ins that can extend DicoogleWebs functionality, like providing statistical information or other analysis results, without any modification on either the DicoogleWeb Core or SDK required.

3.1.6 User Authentication

Serving personal and sensitive medical information requires that only authorized personnel can view and analyze said information.

The previous Dicoogle implementation already defines a simple whitelist/valid user list and authentication system, which consisted of a manually registered list of user-names, passwords and level of clearance of authorized users.

To maintain some compatibility and portability between the previous Dicoogle version and the new one, it is possible to continue to use this list to authenticate web users/clients requests that desire to access the PACS information or configuration. This will continue to guarantee that unauthorized requests cannot access sensitive medical information nor make changes to the system configuration.

3.1.7 User Roles

Unauthenticated users must not be able to access the medical information or system configuration of the solution, and need to provide valid credentials before doing so.

Medical professionals should not have access to the solution configuration options, only system administrators will be able to change those, avoiding erroneous usage/configuration of the solution.

The previous Dicoogle solution already had the concept of user roles implemented in it, so porting this to the new web solution will be pretty straight-forward, and will still maintain the same usage scenario for each user group.

3.2 Non-functional Requirements

3.2.1 Mobile Clients Support

Since the new solution will be web-based and potentially accessible via WAN (Wide Area Network), it would be of great value if medical professionals (doctors) or system administrators could now access both the interface and its data "on the go" without the need to find the nearest desktop computer. This will effectively remove the desktop environment requirement from the solution.

Given the current specifications of most mobile devices (smart-phones and tablets), the solution needs to cope with some of the constrains that these kind of computing devices impose, like limited screen size, processing power and user input methods.

Other constrains on the mobile environment are which technologies are supported by the mobile browser on each device and also the battery drain the solution will create on these devices during its usage. For the former, the Google Android and Apple iOS operative systems have "normalized" which technologies mobile browsers currently support, with HTML5, being supported on at least Android v1.5 and iOS v3.2 [13].

As for the battery usage, maintaining the solution as JavaScript free as possible will minimize the strain/load on the devices CPU allowing it to throttle most of the time, thus saving battery time.

Focusing on the devices screen size, there are already some CSS and JavaScript libraries that adapt and format the web interface according to the screen size of the device it is being presented on (done client side, for each device). This will allow developers of the solution to focus on the solution itself and not on the way it will be presented to the user, while still maintaining usage compatibility with lower/bigger screen devices.

3.2.2 Security

Access to Dicoogle is not strictly bound to users within the same LAN (Local Area Network), meaning that clients outside of the medical institution can access Dicoogle in the same way as if they were inside the medical institution (if the institution network is configured to allow this scenario of course). This creates the necessity of a secure connection between the solution server and each client device, so no medical information can be leaked or accessed by network sniffers.

The HTTP protocol already has a secure mode, HTTPS, which can protect the clients' connection by providing bi-directional encryption between each client and the server. On HTTPS web clients can also validate if they are accessing the "real" DicoogleWeb solution server, and not a fake/spoofed one, by validating the certificate returned by the server.

These measures will effectively protect both server and clients against man-in-the-middle attacks, like ARP and DNS spoofing or the sniffing capture of raw network data.

3.2.3 Performance

The new Dicoogle web architecture, while being substantially different from the previous RMI one, must maintain on average the same, or lower, load during system usage by web clients.

By maintaining server calls to a minimum and providing the ability of client-side caching of resources, including medical data ones, the new web solution can minimize the server load for the same amount of clients, compared with the previous Dicoogle version. And if there's an HTTP gateway (with caching enabled) deployed between web clients and the server, the server load will also be minimized, when consuming medical data resources (like downloading a DICOM file).

Moreover, since the previous solution provided no caching mechanisms, the performance of the new Dicoogle solution should surpass the old solutions performance, while conducting the same tasks. By providing a lowering the time spent by doctors waiting for the solution/data to be ready, the new performance will surely be a strong point of the new solution.

3.2.4 Web Standards

By using and supporting open web standards, a new level of compatibility between Dicoogle and external clients and solutions can be achieved. The previous Dicoogle solution, while fully supportive and compliant of the DICOM protocol which allowed it to interact with other PACS, lacked a proper and open interface of interaction with outside

application/clients. This enforced the use of the only interface available for Dicoogle, the desktop, Java RMI based, application.

Wanting the solution to remain as open as possible, allowing easy and standard interaction with other applications and/clients, the use of open web standards will ensure, at least from a developers point of view, that both the (Dicoogle) information and interface its presented is easily accessible, retrievable and replaced if needed, by any requesting client.

As mentioned above, at 3.2.1, client cross-browser compatibility is of high priority. By using only open web standards, that are well defined and documented, being supported by all major web browsers, it is possible to guarantee that any browser that supports the HTML5 specification will retrieve, render and execute the solution as intended.

Given the current capabilities of the HTML5 specification, like the canvas tag element that allows client-side programmatically driven rendering, and the JavaScript abilities present on any web browser, it is possible to use only open web standards during the development and on the resulting implementation of DicoogleWeb. This will remove the need to 3rd party client browser plug-ins to aid in processing and the display of the user interface, making DicoogleWeb available to any HTML5 compliant browser/device.

One of the scenarios where web clients' browser compatibility is needed will be during the retrieval and rendering of DICOM frames from the server, since most DICOM files have its frame data in either RAW or JPEG2000 format.
Both of these formats are not open web standards and thus are not supported by the majority of the browsers, so server-side conversion of these formats, to one that can be interpreted by all web browsers, must be done.
Two open-web candidate formats arise: JPEG and PNG. PNG was chosen to be the frame delivery format since it provides pixel perfect (loss-less) conversion while still providing average data compression to minimize network usage, compared to the lossy (partial information lost) JPEG format. Another candidate format would be the JPEG-LS (loss-less) format, which could have been chosen instead of PNG. JPEG-LS would provide identical overall image compression and performance versus PNG, however, since the PNG format is a completely open and patent free standard it was chosen as the de-facto format (although some internal JPEG-LS algorithm parts are subject to patents, the

licensing of these patents are free [14], and are only relevant to implementations of the compression/decompression algorithm).

3.2.5 All-in-one Solution

Much like the desktop implementation, this new web solution should be easily deployed onto a medical institution. Meaning a self-contained server solution that requires neither external applications nor previous configuration settings to be run, besides the Java Run-time Environment (which is needed by any Java enabled application).

This feature will continue to allow small medical institutions, as well as larger ones, to easily deploy the solution without the need for expensive dedicated hardware or on-call IT personnel.

4 DicoogleWeb

In order to effectively port the current Dicoogle architecture to a web based one, some architectural changes must be done, while retaining the essential features of the system.

Maintaining the current PACS core implementation and building around it is one of the objectives, removing RMI dependencies, effectively decoupling Dicoogle from any final user interface.

4.1 Core Architecture

DicoogleWebs' core architecture will be very similar to the previous Dicoogle core architecture, although some important changes are in order, both to extend and improve the architecture as-well as implementing the necessary changes for the new interface. Retaining the full functionality of the previous embedded PACS core is a must.

Also, as the project moved on, new issues with concurrent sub-projects of Dicoogle arose, so adding extra functionality, while keeping the new architecture generic enough to avoid specialized interfaces, is one of the strongest points of the new architecture.

The new DicoogleWebs architecture is depicted, in figure 5, bellow:

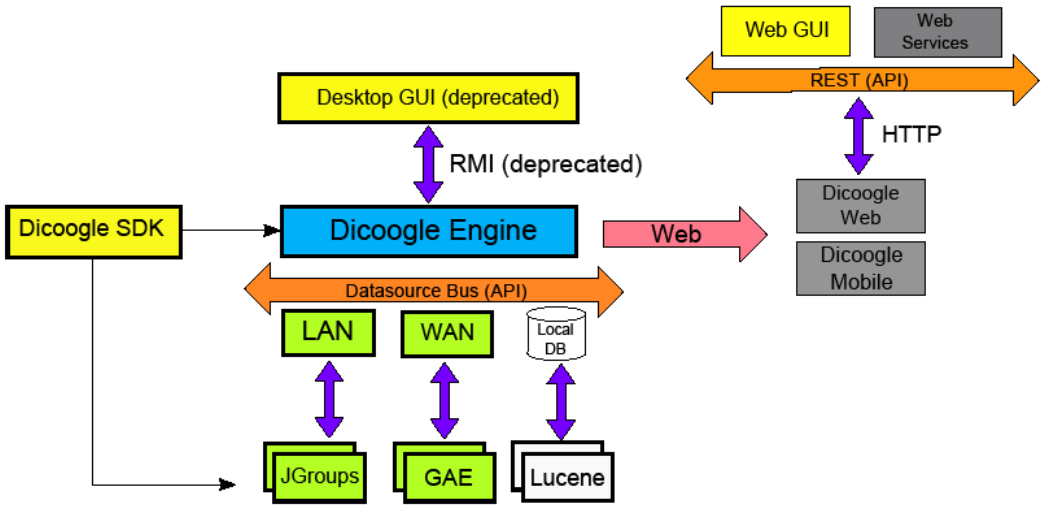


Figure 5: DicoogleWeb architecture diagram.

By keeping the previous Dicoogle core implementation mostly intact, and building the new web interface and API around it, we effectively removed the RMI dependency from the new DicoogleWeb interface.

The RMI interface will still be fully functional, but since it is now deprecated in favor of the new web based interface, it is not the main development target anymore.

The “Dicoogle Web Services” element will now contain the web services API that provide search results, DICOM files, plug-in data, configuration options, indexing core information and control, and authorization information to outside system clients that want to use the new REST API to interact with DicoogleWeb.

The “Web GUI” component will handle the presentation of DicoogleWeb information and configuration onto a client web browser, by providing said information in a user friendly series of web-page/HTML documents.

4.1.1 RMI

One of the objectives of the new platform was to effectively remove the RMI dependency from Dicoogle. However, leaving the current RMI implementation in a working state, and just remove the Dicoogle architecture/platform dependency from it, will leave both communication interfaces fully functional.

This design will make the transition to the new architecture and interface much easier on the medical professionals. Leaving the decision to change user interface to either the system administrator, that can force either RMI or HTTP interfaces to be disabled or to the doctors, which might be accustomed to the previous interface, to do the transition to the leaner and user-friendlier web interface.

4.1.2 SDK

Being the skeletal structure that supports Dicoogle, the SDK must both cope and support the new web architecture, while also improving functionality and compatibility with other distinct Dicoogle sub-projects.

In a joint effort, with the remaining Dicoogle developers, several modifications and improvements were made to the Dicoogle SDK. These changes were in place in order to support the various Dicoogle sub-projects within the Dicoogle developing team(s), while maintaining the SDK generic enough across different projects.

New plug-in interfaces were added to the SDK, further extending the plug-in system functionality. From now, instead of all plug-ins being able to provide querying capabilities if they please, there is a plug-in interface dedicated to querying/searching. Another set of interfaces for requesting/providing access to the graphical interface, providing data/information via HTTP were also added. The base plug-in interface, while remaining basic enough, providing only basic information and command of/for any plug-in, must be implemented/extended by any plug-in interface, including the noted above..

Given the HTTP interfaced plug-in, direct data/information export by plug-ins is now, possible, for each plug-in that desires to do so.

Some extra changes were also made to the SDK, in order to allow for easy development of DicoogleWeb plug-ins, by providing, in the new SDK itself, much of the

basic functionality and tools that most plug-ins would otherwise need to internally support, like the management of their internal settings.

Al of these changes will help new developers to adopt the Dicoogle platform and start (to help) developing for and improving the platform, since now it's easier and faster to develop plug-ins.

4.1.3 Plugins

Since from now on, Dicoogle plug-ins will not be transferred to the client systems', the system will accommodate only server-side plug-ins, while still being able to provide said plug-ins functionality to DicoogleWeb clients.

Adding extra functionality to the plug-in system is also in order, and since now there will be a new vector of communication from clients to plug-in via HTTP requests, an API must be defined to provide all plug-ins with a common functionality of exporting information/data via HTTP.

Another point of interest, is access to the new graphical user interface by plug-ins. Previously, on the "old" (RMI interfaced) Dicoogle, plug-ins had exclusive access to their own Java AWT JPanel, and were free to add and modify its content accordingly to their needs. Mimicking this behavior on the new web-interface would require at the very least a layer to translate AWT objects and their properties to HTML. An extra layer would then be needed to provide post-back feedback for these web components in order to simulate and process the actions triggered by each element event. This would also put an enormous stress on the Dicoogle server, since it would have to cope with all the concurrent clients post-back HTTP requests.

Now with HTML being the main target for a user interface, a system for mediating the access by plug-ins to DicoogleWeb system HTML pages must be devised and implemented. This must be implemented taking into account that there is an HTTP connection between server and client, so there is no "direct" server access by the client, meaning requests and responses will be delayed. Also, trying to maintain HTTP requests to a minimum by the client side interface would be a positive point in favor of mobile devices, from a both a network and battery standpoint.

An overhaul of the current plug-in system must be done, by changing both the SDK logic part as-well as the core plug-in support system, in order to accommodate such structural changes. Minimal changes must also be done to each plug-in currently provided for Dicoogle, mainly minimal interface ones, in order to use them in the newer system architecture.

4.1.3.1 Data Export and Retrieval

One of the major implementations that can revamp the Dicoogle plug-in system resides in providing a way for a plug-in to make available, to any requesting web client, its functionality. This would allow external web client applications to retrieve data directly from each DicoogleWeb plug-in.

During the DicoogleWebs' implementation, a group effort agreement, between distinct Dicoogle developers and maintainers, was reached. This involved the ability to provide each plug-in with its own "realm" within DicoogleWebs' web environment. So by allowing each plug-in to have its own HTTP end-point, DicoogleWeb external client applications can consume information/data directly from each plug-in.

Extra functionality and compatibility can also be achieved by the new plug-in system by allowing plug-is to provide, besides their data, a default XSLT schema that will parse, organize and render the plug-ins returned data directly onto a user-friendly web-page layout. Since modern web browsers possesses this capabilities, this can be done completely on the client-side, saving precious server processing power while providing a standard and decoupled way of presenting the data.

Each plug-in that requires this type of functionality will then be allowed to process HTTP requests and create responses for its own end-point in like fashion:

Request and response 1 for data:

```
\plugin\<pluginname>\<plugindata>
```

Request and response 2 for XSLT (optional):

```
\plugin\<pluginname>\<plugindata>\xslt
```

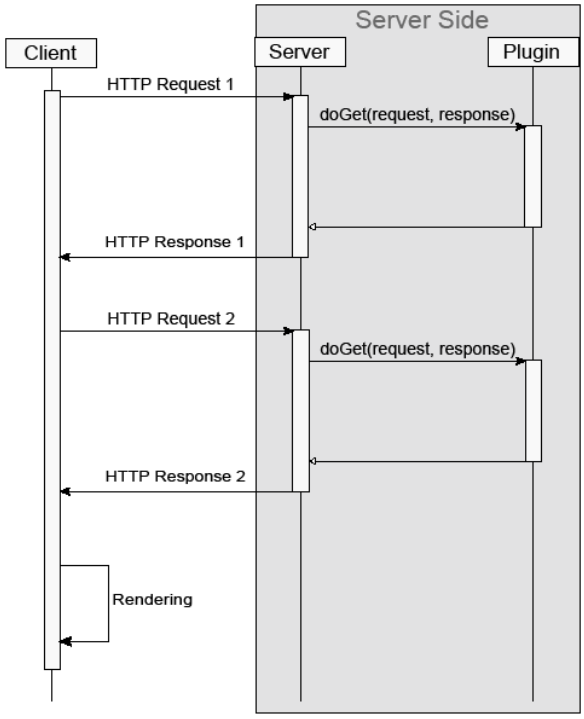


Figure 6: Plug-in data export functionality diagram.

However, there are currently two concurrent data export implementations that need to be completely merged, so a fully standard functionality still does not exist.

4.1.3.2 Access to GUI

Extra work is still being made to allow access to the web interface by Dicoogle plug-ins, the current implementation uses an anchor based system, which allows plug-ins to "inject" in place dynamic HTML code onto a previously exported location by the target page. This is a simple yet very functional, to some extent, interface model.

Each DicoogleWeb web-page decides if it allows HTML code to be injected in it, and on which locations. This is done by calling the `include` method of the `pt.ua.dicoogle.server.web.gui.CodeInjectionPoints.getInstance()` object, for each location where code can be injected. Note that each unique web-page location must also have an unique identifier/name, or code will be injected into multiple places that share the same identifier/name. Upon call of that core function, DicoogleWebs' core will run through all the initialized plug-ins that implement the SDK `pt.ua.dicoogle.sdk.web.gui.InjectsCode` interface which indicates that this plug-in

wants access to the GUI, and ask for the code that it wants to inject on this location, passing the location name and web-page context along to the plug-in. Each plug-in `CodeInjector` object will then modify the web-page context and/or content as it pleases, injecting in-place HTML code and/or changing HTTP headers, and afterwards, the web-page can then be served to the requesting client.

Figure 6 exemplifies, in a simplistic manner, the interaction between Dicoogle and plug-ins that want to inject code onto the requesting web page.

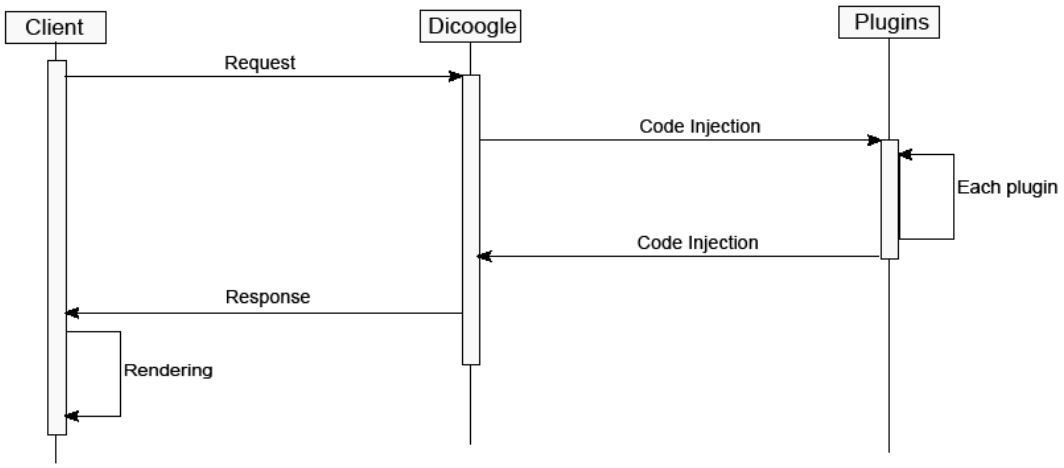


Figure 6: Interaction between Dicoogle and plug-ins that want to inject code onto the requesting web page.

The upside to this implementation is that plug-ins can control the HTTP context of each request/response and of the servlet, allowing, for instance, other authentication methods to be implemented and deployed as plug-ins.

There are some downsides to this implementation, like if one of the plug-ins injects a badly coded piece of HTML (with unclosed or invalid HTML element tags), the remaining of the pages layout can disrupted. And if the target interface is not an HTML browser, rendering of the plug-in injected HTML interface will not be possible.

The major issues with this implementation can be solved by creating a validation and a translation layer, in-place before injecting the code onto the target page, which will close or remove disruptive HTML elements. This layer can also translate the injected code to the target graphical interface (for instance for a Java AWT interface there is no translation to be done, since AWT internally supports HTML elements), removing the requirement of an HTML interface.

4.2 Server-side Architecture

Like in the previous Dicoogle implementation, a server instance of DicoogleWeb is still required to be running at all times. This instance will continue to handle both the inner PACS core and the remaining Dicoogle functionality, with the addition of, now, also providing an HTTP client serving interface to system clients, like doctors, who wish to access and use DicoogleWeb.

The interoperability between Dicoogle and/or other PACS server instances is still fully functional via the DICOM protocol.

Extra functionality required by other external systems can be provided by DicoogleWeb via the HTTP protocol (web API requests).

4.2.1 Web Container

In order to serve HTML and other data types via the HTTP protocol, and since the solution must follow the all-in-one scheme of operation, an embeddable Java HTTP web container is required to serve web clients requests.

A list of available embeddable Java HTTP servers was compiled, comparing each solution capabilities:

	embeddable	HTTPS	JSP	WAR	JAVA version
TJWS	yes	yes	yes	yes	1.2+ (optimized for 1.7)
HttpdBase4J	yes	yes	no	no	1.6+
NanoHTTPD	yes	no	no	no	1.1+
jetty	yes	yes	yes	yes	1.6+

Table 9: Comparing available embeddable Java Web Servers.

Since we are going to serve sensitive and private medical information, the server must support the enabling of a secure HTTP connection.

A server that supports the serving of dynamically generated web pages would be of greater value, since it allows DicoogleWeb to act and response differently to every web client request. And since only two of the solutions support JSP (Java Server Pages), which allows programmers to easily create dynamic web content, the above requirements, between the two JETTY was chosen, since it is a more mature and widely known solution.

Another point of interest of the chosen framework, versus a regular Java REST system, is the default support for HTTP sessions, meaning that it is not possible to hijack or bypass the web interface authentication system (unless due to programming error), thus granting extra security to the system, which is key when dealing with sensitive medical information and data.

4.2.2 Dynamic Content

Since most of the content will be generated during server run-time at a per-request basis, it was chosen to use a web container that supports Java Server Pages. This technology, while not very common within the more commercial web environment versus the more widely known Microsoft ASP technology, still manages to both provide the functionalities that REST-alike web containers provide, while still greatly improving developer easiness and usability by allowing them to easily create a dynamic response content.

Taking this approach for web-page generation will create an extra load on the server for each request received. But since dynamic pages are pre-compiled before being served by the container, this extra load on the server is marginally low compared to the benefits provided to developers of/to the platform.

Decoupling the request processing and response content generation layer from the response content presentation layer, completely separating the two, will transform this architecture into a more Model-View-Controller (MVC) alike one. This provides, not only a cleaner code base, but will also help in the development process, where DicoogleWeb developers might need to debug, server-side, the response code generator for a specific page.

There are already some custom MVC Java libraries that could provide this type of functionality, like the Spring Framework [15], but this functionality can be implemented without the use of external libraries, removing potential JDK and other requirements. Which will also help to keep DicoogleWebs' deployment solution size as low as possible.

4.2.3 Data Retrieval

Given the new web architecture, an opportunity to create a Web API, that external applications can use, arises. This will improve DicoogleWebs' interoperability with other solutions that might want to retrieve/consume data/information from DicoogleWeb and/or its inner PACS core.

The embedded client interface for querying the system (search) is already, for the most part, consuming medical information and data via a custom web API. Improving this API and allowing external applications/systems to use it can prove to be one of the new architectures vantage points.

This new web API can also replace the incomplete WADO interface from the previous Dicoogle version, as-well as extending its functionality.

4.2.3.1 Web Services

Instead of relying on a custom protocol for querying and accessing DicooglesWeb information and data, the new solution will provide web services, with a well defined API and standardized protocol, for this purpose.

Implementing said web services, will improve the usability level of the new DicoogleWebs' architecture, since now, in the web environment, each client application (to DicoogleWeb) can easily implement a small logic layer on-top of their client solution that will allow it to query and obtain data from DicoogleWeb.

This guarantees a high level of interoperability between additional client systems, as-well as re-usability, since the DicoogleWeb client interface, itself, can make use of said web services to query for information and obtain data from.

4.2.3.2 Search

The search results, like the previous implementation, will be provided to system clients in the same fashion of the previous Dicoogle implementation. This means the system can be queried using the same criteria of the desktop implementation, so a simple search and advanced search must be implemented.

This search interface differentiation was kept in working order since it is a proven search method that has previously aided the usage of Dicoogle by medical professionals.

For the simple search, there are only two query parameters: the query string itself and a keywords flag.

The keywords flag notifies Dicoogle of the existence of specific keywords within the input query string, allowing for faster specialized queries to be made by advanced system users. If the keywords flag is not set, then the input query string will be treated as simple text, and a full-text database search will be performed, returning any DICOM file or study indexed information where at least one of the indexed fields matches the query string.

As for the advanced search method, it will be used to obtain very specific search results, that must match on all the criteria defined. An extensible array of fields indicates which criteria will be used to query the system:

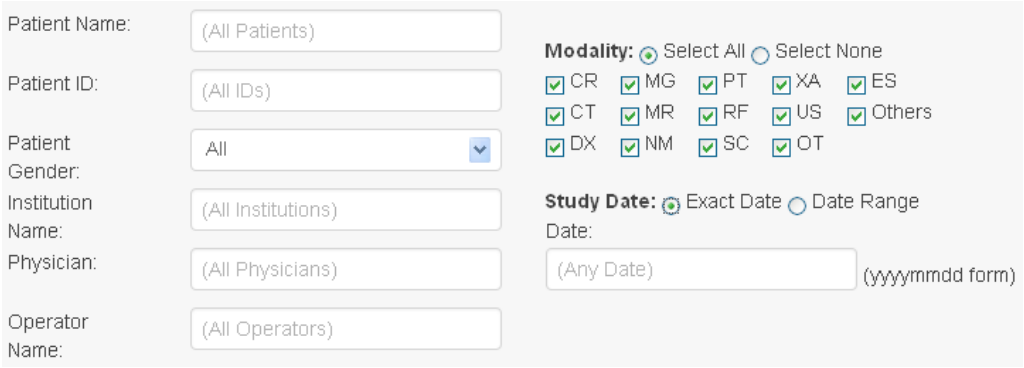


Figure 8: Available fields in the advanced search mode.

These specific fields will greatly improve the search to results, by allowing the medical professional to filter possible unwanted search results.

Besides presenting the search results in the user friendly web interface, both search methods are also available for use to external applications via a web service. Accessing this search web service can be made using the following query schema API:

URI:

`/search`

Parameters:

Name	Accepted Values	Required and Default Value	Validity	Remarks
method	"default", "advanced"	Optional, "default"	Always	Searching Method
query	*	Optional, Empty	Always	Search Query String
keywords	"on", "off"	Optional, "off"	If method = "default"	If the search query should be parsed taking into account that it might have potential search keywords
patientName	*	Optional, Empty	If method = "advanced"	(Part of) The Patient Name
patientID	*	Optional, Empty	If method = "advanced"	Patient ID
patientGender	"all", "male", "female"	Optional, "all"	If method = "advanced"	Patient Gender
institutionName	*	Optional, Empty	If method = "advanced"	Institution Name
physician	*	Optional, Empty	If method = "advanced"	Physician
operatorName	*	Optional, Empty	If method = "advanced"	Operator Name
studyDate	"exact", "range"	Required, "exact"	If method = "advanced"	If the study date should be parsed as an exact or a ranged date
exactDate	date in yyyyMMdd format, where yyyy is the year, MM the month and dd the day	Optional, Empty	If method = "advanced" and studyDate = "exact"	Exact study date
fromDate	"on", "off"	Optional, "off"	If method = "advanced" and studyDate = "range"	Use startDate value in the study date range?
toDate	"on", "off"	Optional, "off"	If method = "advanced" and studyDate = "range"	Use endDate value in the study date range?
startDate	date in "yyyyMMdd" format, where yyyy is the year, MM the month and dd the day	Optional, Empty	If method = "advanced" and studyDate = "range" and fromDate = "on"	Study date range start
endDate	date in "yyyyMMdd" format, where yyyy is the year, MM the month and dd the day	Optional, Empty	If method = "advanced" and studyDate = "range" and toDate = "on"	Study date range end
modCR	"on", "off"	Optional, "off"	If method = "advanced"	If CR modality studies should be included in the results
modMG	"on", "off"	Optional, "off"	If method = "advanced"	If MG modality studies should be included in the results
modPT	"on", "off"	Optional, "off"	If method = "advanced"	If PT modality studies should be included in the results
modXA	"on", "off"	Optional, "off"	If method = "advanced"	If XA modality studies should be included in the results

modES	"on", "off"	Optional, "off"	If method = "advanced"	If ES modality studies should be included in the results
modCT	"on", "off"	Optional, "off"	If method = "advanced"	If CT modality studies should be included in the results
modMR	"on", "off"	Optional, "off"	If method = "advanced"	If MR modality studies should be included in the results
modRF	"on", "off"	Optional, "off"	If method = "advanced"	If RF modality studies should be included in the results
modUS	"on", "off"	Optional, "off"	If method = "advanced"	If US modality studies should be included in the results
modDX	"on", "off"	Optional, "off"	If method = "advanced"	If DX modality studies should be included in the results
modNM	"on", "off"	Optional, "off"	If method = "advanced"	If NM modality studies should be included in the results
modSC	"on", "off"	Optional, "off"	If method = "advanced"	If SC modality studies should be included in the results
modOT	"on", "off"	Optional, "off"	If method = "advanced"	If OT modality studies should be included in the results
modOthers	"on", "off"	Optional, "off"	If method = "advanced"	If Other modality studies should be included in the results

Table 10: Advanced search API request parameters definition.

Usage examples:

- Simple search for all search results currently indexed: `/search`
- Advanced search for only CR modalities of Patients named António Novo: `/search?method=advanced&patientName=Antonio&modCR=on`

When the server-side database search is finished, the results are re-arranged and embedded into a XML document that will contain specific information about each search result. This XML document is then returned to the querying client via the web service response.

The XML format was chosen since it is an open standard for textual data transfers between web services, and all browsers support their parsing without the need of external libraries.

Bellow is the DTD schema containing information about the XML document contents and data format:

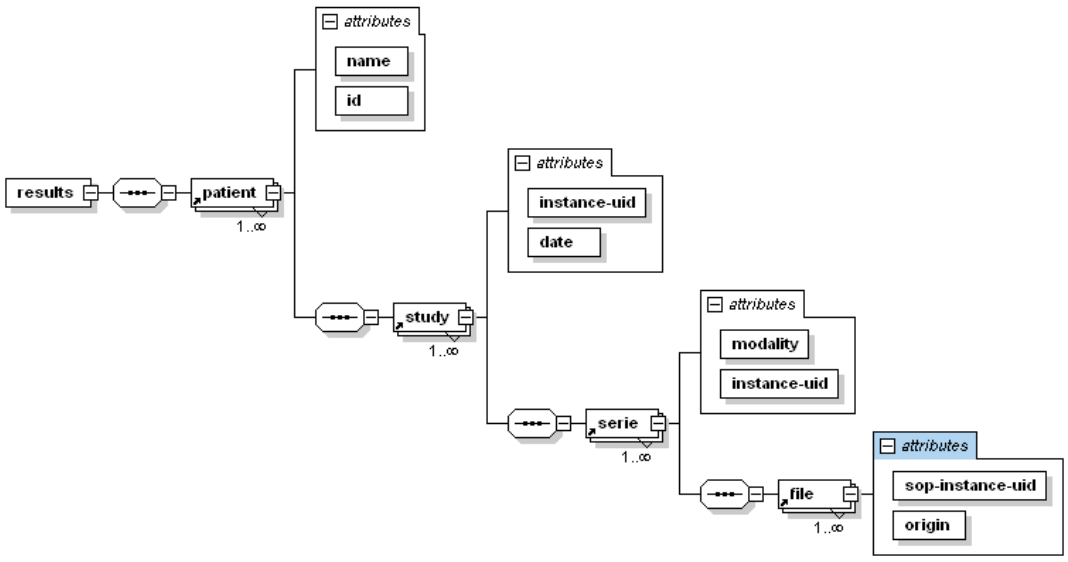


Figure 9: DTD schema of the search results API response.

Note that the current external API returns a small amount of information about each DICOM result, compared to the JSP (web-page) version of the search results. This is by design, since the remaining information can be queried, on a per result basis, if need by the external API requesting application.

4.2.3.3 Tags

Each DICOM file contains a list of tags, defined and present on the DICOM file format specification on PS 3.10 [3], that contains all the relevant information about the patient, which medical exam was the origin of said file and other relevant information about the file itself and its contents. It is this information that Dicoogle uses to index each file within its PACS core.

A doctor might require this extra information to further analyze a search result and/or before providing a diagnosis to the patient, for instance. But, since some medical studies/files can be very large in size, it would be better to provide a way, for system clients, to obtain each DICOM file list of tags, without forcing the clients to fully download each DICOM file so they can just analyze their tags.

Since the SOPInstanceUID is a globally unique identifier for each DICOM file, these requests can be made using only this identifier as a parameter to signal DicoogleWeb

of the target DICOM file. The value of this parameter can be obtained via each search results, which is the main premise scenario for these requests. These values and their functionality will be transparent to typical system users (doctors) since all the functionality will be provided automatically for each file by the interface, without the user having to know or enter said parameter values.

Once again, this functionality is available as a web service and can be queried in like fashion:

URI:

`/dictags`

Parameters:

Name	Accepted Values	Required and Default Value	Validity	Remarks
SOPInstanceUID	*	Required	Always	The unique identifier of a DICOM file

Table 11: DICOM tags API request parameters definition.

Usage example:

- Return all the tags of the DICOM file A0000000, from patient António Novo: `/dictags?SOPInstanceUID=1.2.392.200036.9107.500.305.6918.20130326.102128.187.106918`

The tags contained on the desired file will then be embedded onto a XML document and returned to the requesting client via the web services response.

Depicted bellow is the schema information about the XML document contents and data format:

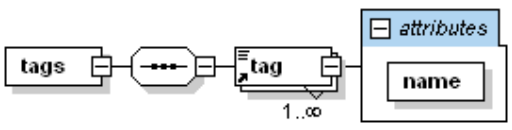


Figure 10: DTD schema of the DICOM tags API response.

4.2.3.4 Frames

As mentioned before, at 3.1.4, not all web clients will have the processing power needed to promptly parse a DICOM file inner images/frames, making the embedded client side viewer unresponsive and even unavailable on entry-level smart-phones, due to having to download and parse the whole DICOM file before rendering.

Furthermore, given the new web architecture, it makes sense to provide such image data in a open and standard way that can be easily retrieved, parsed (sometimes processed) and presented to the medical professionals on any device and/or internet browser.

Given these facts, and since most images inside a DICOM file are encoded either using RAW or JPEG2000 format, both not open web standards and thus unavailable on most browsers, server side conversion of said frames must be performed before returning it to the requesting client. And like depicted earlier, at 3.2.4, the PNG image format was chosen to carry each frames data.

DicoogleWeb will provide these DICOM frames, once again, using a web service that can also be queried by external systems. Said web service can be queried using this format:

URI:

`/dic2png`

Parameters:

Name	Accepted Values	Required and Default Value	Validity	Remarks
SOPInstanceUID	*	Required	Always	The unique identifier of a DICOM file
frame	Integer value >= 0	Optional, 0	Always	Required frame index within the selected DICOM file

Table 12: DICOM frame API request parameters definition.

Usage example:

- Get the first, and only frame, on the DICOM file A0000000, from patient António Novo:

  `/dic2png?SOPInstanceUID=1.2.392.200036.9107.500.305.6918.20130326.102128.187.106918`

- Get the 5th frame from the DICOM file 6.dcm, from patient Camilo Rocha: `/dic2png?SOPInstanceUID=1.3.12.2.1107.5.4.5.35017.4.0.4948412229111107.512&frame=4`

Like the requesting of DICOM Tags, at 4.2.3.3 the SOPInstanceUID parameter, obtained from a previous search result, is used to uniquely identify the target DICOM file.

The desired frame is then converted to the PNG format and returned to the requesting client via the web service response.

This web service, simplifies the implementation of both dedicated desktop DICOM viewers as-well as 3rd party external DICOM viewers, without the clients having to download each DICOM file just to view the more relevant frames to the diagnosis, reducing the time the doctor will have to wait before the DICOM frame is presented on screen.

4.2.3.5 DICOM Files

Like the previous Dicoogle implementation, DICOM files archived within the internal PACS core, can also be retrieved. Although from now on, instead of relying on a custom file transfer protocol, the files will be transferred via the HTTP protocol.

Parallel download of multiple files are possible. Also the system user (a medical professional like a doctor) can continue to use DicoogleWeb normally while the files are being downloaded by the browser, performing searches, for instance. And when the download finished, the client will have a copy of the original desired DICOM file that is archived within the DicoogleWeb PACS.

Requests for this kind of data can be done via the new Web API, using the following request schema:

URI:

`/dcmfile`

Parameters:

Name	Accepted Values	Required and Default Value	Validity	Remarks
SOPInstanceUID	*	Required	Always	The unique identifier of a DICOM file

Table 13: DICOM file API request parameters definition.

Usage example:

- Download the DICOM file A0000000, from patient António Novo: `/dcmfile?SOPInstanceUID=1.2.392.200036.9107.500.305.6918.20130326.102128.187.106918`

Once again, like the requesting of DICOM Tags, at 4.2.3.3, the SOPInstanceUID parameter, obtained from a previous search result, is enough to uniquely identify the target DICOM file.

4.2.4 Configuration

The DicoogleWeb core, as-well as the DicoogleWeb plug-ins, need to retain and save its internal configuration options, for use between system restarts.

On the previous Dicoogle implementation, only the core settings were saved on the server, the remaining configuration options, like plug-ins and Dicoogle interface options, were save on the client computer. But, since the new architecture is aiming at an open and standard web environment, this means that no Dicoogle plug-ins will be downloaded by web clients (removing the requirement of Java Web Run-time plug-ins on the client browser).

Storing the configuration options in a HTTP cookie manner, persisting during HTTP sessions for each authenticated user, would provide a simple and very similar structure to the previous Dicoogle versions way of storing plug-in settings, in a per-client manner. But since DicoogleWeb plug-ins settings and execution will be running solely on the DicoogleWeb server, at all times, it invalidates the option of storing configuration options on HTTP cookie form.

The other option will be to allow the web interface to access, list and allow users to input and set values for each configuration option of each plug-in and DicoogleWeb setting. Although this option is beneficial for the system, by normalizing the current configuration and interface options, it also carries some implementation problems.

Since the new DicoogleWeb architecture relies on the standard HTTP request and response protocol, the semantic information of each plug-in setting, like name and data-type, must be known throughout the HTTP data exchange process between DicoogleWeb server and web client. And because the Dicoogle core does not know every specific detail, and allowed values for each setting which are bound to each plug-ins setting, the trouble of how to parse, set-up and store each plug-in configuration on the server arises.

So a fitting and easily expandable system for managing settings, parsing and rendering them in a user friendly interface with enough semantic information so system users can understand each settings purpose, was devised.

A few components make up this system:

- a translation layer (both from Java class to HTML form input element and from request query parameters to java class object);
- a dynamic page, for each plug-in, that will displaying an HTML form with input elements, one or more per setting, properly converted and rendered from their original Java class;
- a web service that will receive a plug-in or service name and a list of setting names and their new values and applies them as current settings, while also saving them for further use after server restart;

The DicoogleWeb SDK was also changed, like mentioned at 4.1, in order to, while still providing all the previous functionality, guarantee that the new web interface could handle serving, configuring and storing the plug-ins configuration options, leaving the actual value range of each setting to be validated by the plug-in.

By parsing each Dicoogle plug-in or Dicoogle core/interface setting according to data type (Java class) the interface is then capable of creating HTML form input elements that will allow an authorized system administrator to configure each setting, while still

providing semantic value on the graphical interface about each of the settings, be it name, value type and/or range and some help (where needed).

Java Class	HTML Input Element Representation
Boolean	<input type="checkbox" />
Integer	<input type="number" />
Float	<input type="number" />
String	<input type="text" />
Object	<input type="text" />

Table 14: Java class to HTML element translation.

More complex data types (Java classes) or custom ones, will however require that extra information is passed onto DicoogleWeb by the plug-in or setting, in order to correctly parse the setting information onto a valid HTML form input element(s). A generic data type, that can be superseded, was created, as-well as a couple of other pertaining setting data types that were required during the implementation of the new architecture:

Custom Java Class	HTML Input Element Representation
RangeInteger	<input type="range" />
ComboBox	<select><option />…</select>
CheckBoxWithHint	<label><input type="checkbox" /></label>
ServerDirectoryPath	<select /><input type="text" />

Table 15: Custom Java class to HTML element translation.

These are the most relevant custom data types implemented to aid in the development and support of the new web interface.

Extra attention should be pointed at the ServerDirectoryPath one, that allows the user to remotely browse all the directories (available, accordingly to the running server system policy) and select one. This is done via consecutive AJAX requests, to the DicoogleWeb Indexer web-service, upon user selection of a new directory to retrieve its contents from the server. There was already a standard HTML element for browsing and selecting a folder in an HTML interface, however it is local to the client workstation.

The Indexer web-service is one of the instances that will accept and apply supplied requests for configuration. It also allows, as the name implies, the monitoring and modification of the internal DicoogleWeb DICOM file indexer.
It can be queried/commanded by using the following interface:

URI:

`/indexer`

Parameters:

Name	Accepted Values	Required and Default Value	Validity	Remarks
action	“start”, “stop”, “status”, “pathcontents”	Required	Always	Which action is being request
path	*	Required	If action = “pathcontents”	The path which we want to get the contents of

Table 16: DicoogleWeb indexer control API request parameters definition.

Usage example:

- Getting the indexer state and percentage of completion: `/indexer?action=status`
- Get the contents of the server local directory D:\DCM_Files\: `/indexer?action=pathcontents&path=D%3A%5CDCM_Files`

Both these requests, unlike the requests for starting or stopping the indexer, will return a XML document with content relevant to the request made. Here is the schema of both action example requests, respectively:

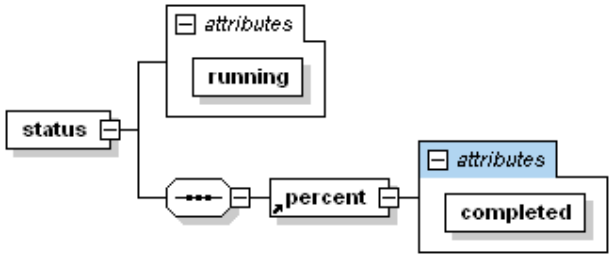


Figure 11: DTD schema of the DicoogleWeb indexer status API response.

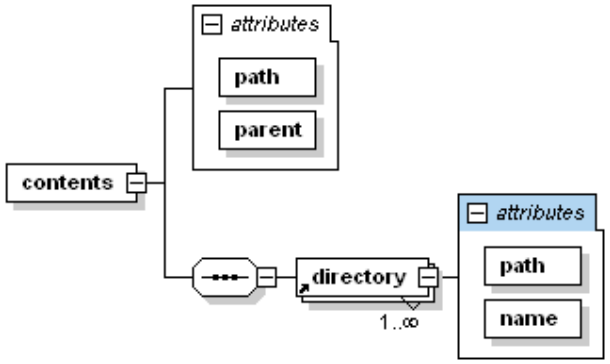


Figure 12: DTD schema of the DicoogleWeb indexer directory contents API response.

Despite all this backing and forth, parsing, validating and converting data-types of each settings' value during the HTTP request and response calls from the client and the server, which can create a small amount of load on the server, the predicted amount of times the configuration options will be accessed and changed are minimal during the run-time of the server in a medical institution, so this extra load on the server can be safely overlooked.

4.3 Client-less Solution

Unlike the previous Dicoogle version, the new DicoogleWeb architecture will require no standalone client side implementation to be deployed (installed) on the client systems. The graphical interface needed to use DicoogleWeb is supplied via HTTP in the form of a series of HTML5 documents, and can be retrieved, parsed and rendered almost instantly, by the client web browser, whenever a client visits a DicoogleWeb web-page.

This new interface guarantees that the system can be used by any client computing system, independently of client-side architecture or installed software, requiring only a network interface that can reach the DicoogleWeb server and a standards compliant web browser.

4.3.1 HTML

Most modern browsers support up to version 5 of the HTML standard. This, still new, version of the HTML standard adds extra extensions to the HTML document model, versus the "old" version 4 of the standard. These new extensions allow the DicoogleWeb server to safely offload some parsing and rendering tasks to clients' browsers, while still maintaining full compatibility between concurrent web browsers. And since the HTML standard is cumulative, new HTML versions will still support older versions, meaning newer versions of the standard will still support the current DicoogleWeb HTML client interface implementation.

The HTML user interface can also be easily customized to meet medical professionals' requirements or implement needed features without creating extra load on the server since all the parsing and rendering of the interface will be done client-side.

4.3.2 Mobile GUI

One of the objectives of this new interface is to support client mobile devices. This will allow doctors and other medical professional, as-well as system administrators, to use available smart-phones, for instance, to browse and use DicoogleWeb, without the need of a typical desktop computer like in the previous implementation.

Most mobile devices, like smart-phones, lack in screen size. Also the typical smart-phone screen resolution is still a lot lower than the desktop system counter parts, which would require constant zooming-in/out by the medical professionals when using the interface in a mobile device.

Including the Bootstrap CSS library in the implementation helped solve this issue with the interface. Being only CSS based, with a few optional JavaScript additions, it is client architecture independent, and thus imposes no limitation to the current solution. With minor changes, adapting the web interface/layout to properly accommodate the Bootstrap CSS library, the web interface now supports both lower and big screen resolutions, adapting itself, client-side, according to the client system screen resolution.

Using the Bootstrap CSS library has also increased the coherence level across the whole web interface, by providing similar layout across distinct pages, effectively improving the system usability by medical professionals.

Here are a series of screen-shots of the same web-page across various screen sizes:

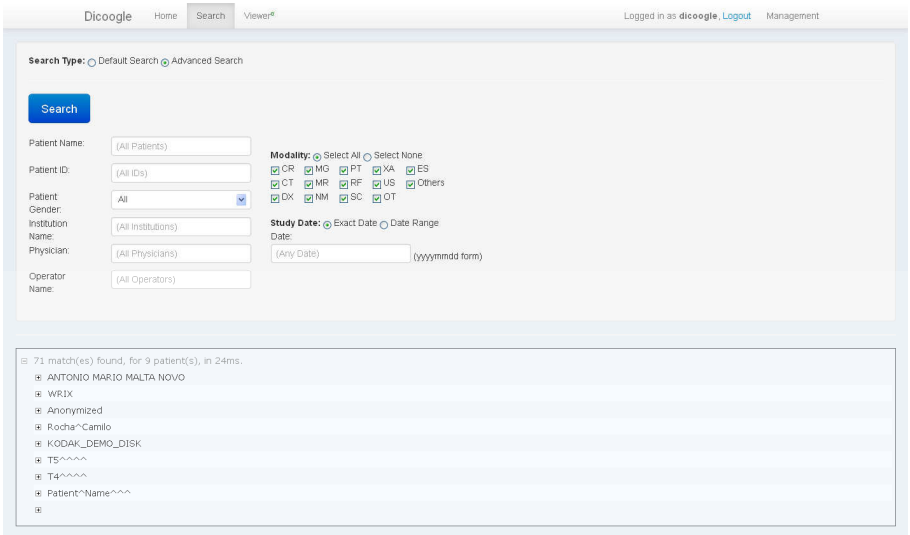


Figure 13: DicoogleWeb search web-page layout on a large screen.

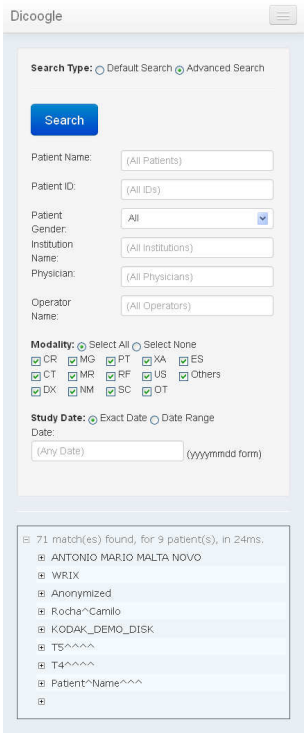


Figure 14: DicoogleWeb search web-page layout on a small screen.

4.3.3 Viewer

A DICOM frame viewer was implemented and integrated within DicoogleWeb. But, since a concurrent, more complete and complex DICOM viewer implementation, was already being implemented by another platform developer, the decision to implement only a simple/example yet functional viewer was made. Later on, the concurrent viewer implementation can easily be merged with DicoogleWeb without requiring many changes being made on the merged viewer.

In order to implement this viewer, there is the need to assess how would the client workstations retrieve and view the DICOM frame/images information.

There is a multitude of web standards, that all major browsers support, that can help accomplish this feature. For instance, using the HTML5 File API would make possible the implementation of a web interface with the same behavior of the current Dicoogle desktop application, meaning that all the parsing and displaying of the DICOM file would be done by the client workstations, after downloading the DICOM file.

Without wanting to emulate the Dicoogle desktop application, where the client workstation retrieves the DICOM file that contains the desired information and then analysis and views its contents using 3rd party tools, a complete embedded web-based solution must implemented and integrated within the DicoogleWeb solution.

One of the target platforms of the web interface are mobile devices, this would enable doctors to still view medical information without having to be near a regular desktop PC. The viewer must, then, be as light and less taxing on mobile devices as possible. So a web-interface to serve all the meta-data and also all the pixel information (images) present on a DICOM file was implemented.

Since each DICOM file already has a unique identifiable ID, the SOP Instance UID, the interface can use this value to uniquely identifying each file over client and server requests and responses. This is very useful because with just an extra parameter, the Frame Number, a web client can request all the pixel information present on a DICOM file in a frame-by-frame series of requests.

Wanting to remain open to web standards, instead of serving the pixel information in the DICOM file, which can be in various formats (including JPEG, JPEG 2000 and RAW and RLE compresses RAW), directly to the client, a server-side conversion of said information is being made to one standard image file format: PNG (Portable Network Graphics). This file/image format is not only open but also a standard in web, and guarantees loss-less (pixel perfect, without information losses) image conversion between the DICOM frames pixel information and the resulting PNG image. And as an added bonus it also implements image compression, so image transfers, between server and client, tax the network less.

Most of these features are already implemented on the new Web API, so the Viewer interface will re-use it in order to retrieve all of the required data by the viewer.

As for the client side displaying of DICOM files pixel information, there are a couple of ways of doing so in a web interface, without requiring external browser plug-ins:

- creating an HTML IMG element on the viewer page for each frame in the DICOM file and loop though them by using JavaScript on the client side;
- creating an HTML5 CANVAS element on the viewer page that will display any pixel information;
- creating a WebGL HTML element on the viewer page that will display any 2D/3D information;

Wanting to emulate some of the major functionalities that 3rd party DICOM viewers provide, like changing the brightness or contrast of the displayed images, that the use of plain HTML IMG elements would not provide, the WebGL or Canvas element must be used in the viewer implementation.

Despite most mobile devices and browsers having the ability to properly display and use a WebGL HTML element, some desktop browsers do not support it. Also WebGL can be very taxing on a mobile device, due to the OpenGL GPU (Graphics Processing Unit) load necessary to display this HTML element. So the HTML5 Canvas element was used for the current web implementation of the viewer.

The current viewer was designed to view the contents of a DicoogleWeb indexed DICOM file, be it just a single frame or a set of frames. It features a small list of capabilities, implemented mainly for a usage example of the new DicoogleWeb architecture and web API.

Among these features, is, of-course, the ability to view the set of frames included in a DicoogleWeb indexed DICOM file, and playing them at the right rate (recording rate), if needed. It is also possible to view, in a user friendly fashion, the list of tags contained on the current DICOM file.

Below is a screen-shot of the current viewer implementation:

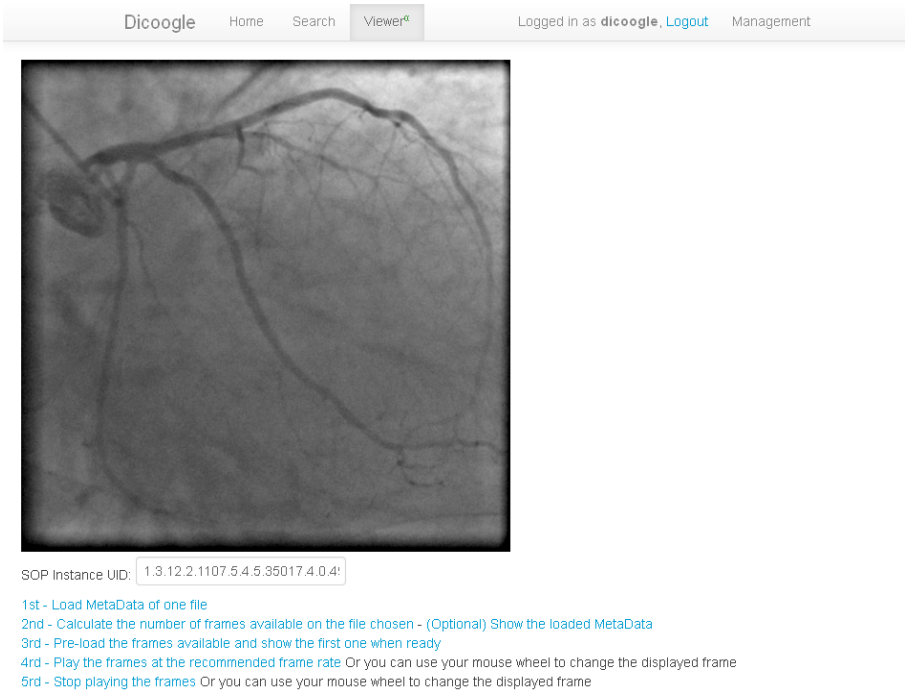


Figure 15: DicoogleWeb embedded DICOM frame viewer web-page.

5 Results

In this section, the resulting software solution, DicoogleWeb, will be discussed. Some comparassion with the previous solution will also be made, both in terms of architectural designing and on a performance basis.

5.1 Overall Solution

One of the most advantageous features of the new web-interface is, and since modern browsers have adopted and support the concept of tabbed browsing, that a medical professional, like a doctor, can be analyzing multiple resources at the same time, without having to close each to conduct a new search on the system and/or analyze other resource, losing the previously collected information. This will greatly improve the usability of the system by doctors, reducing the time spent loading back to previous resources or information and reducing extra load on the DicoogleWeb server.

The ability to use the solution on any web-enabled device (with HTML5 support) and since it is a complete solution (PACS + Search + Viewer) will greatly aid medical personnel on their diagnosis by providing all the tools necessary to do so (for free), not to mention the faster system response time compared to other standalone solutions.

5.2 Performance

Besides easiness of use, the performance and response time of the DicoogleWeb server will contribute immensely for a positive medical professional usage, during periods of high system usage.

Since, in the new DicoogleWeb architecture, both the RMI interface and the Web interface are fully functional, a performance comparison of the two can be conducted. The performance information, presented in this segment, was collected on a wired full-duplex 100Mbps cable network consisting of two, connected, computer systems:

- DicoogleWeb server:

```
CPU: AMD Athlon Neo MV-40 – 1 Core – 1 Thread @ 1.6GHz
RAM: 2GB DDR2 @ 333MHz CL5 – Single Channel
OS: Windows 7 Home Premium SP1
JRE: build 1.7.0-b147
PACS index size: 11 DICOM files – 34MB
```

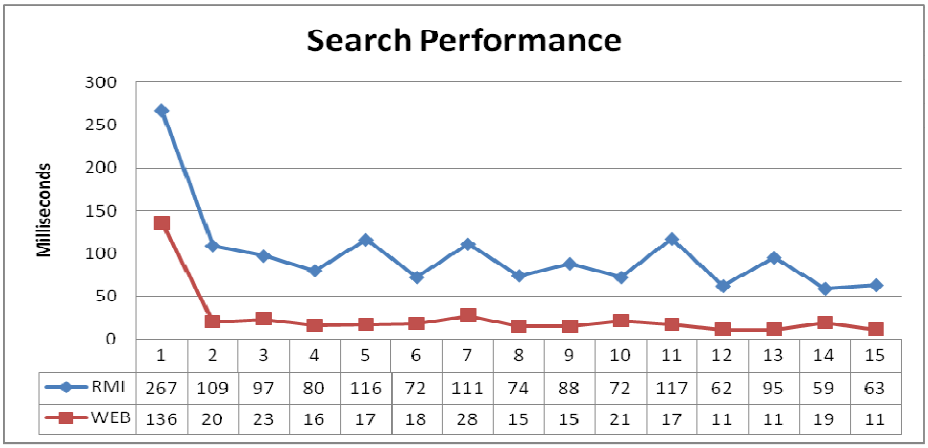
- DicoogleWeb client:

```
CPU: Intel Pentium 4 650 - 1 Core - 2 Threads @ 3.74GHz
RAM: 2GB DDR2 @ 294MHz CL3 - Dual Channel
OS: Windows XP SP2
JRE: build 1.7.0_10-b18
Web Browser: Firefox v22.0 beta
```

5.2.1 Search

The search performance results were gathered by conducting the same "Default search" query of "Novo" (un-key-worded) over the two interfaces: RMI and Web interface. And the server response times, taken to conduct each search, were logged.

The search, independently of the interface used, will return two matches for one patient. Each interface tests were done in a streaming (continued and consecutive) manner, with server and client restarts between interface changes. Here are the results:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
RMI	267	109	97	80	116	72	111	74	88	72	117	62	95	59	63
WEB	136	20	23	16	17	18	28	15	15	21	17	11	11	19	11

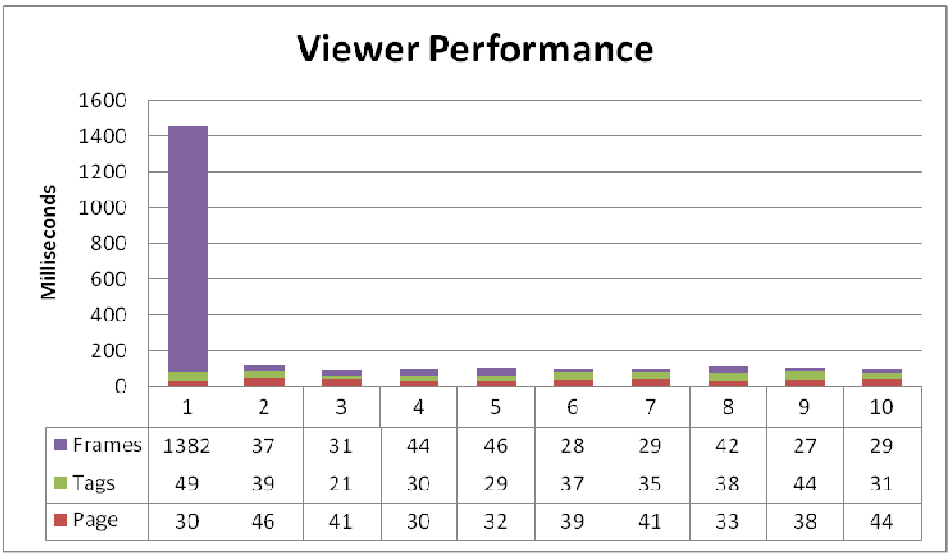
Figure 16: Comparison of RMI and Web interfaces performance during consecutive searches.

As expected, the searches performed using the web environment provide, on average, an 80% reduction in the time spend by the server conducting the search and returning the results to the client. This is mainly due to the removal of the RMI overhead on the connection between client and server.

5.2.2 Viewer

Since the previous version of Dicoogle didn't provide any DICOM viewer capabilities, the two cannot be directly compared.

However, the time spent by the client waiting for the Viewer session to be ready, meaning time taken for DICOM file frames and tags to download, parse and rendering, can be measured. Here are the results for the same 3.1MB "A0000000" DICOM file, containing one shoulder X-Ray frame of patient António Novo:



	1	2	3	4	5	6	7	8	9	10
Frames	1382	37	31	44	46	28	29	42	27	29
Tags	49	39	21	30	29	37	35	38	44	31
Page	30	46	41	30	32	39	41	33	38	44

Figure 17: Detailed Web interface performance during the setup of consecutive viewer sessions.

Figure 17 depicts the various stages/procedures of mounting a viewer session and the time spent by the client retrieving, and the system serving, each one. While these stages progression is completely transparent to the user, the time spent processing each can greatly impact the user experience.

When a user requests for a viewer session, of a DICOM file, the first information retrieved from the server is the web page, "Page" on figure 17. This includes the HTML file, its CSS and all the linked JavaScript needed by the solution to provide a functional and user friendly DICOM viewer.

The second stage of this procedure is the request of the DICOM tags of the target DICOM file, Tags on figure 17. These tags hold pertaining information about which patient and medical study/series the DICOM file that will be analyzed belongs to.

Information about the DICOM file internal structure, like number of DICOM frames, can also be retrieved by analyzing the DICOM tags of said file.

The last, and the most taxing stage of a viewer session, is the retrieval of the frames of the DICOM file being analyzed, Frames on figure 17. The client browser will request as many frames as available on the DICOM file, dictated by the DICOM tags obtained before, so that the doctor conducting the analysis can view and analyze each one that is relevant to the diagnosis. The time spent by this stage is the sum of the time that takes the server to retrieve, convert and then serve, to the client, each DICOM frame of the DICOM file being analyzed.

While the first request for a viewer session might take more than a second, time spent by the server retrieving, converting and returning the requested DICOM frame, the consecutive calls for the same viewer session will take a significantly less amount of time to setup. This is mainly due to some optimizations made during the development of the solution (see 5.3), which allows the server and client-side caching of some resources, thus reducing greatly the time clients spent obtaining resources from the server.

Also, for DICOM files that contain more than one frame, it is now possible for the doctor to start analyzing frames that have already arrived, from the DicoogleWeb server to the doctor browser viewer session, before all of the frames have been converted and served. This is an improvement versus the old solution where doctors needed to wait before the complete DICOM file was downloaded from DicoogleWeb before they could conduct the analysis of said file.

5.2.3 DICOM File Transfer

Once again, both solutions continue to support raw DICOM file transfers from server to client, allowing doctors to download pertaining DICOM files in order to further analyze them onto another, more specialized, DICOM viewer, or to simply share them with another, unconnected, workstation.

The next picture, figure 18, presents a comparison of both interfaces performance in this area during consecutive file transfers:

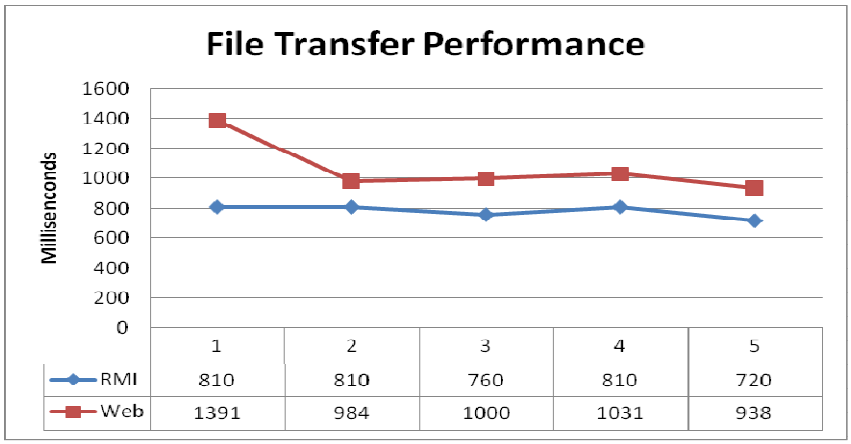


Figure 18: Comparison of RMI and Web interfaces performance during consecutive file transfers.

The, now deprecated, RMI interface still provides better performance in this area, due to the transfer of raw data packets across the network, without the HTTP protocol overhead of the web interface. Still, with only 20% less performance on average compared to the RMI interface, the web solution performance is well within the targeted performance for the new architecture.

This usage scenario, downloading DICOM files, will now be much less relevant on the new architecture than it was on the previous one, since now doctors will have the ability to analyze each DICOM file, by using the embedded DICOM viewer, without having to download it. This will provide a better overall performance and experience to system users (doctors).

5.2.4 GUI

One of the strongest points in favor of the web-based GUI is the openness of the interface, allowing the user to carbon-copy the data rendered on the interface. This was not possible on the previous interface, and can be very useful to medical staff, due to rather large DICOM tag values or patient information that might be needed as input values on other interfaces.

Another positive point is the readiness of said interface, which can be loaded, by each client, much faster than the previous desktop RMI interface.

This new web interface also allows external applications/tools to include the DicoogleWeb interface easily within their own interface. Be it a standard desktop or web

application, simply including a web-frame pointing to the DicoogleWeb server is sufficient to provide the full functionality of DicoogleWeb to clients.

5.3 Improvements

Given the current openness of the new platform, some improvements were, easily, added to the DicoogleWebs' implementation, that will greatly improve not only system usability but also decrease server load during high system usage. These will be discussed on the next points.

5.3.1 Frame Caching

Multiple doctors can be requesting and analyzing the same DICOM frames. This can happen, for instance, during a cooperative video call between doctors where they compare notes on the same frames. This will create multiple requests and image conversions for these same frames.

Since each DICOM frame that gets requested from the DicoogleWeb server must be first retrieved from the file, then converted to PNG and sent to the client, this creates a lot of stress on the server. Creating a simple caching mechanism that will keep recently converted DICOM frames, in a temporary disc folder, will greatly help the system by eliminating unnecessary and repeated DICOM frame conversions.

The current implementation of the server local caching of converted frames, will keep these served frames for a maximum non-usage period of time, thus decreasing the load put on the server during same study access by multiple clients/doctors.

5.3.2 Minimalistic JavaScript

Given JavaScripts' run-time evaluation and execution, its execution can be very taxing on client systems, especially mobile devices.

By keeping JavaScript code, and subsequent use of JavaScript libraries, like jQuery, to a minimum, relying on native browser capabilities and support whenever possible, we can minimize the loading times and reduce the stress that mobile devices will be put under when using the DicoogleWeb interface.

While this might add a small extra load on the server while creating some specific interface web pages, this will also greatly improve the overall user experience on mobile devices, by allowing almost instant page rendering and a lag-free user interface, as well as better battery life.

5.3.3 HTTP Caching

The HTTP protocol documents and supports client-side caching of previous server responses, allowing this data to be used by the client on future request copies, without the need to re-download the same data from the server.

Since most modern browsers are able to keep static content cached client-side, a small logic layers was implemented on some of the DicoogleWeb services, mainly the resource export ones, which will allow this data to be cached and managed client-side on/by each client browser, improving DicoogleWebs' response when a client re-requests the same data or information.

Also, since neither DICOM frames nor tags, within the same DICOM files, will be changed once they are archived within DocoogleWebs' embedded PACS core, this data, once returned to the client that requests it, can be cached client-side for future use.

An extra logic layer, that provides extra browser support while still maintaining full HTTP protocol compatibility, was then added on the DICOM frame and tags web services of DicoogleWeb. This layer will allow each web service to correctly process and response browsers HTTP HEAD requests for resource changes. Making the DicoogleWeb server client-side caching aware and allowing client browsers to cache said the returned information, if they please.

Unfortunately, since new DICOM files keep will entering the system, this same logic layer can't be directly linked to the search results web service, since caching these results can potentially void the ability of web clients to obtain all the valid search results for each query. Still, with a few modifications to the DICOM indexing mechanism, by creating a simple flag that will indicate when was the last file indexed, this caching problem can be bypassed, and thus allowing web clients to parse, for a finite amount of time, search results. As long as no new DICOM data enters the indexing mechanism, the system can assure that the clients cached search results will still be valid, until another

DICOM file is indexed by the server (this can be accomplished by adding the HTTP HEADER directive "`Cache-Control: must-revalidate`" to each search result web service response).

The HTTP specification also supports request pipe-lining, allowing parallel requests for resources to be made, instead of the traditional "wait for the previous response before conducting the next request".

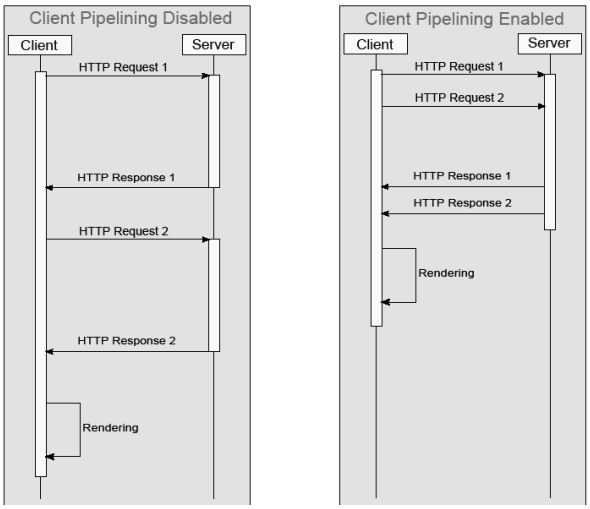


Figure 19: Comparassion between an HTTP pipelining disabled and enabled client.

Since most modern browsers also support HTTP pipe-lining, multiple DICOM frames and other resources can be downloaded from the DicoogleWeb server at the same time, as fast as the server will be capable to provide them, greatly reducing the client waiting time for resources, be it data or information. This will impact the embedded viewer greatly, and in a positive way since various DICOM frames can be downloaded in parallel, the viewer session data will be ready for rendering faster.

6 Conclusion

DicoogleWeb now features a GUI agnostic interface, meaning that using standard web requests, a whole range of various GUIs can be created. This means that 3rd party desktop applications for using DicoogleWeb can still be created, and will interact with the server via the defined HTTP request and response API for the various DicoogleWeb web services. This guarantees a new compatibility level between client systems that the previous Dicoogle version lacked.

The new revamped web architecture will, not only, allow platform developers to more easily request and consume information from DicoogleWeb, given the new open-standards based Web API, but will also allow medical personnel to use the system in a more streamlined manner. Meaning, doctors will spend less time waiting for resources to be prepared and presented to then, compared the previous Dicoogle solution, where there was the need for external tool to be loaded and pointed to each pertaining resource.

While the new DicoogleWeb solution provides the same, basic, functionalities of previous Dicoogle versions, the system can now be extended easily, without the need to deploy a new version of the solution across all clients. Meaning only the DicoogleWeb server needs to be updated to provide all the extra functionalities to any (web) client within the same medical institution. This will greatly speed up the adoption of new/improved/updated DicoogleWeb version by medical institutions, since there is no need to deploy the new version across the multitude of client devices within the institution.

Most of the goals set at the start of the project/paper were met, meaning that the new implementation is able to provide a complete storage/archiving, searching and analysis tool for medical imaging, allowing doctors to spend less time waiting till a solution is ready with the medical data to analyze, improving diagnosis performance. This will, ultimately, help the institutions were DicoogleWeb will be used. Doctors are now able to provide diagnosis a bit faster than with the previous solution, given that the solution can be accessible from any computing device with a network connection, even from outside of the medical institution. Also by removing the need to acquire other DICOM file analysis tools, the cost of operations of these institutions will also lower, while doctor and diagnosis productivity will still improve.

While the new architecture is well defined and provides great functionality, there are still, as with any on-going software solution, some implementation issues to be solved and further work to be put onto the solution, as with any software solution.

6.1 Implementation Issues

During the development of the new solution, some implementation related issues arose, most of which were solved. Some remained, mainly due to lack of developing time. Here is a list of the current implementation issues:

- The main issue with the new implementation, much like the previous one, is related to the fact that DicoogleWeb relies on a single server architecture, which make it a perfect target platform for Denial of Service (DoS) attacks.
- Another implementation issue was due to Jetty and the JSPF plug-in system configuration problems, which invalidated the possibility of implementing a MVC alike architecture for delivering web content. While this is currently a minor issue, it is possible that, with proper Jetty configuration options, this might not be an issue anymore.

6.2 Further Work

Some further work can be put into the current solution, in order to improve it. Here are some suggestions for what is yet to be implemented:

- Choose one of the available/concurrent interfaces for accessing plug-ins data export, making it the definitive one.
- Integrate the concurrent, and more complex, viewer onto the current architecture.
- Create, configure and deploy a certificate that will be used to cipher the Secure HTTP (HTTPS) connection between DicoogleWeb server and clients.
- Creating a translation layer between GUI injected code and the resulting code injection, fixing potential unclosed HTML tags and/or interface layout problems.
- Server-side caching of search results, taking into account the when the last search was made and if new DICOM files arrived to the PACS core since then.

7 References

[1, 3, 6, 9, 16-18]

1. Costa, C., et al., *Dicoogle-an open source peer-to-peer pacs.* Journal of Digital Imaging, 2011. **24**(5): p. 848-856.
2. Huang, H.K., *PACS and Imaging Informatics: Basic Principles and Applications*2010: Wiley.
3. Association, N.E.M., *Digital Imaging and Communications in Medicine (DICOM)*1993: The Association.
4. Anklesari, M., Lindner, Johnson, Torrey & Alberti. *The Internet Gopher Protocol*. 1993 [cited 2013 06-12]; Available from: http://www.ietf.org/rfc/rfc1436.txt.
5. W3C. *Hypertext Transfer Protocol - HTTP/1.1*. 1999 [cited 2013 05-06]; Available from: http://www.w3.org/Protocols/rfc2616/rfc2616.html.
6. Wikipedia. *Usage share of web browsers*. 2013 [cited 2013 05-06]; Available from: http://en.wikipedia.org/wiki/Usage_share_of_web_browsers.
7. ISO/IEC, *Information technology - Document description and processing languages - HyperText Markup Language (HTML)*, 2000.
8. W3C. *HTML 4.01 Specification*. 1999 1999 [cited 2013 06-06]; Available from: http://www.w3.org/TR/REC-html40/.
9. Dreyer, K.J., A. Mehta, and J.H. Thrall, *PACS: a guide to the digital revolution*2002: Springer Verlag.
10. Biedert, R. *jspf - Java Simple Plugin Framework*. 2011 [cited 2013 25-01].
11. Foundation, T.A.S. *Apache Lucene*. 2013 [cited 2013 25-01]; Available from: http://lucene.apache.org/.
12. Ferreira, C.A.M.V., *Peer-to-peer network for medical imaging*, 2010, Universidade de Aveiro.
13. Firtman, M. *Mobile HTML5 - compatibility*. 2013 [cited 2013 03-06]; Available from: http://mobilehtml5.org/.
14. Hewlett-Packard. *License Of Certain Hewlett-Packard Patents Relating To Lossless and Near-Lossless Image Compression*. 1995; Available from: http://www.hpl.hp.com/research/info_theory/loco/JPEGLSTerms.htm.
15. Go Pivotal, I. *Spring Framework*. 2013 [cited 2013 05-06]; Available from: http://www.springsource.org/spring-framework.
16. Costa, C., et al., *Indexing and retrieving DICOM data in disperse and unstructured archives.* International journal of computer assisted radiology and surgery, 2009. **4**(1): p. 71-77.
17. Koutelakis, G.V. and D.K. Lymperopoulos. *PACS through web compatible with DICOM standard and WADO service: advantages and implementation*. in *Engineering in Medicine and Biology Society, 2006. EMBS'06. 28th Annual International Conference of the IEEE*. 2006. IEEE.
18. Pianykh, O.S., *Digital Imaging and Communications in Medicine (DICOM): A practical introduction and survival guide*2011: Springer.