J Reliable Intell Environ (2015) 1:173–187
DOI 10.1007/s40860-015-0007-1

ORIGINAL ARTICLE

# A secure, scalable and versatile multi-layer client–server architecture for remote intelligent data processing

**Gabriele Piantadosi**[1] · **Stefano Marrone**[1] · **Mario Sansone**[1] · **Carlo Sansone**[1]

Received: 29 July 2015 / Accepted: 15 October 2015 / Published online: 5 November 2015


**Abstract** In recent years, the need for data collection and analysis is growing in many scientific disciplines. This is consequently causing an increase of research in automated data management and data mining to create reliable methods for data analysis. To deal with the need for smart environments and big computational resources, some previous works proposed to address the problem by moving on remote processing, with the aim of sharing supercomputer resources, algorithms and costs. Following this trend, in this work we propose an architecture for advanced remote data processing in a secure, smart and versatile client–server environment that is capable of integrating pre-existing local software. In order to assess the feasibility of our proposal, we developed a case study in the context of an image-based medical diagnostic environment. Our tests demonstrated that the proposed architecture has several benefits: increase of the system throughput, easy upgradability, maintainability and scalability. Moreover, for the scenario we have considered, the system showed a very low transmission overhead which settles on about 2.5 % for the widespread 10/100 mbps. Security has been achieved using client–server certificates and up-to-date standards.

**Keywords** JAAS · DCE-MRI · OsiriX · Secure · NIST · Biomedical · Image processing · TLS/SSL

## 1 Introduction

As technology advances at a rapid pace, data collection and analysis is becoming more important. Research on automated data management (storage and acquisition) and data mining has increased with the aim to create more reliable methods [1]. For example, in retail business, Wal-Mart Stores, Inc. handles more than 1 million customer transactions every hour. A data warehouse estimated to contain more than 3 PB of data shows record of every single purchase by their point-of-sale terminals in each of their 6000 stores worldwide. By applying machine learning on this data, they can extract patterns indicating the effectiveness of their pricing and advertising strategies ensuring a wide knowledge for better inventory management, supply, promotion and advertising campaign [2]. Medical diagnostic imaging usually requires massive processing of huge data, often with strong temporal deadline constrains. For example, a small clinical centre equipped for Magnetic Resonance Imaging (MRI) can produce up to 20–25 MRI scans per day, resulting in about 4 GB of raw data per day, which can easily tenfold after processing. Such amount of data is likely to quickly grow because of diagnostic imaging technologies improvements. Local workstation cannot handle resource-consuming processing on big dataset without involving a large amount of time and vice versa due to the limited hardware capabilities. The constraints for medical image processing in small and medium companies are the costs (for several advanced workstation) and the

**Electronic supplementary material** The online version of this article (doi:10.1007/s40860-015-0007-1) contains supplementary material, which is available to authorized users.

✉ Carlo Sansone
carlosan@unina.it; carlo.sansone@unina.it

Gabriele Piantadosi
gabriele.piantadosi@unina.it

Stefano Marrone
ste.marrone.g@gmail.com

Mario Sansone
mario.sansone@unina.it

[1] Department of Electrical Engineering and Information Technology (DIETI), University of Naples Federico II, via Claudio 21, 80125 Naples, Italy

required computational power which is not readily available on conventional medical workstations.

To deal with the need of a smart environment and big computational resources, some previous works proposed to address the problem moving on remote processing, with the aim of sharing supercomputer resources, algorithms and costs. In literature, various frameworks can be found that aim to apply remote processing and modular composition to various research fields.

In this work we propose an architecture for remote data processing enabling pre-existing local applications to access remote intelligent environments. The proposed infrastructure has been applied on a medical image analysis environment. This allowed us to address some additional problems that are also strongly linked with medical data and their analysis. As shown in the next section, general approaches often forgot these important points:

- Data analysis systems must be able to interact with different software (acquisition and examination software for the medical case study) and instrumentation (different vendors equipment);
- Context scalability and versatility (to meet potential growing of requests);
- Operational time (a strict constraint such as clinical environment time);
- Data sensitivity (privacy must be guaranteed).

Those requirements pushed the research towards the development of systems, infrastructure and architecture able to operate on big data analysis and, at same time, managing computational complexity, scalability, upgradability and costs [1].

The aim of this study is to propose an architecture for advanced remote workflow execution in a secure and versatile client–server intelligent environment that is capable of interacting with a widespread software. In particular, we propose an architecture based on four ideas:

- It is crucial that operators can continue to make use of their usual software and tools to encourage the spread of the proposed architecture without impacting the learning curve;
- The architecture should be as independent as possible from the particular implementation, allowing server-side to enlarge or evolve according to client needs;
- Operators should not be forced to wait until the job ending. It is very important that users can continue their work, receiving information about the status of remote processing;
- Data transmission should meet up-to-date security standards.

This means that the proposed architecture should not be considered as a batch execution software by itself, but as a remote support tool designed to extend the pre-existent local software (i.e., client-side medical image processing software in the considered case study) with new advanced functionalities enabling access to intelligent environments.

The paper is organized as follows: in Sect. 2, we discuss some related works in remote data processing as well as trends in clinical environment. In Sect. 3, we present the proposed system starting from the layers overview and then focussing on a practical implementation in the specific context of medical image processing and automatic breast lesion detection by a dedicated CAD system (Sect. 4). The obtained results are presented in Sect. 5 and discussed in Sect. 6, where we also draw some conclusions. Finally, in Sect. 7, we shortly explain how to access our architecture implementation.

## 2 Related works

Some studies proposed very general purpose architecture for remote processing in different fields (domain-independent frameworks). *Triana* [3] is a visual workflow-based problem solving software, developed at Cardiff University. Originally, it was developed for a gravitational wave detection project. Subsequently, it has been extended to incorporate a range of modules, such as peer-to-peer communication, grid services and web services integration with the purpose of providing a wider integration with existing grid technologies [4]. The *Kepler Project* [5] is a scientific project that offers construction, composition, and orchestration engine. The focus is on data analysis and modelling, which influenced the design in that it is suitable for modelling processes in a wide variety of scientific domains from physics via ecosystems, to bioinformatics web services. Instead of trying to provide a generic semantic for all possible types of processes encountered in these domains, Kepler separates the execution engine from the workflow model and assigns one model of computation to each workflow [4]. The *LONI Pipeline Processing Environment* [6] is a java-based modules oriented visual programming interface for the simplified design and execution of remote pipelines. It was initially proposed to investigate for image processing in brain mapping. The *Taverna Suite* [7] is a powerful scientific workflow management system born to satisfy the needs of bioinformaticians who need to build scientific workflows from numerous remote web services. It offers a graphical designing tool, an authoring client, a workflow representation language and different additional components such as a service directory, data and meta-data repositories and others.

All are able to design, execute and manage complex and heavy workflows helping developers to move remote supercomputers, parallel architectures or cloud system. This

usually requires a migration from the pre-existent local software to a tool which is more compliant with the architectures. The server-side workflows need to be adapted or even rewritten to be compatible with the architecture language. Moreover, security and privacy issues, associated with the cloud computing of sensitive data, such as biomedical data [8], have been raised.

The case study addressed in this work can be considered part of a wider field known in literature as "Computer-Aided Detection (CADe)" or "Computer-Aided Diagnosis (CADx)". These systems can support physicians in the examination of medical images both automatically detecting suspicious regions of interest (ROIs) and suggesting diagnosis. To handle the large amount of data typically managed in radiological units and to provide more efficient services, different CAD architectures have been proposed in the literature. Scheinine et al. [9] proposed an Object-Oriented client–server System for Interactive Segmentation of Medical Images based on JAVA for the client and CORBA for the distributed system, connected by a TCP/IP socket protocol. Mayer et al. [10] implemented a "processing on demand" client–server architecture for 3D image processing in which the computation load is all on the server-side, while the client requests the desired images one slice (2D) at a time.

Yacoub et al. [11] presented an evolution of the open standard DICOM able to support communication between DICOM entities over a TCP/IP network. A more modern architecture has been proposed by Crane et al. [12] integrating the University of California at San Francisco Radiology Department's magnetic resonance (MR) scanners with its high-performance computing (HPC) grid. With the diffusion and the improvement in client-side software such as web browsers, a low-impact proposal has been assessed by Mahmoudi et al. [13] with a web oriented visualization software.

Some of above cited papers were mostly designed for computing systems belonging to several generations ago. Moreover, to the best of our knowledge, a complete remote and scalable CAD system has never been proposed.

## 3 The proposed architecture

In this paper, we propose a generic architecture for remote processing of data, with a special attention to security and scalability issues. Moreover, a preliminary implementation of our proposal has been realized using low-impact technologies and up-to-date standards.

We designed the system with the following three constraints in mind:

- easy integration into third-party client-side software;
- TCP/IP network infrastructure;
- up-to-date open standards.

To provide easy integration, we have chosen to divide responsibilities between the pre-existent client-side software and the client-side application of proposed architecture: the first one has to interact with the user, making all operation required to interact with proposed architecture totally transparent to the operator, while the second one has to manage the data transmission and results reporting. The proposed strategy allows meeting the easy integration goal as long as the third-party client implements (with the aim of a plug-in, API or other form of customization) the steps required to interact with the proposed system client-side application.

Next, we present the general structure and a case study implementation.

### 3.1 Architecture layers

The system is organized as a multilayer client–server architecture implementing a multi-client/single-server model. Within the same node, different layers communicate using the file system (see Fig. 1).

Each layer has a well-defined role:

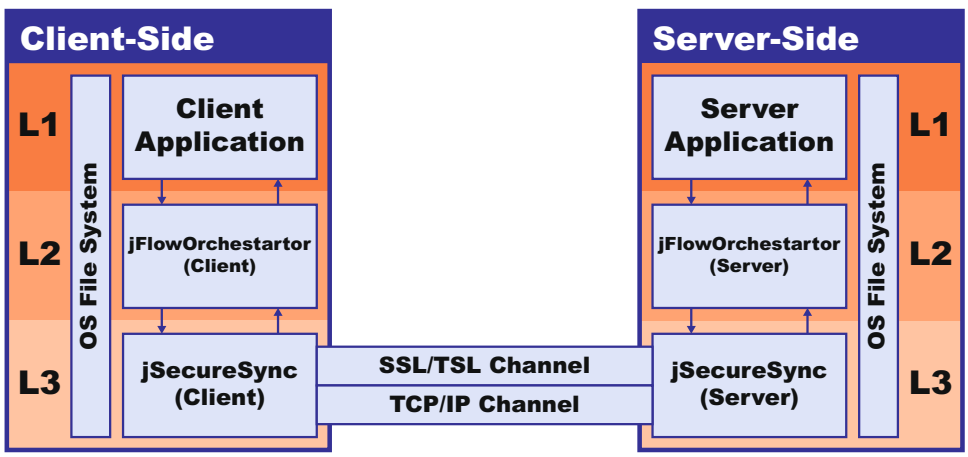


**Fig. 1** Layers overview of the proposed architecture that provides: communication (*L1*), orchestration (*L2*) and application (*L3*)

#### 3.1.1 L1—communication

This layer manages every detail of transmission and synchronization (security, physical interconnection, etc.). It creates communication channels between client and server for data transfer.

#### 3.1.2 L2—orchestrator

This layer manages communication flow between client and server. It adapts the data generated from the application on the client (layer 3) in a suitable format to be adequately processed by the application layer on the server-side. This layer manages the job required by the client and sends notifications in regards to the job's status.

#### 3.1.3 L3—application

This layer represents the application used both on client and server-side. On the client, it represents the pre-existing image processing software plus our client-side tools, while on the server-side it is composed of all software, tools and means needed to provide required operations done, if necessary, as parallel operation.

It is worth to underline that the Communication layer can be completely implemented without knowledge of the final context and application. The Orchestrator layer can be only partially decoupled from the specific context, while the Application layer cannot be delineated without a specific context and application.

### 3.2 Communication layer

The component which implements connection establishment and management, command transmission and file-state synchronization is called *jSecureSync*. It has been coded in Java in order to increase portability.

Running both on client-side and on server-side, it implements the multi-client/single-server model: the client starts communication, then the server creates a new thread for each accepted incoming connection. Once communication is established, client and server-side operate symmetrically: on both sides jSecureSync runs a Connection Manager module that manages every aspect of file synchronization over the opened channel. After the Connection Manager starts, client and server act like peers.

Client and server modules establish two different channels (Fig. 1) exploiting Java TCP/IP sockets:

– a non-secure channel (via plain TCP/IP): it is used to transmit heavy data (like volume data) and does not carry private sensible information;

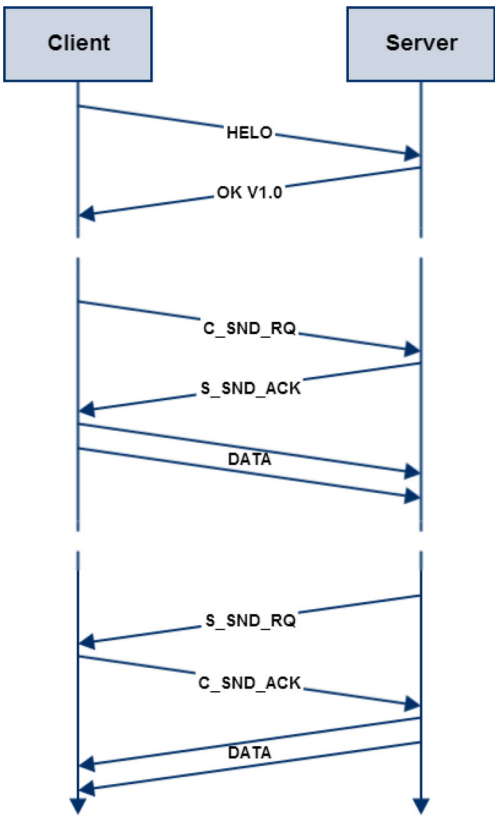


**Fig. 2** Client–server communication protocol timing

– a secure channel (via SSL over TCP/IP): it is used to transmit commands and private sensible information.

Clients and server communicate through a protocol with low overhead (coded in hexadecimal format). Figure 2 presents the protocol commands:

– *HELO* Client asks to open a channel (for synchronization process) with the server;
– *ALIVE* Used by peers to communicate, to be active, or to ask for information about the status of the other peer;
– *OK v1.0* The server acknowledges a client request and, at the same time, it instructs the client on the version of the protocol used;
– *SND_RQ* The peer requests to synchronize a file over the secure channel (SSL over TCP/IP);
– *INSECURE_SND_RQ* The peer requests to synchronize a file over the non-secure channel (plain TCP/IP);
– *SND_ACK* Acknowledges to allow the other peer to send over the secure channel (SSL over TCP/IP);
– *INSECURE_SND_ACK* Acknowledges to allow the other peer to send over the non-secure channel (plain TCP/IP);
– *CLOSING_CONNECTION* Calling peer informs the other one that after this command it will stop the communications by closing all channels (ack is not requested).

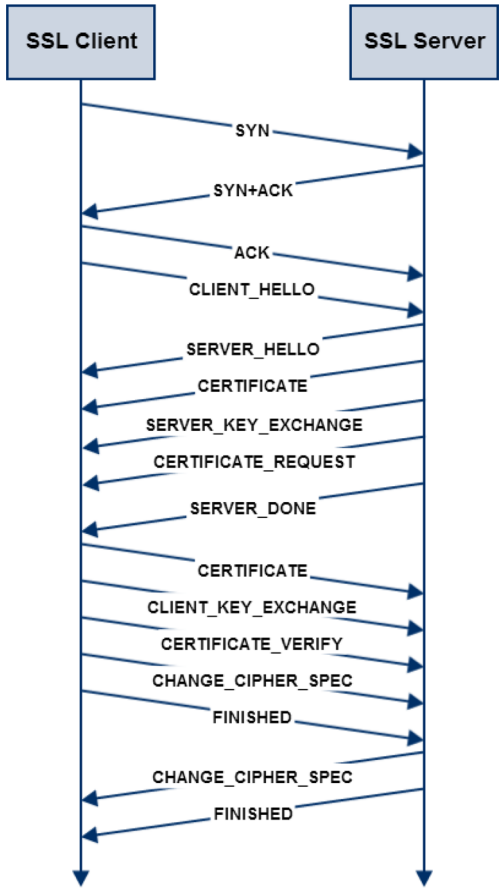


**Fig. 3** SSL client-authenticated mode timing protocol

SSL/TLS channel security has been realized using a symmetric authentication policy (client-authenticated mode [14–16]), issued during user registration as an authorized client (see Fig. 3). This means that every new client has to successfully achieve a one time registration step to be recognized as a real, authorized client and receive its couple of security certificates (one to verify server authenticity, and the other as proof of being a trusted client to the server). Certificates contain a Public Key and a Signature, both used to identify the user on server-side and to grant access to server features. The procedure for authentication and authorization through certificates is achieved by means of Java services using Java Authentication and Authorization Service (JAAS) [17,18] classes. This choice guarantees compatibility over different networks architecture supporting the TCP/IP stack protocols and portability (all virtualization is made by the Java Virtual Machine). Our implementation uses elliptic curve Diffie–Hellman (ECDH) [19,20] as session key agreement protocol, while the channel is secured by using an AES 128bit encryption with Cipher Block Chaining (CBC) as cipher block modality [16,21]. The use of client-authenticated mode allows the server to extract client credential directly from SSL session, making authentication and authorization step achievable through JAAS services without using an ad-hoc access protocol. Digital certificates are provided according to X.509 standard and key pairs are encrypted with a 3072bit RSA algorithm. The SSL encoded configuration string is "TLS_ECDHE_RSA _WITH_AES_128_CBC_SHA" [22,23].

JSecureSync does not perform any kind of analysis on the file to be transmitted. It only synchronizes all files in a specific folder (between client and server), providing a trusted isolated server disk quota to every client.

Before sending, every file undergoes a compression. The compression phase can affect both transmission time (heavy compression files are smaller and require lower transmission time) and required computational power (heavy compression algorithms often require dedicated computational capability).

### 3.3 Orchestrator layer

The middle layer (L2) has been implemented as an abstract object (jFlowOrchestrator) that provides all abstract methods to perform brokerage between L3 and L1, by orchestrating the steps of each request and the server-side computational flow progress. It also provides adaptation for the application layer (L3) data inputs and outputs: different software can run on the application level both on server and client-side requiring different file format or data arrangement. For example, as in the case study we will present in another section, a medical image software could require a remote processing of a medical study. However, it is not required to send all the data to the CAD server (e.g. private information concerning patient name, age, etc. must be dealt with separately). Exploiting the underlying layer (L1) services, the Orchestrator layer (L2) takes charge over the adaptation and coordination of the data flows. The jFlowOrchestrator provides three main abstract methods:

- *bool fileFilter* (*File file*)
  This method is used to define which kind of element (including file name, file type, folder, etc) needs to be monitored for changes
- *bool serveFileFromPeer* (*File file*)
  This method is used every time a monitored file is modified by the other peer
- *bool serveFileToPeer* (*File file*)
  This method is used every time a monitored file is modified by the higher (L3) layer.

### 3.4 Application layer

The top layer (L3) has to be implemented according to client-side software and server-side advanced processing software and tools. Client and server-side software must provide a

way to implement a communication with L2 layer (over file system) to:

- request a job (client);
- serve a job request (server);
- execute progress synchronization (client and server);
- provide results (server);
- retrieve results (client).

## 4 A case study

### 4.1 Clinical scenario

Dynamic Contrast Enhanced Magnetic Resonance Imaging (DCE-MRI) is a widespread methodology for cancer detection and evaluation. In particular, in this study, we focussed on the breast DCE-MRI which has a large potential in early detection of cancer and therapy response assessment, especially for young women.

Due to the large amount of data acquired within a DCE-MRI session, the physician has to deal with the complex task of accurately detecting suspicious ROIs and evaluating tumour aggressiveness using 4D data (3D spatial data and 1D time data) [24,25].

In this context, many scenarios could be imagined in which the proposed architecture could be useful: let us consider the processing of multiple data corresponding to the same patient (patient follow-up) or the concurrent processing from multiple radiologists within a single medical centre.

We now describe the implementation of our proposal realized to support this application context.

### 4.2 Orchestrator layer implementation

In medical image analysis context, all abstract methods have been implemented to work on biomedical image data and server processing software: the resulting module is called *jM-FlowOrchestrator*.

In this context it is worth mentioning that DICOM is the de-facto standard for biomedical images [26]. It is a strongly structured file composed of image row data and patient meta-data used in human and veterinary medicine diagnostic imaging. The DICOM standard supports the handling, storing, printing, and transmission of medical imaging. It includes a file format definition and a network communications protocol. The communication protocol is an application protocol that uses TCP/IP to communicate between systems. We intentionally decided to not use DICOM standard transmission features with the aim of decoupling proposed architecture effectiveness from any particular application context.

For advanced data processing on the server-side, at layer L3, we have chosen to use MATLAB [27] (R2013b version) because (i) it requires no special handling of larger sized vector or matrix (images), (ii) it has advanced ToolBoxes (Image processing, Parallel and Distributed Computing, Artificial Intelligence) and (iii) it has the ability to invoke external classification and data mining tools (Fig. 6). In particular, in our previous work [25] we implemented a set of methods to interact with Weka [28] machine learning and data mining suite. This choice makes it possible also to parallelise client request executions by different MATLAB instances, each strongly bound to a specific CPU using MATLAB Parallel Toolbox. It is worth noting that the MATLAB environment has to be installed only on the server-side, while on the client-side the pre-existing software can still be used. According to the DICOM standard the image data can be compressed using a variety of compression standards, including JPEG, JPEG Lossless, JPEG 2000, and Run-length encoding (RLE). In the proposed architecture, signal intensity data are compressed as 4D volume to exploit spatial correlation between near slice of the same time series and temporal correlation between the same slice across different time series. For this reason, the architecture always has a compression stage, independently on the image format used in the DICOM file. Also we observed and properly motivated that the format file required by remote processing tools is achieved with the appropriate organization of the data for a better compression ratio (see Fig. 7). For this reason DICOM files, after a suitable processing step, are compressed in MATLAB format. To achieve MATLAB compression without the MATLAB software, the open-source Java JMatIO library was used.

On the client-side, jM-FlowOrchestrator performs:

- DICOM file analysis to generate two different files: one for volume data ("volume.mat" containing only signal intensity) and one containing only meta-data ("patient.mat" containing only privacy sensible information). Both files are directly stored in MATLAB file format and sent to client by exploiting the potential of the lower layer (jSecureSync);
- Update L3 on the actual processing state (for example, the remaining time) of the requested feature on server-side.

On the server-side, jM-FlowOrchestrator performs:

- Data preparation according to server L3 application layer's required formalism (as described in Sect. 4);
- MatLab results, data gathering and transmission to server-side L1 layer;
- Client-side L2 layer status updating.

Remote processing specifications (both job required and status updates) are memorized in an XML file transmitted over the secure channel together with MATLAB formatted files

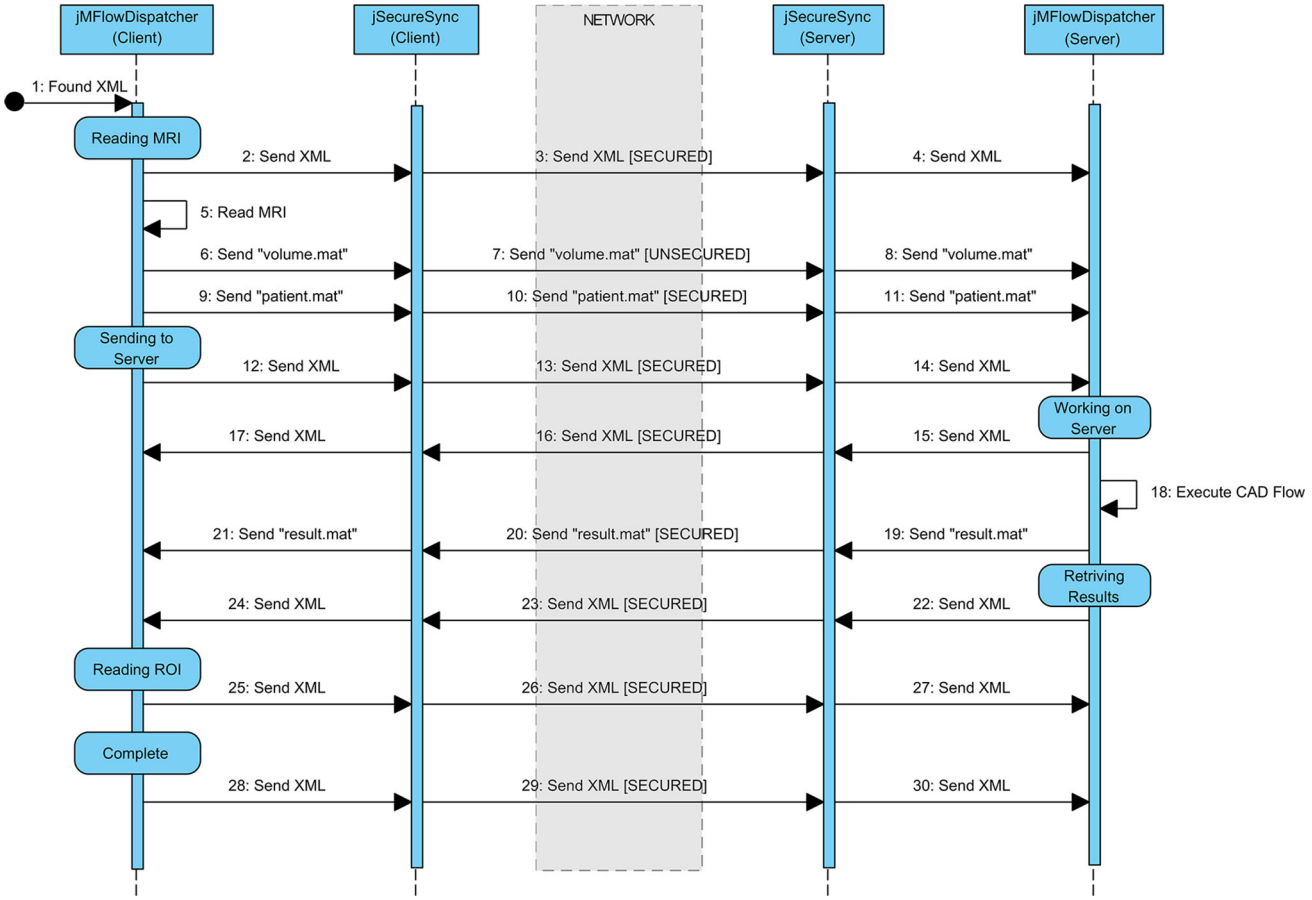


**Fig. 4** Component level sequence diagram for a single job request execution. *Rounded rectangles* on temporal line represent state changing

containing sensitive data. That file is periodically exchanged between client and server.

A single DCE-MRI study can contain as many as 800 DICOM files. Each file contains a single 2D slice and the meta-data DICOM tag about the study and the patient. By exploiting the potentiality of the lower layer (secure and non-secure channel offered by L1), the DICOM files are split. The volume information extracted for all the DICOM files of the same patient consist of 12 bit coded signal intensity for each voxel and is stored in a 4D matrix. Stripped of all sensitive information, it can be send over the non-secure channel (step 6 of sequence diagram in Fig. 4). The meta-data provided by DICOM format normalized (often with an high internal redundancy and with information that is not strictly necessary for the purpose of processing) and compressed (in MATLAB format) are prepared to be transmitted over the secure channel (step 9 of sequence diagram in Fig. 4). All compression tests on patient data were performed considering both meta-data and signal intensity as 4D volume. Results show as MATLAB achieves a better compression ratio (CR) than the Zip4j protocol (the runner up), exploiting the spatial and temporal correlation typical of DCE-MRI images, and the MATLAB sparse-matrix data representation. For those reasons, MATLAB compression is chosen at the middle level (L2 jM-FlowOrchestrator) only for the DICOM content (both on images and meta-information). Other file types (XML) are compressed using Zip4j protocol to preserve their human-readability.

The component level sequence diagram in Fig. 4 shows the message passing protocol sequence in the described context for a flow execution scenario:

- Rounded rectangles over the temporal line represent state changing;
- Intra layer communication (a different layer on the same machine) is performed by using file system services;
- Inner layer communication (same layer on a different machine) is performed through communication channel provided by L1 layer.

### 4.3 Application layer implementation

On the client-side, we decided to support the widespread OsiriX medical image tool [29], due to its powerful interface and since it has an open plug-in system.

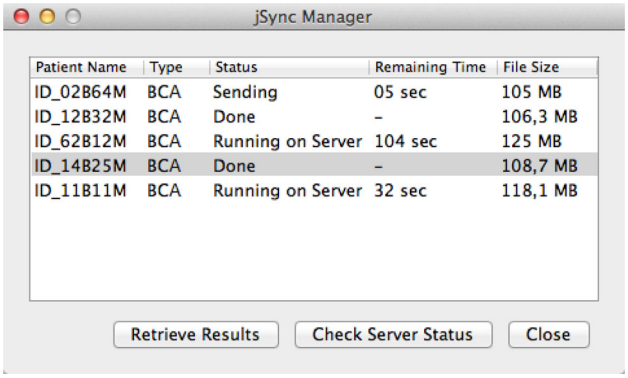


**Fig. 5** OsiriX GUI example showing how the user can check the status of each job and can retrieve results within OsiriX main window

In particular, we developed an OsiriX plug-in capable of interacting with jM-FlowOrchestrator (using the file system), to give the end user the ability to:

- Export selected patient DICOM files to jM-Flow Orchestrator;
- Use GUI for required operation on server-side, with the possibility to change any parameter;
- Progress display for each required operation;
- Import obtained results from jM-FlowOrchestrator and view them by using the powerful OsiriX interface.

The plug-in allows the user to send patient's data (selecting the required operation), to check in-service operation (with an estimation of the remaining time) and to gather the results. On the arrival of the results, the system alerts the user by means of a pop-up. Clicking the 'Retrieve Results' button (Fig. 5), the OsiriX 3D viewer interface is loaded and the results are shown (in this case, the automatically detected lesions).

On the server-side, we developed an advanced modular system for the waterfall execution of any job flow (made up of subsequent steps) called *MatlabFlowManager*. Many simple modules can be combined to form a more complex system easily through a simple XML flow configuration file. The XML configuration file contains the module internal setting and the module interconnection setting. In our previous studies [24,25], we presented a system for automatic lesion detection in breast DCE-MRI based on a SVM classifier (Breast Lesion Automatic Detection System, BLADeS). It has been used in this work as a case study (Fig. 6).

The DCE-MRI data segmentation flow processing defined, in accordance with BLADeS, is achieved by means of the followings steps:

- 4D volume extraction from patient DICOM images;
- BreastMask extraction (a binary mask representative of voxel belonging to breast parenchyma, excluding background, bone, skin and pectoral muscles voxel);
- Pre-selection Mask (a binary mask that excludes non-suspect voxel using an approach presented in our previous works [25,30–34];
- Feature extraction;
- Classification (using weka [28] and libsvm [35] packages).

Each step is obtained using one or several modules developed in MATLAB. The specific XML file that is invoked for each execution of BLADeS calls the following modules:

- *BreastMask extraction* reads the MATLAB formatted files sent by the client and performs the BreastMask extraction procedure. Saves a binary 3D matrix on the File System;
- *Image pre-processing* reads the MATLAB formatted files and performs a median filtering (with a 3 px windows) according to our previous work [25]. Saves a new MATLAB file containing the pre-processed 4D volume.
- *Voxel pre-selection* reads the pre-processed MATLAB file and the BreastMask MATLAB file and performs a voxel-by-voxel pre-selection. Saves the pre-selection

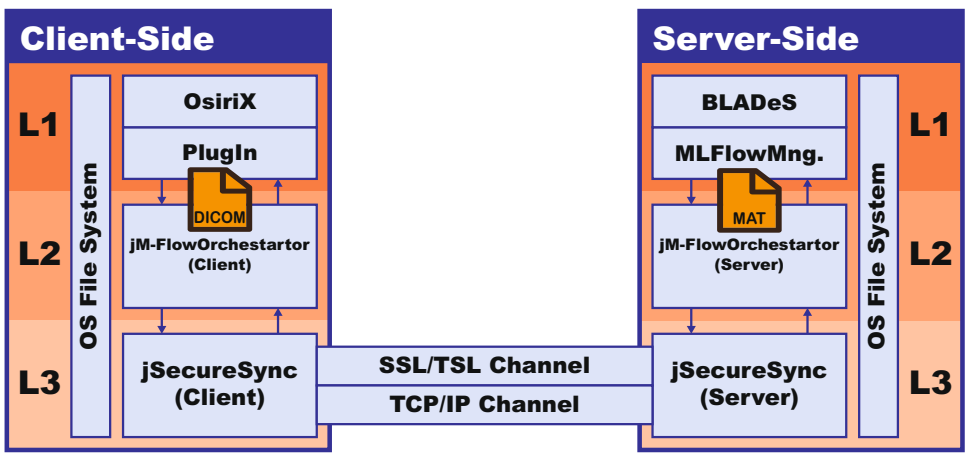


**Fig. 6** The proposed architecture in the specific context of DCE-MRI evaluation, by using OsiriX and a previously proposed CAD tool (BLADeS [24,25])

mask (a binary 3D mask that marks only the suspected voxel) [30–34];
- *Feature extraction* reads the pre-processed MATLAB file, the pre-selection mask MATLAB file and the list of feature to be extracted. Saves a MATLAB file containing the dataset to classify;
- *Classification* performs the classification using weka [28] and libsvm [35] packages. saves the results into a new MATLAB file.

### 4.4 Architecture behaviour

To better understand the whole architecture, we present a step-by-step scenario of the medical context.

The physician loads the patient DCE-MRI data (from the local files or from the clinical PACS server) into OsiriX; then loads the installed plug-in, selects the remote required task in a a drop-down menu (in that case, the name of the remote task that implements BLADeS on the server was "BCA"—"Breast Cancer Analysis") and starts a new remote job request. The plug-in takes into account the DCE-MRI data as a whole and copies the DICOM files into the specific folder of the Java client-side of the architecture by creating a new sub-directory named with a new unique ID. Finally, the plug-in also creates an XML file containing the required remote job, and the status "Reading MRI". The folder monitor daemon, implemented into jM-FlowOrchestrator (L2), catches the XML file (step 1 of Fig. 4) and acquires the DICOM files (step 5 of Fig. 4) by creating two different files in the MATLAB format: one file ("volume.mat") containing the whole 4D volume of the DCE-MRI image data without any patient information or any kind of private data, and one file ("patient.mat") containing the sensitive data (such as patient name, age, height, weight, etc.). jM-FlowOrchestrator (L2) saves each of these files (the XML file—step 2 of Fig. 4—and the 2 MATLAB formatted file—steps 6 and 9 of Fig. 4) into the folder of the lower level (L1—jSecureSync), marking the XML and the "patient.mat" file as secure and "volume.mat" as non-secure. The folder monitor daemon, implemented into jSecureSync (L1), as in the upper level, catches that files and transmits over the right channel (either secure or non-secured) previously instantiated with the server-side according to the remark of the upper Layer (steps 3, 7 and 10 of Fig. 4).

On the server-side, the jSecureSync (L1) module receives all three files and moves them to the upper level folder (steps 4, 8 and 11 of Fig. 4). Into the jM-FlowOrchestrator (L2) server module, unlike what is performed on the client-side, no specific operation is performed but simply the files are moved from the lower transmission level to the upper application level starting the required task (in the specific context the CAD segmentation workflow—step 18 of Fig. 4).

Each status update is performed by updating the XML file and re-syncing it between the peers. For example, once the jM-FlowOrchestrator (L2) server module starts the workflow execution, the status into the XML changes into "Working on Server" and the folder monitor daemon of each level provides to move the XML file through each level and, of course, between peers (steps 15, 16 and 17 of Fig. 4). Each GUI refresh shows the punctual status of each job request (Fig. 5).

Once the required task is completed, the result is saved on the server-side (into MATLAB format) and, with the same strategy of the client-to-server communication, is moved to the client-side (steps 19, 20 and 21 of Fig. 4) where jM-FlowOrchestrator (L2) performs a conversion from the MATLAB format to the XML format. It is then ready to be retrived into the OsiriX software upon user request.

Furthermore, once the GUI receives an XML file containing the status "Retrieving Results", it alerts the user with a pop-up message.

With respect to the execution time, the whole architecture, therefore, only adds the time needed to transmit files and adapt data formats from client to server and vice versa. Transmission times include: time needed to adapt data from both applications layers; compression and decompression times and the actual sending time via TCP or SSL channels. In the performance assessment, we also focused on how much the architecture impacts the execution time by evaluating the overhead percentage.

### 4.5 Exploit parallelism

For the specific case study, the CAD system for automatic lesion detection in the breast DCE-MRI we deployed on the server-side is not yet ready to be directly parallelised. On the other hand, the architecture is able to schedule different executions of the same CAD elaboration, one for each request. Each MATLAB instance is bound to a single job request and to a specific CPU with the aim to improve the local performance. In this way, the architecture is able to increase the overall system throughput, and also the one of a non-parallelised service. The jM-FlowOrchestrator (L2) together with MatlabFlowManager (L3) takes care of the binding with the local CPUs or the dispatching of the job to a least busy server, exploiting the parallelism in a cluster manner too.

### 4.6 Performance assessment

#### *4.6.1 Data compression tests*

We compare different compression protocols: Java Native Zip library, Lempel–Ziv–Markov (LZMA), the Zip4j open-source library and MATLAB file format compression. The Zip4j library has been chosen due to its large diffusion. It provides several levels of compression [(0) store, (1) fastest, (3) fast, (5) normal, (7) maximum, (9) ultra] requiring different times. Of course, the compression ratio and compression

times affect the transmission times. Therefore, in order to choose the optimal compromise between compression ratio and the overall transmission time, we have tested several compression levels evaluating the compression ratio (CR) as in the Eq. (1):

$$\text{CR}\ (\%) = \left(1 - \frac{\text{compressed size}}{\text{original size}}\right) \times 100 \quad (1)$$

As the DICOM files include private information that must be sent on the secure channel (see Sect. 3.2), we did not directly send the compressed DICOM images.

Instead, the Orchestrator layer (see Sect. 3.3) extracts meta-information, sending it only one time for each patient, before signal intensity 4D data. This has the advantage of not sending the same information multiple times, i.e. the patient meta-data.

#### *4.6.2 Transmission and execution tests*

To evaluate transmission and execution times, we compared the total execution times made up of both transmission times (from client to server and vice versa) and server computational time.

The transmission time includes all the operations required by the proposed architecture (such as the compression/decompression time or the time needed to adapt data from both application layers) to move the execution of the required job from a local context to a remote environment.

Moreover, the overhead (OH) has been calculated according to the equation 2:

$$\text{OH}\ (\%) = \frac{\text{Transmission Time}}{\text{Transmission Time} + \text{Execution Time}} \times 100 \quad (2)$$

#### *4.6.3 Scalability test*

To evaluate performance and feasibility of the proposed system, local execution vs. remote server execution has been evaluated. We compared the total execution time (constituted by client-to-server transmission time, server processing time, server-to-client transmission time) in the client–server architecture with the time needed by the same operation when it is entirely performed on physician's workstation.

Scalability has been tested by evaluating the system throughput (number of produced output per time unit, considered as the number of results produced by the system in one hour under a huge load regime) in four different configurations:

- Local: job is entirely performed on the radiology workstation;
- 1 CPU: job is performed through server using one CPU;
- 2 CPUs: job is performed through server using two CPUs;
- 4 CPUs: job is performed through server using four CPUs.

Throughput is defined in Eq. (3) (for the local configuration) and in Eq. (4) (for the remote configurations) and the unit of measurement is: $\frac{\text{jobs}}{h}$.

$$\text{TP}_{\text{local}} = \left(\frac{1\ \text{hour}}{\text{single\_job\_execution\_time}}\right) \quad (3)$$

$$\text{TP}_{\text{server}} = \left(\frac{1\ \text{hour}}{\text{single\_job\_execution\_time}}\right) \times \#\text{CPU} \quad (4)$$

For each remote configuration, the speedup has been calculated according to Eq. (5) as proposed in [36].

$$\text{speedup} = \frac{\text{TP}_{\text{remote}}}{\text{TP}_{\text{local}}} \quad (5)$$

## 5 Case study results

### 5.1 Patient dataset

The dataset used for the following tests included breast DCE-MRI data from 33 patients (average age 40 years, in range 16–69) with benign or malignant lesions histopathologically proven: 19 lesions were malignant and 14 were benign. Although the system architecture is independent from the data acquisition protocol, in order to evaluate its performance in a real case, we considered its application to the analysis of breast DCE-MRI T1-weighted FLASH 3D coronal images (TR/TE 9.8/ 4.76 ms; flip angle 25°; field of view 330 × 247 mm × mm; matrix 256 × 128; thickness 2 mm; gap 0; acquisition time 56 s; 80 slices spanning the entire breast volume). For each patient, ten series were acquired: one series ($t_0$) was acquired before and nine series ($t_1$–$t_9$) after the intravenous injection of 0.1 mmol/kg of a positive paramagnetic contrast agent (Gd-DOTA, Dotarem, Guerbet, Roissy CdG Cedex, France). Signal intensity data are stored as an uncompressed DICOM file. A single patient dataset occupies about 111 MB.

### 5.2 Compression tests

We compare four different compression protocols: Java Native Zip library, Lempel–Ziv–Markov (LZMA achieved using 7z open-source java library), the Zip4j open-source java library and MATLAB file format compression. Transmission times of whole the dataset (for each of 33 patients) have been evaluated on a 10/100 mbps Ethernet LAN. Table 1 reports the average results of the compression tests for Zip4j. Zip4j can achieve a compression ratio of about 70 %

**Table 1** Zip compression level impact on transmission time (evaluated over 33 patients)

| Compression level | Original size (MB) | Compressed size (MB) | Compression ratio (%) | Compression time (s) | Transmission time (s) |
|---|---|---|---|---|---|
| 0 (store) | 111 | 111 | 0.00 | 0.43 | 8.38 |
| 1 (fastest) | **111** | **34** | **68.78** | **0.83** | **6.38** |
| 3 (fast) | 111 | 34 | 68.78 | 1.12 | 6.42 |
| 5 (normal) | 111 | 32 | 70.33 | 3.22 | 6.13 |
| 7 (maximum) | 111 | 31 | 71.28 | 10.26 | 5.97 |
| 9 (ultra) | 111 | 31 | 71.48 | 26.20 | 5.91 |

The average size of a original DICOM study is about 111 MB. The compression ratio of Zip is about 70 % for all levels. The level 1 (reported in bold) is the best trade-off between short transmission and compression times

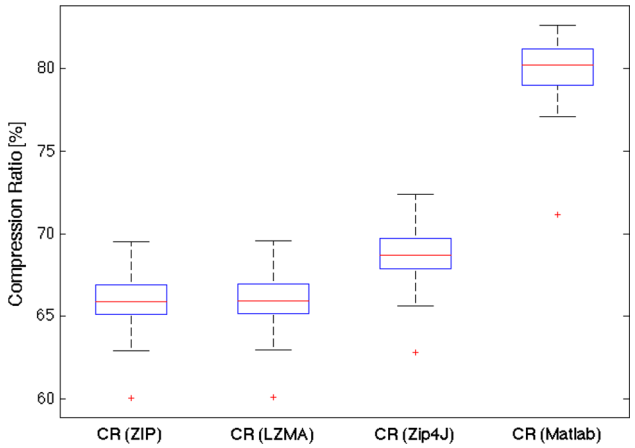


**Fig. 7** Compression ratio comparison between different compression methods, including MATLAB

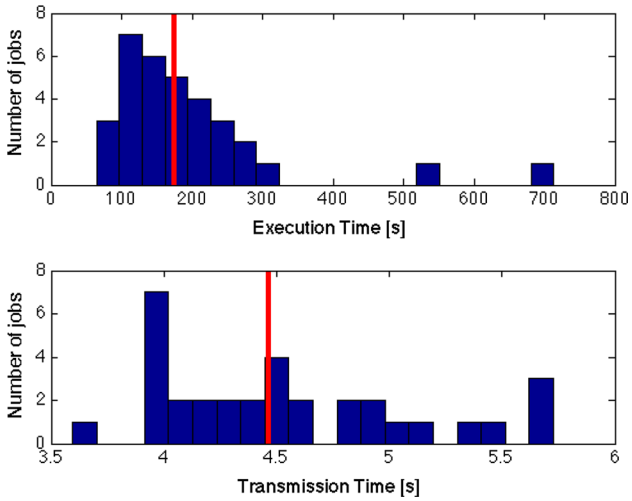


**Fig. 8** Distribution of remote execution and transmission times using 1 CPU on server node. Median values of execution (175 s) and transmission (4.5 s) have been indicated as *red lines*

for all levels. Higher levels of compression do not seem to significantly improve the compression ratio. They dilate compression times without dramatically reducing the transmission times (that follows an exponential decrease with a lower bound to about 6 s). The level 1 is the best trade-off between short transmission and compression times. It is worth noting some communication errors that alter the transmission time (on the Level 3 test) due to the best-effort feature of the TCP/IP protocol, over which, we have carried out these tests.

Figure 7 shows the average compression ratio computed over the overall patient dataset for each compression protocol. For each test, data are composed of both meta-information and signal intensity data. It is possible to appreciate that Zip4j algorithm (the runner up) has a compression ratio approximately 10 % lower than MATLAB results. All other files (XML) are compressed with Zip4j.

### 5.3 Transmission and execution tests

Figure 8 shows the distribution of execution and transmission times. Median values of execution (175 s) and transmission (4.5 s) times have been indicated as red lines.

The server-side has been implemented on an Intel Core i7-3630QM 64 bit Quad Core 2.4 GHz equipped with 12 GB RAM. Also in this case the transmission times have been evaluated on a 10/100 mbps Ethernet LAN, a 100/1000 mbps Ethernet LAN, a cheap xDSL (nominal 4 mbps) and a MPLS (nominal 10 mbps). Each average overhead was calculated according to Eq. (2) and is shown in Table 2 (for decreasing values of overhead).

In Fig. 8, it can be observed that the transmission time has spread over a small interval (3–6 s), while the execution time has two clear outliers of about 549 and 713 s. These are due to two patients having a very high number of voxels to analyse.

### 5.4 Scalability evaluation

In Table 3, we compare the average execution times of four different server-side architecture configurations according to Sect. 4.6.3. Remote execution times include image processing and transmission between client and server (and vice

**Table 2** Overhead in different network configuration (the execution time assess on about 188 s)

| Network | Nominal speed (mbps) | Transmission time (s) | Overhead (%) |
|---|---|---|---|
| xDSL | 4 | 17.104 | 9.15 |
| LAN | 100 | 4.541 | 2.62 |
| MPLS (PPPoA) | 10 | 2.832 | 1.66 |
| Fast LAN | 1000 | 2.817 | 1.65 |

It is worth noticing that transmission time could be afflicted by network load (best-effort treatment)

**Table 3** Average execution times, throughput and speedup in the different cases

| | Execution time | | Throughput (jobs/h) | Speedup |
|---|---|---|---|---|
| | Mean (s) | Standard deviation (s) | | |
| Local (1 job) | 229.34 | 145.19 | 15.7 | – |
| Remote (1 CPU, 1 job) | 192.60 | 128.05 | 18.7 | 1.2 |
| Remote (2 CPUs, 2 job) | 199.40 | 130.41 | 36.1 | 2.3 |
| Remote (4 CPUs, 4 job) | 204.31 | 130.59 | 70.5 | 4.5 |

versa). In the client–server approach, the server has been implemented on an Intel Core i7-3630QM 64 bit Quad Core 2.4 GHz equipped with 12 GB RAM. The local evaluation has been tested on a typical OsiriX workstation (Apple iMac with Intel Core 2 Duo 2.0 GHz equipped with 3 GB RAM) with lower characteristic than the client–server approach. Also in this case, transmission times have been evaluated on a 10/100 mbps Ethernet LAN.

It is worth noticing that the execution times remain almost unchanged in all the considered configurations. However, as Table 3 shows, the throughput and the speedup ratio increase is almost proportional to the number of processors used. This justifies the choice of a remote execution and brings benefits in terms of upgradability and maintainability of the segmentation algorithms and result sharing.

## 6 Discussion and conclusions

In this work we proposed an architecture for advanced remote data processing in a secure and versatile client–server environment.

To demonstrate the feasibility of the proposed system and the advantages over past studies, we have implemented a medical case study for automatic lesion detection in breast DCE-MRI [24].

The result is a process-on-demand architecture, allowing the radiologist to have access to a secure, versatile and powerful remote CAD system.

The aim of the proposed architecture is the possibility of easily integrating a pre-existing medical image processing software within a complete CAD system deployed on a server machine and shared with many workstations. This has the benefit to allow the physician to not change the user interface he is accustomed to, but to extend the pre-existent software via plug-in. This is one of the main novelties with respect to previous works that proposed interaction with the user through proprietary interfaces [13].

The plug-in's main role is to interact with the physician making all required tasks (XML file creation, DICOM exporting, status monitoring and alerting, result reporting) totally transparent to the operator. The jM-FlowOrchestrator (client) role is to extract sensitive data from DICOM files, prepare the MATLAB formatted files to the lower transmission layer (jSecureSync) and manage status updates working jointly with its counterpart on the serves side.

To make the proposed system available for a new client-side image processing application, only the small plug-in development is needed, as the JM-FlowOrchestrator is written in Java (making it portable and transferable to different operative systems with ease). The proposed strategy allows us meeting the integration goal easy, as long as the third-party client implements a customization such as a plug-in or API philosophy supported by some form of SDK toolboxes. As a result, the use of a plug-in together with jSecureSync and jM-FlowOrchestrator makes the proposed architecture versatile and has "low-impact" on the pre-existent infrastructure.

The use of a remote server for advanced operation execution allows an increase in number and type of services offered. It also improves withstanding services without any modification on the client-side. Moreover, during server execution, the client user can utilize his workstation for other kinds of work, optimizing task time. Table 3 shows that the use of a multiprocessor server (or a cluster of multiprocessor servers) yields a system that has an execution time comparable with local processing system, but with a higher throughput value, which is quite proportional to the number of processor used. The multi-client/single-server structure optimizes the use of resources by allocating different atomic jobs on different CPUs. Table 3 also shows that, by increas-

ing the number of the available CPUs and jobs, the mean execution time only slightly increases a little. This allows an easy handling of small and big increase of requests by adding additional CPUs or multiprocessors servers to the cluster.

It is worth noticing that the proposed architecture, with reference to a typical radiological workstation, does not require further expensive hardware, interconnection networking or operating system (all tests were performed on commercial notebook/pc/mac and on conventional network infrastructures), making it suitable for small clinical structures with of few radiologists, up to big hospitals or structures in which tens of radiologists are involved.

In order to limit transmission time, image files need to be compressed before being sent over the network. For the specific implementation of the architecture reported in this paper, compression tests showed that, due to the significant correlation of medical image data and the extraction of repeated meta-data information, the use of MATLAB format can produce results even better than those obtained by using the fastest compression offered by Zip4j (level 1). The compression ratio was about 70 % for Zip4j, while MATLAB format can achieve a compression ratio up to 80 %. All other files not in DICOM format (XML) are compressed with the Zip4j protocol. Transmission and execution time measurements were then performed on some plausible scenario, for a practical clinical environment, corresponding both to a private network among different structures from the same company and to different companies connected through available network infrastructures (e.g. optical fibre). The overhead added by network infrastructure (Table 2), in the case study (heavy remote computation) we have considered, settles on about 2.5 % (for the widespread 10/100 mbps) of the remote processing execution time. This overhead reaches a still acceptable value of 10 % in the worst case of a cheaper xDSL network WAN connection. This could make it possible to apply the proposed architecture even over a more diffused network (such as xDSL) without a significant decrease in performance.

Remote processing of medical images proposed in previous studies (such as in [9,10]) only offered basic image processing or 3D reconstructions, without heeding the associated complex and severe security issues. When analysed, security issues are typically resolved by considering server and client connected by a private dedicated network [12]. Our system can be safely used over every kind of interconnection network that supports TCP/IP stack protocol, allowing the development of multi-centre interconnections distributed over a wide geographical area. All the security evaluation is thorough using SSL, AES and RSA parameters meeting the 2012 NIST standard requirements [22] and will ensure channel privacy through to the year 2030 at minimum. This will allow for a distribution of the system cost over a longer period. On the other hand, the use of authorization certificates, means a simple and safe authentication and authorization phase.

Although the DICOM standard provides all functionality needed in our architecture, we intentionally decided not to use DICOM standard with the aim of decoupling our architecture from any particular application context. The proposed architecture has been demonstrated with reference to biomedical image processing, but it is possible to apply it even in different contexts (i.e. not using DICOM).

Recent studies [37] claim that in the near future, "computation on demand" will be available to everyone and at a low cost, causing an even higher interest in distributed computing services. Thus, future works will focus on improving server side by supporting multiple servers (cluster) architectures. Load balancing will be achieved by the use of a distributed decision protocol. Each node in the cluster knows the others nodes load status: when a client asks to connect it will be redirected to the least busy server node. This system will be also designed to allow hot-plug (new server connection during execution) and automatic dead server recovery, resolving also the Single Point Of Failure (SPOF) problem typical of a central server system. For improving dependability and recoverability of the offered services, is mandatory to uncouple the source and destination of data transfers. This can be done by means of a new module which asynchronously feeds workers with new units of work on an on-demand basis, and on a special feeding strategy based on bookkeeping the status of each work-unit as in [38] and dynamically load balancing as in [39].

In conclusion, the proposed architecture has a secure, versatile and low-impact approach, making this work suitable to enable an easy integration into intelligent environments.

## 7 Mode of availability

The whole architecture is the result of a collaboration (still active) with radiologists and physicians who are testing the functionalities of the software. However, we have not yet delivered a public plug-in. The communication core (jSecureSync) and interfaces for implementing the processing flow level (jFlowOrchestrator) are available by writing to carlo.sansone@unina.it

**Acknowledgments** The authors are grateful to Dr. Antonella Petrillo, Head of Division of Radiology, Department of Diagnostic Imaging, Radiant and Metabolic Therapy, "Istituto Nazionale dei Tumori Fondazione G. Pascale"-IRCCS, Naples, Italy, for providing access to DCE-MRI data. Moreover, we would like to thank PhD Roberta Fusco, from the same institution, for useful discussions.

## References


1. Buyya R, Yeo CS, Venugopal S (2008) Market-oriented cloud computing: vision, hype, and reality for delivering it services as computing utilities, In: 10th IEEE international conference on high performance computing and communications, 2008. HPCC2008, IEEE, New York, pp 5–13
2. Bryant R, Katz RH, Lazowska ED (2008) Big-data computing: creating revolutionary breakthroughs in commerce, science and society
3. Majithia S, Taylor I, Shields M, Wang I (2003) Triana as a graphical web services composition toolkit. In: Proceedings of the UK eScience all hands meeting, pp 2–4
4. Curcin V, Ghanem M (2008) Scientific workflow systems-can one size fit all? In: Cairo international biomedical engineering conference, 2008. CIBEC 2008. IEEE, New York, pp 1–9
5. Altintas I, Berkley C, Jaeger E, Jones M, Ludascher B, Mock S (2004) Kepler: an extensible system for design and execution of scientific workflows. In: Proceedings of the 16th international conference on scientific and statistical database management, 2004. IEEE, New York, pp 423–424
6. Rex DE, Ma JQ, Toga AW (2003) The loni pipeline processing environment. Neuroimage 19(3):1033–1048
7. Wolstencroft K, Haines R, Fellows D, Williams A, Withers D, Owen S, Soiland-Reyes S, Dunlop I, Nenadic A, Fisher P et al (2013) The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. Nucleic Acids Res gkt328
8. Svantesson D, Clarke R (2010) Privacy and consumer risks in cloud computing. Comput Law Secur Rev 26(4):391–397
9. Scheinine AL, Donizelli M, Pescosolido M (1998) An object-oriented client–server system for interactive segmentation of medical images using the method of active contours. In: Bildverarbeitung für die Medizin 1998. Springer, New York, pp 308–312
10. Mayer A, Meinzer H-P (1999) High performance medical image processing in client/server-environments. Comput Methods Programs Biomed 58(3):207–217
11. Yacoub SM, Ammar HH (1999) The development of a client/server architecture for standardized medical application network services. In: Proceedings of the 1999 IEEE symposium on application-specific systems and software engineering and technology, 1999. ASSET'99. IEEE, New York, pp 2–9
12. Crane JC, Crawford FW, Nelson SJ (2006) Grid enabled magnetic resonance scanners for near real-time medical image processing. J Parallel Distrib Comput 66(12):1524–1533
13. Mahmoudi SE, Akhondi-Asl A, Rahmani R, Faghih-Roohi S, Taimouri V, Sabouri A, Soltanian-Zadeh H (2010) Web-based interactive 2D/3D medical image processing and visualization software. Comput Methods Programs Biomed 98(2):172–182
14. Dierks T, Allen C (1999) The TLS protocol version 1.0, RFC 2246 (proposed standard), obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176. http://www.ietf.org/rfc/rfc2246.txt
15. Dierks T, Rescorla E (2006) The transport layer security (TLS) protocol version 1.1, RFC 4346 (proposed standard), obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176. http://www.ietf.org/rfc/rfc4346.txt
16. Dierks T, Rescorla E (2008) The transport layer security (TLS) protocol version 1.2, RFC 5246 (proposed standard), updated by RFCs 5746, 5878, 6176. http://www.ietf.org/rfc/rfc5246.txt
17. Oracle Corporation (2011) Java authentication and authorization service reference guide. http://docs.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html. Accessed February 2014
18. Lai C, Gong L, Koved L, Nadalin A, Schemers R (1999) User authentication and authorization in the Java TM platform. In: Proceedings of the 15th annual computer security applications conference, 1999 (ACSAC'99). IEEE, New York, pp 285–290
19. Barker E, Johnson D, Smid M (2007) Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography. In: NIST special publication 800–56A
20. Certicom Research (2000) Standards for efficient cryptography SEC 1: elliptic curve cryptography. http://www.secg.org/collateral/sec1_final.pdf. Accessed February 2014
21. Chown P (2002) Advanced encryption standard (AES) ciphersuites for transport layer security (TLS), RFC 3268 (proposed standard), obsoleted by RFC 5246. http://www.ietf.org/rfc/rfc3268.txt
22. Barker E, Barker W, Burr W, Polk W, Smid M (2006) Recommendation for key management—part 1: general (revision 3). In: NIST special publication, pp 800–857
23. Blake-Wilson S, Bolyard N, Gupta V, Hawk C, Moeller B (2006) Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS), RFC 4492 (informational), updated by RFCs 5246, 7027. http://www.ietf.org/rfc/rfc4492.txt
24. Piantadosi G, Marrone S, Sansone M, Sansone C (2013) A secure OsiriX plug-in for detecting suspicious lesions in breast DCE-MRI. In: Algorithms and architectures for parallel processing. Springer, New York, pp 217–224
25. Marrone S, Piantadosi G, Fusco R, Petrillo A, Sansone M, Sansone C (2013) Automatic lesion detection in breast DCE-MRI. In: Image analysis and processing—ICIAP 2013. Springer, New York, pp 359–368
26. Mustra M, Delac K, Grgic M (2008) Overview of the DICOM standard. In: 50th international symposium ELMAR, 2008, vol 1. IEEE, New York, pp 39–44
27. I. The MathWorks (2013) Matlab. http://www.mathworks.com/products/matlab
28. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. ACM SIGKDD Explor Newsl 11(1):10–18
29. Rosset A, Spadola L, Ratib O (2004) OsiriX: an open-source software for navigating in multidimensional DICOM images. J Digit Imaging 17(3):205–216
30. Fusco R, Sansone M, Sansone C, Petrillo A (2011) Selection of suspicious ROIS in breast DCE-MRI. In: Image analysis and processing—ICIAP 2011. Springer, New York, pp 48–57
31. Fusco R, Sansone M, Sansone C, Petrillo A (2012) Segmentation and classification of breast lesions using dynamic and textural features in dynamic contrast enhanced-magnetic resonance imaging. In: 25th international symposium on computer-based medical systems (CBMS), 2012. IEEE, New York, pp 1–4
32. Fusco R, Sansone M, Petrillo A, Sansone C (2012) A multiple classifier system for classification of breast lesions using dynamic and morphological features in DCE-MRI. In: Structural, syntactic, and statistical pattern recognition. Springer, New York, pp 684–692
33. Fusco R, Sansone M, Maffei S, Raiano N, Petrillo A (2012) Dynamic contrast-enhanced mri in breast cancer: a comparison between distributed and compartmental tracer kinetic models. J Biomed Graph Comput 2(2):23
34. Fusco R, Filice S, Granata V, Mandato Y, Porto A, DAiuto M, Rinaldo M, Di Bonito M, Sansone M, Sansone C (2013) Can semi-quantitative evaluation of uncertain (type ii) time–intensity curves improve diagnosis in breast DCE-MRI? J Biomed Sci Eng 6:418
35. Chang C-C, Lin C-J (2011) Libsvm: a library for support vector machines. ACM Trans Intell Syst Technol (TIST) 2(3):27
36. Hennessy JL, Patterson DA (2011) Computer architecture: a quantitative approach. Elsevier, London
37. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25(6):599–616

38. De Florio V, Deconinck G, Lauwereins R (1997) An application-level dependable technique for farmer–worker parallel programs. In: High-performance computing and networking. Springer, New York, pp 644–653

39. Leeman M (2012) A resource-aware dynamic load-balancing parallelization algorithm in a farmer–worker environment. Innov Approach Resilient Adapt Syst 88:88–104