In Depth Tutorials and Information

A Software Tool for Reading DICOM Directory Files

abstract

DICOMDIR directory files are useful in medical software applications because they allow organized access to images and information sets that come from radiological studies that are stored in conformance with the digital imaging and communication in medicine (DICOM) standard. During the medical application software development, specialized programming libraries are commonly used in order to solve the requirements of computation and scientific visualization. However, these libraries do not provide suitable tools for reading DICOMDIR files, making necessary the implementation of a flexible tool for reading these files, which can be also easily integrated into applications under development. To solve this problem, this work introduces an object-oriented design and an open-source implementation for such reading tool. It produces an output data tree containing the information of the DICOM images and their related radiological studies, which can be browsed easily in a structured way through navigation interfaces coupled to it.

Introduction

The digital imaging and communications in medicine (DICOM) standard (National Electrical Manufacturers Association (NEMA0, 2004a; Revet, 1997) was published in 1993. Its main goal was to establish norms for handling, storing, and interchanging medical images and associated digital information within open systems. Also it was to facilitate the interoperability among acquisition equipments and other medical devices, as well as their integration within specialized information systems in the medical and health care area.

Since then, the appearance and use of computer-assisted medical applications have increased, as a result of the accelerated technological development and the standardization process of medical information representation and handling, which generated a greater demand of development tools for those applications.

These applications range from health care information systems and picture archiving and communication systems (PACS) () solutions, to technological support systems for medical procedures, such as image-based diagnosis and surgical planning, which previously depended on the knowledge and expertise of the physicians.

In such applications, the handling of images coming from different acquisition modalities is essential. These images generated from radiological studies and stored according to the specifications of parts 10, 11 and 12 of the DICOM standard (NEMA, 2004e, f,& g) must be retrieved from storage media as a bidimensional display or in tridimensional reconstructions and other special processes, such as fusion and segmentation of images. The use of DICOMDIR directory files is almost mandatory for searching, accessing, and browsing medical images because they index the files belonging to the patient on whom the studies were performed, thus making it easier to access to those images and their associated medical information.

During the medical application software development, the use of programming interfaces (APIs) or class libraries is frequent in order to solve the computation and visualization needs, as well as for providing DICOM support to the applications. In that sense, there exist numerous public domain applications that can be used by radiologists and other specialists for reading and displaying DICOM images files and even for reading DICOMDIR index files, which cannot be integrated into applications under development because of their proprietary code.

Companies, such as Lead Technologies, ETIAM, Merge, Laurel Bridge, and DeJarnette, have commercial software development kits (SDKs) that provide complete implementations of the DICOM standard, but the acquisition costs for these SDKs are high. Open-source libraries are an alternative choice for integrating DICOM support into applications. Regarding this matter, libraries, such as visualization tool kit (VTK) (), insight segmentation and registration tool kit (ITK), DICOM tool kit (DCMTK), and virtual vision machine (VVM), allow the reading of DICOM images, but they do not provide mechanisms for reading DICOMDIR files. Like in the DCMTK case, there are other libraries that provide tools for a basic and low-level access to the information contained in the files. However, they have disadvantages, such as troublesome information retrieving process and reading tools, which are difficult to integrate into the applications.

Due to the lack of an adequate tool for reading and handling DICOMDIR files in a structured and simple way, which could be also easily coupled to browsing interfaces and attached to medical application under development, we introduce in this article the design and implementation of a DICOMDIR files reader. This tool has been successfully integrated into an application for neurosurgery preoperative planning (Montilla, Bosnjak, Jara, & Villegas, 2005), but it also can be attached to any other software under development that requires the handling of DICOM images and DICOMDIR directory files.

The next sections include the revision of related works, the essential theoretical background that frames this work within the DICOM standard context; the description of the methodology used for the implementation of the tool; and, finally, the discussion and conclusions obtained from the integration and test of the implemented reader into a medical application.

Antecedents and Related Works

The creation of the American College of Radiology (ACR)-NEMA committee in 1983 was the product of earlier attempts by the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) to establish a normative for exchanging, transmitting, and storing medical images and their associated information. The version 1.0 of the standard was published by this joint committee in 1985, under the document ACR-NEMA No. 300-1985, followed in 1988 by the document ACR-NEMA No. 300-1988 of version 2.0. Previous to these norma-tives, medical images were stored in files by the acquisition devices under their own proprietary formats and transferred through point-to-point communication or by removable storage media. Versions 1.0 and 2.0 established a standardized terminology and information structure, as well as hardware interfaces, software commands sets, and consistent data formats.

The most recent version, known as DICOM 3.0 (NEMA, 2004a) was published in 1993, and it was structured in parts, or documents, to facilitate its support and extension. In the last version, objects for the representation of patients, studies, reports, and other data sets were added, as well as unique identifiers for these objects, enabling the transmission of information through communication networks using the TCP/IP protocol. The DICOM 3.0 standard facilitates the transfer of images and related information within open systems containing different medical equipments and the integration with medical information systems and applications.

DICOM parts 3, 6, 10, and 12 (NEMA, 2004b, d, 4e, & g) were of particular interest for our tool design. They define and describe the DICOMDIR directory object and other information objects; attributes and representation values of DICOM information model entities; and specifications for files formats and information storage in physical media.

We have not found formal research papers related to the design and implementation of open-source tools for reading DICOMDIR files and their integration within medical applications. Nevertheless, there exist documented development libraries that include support for this kind of file. On the other hand, regarding the complexity of searching and decoding information contained in DICOM files and the fact that the tool had to be integrated into a medical application developed with C++ language, we decided to search for open-source development libraries, based upon C++ and with DICOM support in order to use them as a base for the tool development.

Just a few development libraries, besides the expensive commercial ones, enable the reading and analysis of DICOMDIR files. Due to their structures or features only three open-source libraries, based upon C++ language, were deemed appropriate to be used as reference for the tool design and implementation; the remaining APIs were found either to have a complicated structure or were based upon Java language.

GDCM (GDCM, 2005) is an API supported by Centre de Recherche et d'Applications en Traitement de l'Image et du Signal (CREATIS), which provides a fairly complete support for reading DICOMDIR files and a simple access to the information extracted from them. However, it implements only part 5 of the DICOM standard; therefore, it would be of little use as a base library for medical applications that require the implementation of other features defined by the standard.

Dicomlib library (DicomLib, 2005) provides a fuller implementation of the DICOM standard, and it also features the reading of DICOMDIR files. Although this library tries to ease the huge intrinsic complexity in the use of the DICOM standard, its access and presentation of the DI-COMDIR compiled information is not the best one to be integrated into the applications.

Finally, DICOM tool kit (DCMTK) (DCMTK, 2005) from Oldenhurger Forschungs und Entwicklungsinstitut fur Informatik-Werkzeuge und Systeme (OFFIS) is another complete library, having several years of evolution and continous use in medical applications development. Although DCMTK provides support for the creation, modification, and opening of DICOMDIR files, it does not offer structured and simple access to the information gathered from the files. However, we selected DCMTK as the base library for the reader development due to its robustness, flexibility, and ability to handle of DICOM images.

Thus, starting from the basic functions for information searches and decoding that provides DCMTK, it was possible to develop the tool for reading the information contained in DICOMDIR files and attach this tool to medical applications that require organized access to DICOM image sets from medical studies.

theoretical background

A lot has been written about the DICOM standard ever since it was published, including revisions and extensions. The scope of DICOM is so wide that the researcher certainly gets overwhelmed by the amount and complexity of the information contained in the standard documents. In our case, as in any other case of software development that involves the handling of information within the DICOM scope, the revision of basic fundamentals for real-world information representation according to the DICOM standard was necessary.

Within DICOM's structure, the term "information" refers to medical images coming from different acquisition modalities, signals, curves, look-up tables, and structured reports, as well as to other information gathered during patient visits to the healthcare specialist and the studies derived from them. This information and its generating agents are represented by the standard through models.

DICOM Application and Information Models

DICOM structures and organizes medical data and information through models that emulate the real-world hypothetical situation, where a patient visits a health care specialist, who later orders a set of radiological studies as shown in Figure 1.

Figure 1. Correspondence between the radiological exam environment and the DICOM Information Model

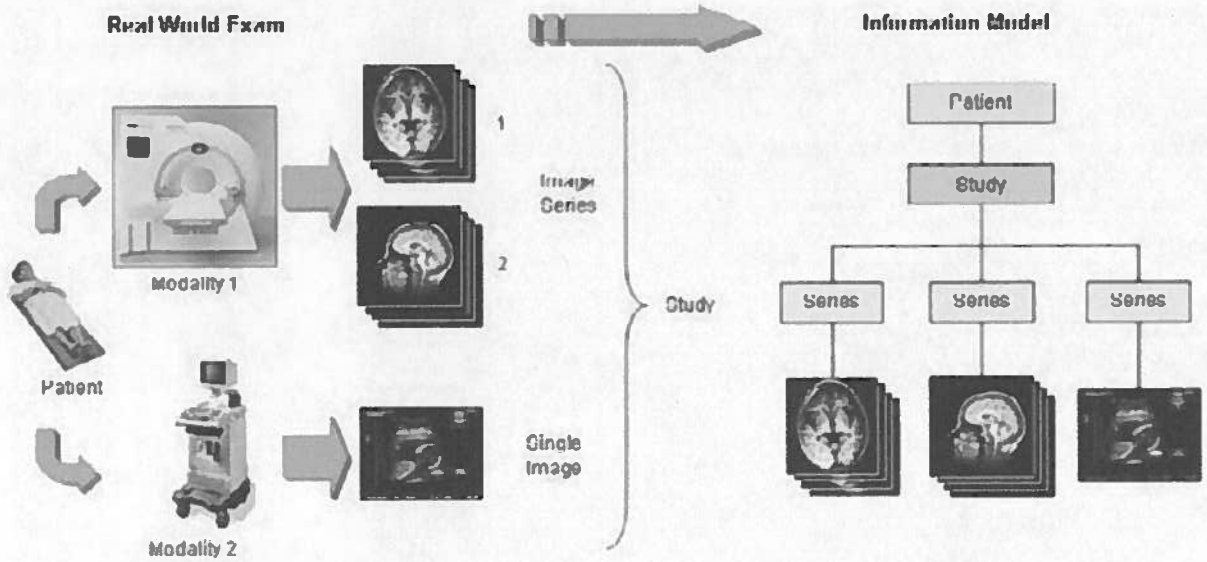
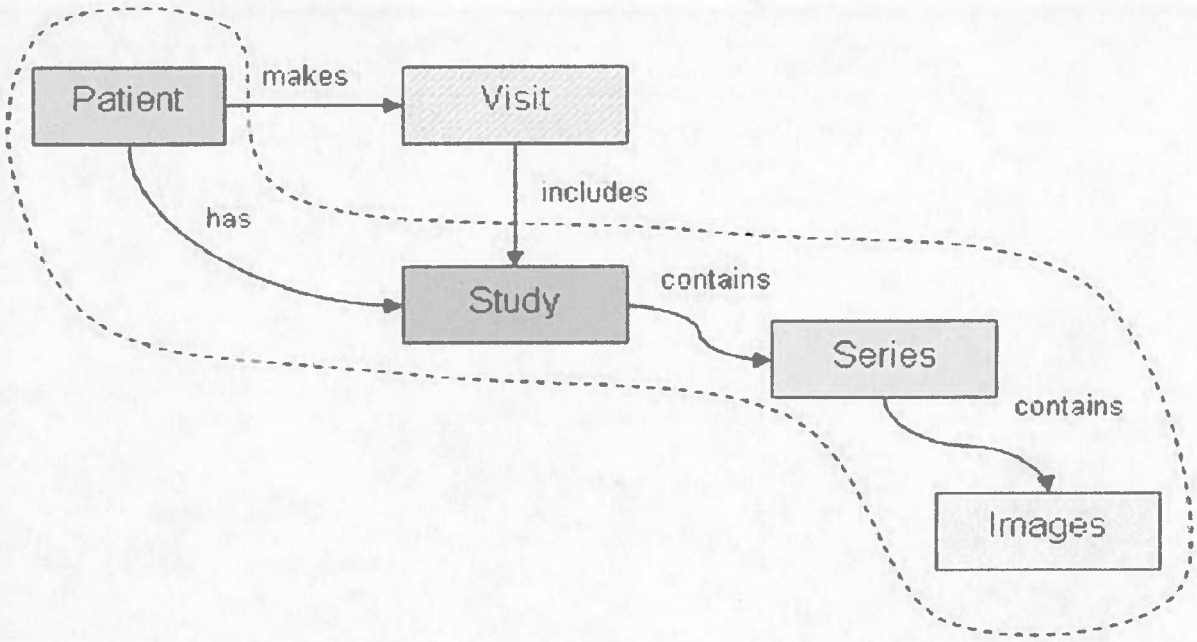


Figure 2. Simplified DICOM application model showing information entities and relationships among them



An application model is defined as an entity/ relationship diagram (see Figure 2) that relates real-world objects inside the standard scope. Its diagram derives from the way hospitals' radiology departments handle images from one or more acquisition modalities. Those images are ordered in a series, according to some spatial or temporal relationship, and then stored in a folder for each patient.

Although the application model contains entities for representing several DICOM objects, such as exams results, medical reports, and study procedures, in this work only the patient, study, series, and image entities were considered because they are directly associated to data contained within the images. This consideration produces the simplified version of the DICOM application model shown in Figure 2, where the dashed region contains the entities of interest.

The DICOM information model derives from the application model and its entities are known as information object description (IOD). In an abstract way, an IOD describes real-world objects that share the same properties (NEMA, 2004b). This abstraction keeps a close relationship to the object-oriented design and programming paradigm.

Information model entities are featured by attributes whose types, multiplicities, and contents change depending upon the entity to which they belong. Attributes or data elements are defined in part 5 of the standard (NEMA, 2004c), and they are cataloged in part 6 of the data dictionary (NEMA, 2004d). An attribute is classified according to its presence in obligatory (types 1 and 2), conditional (types 1C and 2C), and optional (type 3).

Attributes are identified in the data dictionary through a tag composed from an ordered 16-bit number duple (gggg,eeee) expressed in hexadecimal form. These numbers represent the group and the element number within that group. The standard attributes have an even group number, different from 0000, 0002, 0004, and 0006, whereas private attributes, not contained in the data dictionary, have an odd group number, different from 0001, 0003, 0005, 0007, and FFFF. All aforementioned group numbers are reserved by the standard.

In addition to the tag, there are other data fields that also belong to the attributes structure, such as value representation (VR), value multiplicity (VM), length, and contained value. The VR describes, through a 2-byte character string, the type and format of the data contained in the attribute value field, such as integer or floating numbers, dates, string characters, and sequences. The VM specifies the cardinality or number of values that are codified in the value field. The length contains the attribute's value size in bytes. Finally, the value field stores the attribute data, according to its respective presence type.

For each IOD there are defined operation sets and named services that are executed on the information objects. When an entity needs to perform an operation over an IOD, it must request the proper service to another entity, which behaves as a server. Each object-defined service establishes a service-object pair (SOP), and the whole service set that is applicable to a particular IOD is a named SOP class.

The media storage service class and the que-ryrRetrieve service class are examples of SOP classes. The first one comprises the M-READ, M-WRITE, and M-DELETE services set, applied to the reading, writing, and deleting of files with image IODs coming from acquisition modalities. The second class groups the C-FIND, C-MOVE, and C-GET services that can be requested for querying and transferring information from IODs associated with different entities.

An SOP instance is the occurrence of an IOD that is operated within a communication context, through a specific service set. For example, DI-COMDIR directory files are SOP products from requesting the information writing service from a directory IOD to a physical storage media.

The simplified model from our research considers only four entities and their corresponding IODs. The patient entity contains data of the patient for whom radiological studies were performed as described by the study entity. The series entity models information resulting from radiological studies, such as images or signals, and keeps some kind of spatial or temporal relationship among them. The image entity represents the images coming from some of the existing acquisition modalities, for example, computed tomography (CT), magnetic image resonance (MRI), or ultra sound (US).

DICOM File Format

The DICOM file format, described in part 10 of the standard (NEMA, 2004e), defines the way data representing a SOP instance is stored in a physical storage media. The data is encapsulated as a stream of bytes, preceded by a header with metainformation required for identifying the SOP instance and class.

The header has an organized sequence of components, named file ID, which organizes files hierarchically. An ID has up to eight components, where each one is a string of one to eight characters separated by backslashes. The file ID generally corresponds to a directory path and to a filename, for example, SUBDIR1\SUBDIR2\ SUBDIR3 \ABCDEFGH.

Located after the header is the data set associated with the information model entity that is stored in the file. Depending upon the entity nature, this stream of bytes could

represent some of the following objects: images, curves, signals, overlay annotations, lookup tables for transforming images pixel values according to acquisition modalities or values of interest, presentation images descriptions, structure reports, or raw data.

Within the context and scope of our research, we considered only DICOM files containing images associated with studies performed on patients. Therefore, the stored data describes the image plane and the pixels features, as well as values for mapping the image to color or gray scales, overlay planes, and other specific features.

DICOM files are gathered in collections sharing a common name space, such as storage volumes or directory trees, and having unique file identifiers within it. The file collection is an abstraction of a container where files can be created or read. Each collection must be accompanied by an index file with a DICOMDIR identifier, corresponding to a DICOM directory object instance. Part 12 of the standard (NEMA, 2004g) describes the way the DICOM file information is encoded inside the physical storage media. It depends upon the file system used by the computer system for the files creation and interchange, as well as the physical media used for it.

DICOMDIR File Format

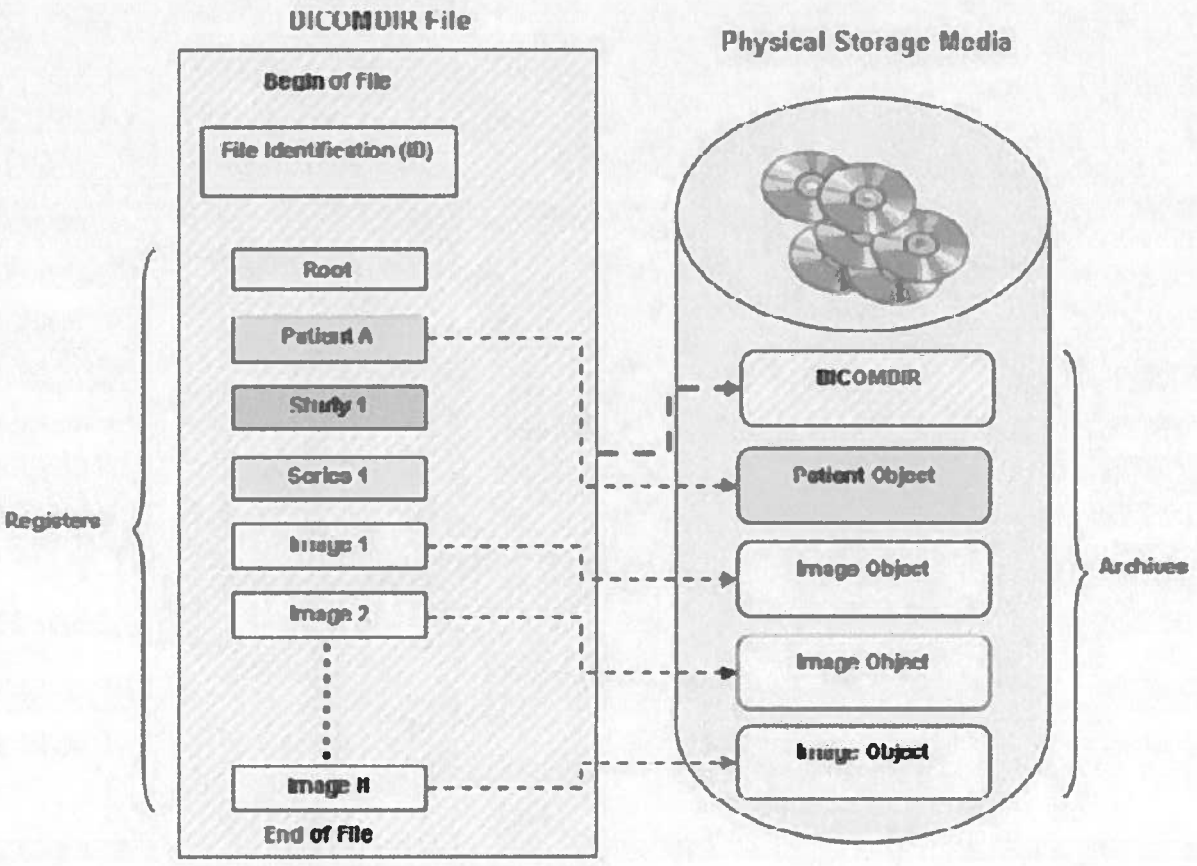
The DICOM standard defines a special object class as a named basic directory object, whose purpose is to serve as an organizing index for DICOM files stored in a physical media. The instance of a DICOM directory class object is a file with a unique filename and ID named DICOMDIR. The formal definition of the DICOMDIR object and its content are in part 3 annex F of the DICOM standard (NEMA, 2004b), whereas its structure complies with the DICOM files format specified in section 7 of part 10 (NEMA, 2004e).

There are registers in the DICOMDIR file with the information associated to objects stored in a DICOM files set, and it does not make reference to files that do not belong to the set. Each register contains a field identifying the represented information type, such as patient, study, series, and image, besides a group of specific fields with attributes extracted from the stored SOP instance. The registers are hierarchically sorted, and they are linked among themselves in the same hierarchy level and to the next lower level in the hierarchy (Figure 3).

The preceding information is generally present in the file, such as indicated in the standard, but some data fields can be optional, such as those related to the complementary information of the patient (birth date, sex, and age) and the studies (referring physician, institution, protocols, and diagnoses), as well as image descriptive information, for example, dimensions, samples by pixel, photometric interpretation, and gray-level window.

There should be a unique DICOMDIR file for each DICOM files set contained in a storage media. The DICOMDIR file location is related to the storage media directory organization, and it is commonly found at the root directory. DICOM-DIR files help to make fast queries and searches throughout media contained images, without the need for reading whole file sets. Otherwise, searching and browsing images and information within file sets becomes an intensive, tedious, and difficult task.

Figure 3. Structure of a DICOMDIR file and its representation in a physical storage media



DICOMDIR FILE READER IMPLEMENTATION

Description

The implemented tool enables DICOMDIR files to be opened in order to obtain the most relevant information from the DICOM-file collection they index. This information is organized in a hierarchical data structure, which can be easily consulted, and when coupled to a suitable graphic interface, permits the interactive browsing of the

information related to the collection. The implementation was made using C++ language, based upon an object-oriented design that allows new DICOM data fields to be added to the reader. Some DCMTK library classes and methods (DCMTK, 2005) were used to facilitate the searching and decoding of the data contained within DICOM-DIR files, thus avoiding the inherent complexity in the handling of the information stored under DICOM standard specifications.

Data structures

The references to SOP instances contained in DICOMDIR files are linked according to the entities' hierarchy ofPATIENT-STUDY-SERIES-IMAGE, which is implicitly established in the DICOM information model, thus establishing a natural correspondence between the hierarchy and a tree data structure. This tree has heterogeneous content nodes that correspond to the DICOM simplified application model (Figure 2), so there are five node types: root, patient, study, series, and image. The entities' relationship cardinality sets the offspring multiplicity, that is, a patient could be object from several studies, whereas each one of them could have several image series.

During the reading of a directory file, all of the file's hierarchical links are traveled, and the related collection file information is gathered for filling the nodes' specific fields, thus building the data tree structure. At the end of the reading process, the tree has the relevant collection information, having a structure similar to that shown in Figure 4. This data structure could be browsed in order to consult the information without the need of accessing the whole file set again.

An approach was considered for the tree structure implementation where the nodes behave simultaneously as structural elements and data containers. In this way, the nodes establish the tree hierarchical structure and each one of them also stores the information associated to the corresponding entity. Hierarchy levels are made up of linked node lists, and each one of these lists have a children's list, corresponding to elements from the next lower hierarchy level (Figure 5). This approach facilitates the travelling of the tree and its coupling to browsing interfaces.

Figure 4. Structure of the data tree structure built by the reading tool

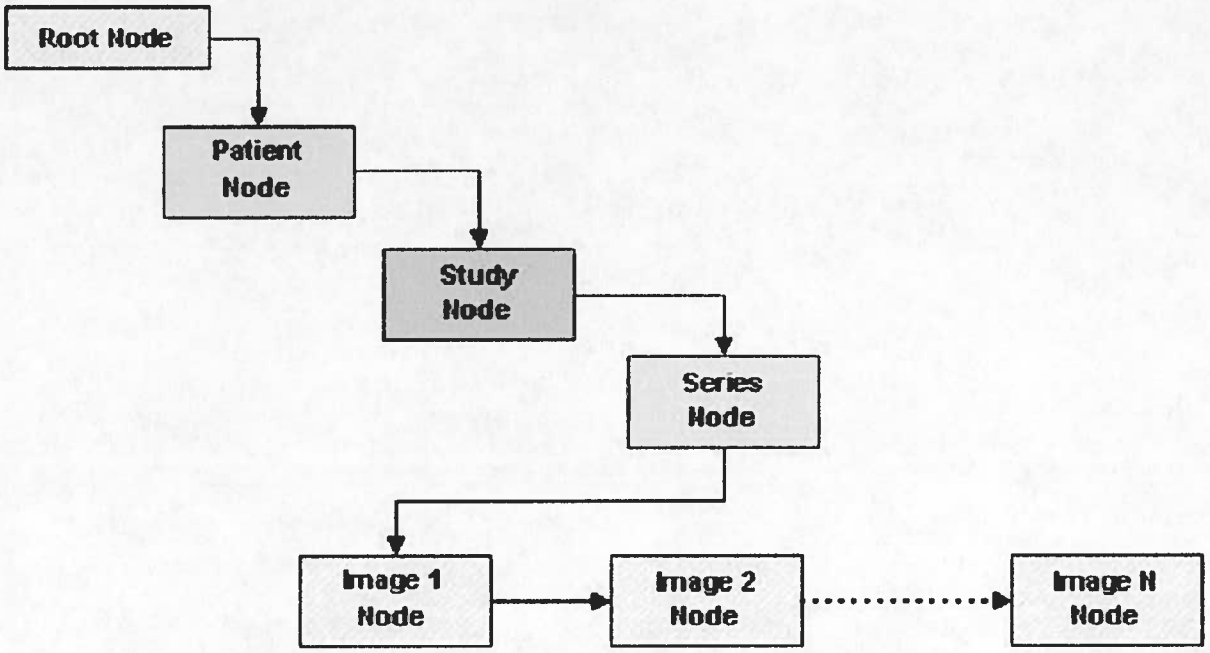
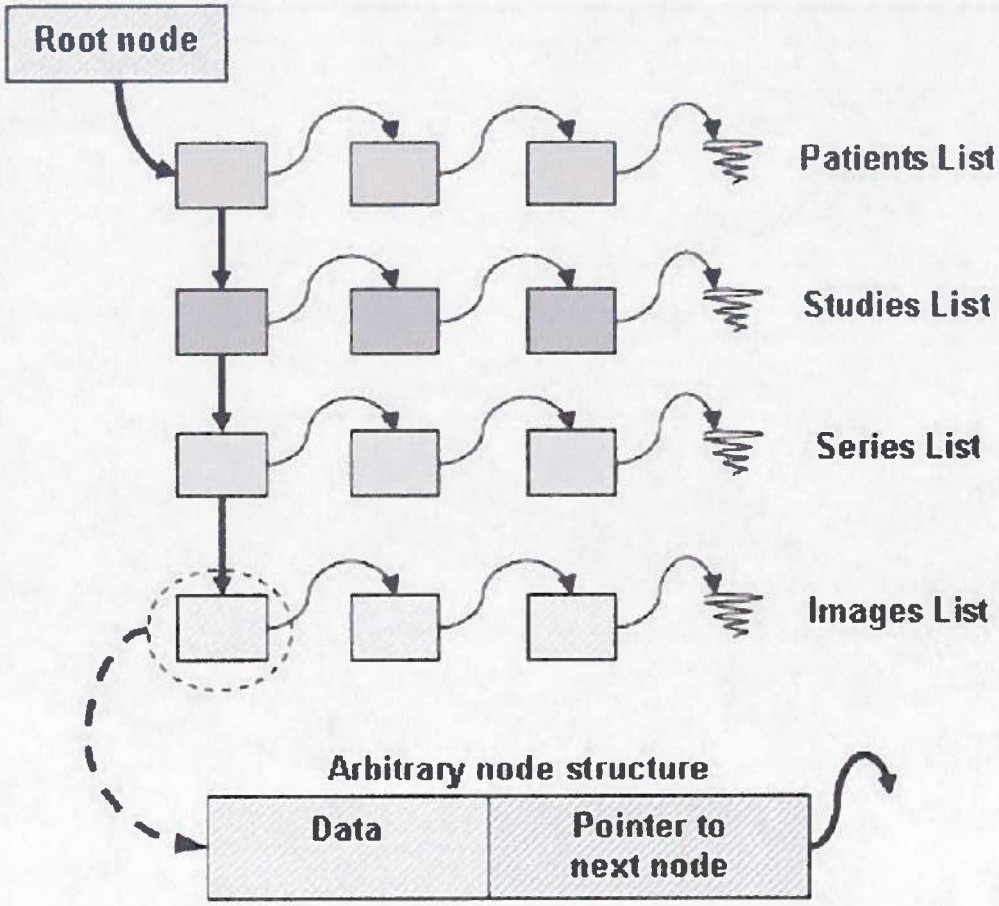


Figure 5. Data tree implementation using linked node lists as data containers



A class hierarchy was defined for enabling the nodes polymorphic handling through virtual methods. In this hierarchy, the base class, DICOMDIRNode, defines the basic structure for the other nodes, as well as the virtual methods for accessing and travelling the tree. All subclasses deriving from it, that is, DICOMDIRRootNode, DICOMDIRPatientNode, DICOMDIRStudyN-ode, DICOMDIRSeriesNode, and DICOMDIR-ImageNode, redefine their contents and behavior according to the corresponding DICOM application model entity to which they belong.

collected DICOM Data Fields

DICOM files contain rather large data-element groups that are cataloged in the DICOM data dictionary (NEMA, 2004d). However, not all of these elements are useful in the scope of software applications that handle medical images, and for that reason, a selection of relevant elements was made in order to limit the number of data fields gathered during the DICOMDIR files reading process. Table 1 shows the collected data fields with their tags, value representations, multiplicities, presence attributes, and corresponding DCMTK classes and C++ standard types.

One selection criteria for the data fields is based upon the information provided by the image processing. Among the selected fields are the image dimensions, the samples per pixel number, and the gray-level window. Another criterion is the general information that guides the user during the search and selection of images, for example, patient's data, description, and modality of the study. Complementary fields that do not provide significant information for the image processing or to the user applications were not used for the gathering process, for example, address, occupation, and medical antecedents of the patient, and the technical information from the image acquisition devices.

Reading Process

The reader uses Algorithm 1 for the DICOMDIR information retrieving process.

It can be observed that in this algorithm starting from the parent nodes, each tree node and its children's nodes are recursively travelled, level by level, until reaching the deepest tree level. Only nodes from PATIENT, STUDY, SERIES, and IMAGE types are taken into account, though it is possible to extend the consideration to other types of nodes. For each visited node, the selected fields are compiled according to the node type (see Table 1). However, as explained in the DICOMDIR File Format section, not all the fields have an obligatory presence inside the DICOMDIR file, therefore the missing information must be completed from the corresponding DICOM image file, once its file identifier is known.

Some methods from the DCMTK library are used to travel through the DICOMDIR structure and retrieve its data. These methods can handle data elements encoded either in big Endian or little Endian byte ordering, or with any kind of representation values and cardinalities. The retrieving methods are invoked by the data element tag each time a node is visited, whereas the travelling methods are used during recursive calls to the tool's main reading method.

Table 1. DICOM attributes compiled during DICOMDIR file reading process. Attributes are grouped by node type, showing main DICOM data fields and corresponding DCMTK/ C+ + data types for each one of them

Node type	Tag	Field name	VR	VM	Presence	DCMTK data type	C++ data type
PATIENT	(0010,0010)	Patient name	PN	1	Obligatory	DcmPersonName	char[ ]
	(0010,0020)	Patient ID	LO	1	Obligatory	DcmLongString	char[ ]
	(0010,0030)	Birth date	DA	1	Optional	DcmDate	char[ ]
	(0010,0040)	Sex	CS	1	Optional	DcmCodeString	char[ ]
	(0010,1010)	Age	AS	1	Optional	DcmAgeString	char[ ]
STUDY	(0020,0010)	Study ID	SH	1	Obligatory	DcmShortString	char[ ]
	(0008,0020)	Study date	DA	1	Obligatory	DcmDate	char[ ]
	(0008,0030)	Study time	TM	1	Obligatory	DcmTime	char[ ]
	(0008,1030)	Description	LO	1	Obligatory	DcmLongString	char[ ]
	(0018,1030)	Protocol name	LO	1	Optional	DcmLongString	char[ ]
	(0008,0090)	Referring physician	PN	1	Optional	DcmPersonName	char[ ]
	(0008,0080)	Institution	LO	1	Optional	DcmLongString	char[ ]
	(0008,1080)	Diagnoses	LO	1..N	Optional	DcmLongString	char[ ]
	(0020,0011)	Serie number	IS	1	Obligatory	DcmIntegerString	signed long
	(0008,0060)	Modality	CS	1	Obligatory	DcmCodeString	char[ ]
	(0018,0015)	Body part	CS	1	Optional	DcmCodeString	char[ ]
SERIE	(0008,0021)	Serie date	DA	1	Optional	DcmDate	char[ ]
	(0008,0031)	Serie time	TM	1	Optional	DcmTime	char[ ]
	(0008,103E)	Description	LO	1	Optional	DcmLongString	char[ ]
	(0008,1050)	Performing physician	PN	1..N	Optional	DcmPersonName	char[ ]

Node type	Tag	Field name	VR	VM	Presence	DCMTK dat type
	(0020,0013)	Image number	IS	1	Obligatory	DcmIntegerSt
	(0008,0008)	Image type	CS	1..N	Optional	DcmCodeStri
	(0008,0023)	Image date	DA	1	Optional	DcmDate
	(0008,0033)	Image time	TM	1	Optional	DcmTime
	(0028,0010)	Rows number	US	1	Optional	Uint16
	(0028,0011)	Columns number	US	1	Optional	Uint16
	(0028,0100)	Bits allocated	US	1	Optional	Uint16
	(0028,0101)	Bits stored	US	1	Optional	Uint16
IMAGE	(0028,0102)	High bit	US	1	Optional	Uint16
	(0028,0002)	Samples per pixel	US	1	Optional	Uint16
	(0028,0103)	Pixel representation	US	1	Optional	Uint16
	(0028,0004)	Photometric interpretation	CS	1	Optional	DcmCodeStri
	(0018,0050)	Slice thickness	DS	1	Optional	DcmDecimal!
	(0028,0030)	Pixel spacing	DS	2	Optional	DcmDecimal!
	(0028,1050)	Window center	DS	1..N	Optional	DcmDecimal!
	(0028,1051)	Window width	DS	1..N	Optional	DcmDecimal!
	(0028,1053)	Rescale slope	DS	1	Optional	DcmDecimal!
	(0028,1052)	Rescale Intersection	DS	1	Optional	DcmDecimal!
	(0004,1500)	Referenced file ID	CS	1..8	Optional	DcmCodeStri

Results and Discussion

The DICOMDIR reader was integrated for its test and validation into software for neurosurgery and brachytherapy planning, developed with the VVM library (Montilla, Bosnjak, & Villegas, 2003). A graphical interface was coupled to the reader in order to enable the navigation of the information in a simple way. By just opening the associated DICOMDIR file, users will be able to have the interface display the patients' study images and their related medical information. Control widgets provided by the interface were used for interactively browsing and selecting the images.

Algorithm 1.

Open DICOMDIR file for reading

Assign DICOMDIR file root register to NODE variable

With NODE, do recursively /* Begin of recursive block */

Assign NODE first child to Next_Register variable

While (Next_Register <> NULL)

{

Select according to Next_Register type {

Case PATIENT:

Get patient node related information Create a new DICOMDIR_PatientNode node object Fill node members with gathered information Insert node in the data tree Assign Next_Register to NODE Call recursive block with new NODE value Complete missing patient information extracted from related image file Case STUDY:

Get study node related information Create a new DICOMDIR_StudyNode node object Fill node members with gathered information Insert node in the data tree Assign Next_Register to NODE Call recursive block with new NODE value Complete missing study information extracted from related image file Case SERIES:

Get series node related information Create a new DICOMDIR_SerieNode node object Fill node members with gathered information Insert node in the data tree Assign Next_Register to NODE Call recursive block with new NODE value Complete missing series information extracted from related image file Case IMAGE:

Get image node related information

Create a new DICOMDIR_ImageNode node object

Fill node members with gathered information

Insert node in the data tree

Assign Next_Register to NODE

Call recursive block with new NODE value

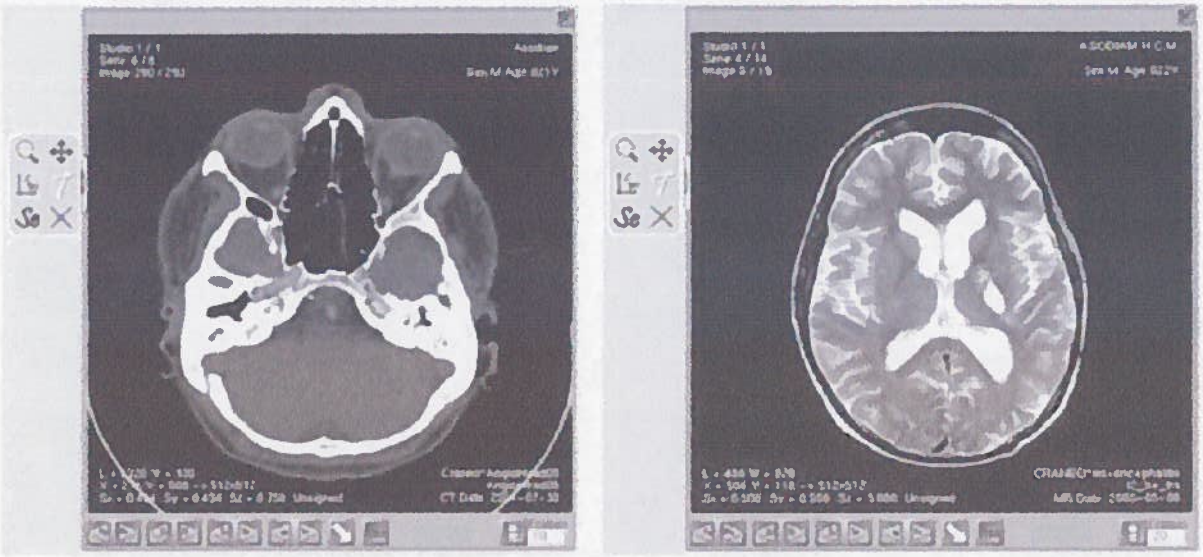
}

Assign NODE next child to Next_Register

}

} /* End of recursive block */

Figure 6. Results achieved from the integration of the DICOMDIR reading tool into a medical application for neurosurgery planning. The tool is coupled with a graphical interface for browsing throughout studies, series and images.



The reading process was verified with DI-COMDIR files associated to several image sets, coming from studies of CT and MRI performed on different patients. It could be proved that the reader recollected the information specified by the class definitions for each type of node, enabling display of its structure through the interface. Two examples from the tests are shown in Figure 6. In both examples, the studies were made on the patient's head but with two different modalities. The CT study has an eight-image series, whereas the MRI study contains 14 images.

It was observed that access to the information collected from the studies can be made in an organized and fast way, making transparent the navigation process of data structures and improving the efficiency in the use of the software. The tool is able to be integrated with other applications under development that require the handling of DICOM images and DICOMDIR files.As is usual in software projects implying code reusability, benefits from using our tool are going to be directly reflected in ease and speed of development.

conclusion

The design used for the DICOMDIR file reader allows the medical application programmer to effortlessly incorporate the feature for the handling of this kind of file in software under development. Also, it avoids exhaustively understanding the DICOM standard and DCMTK classes and methods, which can be really hard and bothersome, enabling the programmer to focus on the integration of the tool into the application, as well as on the development of a navigation interface, according to the application's needs. Because the reading tool has open-source code, it can be reused and modified at will by the programmers. In addition, the design used for the tool development enables inclusion of new data fields to the presently used entities, as well as to add other information model entities to the data tree structure.

By integrating the tool into a medical application that handles DICOM image sets, it was proved it facilitates the browsing and searching of the information contained in the image sets, accelerating the fulfillment of these tasks and improving the efficiency and performance of the application. Open-source programming tools of this type also facilitate the development of medical applications and help to reduce software costs, thus making it more accessible to health institutions, physicians, and patients, particularly in regions where investment in health care solutions is either limited or not a priority issue.

FUTURE WORKs

We are considering the future implementation of tools for converting images from other formats to the DICOM format and for the creation of DICOMDIR files from nonindexed studies files. A tool for anonymization of DICOM fields also could be useful in protecting patient confidentiality during the sharing of clinical data among research teams. We hope that the implementation and subsequent use of this tool set will represent an incremental increase in the efficiency, quality, and speed of development of medical informatics software, producing applications that will be more complete and flexible at same time.

Next post: Technology in Primary and Secondary Medical Education

Previous post: Application of Text Mining Methodologies to Health Insurance Schedules

- **Related Links**
  - Medical Informatics
    - Evaluation of Health Information Systems: Challenges and Approaches
    - Assessing E-Health
    - The Competitive Forces Facing E-Health
    - Health Portals: An Exploratory Review
    - The Telehealth Divide

- :: Search WWH ::

Search

Google™ Custom Search