Journal of Digital Imaging

Mastering DICOM with DVTk

Glenn Potter,[1] Rick Busbridge,[1] Michael Toland,[2] and Paul Nagy[2]

The Digital Imaging and Communications in Medicine (DICOM) Validation Toolkit (DVTk) is an open-source framework with potential value for anyone working with the DICOM standard. DICOM's flexibility requires hands-on experience in understanding ways in which the standard's interpretation may vary among vendors. DVTk was developed as a clinical engineering tool to aid and accelerate DICOM integration at clinical sites. DVTk is used to provide an independent measurement of the accuracy of a product's DICOM interface, according to both the DICOM standard and the product's conformance statement. DVTk has stand-alone tools and a framework with which developers can create new tools. We provide an overview of the architecture of the toolkit, sample scenarios of its utility, and evidence of its relative ease of use. Our goal is to encourage involvement in this open-source project and attract developers to build off and further enrich this platform for DICOM integration testing.

KEY WORDS: DICOM, systems integration, PACS DICOM IHE conformance, Health Level 7, IHE

INTRODUCTION

Digital Imaging and Communications in Medicine (DICOM)[1] plays a major role in the health care information technology (IT) field as the standard for medical images and communication throughout the hospital. With the organization of the DICOM Standards Committee in 1996 and the support of major medical groups and imaging vendors worldwide, DICOM has become a dominant integration mechanism in the hospital enterprise. The DICOM standard version 3.0 contains 16 parts (more than 2,000 pages), and the standard itself is constantly evolving as new software and imaging technologies are developed. Mastering a technology like DICOM can be a daunting task, but the successful student will be on the right path with the DICOM Validation Toolkit (DVTk) from DVTk.org. Consider DVTk the lab kit that is handed out with the DICOM standard on the first day of DICOM 101 class.

As with any other technology, true learning and understanding require hands-on experience.[2] Although the standard itself does not define or identify testing or validation procedures to assess conformance, a number of third-party tools have been developed to fill that role. DVTk is a powerful tool for mastering the intricate DICOM file format and transfer syntax. The mantra of DVTk is to make DICOM easy. If a picture archive and communication system (PACS) adminsitrator, for example, is having a problem with imaging device integration, DVTk can be the first line of defense in tracking the problem. When vendors are pointing fingers at one another, DVTk can help them past recriminations and on to real solutions. For medical software developers, integrators, and testers, DVTk can help to more quickly produce robust systems.

The DICOM Validation Tool (DVT) test framework is a flexible architecture and uses service–object pair (SOP) class definition files, making it adaptable as the DICOM standard evolves. It includes a graphical user interface (GUI) and a command line interface, DICOM media validation, service class user (SCU) and service class provider

[1]From the AGFA HealthCare, Mortsel, Belgium.

[2]From the Radiology IT, University of Maryland School of Medicine, Baltimore, MD, USA.

Correspondence to: Glenn Potter, AGFA HealthCare, 590 North Shore Drive, Hartland, WI, 53029, USA; tel: +262-369-0900; e-mail: glenn.potter@agfa.com.

Online publication 7 August 2007

doi: 10.1007/s10278-007-9057-0

(SCP) emulators, and a rich scripting language. For more advanced testing, VBScript (Microsoft Visual Basic Scripting edition; Microsoft; Redmond, WA) or JScript (Microsoft) can be executed by DVT, and a set of .NET assemblies is available for developing stand-alone test tools with languages such as Visual Basic.NET (Microsoft) and C# (Microsoft).

If a validation tool is to be taken seriously, it must be vendor neutral. With the release of DVT 2.1 in 2005, DVTk now exists as an open-source community project. Not only does this encourage more vendors to contribute (because of the incentive of reduced development and integration costs), but the adoption and proliferation of standards make it easier for individual developers to contribute to open-source projects such as DVTk. Independent software developers are sometimes discouraged from proceeding because of the chance that their work will provide a "one-off" solution only and not be widely used. DVTk, with the backing of the mature DICOM standard, is attracting talented developers with a desire to create something useful and lasting. This success assures that time devoted to learning this toolkit will not be wasted.

This article focuses primarily on the DVT main application. We provide background on the toolkit and specific examples about the two intended roles of DVT: service and development. Current efforts within the DVTk project and future directions are also highlighted.

DVTK HISTORY AND ARCHITECTURE

In 2000, Agfa HealthCare (Mortsel, Belgium) and Philips Medical Systems (Eindhoven, The Netherlands) decided to coordinate activities around DICOM validation testing by bringing together efforts already started by both companies under a joint DVT project. The intention was to produce a DVT that could not only be used internally by both companies to test their own products but also made freely available to other original equipment manufacturers (OEMs) as a means of testing their products to the same level of detail. The ultimate aim was to reduce the time spent integrating proprietary systems by first exposing these systems' equipment to tests run using DVT.

DVT project has a steering committee with responsibility for guiding legal, technical, and commercial aspects. The steering committee meets every 6 months to discuss past progress, current issues, and future requirements. A project manager was elected by the steering committee to manage the DVT project on a daily basis and report back to the committee. Development tasks were divided up based on the available skills of developers who report to the project manager. In its first years, Agfa and Philips provided the personnel to staff the DVT project.

In September 2005, DVT was made into an open-source project (http://www.dvtk.org, DICOM Validation Toolkit; last accessed May 2007) under SourceForge (http://sourceforge.net/projects/dvt, SOURCEFORGE.NET ; last accessed May 2007) as DVTk and is licensed under the GNU Lesser General Public License (LGPL). The steering committee decided that the time had come to begin promoting DVTk to a wider audience, with the aim of attracting other companies who might join and supply development resources. Around this time, ICT Healthcare (Eindhoven, The Netherlands), which had already been supplying development resources to the project, joined as a full member with representation on the steering committee. The goal was to make DVTk the independent gold standard for DICOM validation and thereby improve the interoperability of all vendor products using the DICOM interface.

The dvtk.org Web site is now the location for the latest downloads, defect tracking details, and forums on using DVTk. Interested individuals and companies may join the project through the Web site. A weekly project telephone conference coordinates the activities of the development team.

DVTk-based applications include DVT, the main application, which together with the core forms what is now referred to as the DICOM Validation Framework (Fig. 1). The core includes a DICOM testing data model and object-oriented class structure. The DVTk and DVTkData libraries provide access to this data model and class structure via the managed code adapter. The core includes the DICOMScript language that is supported by DVT. This language can be separated into basic programming, which includes the SEND and RECEIVE commands for simulating DICOM SCPs and SCUs, and advanced programming. The advanced programming language includes commands such as SYSTEM, for executing operating system native applications, and a number of commands for working with the Data Warehouse feature. The Data Warehouse is a run-time memory structure in

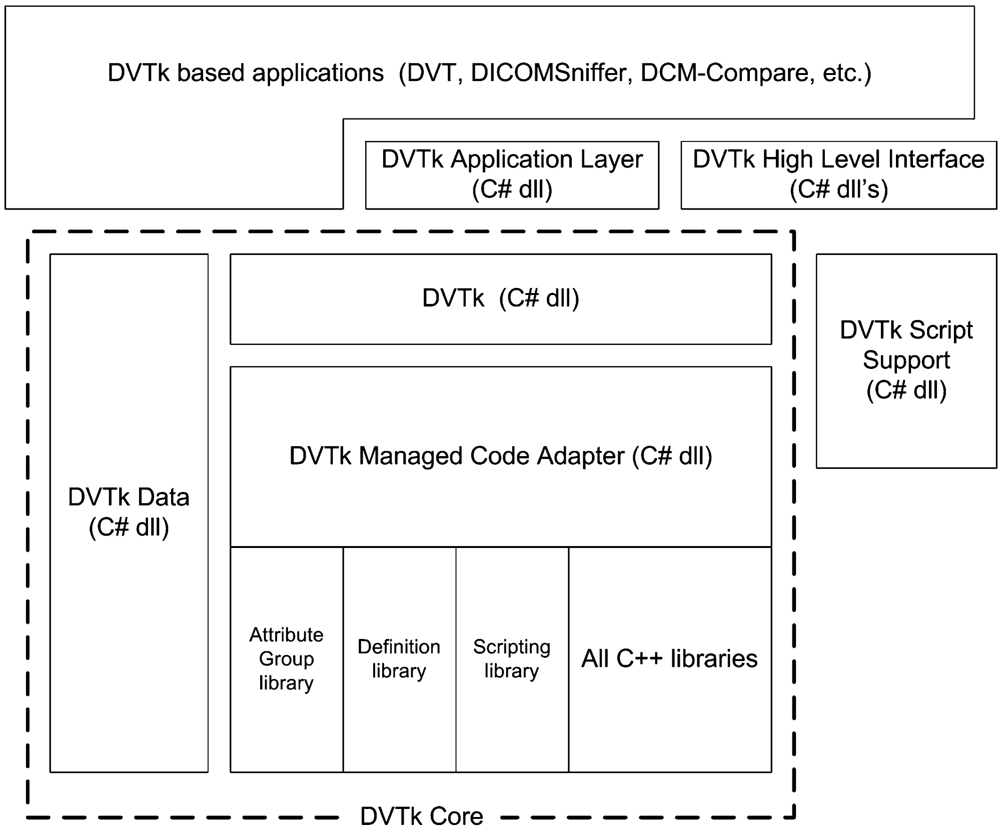

DVTk based applications (DVT, DICOMSniffer, DCM-Compare, etc.)
DVTk Application Layer (C# dll)
DVTk High Level Interface (C# dll's)
DVTk (C# dll)
DVTk Script Support (C# dll)
DVTk Managed Code Adapter (C# dll)
DVTk Data (C# dll)
Attribute Group library
Definition library
Scripting library
All C++ libraries
DVTk Core

Fig 1. DICOM Validation Framework.

which the user can store Association Control Service Element (ACSE) requests and responses, DICOM commands, and DICOM objects for reuse across test scripts and sessions.

For more advanced testing scenarios, DVTk includes the Script Support library and the High-Level Interface (HLI) library. The HLI library is a newer abstraction built on top of the core that makes it easy to write multithreaded tests. The application program interface (API) exposed by this library encapsulates many of the low-level core API classes and methods that make writing VBScripts similar to writing scripts in DVTk's native DICOMScript language. VBScripts can be executed entirely within the DVT GUI application or command-line executable or debugged in Visual Studio .NET. Other DVTk-based applications built on the core and currently available on dvtk.org include: DICOM Network Analyzer, a network sniffer and DICOM protocol analyzer; DICOM Editor, for displaying and editing DICOM files; DICOM Compare Tool, for comparing the attributes and values of two DICOM files; DICOM Attribute Validator, for validating DICOM files, including Structured Report objects, against definition files; DICOM File Stripper, for removing all but mandatory attributes from a DICOM file; and DICOM File Anonymizer, for removing patient and physician information from a DICOM file.

To date, the following contributions have been made to the DVTk project by outside parties, companies, and/or institutions:

- Medical Communications (UK) provided the underlying Transmission Control Protocol/Internet Protocol (TCP/IP) Capture File to the DICOM protocol data unit (PDU) conversion utilities used by the DICOM Network Analyzer application.
- The National Institute of Standards and Technology (NIST; Gaithersburg, MD) provided the Health Level Seven (HL7)[3] validation Web services used in the latest DVTk HL7 validation components.

GETTING STARTED WITH DVTk OUT OF THE BOX

The latest version of DVT can be downloaded from http://www.dvtk.org . It comes packaged as a Microsoft Windows InstallShield application. The installation subfolder includes a user's guide and Windows help files for the extensible .NET assemblies (available only if MS Visual Studio .NET is installed). A large number of examples on how to use the many features of DVT are included in the subfolder.

Starting DVT presents the user with an empty workspace. DVT is designed to work on a single project at a time, although multiple views of the project may be opened from which multiple tests may be simultaneously executed. A DVT project is a container for one or more test sessions. A session is a container for the configuration of one or more tests to be performed against a system under test (SUT). Project and session configuration properties are stored in flat files. The DVT GUI exposes most of the configuration properties, although some of the more advanced settings, such as STRICT-VALIDATION, are available only by directly editing the session file. Some settings can also be modified in script files. For example, the STRICT-VALIDATION script command overrides the value of the STRICT-VALIDATION session property. Some settings, such as CALLED_AE_TITLE, also have built-in default values that are assumed if the property is not defined in a script or session file.

Session files come in three types: emulator, script, and media. Emulator sessions are used when DVT should act as an SCU or SCP emulator to test the DICOM transfer syntax. The emulators have support for Verification SCU/SCP, Storage SCU/SCP, and Print SCP. Script sessions are used when DVT is used to execute a DICOMScript, DICOMSuperScript, or VBScript. A DICOMSuperScript is simply a script that calls one or more DICOMScripts. Script sessions have support for network SCP/SCU message exchange and DICOM media file creation. Media sessions are used to validate the DICOM file format as a DICOMDIR and/or DCM media files. When validating a DICOMDIR file, any referenced files are also validated.

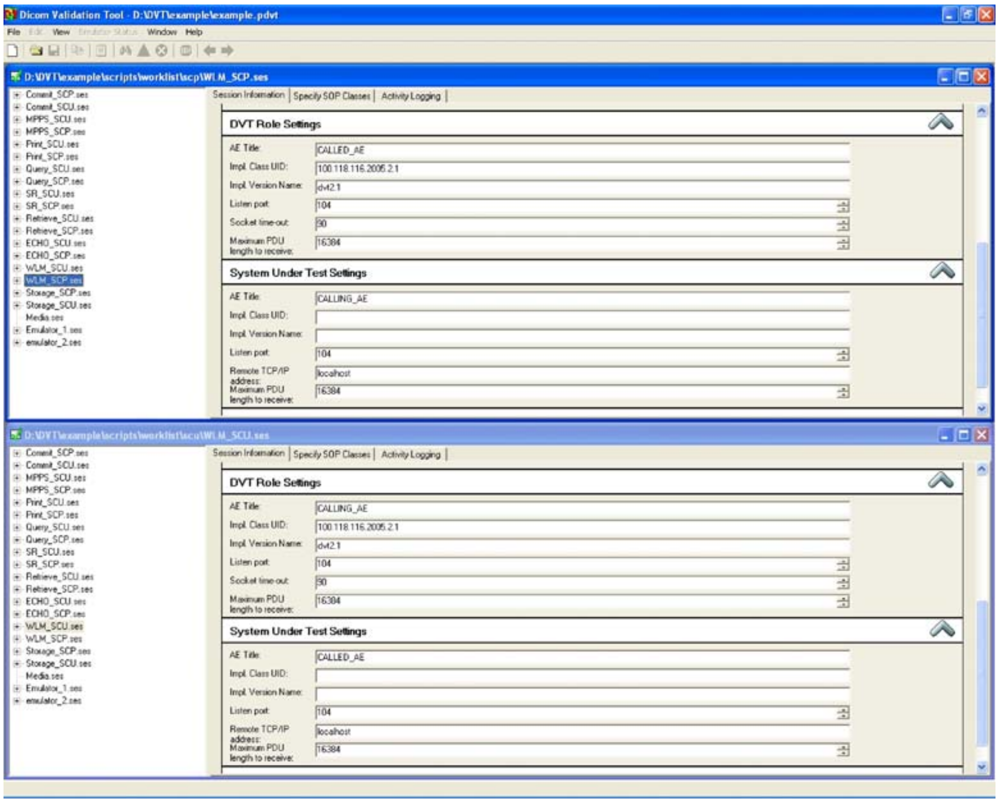
An easy way to gain familiarity with DVT is go through the examples in the subfolder using DVT as both the test tool and the SUT. Opening up the example project displays a tree view of all test sessions defined in the project and provides details on the session currently selected in the tree. Clicking on the Window menu and selecting the New Project View and Tile item displays two views of the project.

The next example will walk through the Modality Worklist (WLM) SCP/SCU test script sessions. Selecting the WLM_SCP session in the top view and the WLM_SCU session in the bottom view presents a view from which two test sessions may be executed simultaneously (Fig. 2).

At the top of the Session Information tab are general settings, including the session type, session ID (used in uniquely naming results files), and location of files used/created by the test session. The DVT Roles Settings section defines the Application Entity (AE) title and some connection settings that DVT assumes during the test session. The SUT Settings section defines the AE title and some connection settings, including the TCP/IP address, of the system the session is testing. Because this example uses DVT as both the testing system and the SUT, the DVT role settings in the top view are synchronized to the SUT settings in the bottom view, and the SUT settings in the top view are synchronized to the DVT role settings in the bottom view.

Locations of the SOP class definition files required for the test session are selected under the Specify SOP Classes tab. The selected definition files are loaded into memory when the test session starts, and DVT uses these to validate the DICOM messages and objects it sends and receives. A definition file describes a single DICOM SOP class in terms of the combination of DICOM Message Service Element (DIMSE) commands and Information Object Definitions (IODs) that make up the SOP class. In this case, both test sessions specify the Modality Worklist Information Model–FIND SOP Class item. The standard definition files that come with DVT in the definitions subfolder are taken directly from the DICOM standard parts 3 and 4. Private definition files can be made that extend the validation capabilities of DVT by copying one of these standard files and modifying it with private user IDs (UIDs), modules, and attributes. A typical customization might specify that when a particular SUT is known to always send a value for the Patient Name (0010,0010) attribute, the corresponding definition file can be modified to define the Patient Name attribute as a type 1 (mandatory, nonzero-length) attribute.

Opening the WLM_SCP tree node in the top view and selecting the DICOMScript, 1.ds, displays a Script tab in the right side of the view. This script



D:\DVT\example\scripts\worklist\scp\WLM_SCP.ses

DVT Role Settings

AE Title:	CALLED_AE
Impl. Class UID:	100.118.116.2005.2.1
Impl. Version Name:	dvt2.1
Listen port:	104
Socket time-out:	90
Maximum PDU length to receive:	16384

System Under Test Settings

AE Title:	CALLING_AE
Impl. Class UID:	
Impl. Version Name:	
Listen port:	104
Remote TCP/IP address:	localhost
Maximum PDU length to receive:	16384

D:\DVT\example\scripts\worklist\scu\WLM_SCU.ses

DVT Role Settings

AE Title:	CALLING_AE
Impl. Class UID:	100.118.116.2005.2.1
Impl. Version Name:	dvt2.1
Listen port:	104
Socket time-out:	90
Maximum PDU length to receive:	16384

System Under Test Settings

AE Title:	CALLED_AE
Impl. Class UID:	
Impl. Version Name:	
Listen port:	104
Remote TCP/IP address:	localhost
Maximum PDU length to receive:	16384

Fig 2. DVT-Worklist Management SCP and SCU test sessions.

defines the test steps for the SCP. Simple DICOM-Scripts such as this can be written using only the SEND and RECEIVE commands. A large number of additional commands are available for more advanced testing scenarios. The Script tab is read-only, but Windows Notepad can be launched from a context menu on the script name in the tree view. A handy DVT DICOM script reference (dvtDICOM-script.hlp) file is in the docs folder. This script first executes a RECEIVE command for an association request message that specifies a single presentation context, consisting of the Modality Worklist Information Model–FIND SOP Class and three possible transfer syntaxes. The SCU script in the bottom view SENDs an association request message. The only required parameter for the ASSOCIATE-RQ message is the presentation context. For the SCP's RECEIVE command, the ASSOCIATE-RQ message is referred to as the reference or expected object. This will be compared to the received ASSOCIATE-RQ message. DVT will perform validation in two steps:

1. The received object is first validated against any loaded definition files. This step ensures that the correct attributes are present in the object and that they are encoded correctly.
2. This step is optional and applied only if a reference object is present in the script. Checks made here are that the expected number of attribute values has been received and that each attribute value matches the corresponding reference object attribute value.

Often, it is not known with what values an SUT will respond, in which case no attributes can be specified in the reference object, so that only step one validation occurs (http://www.dvtk.org "DVT User Guide", version 2.1, August 2005). The VALIDATION script command determines how the DICOM objects are validated by DVT. By default, validation is ENABLED, meaning full validation will occur. The value of the STRICT-VALIDATION session property determines how the result of the validation is handled by DVT. If STRICT-VALIDATION is enabled, the presence of attributes in the received message must match the definitions exactly. If they do not match, then DVT reports a FAILED validation and aborts further DICOMScript interpretation. If STRICT-VALIDATION is disabled and attributes do not match, then DVT reports a WARN message. By default, STRICT-VALIDATION is disabled.

Following a successful association validation and negotiation, the SCU script SENDs a C-FIND-RQ message of type Modality Worklist–FIND. The SCP script RECEIVEs the C-FIND-RQ, validates it, and proceeds to SEND three hard-coded C-FIND-RSP messages. The SCU script was written to mirror the SCP script, so the three reference C-FIND-RSP messages it defines match exactly with those sent by the SCP. These scripts also demonstrate two DVT scripting features: Value Mapping and Value Representation (VR) Keywords. Value mapping allows the user to substitute a label for a value defined in a script, generated by DVT, or received from the SUT, and then refer to that value with the label further down in the script. The LABEL: keyword is used to map an attribute value defined in the script or received from the SUT. The NEW: keyword tells DVT to generate a new value and assign it to the given label. An example of this is in the SCP script, where the Study Instance UID attribute (0020,000D) is assigned a new value generated by DVT and named with the label StudyInstanceUid1. The generated UID value can be referred to subsequently in the script as StudyInstanceUid1 (although this was not done in this example). VR keywords, such as the AUTOSET keyword used in both the SCU and SCP scripts, tell DVT to generate a value and assign it to the corresponding attribute. In the example case, AUTOSET is used in the Scheduled Procedure Step Start Date attribute to ensure that values match between the two scripts. Both Value Mapping and VR Keywords are sensitive to the type of attribute to which they are applied.

This test is begun by right clicking on the SCP script and selecting execute. When DVT is executing a test session, all tabs except the Activity Logging tab are hidden, the session tree in the corresponding view is disabled, and the test stop button in the toolbar is enabled. The SCP's Activity Log will indicate that it is waiting for a connection on port 104. Executing the SCU script in the lower window will result in a number of messages in the Activity Logs, followed by the termination of the test sessions. Comparing and correlating the SCP and SCU activity log entries provides a good visual reference for the DICOM message flow between two AEs.

By default, the AUTO-TYPE-2-ATTRIBUTES session property is set to true in the session file, which means that DVT will automatically add any zero-length type 2 (type 2 attributes must be included; however, they may be encoded with a zero-length value or no value) attributes from the definition file to the dataset before sending to the SUT. This behavior is recorded in the SCU's activity column by the “Automatic Type 2 Attributes population...” statement. This feature means that only type 1 (required, nonzero-length) attributes must be explicitly stated in the scripts.

After completion of the scripts, each view displays the Validation Results tab, where the results of the test sessions are displayed. DVT stores the results of each test in a _res.xml file in the test session's configured results directory. The file name takes the form <Detail|Summary>_nnn_<scriptName>_res.xml, where nnn is the session ID from the session properties. Varying the session ID from one test execution to the next allows storage of multiple sets of results in the results directory. Two session Boolean properties define whether summary or detailed results are generated: SUMMARY-VALIDATION-RESULTS and DETAILED-VALIDATION-RESULTS. If the test session's STORAGE_MODE property is set to as-media or as-dataset, any media files received by DVT during the test are also stored in the results directory, again using the session ID to help uniquely name the files. The generated results files are listed in the session tree under the script name. Selecting one of the results files displays it in the Validation Results tab. The Validation Results tab is a Hypertext Markup Language viewer allowing navigation between the summary results, detailed results, and any generated media files using hypertext links. DVT does not include a DICOM file viewer. To automatically view a generated .dcm file by clicking on a link in the Validation Results tab, a DICOM image viewer must be installed and associated with the .dcm file type.

The SCP's Validation Results tab will show a test result of PASSED, and a .dcm media file is created that contains the Modality Worklist–FIND dataset received in the C-FIND-RQ message sent from the SCU. The SCU's Validation Results tab also shows a result of FAILED, with nine errors reported. The Summary Results File lists the errors; in this case, three type 1 attributes are missing from each C-FIND-RSP message sent from the SCP. Clicking on a Link to Detailed Result link displays the errors in the context of the complete C-FIND-RSP message. The detailed results file contains the complete contents of each message sent and received in chronological order. Any comment lines in the DICOMScript that begin with ## are copied directly to the detailed results file. This is a good way to insert additional test information directly into the results. For additional troubleshooting output in the Activity Logging tab and detailed results file, the LOG-RELATION, LOG-DEBUG, LOG-DULP-STATE, and PDU-DUMP test session properties can be enabled. In the example described here, it is left as an exercise for the user to add the missing type 1 attributes to the SCP script so that the SCU test session result becomes PASSED.

DICOMSuperScripts (script files with a .dss extension) enable the reuse of DICOMScripts in various test scenarios. The storage example script sessions included with DVT demonstrate the benefits of DICOMSuperScripts.

DVTk FOR SERVICE TROUBLESHOOTING

As demonstrated in the previous section, DVTk can be a useful tool when troubleshooting modality worklist problems. Another common use case would occur when adding new modalities to a PACS network. It can be frustrating to ensure that AE titles, ports, and IP addresses match between the modality and PACS configuration. This section will illustrate ways in which DVT can be used to troubleshoot a PACS-modality interface problem and describe another, more service-orientated tool available from DVTk.org.

Because DVT can act as both an SCP and SCU, it is an excellent starting point for troubleshooting PACS-modality problems. One of the most common and most frustrating failures is when a modality does not send images to the PACS. Most modalities provide little or no error information when image storage problems occur with the PACS. The PACS system is often equally unhelpful. Without an error message of some kind, PACS administrators have no guidance on where to begin in addressing the problem. DVT can simulate the modality or the PACS to pinpoint the source of the problem while obtaining hard evidence that can be used to engage the involvement of additional service layers, such as the hospital networking group or modality vendor. The Emulator_1 test session in the example project that comes with DVT installation can be used to emulate the problem modality. The problem modality's AE title and PACS connection properties must be copied to the Emulator_1 session's properties. Although not absolutely necessary, the modality being emulated should be taken off the network to prevent any AE title conflicts during troubleshooting. The first test to perform is a simple Verification to ensure that there is DICOM network communication between the modality AE and the PACS. The test is started by opening the Emulator_1 session's tree node, right clicking on the Storage SCU Emulator node, and selecting the Execute menu item. All tabs except the Activity Logging tab are hidden, and a dialog box is displayed. Clicking on the Echo button tells DVT to send a C-ECHO message to the SUT. The activity logging tab will display the operations performed. For the purposes of this article, the tests are run against the open-source DICOM PACS DCM4CHEE (http://www.dcm4che.org, dcm4chee-2.x DICOM Clinical Data Manager system; last accessed May 2007).

If the verification test succeeds, then one can assume the patency of DICOM network connectivity between the modality and PACS. The next test is to attempt image storage to the PACS. This test requires a set of DICOM test images, preferably from the modality that is experiencing the problem. The images should contain no names, IDs, or UIDs that will conflict with real-world data. It is important, however, that the data used to perform the test matches the real-world data in structure as closely as possible, including any private data elements the modality creates. A copy of the modality's DICOM conformance statement comes in handy here. This will help simulate the behavior of the modality as closely as possible. Under the Specify Transfer Syntax (TS) button on the Session Information tab, all transfer syntaxes supported by the modality as a storage SCU can be selected. For example, to test a computed tomography (CT) modality, one would select the CT Image Storage SOP Class (1.2.840.10008.5.1.4.1.1.2). For this test, the storage SOP class and transfer syntax that match the DICOM test images that will be stored to the PACS should be selected. Note that this step is not actually necessary; when emulating a storage SCU, DVT will automatically add to the association negotiation the transfer syntaxes and SOP Classes from the

DICOM media files being stored. Selecting the correct transfer syntaxes and SOP Classes is necessary when emulating a storage SCP, however. Executing the Storage SCU Emulator opens the Storage SCU Emulator dialog box. In the dialog box, the Add button is used to add the test images to the list that will be sent to the SCP. Selecting the Validate before export option tells DVT to validate the media files against the corresponding definition file prior to sending them. The number of associations the modality uses when sending multiple images is also specified here (most modalities would send multiple images on a single association). Finally, clicking the Send button will tell DVT to execute the storage test and close the dialog. DVT's detailed results will show the verification of each media file followed by each storage transaction. The Storage SCU emulator does not automatically do storage commitment as the SCP emulator does. To simulate the modality performing a storage commitment request, one could create a DICOMScript session similar to the Commit_SCU example session, modifying the SOP instance UIDs to match the images stored. This test should be performed immediately after the storage test.

To demonstrate some of the more advanced DICOMScript features and the Data Warehouse in DVT, one can imagine a scenario where a CT modality has recently undergone a software upgrade. Since the upgrade, technicians have been reporting that although they can successfully post-process images on the modality before sending them to the PACS, processing of images retrieved from the PACS back to the modality fails. No errors have been reported by the PACS or modality on storage or retrieval. One possible reason is that the PACS is doing something to the stored images that is preventing the modality from processing them. To evaluate, one can perform what is called a PACS transparency test. The idea is to make what the PACS is doing (if anything) to the images transparent by comparing the before and after images. This test will require a DICOM CT image file from the modality on the DVT workstation and a DVT project with two sessions. A script session playing the role of SCU is used to put the image in the DVT Data Warehouse, store it to the PACS, and initiate a C-MOVE from the PACS back to DVT. An Emulator SCP session is used to receive the image moved back from the PACS and for performing the validation of the received image against the original image in the Data Warehouse.

The first commands the script performs are to reset the Data Warehouse and read the test image into the Data Warehouse (Fig. 3). The first READ command loads the DICOM image into the Data Warehouse and uses the value of attribute 0×00080018, the SOP Instance UID, as a reference value. This will allow the SCP Emulator that subsequently receives the image back from the PACS to automatically compare it to the image in the Data Warehouse based on the fact that the SOP Instance UID values are the same. The second READ command loads the same image again into the Data Warehouse but this time references it with CTIMAGE1. This reference will be used to export the image directly from the Data Warehouse to the PACS. As an alternative to READing the image into the Data Warehouse, one could use the CREATE and SET commands to build an image object directly in the Data Warehouse. DVT has the ability to generate pixel data patterns for VRs of type other byte (OB), other float (OF), and other word (OW). For the purposes of this example, it is best to use an image from the problem CT modality. The next operation the script performs is to establish a C-STORE CT Image association with the PACS. After establishing the association, the script creates a C-STORE-RQ command object in the Data Warehouse and exports the CT Image referenced by CTIMAGE1 using the C-STORE-RQ command referenced by CSTOREREQID, over the established association (Fig. 4). After closing the storage association, the script creates a new association on which to perform the C-MOVE, sends the request to move the image from the PACS to the DVT AE, and then closes the association (Fig. 5). Note that the Patient ID (0×00100020) and SOP Instance

```
RESET ALL

READ "..\Storage_Emulator\Images\00000320.dcm" 0x00080018
READ "..\Storage_Emulator\Images\00000320.dcm" "CT Image" CTIMAGE1
```

Fig 3. PACS transparency test using DICOMScript-reading image into the Data Warehouse.

```
CREATE C-STORE-RQ CSTOREREQID

EXPORT C-STORE-RQ CSTOREREQID "CT Image" CTIMAGE1
RECEIVE C-STORE-RSP((0x00000900, US, 0x0000))
```

Fig 4. PACS transparency test using DICOMScript-exporting image from the Data Warehouse.

UID (0×00080018) were manually copied from the CT Image test file. Also, before executing this test, the SCP Emulator must already be started and waiting for the C-STORE from the PACS in response to the C-MOVE request. If the test executes successfully, the statement "Reference Dataset with identifier '1.3.46.670589.10.900123.19970114.35042000040' found in Warehouse" should appear in the Activity Log indicating that DVT found the same CT image previously loaded into the Data Warehouse and will use it to perform a comparison of the dataset attributes and values.

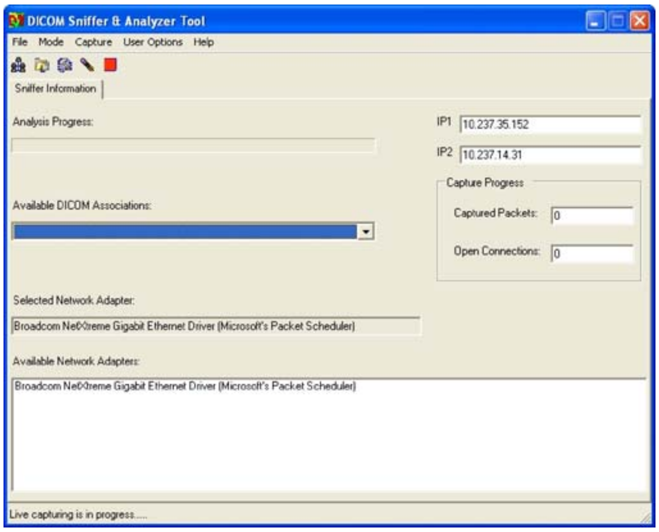
This test requires that the PatientRootQueryRetrieve-MOVE.def definition file is loaded in the script session and the CTImageStorage.def definition file is loaded in the Emulator SCP session. If, as suspected, the PACS is modifying the CT Image object before returning it to the modality, the Emulator SCP results data will identify where these modifications occurred. In this case, the hypothetical tester notices that DVT is indicating that a set of private attributes in the Data Warehouse copy of the image are missing from the image returned from the PACS. Upon further investigation, it turns out that the software upgrade performed on the modality included the addition of a set of new private attributes required by the modality to perform new post-processing algorithms on the images. The PACS does not support these new private attributes for some reason and is stripping them from the image objects.

Instead of using a combination of a DICOMScript test session and Emulator test session, this test could have been implemented with a single VBScript session using the HLI library. In this case, separate threads would be created for the SCU and the SCP parts. This approach is left up to the user as an exercise.

Although DVT can analyze all aspects of DICOM interfacing, it tends to be overloaded for field service issues. DVTk.org has a number of service-orientated tools also based on the DVTk framework. One of these is the DICOMSniffer & Analyzer Tool. The tool is a GUI-based network sniffer and DICOM protocol analyzer. This tool uses the WinPcap open-source network library for packet capturing and filtering and includes a user guide. This tool can sniff a live DICOM network stream given two endpoints, such as a modality and PACS. The troubleshooting power of this tool should not be underestimated. It can perform message validation between two devices similar to DVT but does not require replacing an SUT device with the testing tool. Figure 6 shows the tool capturing a modality worklist query/response communication between two IP addresses.

```
SEND ASSOCIATE-RQ(
   PRESENTATION-CONTEXT-ITEMS(
      "Patient Root Query/Retrieve Information Model - MOVE SOP Class",
      "Implicit VR Little Endian"))
RECEIVE ASSOCIATE-AC(
   PRESENTATION-CONTEXT-ITEMS(
      "Patient Root Query/Retrieve Information Model - MOVE SOP Class",
      0,
      "Implicit VR Little Endian"))

SEND C-MOVE-RQ "Patient Root Query/Retrieve - MOVE"(
   (0x00000002, "Patient Root Query/Retrieve Information Model - MOVE SOP Class")
   (0x00000600, AE, "DVT")
   (0x00080052, CS, "IMAGE")
   (0x00100020, "0000000867")
   (0x00080018, "1.3.46.670589.10.900123.19970114.35042000040"))
RECEIVE C-MOVE-RSP(
   (0x00000002, "Patient Root Query/Retrieve Information Model - MOVE SOP Class")
   (0x00000900, 0x0000))

SEND RELEASE-RQ
RECEIVE RELEASE-RP
```

Fig 5. PACS transparency test using DICOMScript-moving the image back to DVT.

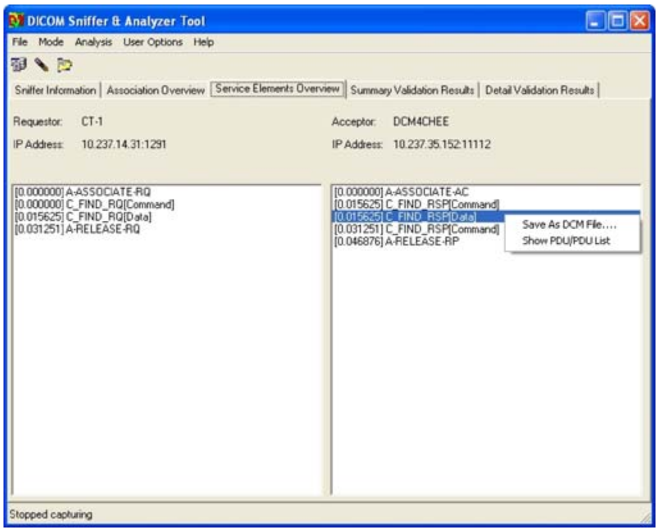

DICOM Sniffer & Analyzer Tool
File Mode Capture User Options Help
Sniffer Information

```
Analysis Progress:
IP1 10.237.35.152
IP2 10.237.14.31
Capture Progress
Captured Packets: 0
Open Connections: 0
Available DICOM Associations:
Selected Network Adapter:
Broadcom NetXtreme Gigabit Ethernet Driver (Microsoft's Packet Scheduler)
Available Network Adapters:
Broadcom NetXtreme Gigabit Ethernet Driver (Microsoft's Packet Scheduler)
Live capturing is in progress.....
```

Fig 6. DICOM Sniffer & Analyzer tool in capturing mode.

After the capture process, the user is presented with a per-association analysis of the DICOM control and datasets that were captured. The Association Overview tab shows the requested and accepted services and each field of the association PDUs. In the Service Elements Overview tab (Fig. 7), one can see all of the DIMSE messages that were transmitted during the association, save a copy, and view the PDUs. The summary and detailed validation results are similar to those produced by DVT, including message validation against SOP Class definition files. The tool also has the option of saving the captured data to a capture file for latter analysis or for sending on to additional service personnel. The DICOM Sniffer & Analyzer is a tool every PACS administrator should have on his or her workstation.

Two more tools built on the DVTk framework that should be in the imaging professional's toolbox in working with DICOM on a regular basis are DICOM Compare and DICOM Editor. DICOM Compare can compare the attributes and values of two DICOM files. It can also filter out of the compare process any attributes and sequences to make it easier to identify offending elements. A known "good" DICOM object can be compared to


DICOM Sniffer & Analyzer Tool
File Mode Analysis User Options Help
Sniffer Information
Association Overview
Service Elements Overview
Summary Validation Results
Detail Validation Results
Save As DCM File....
Show PDU/PDU List

```
Requestor:  CT-1                      Acceptor:   DCM4CHEE
IP Address: 10.237.14.31:1291         IP Address: 10.237.35.152:11112

[0.000000] A-ASSOCIATE-RQ             [0.000000] A-ASSOCIATE-AC
[0.000000] C_FIND_RQ[Command]         [0.015625] C_FIND_RSP[Command]
[0.015625] C_FIND_RQ[Data]            [0.015625] C_FIND_RSP[Data]
[0.031251] A-RELEASE-RQ               [0.031251] C_FIND_RSP[Command]
                                      [0.046876] A-RELEASE-RP
Stopped capturing
```

Fig 7. DICOM Sniffer & Analyzer tool in analyzing mode.

one causing a problem that was captured from the network with DICOM Sniffer.

With the DICOM Editor, one can add/delete/modify any attribute or sequence and sequence item and save the modified DICOM file to any location. This tool goes hand-in-hand with the DVT, DICOM Sniffer, and DICOM Compare tools. After those tools have identified the DICOM elements that may be causing a DICOM object storage problem, the editor can be used to manually fix the DICOM object and then resend it to the system where the failure is occurring. If the store succeeds, chances are the source of the problem has been identified.

DVT AS A DEVELOPMENT TEST TOOL

Another use of DVT is automated unit or system testing in a software development environment. Most modern source code management systems include automated build-and-test subsystems. Scripts can be created to test the DICOM interfaces of a vendor's product, and the command line version of DVT can then be launched to execute those scripts against the product as part of the normal automated build-and-test processes. The DVT results and output files can then be saved along with the rest of the build/test artifacts. The command line version of DVT can be called on a DICOMScript as follows: dvtcmd Modality_System_Test.ses Modality_System_Test.ds. The summary and detailed results Extensible Markup Language files output by DVT make it easy for an automated test system to determine the results of the test, generate a report, and take any other appropriate action, such as e-mailing the development team.

A tool as programmable and rich in output as DVT is valuable to the entire iterative develop-

```
Imports System
Imports System.Collections
Imports System.IO

Imports VR = DvtkData.Dimse.VR
Imports DimseCommand = DvtkData.Dimse.DimseCommand
Imports Attribute = DvtkHighLevelInterface.Dicom.Other.Attribute
Imports DvtkHighLevelInterface.Dicom.Messages
Imports DvtkHighLevelInterface.Dicom.Other
Imports DvtkHighLevelInterface.Dicom.Threads
Imports DvtkHighLevelInterface.Common.Threads
Imports DvtkHighLevelInterface.Common.UserInterfaces
…
Module Module1
  Sub Main(ByVal CmdArgs() As String)
    Dvtk.Setup.Initialize()
    Dim theThreadManager As ThreadManager = New ThreadManager

    … validate command line arguments …

    Dim theMainDicomThread As MainDicomThread = New MainDicomThread( _
      System.Convert.ToString(numModalities), _
      System.Convert.ToString(numImages), _
      defsDir)

    theMainDicomThread.Initialize(theThreadManager)

    … set some main thread options …

    ' Set the identifier.
    theMainDicomThread.Options.Identifier = "Main DICOM Thread"

    ' Care that any activity logging will be displayed to the HliForm.
    Dim theHliForm As HliForm = New HliForm
    theHliForm.AutoExit = False
    theHliForm.Attach(theMainDicomThread)

    ' Start the actual execution of theMainDicomThread.
    System.Console.Out.WriteLine("Starting main DICOM thread")
    theMainDicomThread.Start()

    ' Wait until all threads have finished executing.
    theThreadManager.WaitForCompletionThreads()

    … output some test performance statistics …

    Dvtk.Setup.Terminate()
```

Fig 8. Multithreaded stress test using VB.NET-main subroutine and imports.

ment process. Software validation teams can link test session files, scripts, and result files to software defect issues, so that a developer can then use them to reproduce the defect. A validator can subsequently use those same scripts to test the fixed software. Maintaining a repository of DVT scripts is also an excellent way to run regression tests.

If the DICOM software being tested is in the .NET language family, tighter testing integration with DVTk can be achieved via NUnit,[4] the open-source unit testing framework for .NET. This integration uses a DVTkInvoker class to invoke the command line version of DVT with a session file, script file, and results file as arguments. One can use DVT Clients and Servers within NUnit to handle the various DICOM SOP Classes, starting and stopping DVT as necessary and obtaining validation results information. One future goal is to be able to start up and test an entire workflow scenario from NUnit using DVT.

DVTk makes it quick and easy to perform repeatable, detailed testing of DICOM interfaces. In the following example, we posit a stress test to run against a storage SCP service. Because this is a stress test, the test application must hit the SCP simultaneously from multiple SCUs, so that multiple threads will be needed. Writing multithreaded Visual Basic (VB) test applications is a breeze with the new HLI library. In fact, it looks quite similar to a DICOMScript, with high-level abstractions for creating and releasing associations and for sending and receiving messages. This example using the HLI requires DVTk alpha version 2.1.007.006 or greater , which can be downloaded from dvtk.org after registering as a Plus User. Plus Users have access to early releases and draft documents, such as the draft version of the HLI API help file.

```
Class MainDicomThread
  Inherits DicomThread

  Private modalityCount As Integer
  Private numImagesPerAssociation As Integer
  Private defsDir As String

  …
  Protected Overrides Sub Execute()

    Log("Starting " & modalityCount & " modalities")
    For i As Integer = 1 To modalityCount

      ' Create and start a CT modality
      Dim storageScu As CtStorageScu = New CtStorageScu( _
        numImagesPerAssociation, defsDir)
      storageScu.Initialize(Me)
      storageScu.Options.Identifier = "CT Storage SCU # " & i
      storageScu.Options.SessionId = System.Convert.ToUInt16(i)
      storageScu.Options.Name = "Storage_SCP_Stress_Test - " & i

      storageScu.Options.LogChildThreadsOverview = False
      storageScu.Options.LogThreadStartingAndStoppingInParent = False
      storageScu.Options.LogWaitingForCompletionChildThreads = False

      storageScu.Options.DvtAeTitle = "DVT_" & i
      storageScu.Options.DvtPort = 103 + i

      storageScu.Options.SutAeTitle = "DCM4CHEE"
      storageScu.Options.SutIpAddress = "localhost"
      storageScu.Options.SutPort = 11112

      storageScu.Options.LoadDefinitionFile( _
        defsDir & "\CTImageStorage.def")
      storageScu.Options.StrictValidation = False

      storageScu.Options.StartAndStopResultsGatheringEnabled = False
      storageScu.Options.GenerateDetailedResults = False
      storageScu.Options.ShowResults = False
      storageScu.Options.StorageMode = Dvtk.Sessions.StorageMode.NoStorage

      storageScu.Start()

    Next i
    Log("Modalities started")

  End Sub
```

Fig 9. Multithreaded stress test using VB.NET-MainDicomThread class.

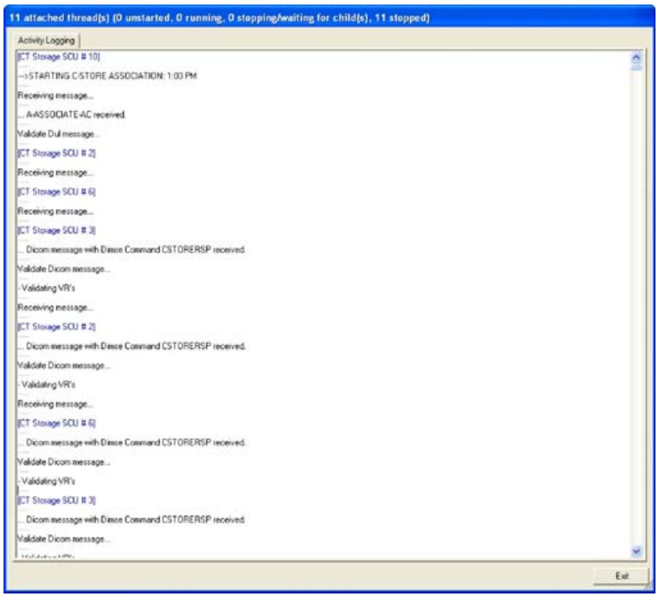
```
Class CtStorageScu
  Inherits DicomThread

  Private numImages As Integer
  Private ctIodTemplate As DicomMessage = New DicomMessage( _
    DimseCommand.CSTORERQ)
  Private uidGenerator As DvtkData.Dimse.UniqueIdentifier = _
    New DvtkData.Dimse.UniqueIdentifier
  Private xActionTimeMs As Long

  Public Sub New(ByVal numImages As Integer, ByVal defsDir As String)
    Me.numImages = numImages
    Me.ctIodTemplate.DataSet.Read(".\images\ct_image.dcm", _
      defsDir & "\CTImageStorage.def")
  End Sub

  Protected Overrides Sub Execute()

    'Generate some unique values for the study
    Dim studyUid As String = uidGenerator.GenerateUidValue()
    Dim seriesUid As String = uidGenerator.GenerateUidValue()
    Dim uniqueNum As Long = System.DateTime.Now.Ticks
    Dim accNum As String = "ACC_" & uniqueNum
    Dim patientName As String = "PATIENT_LAST_NAME_" & uniqueNum _
      & "^PATIENT_FIRST_NAME"
    Dim patientId As String = "PATIENT_ID_" & uniqueNum
    Dim studyId As String = "STUDY_ID_" & uniqueNum

    Dim startTime As System.DateTime = System.DateTime.Now
    Log("Starting C-STORE Association: " & startTime.ToLongTimeString())

    SendAssociateRq _
    ( _
       New PresentationContext("1.2.840.10008.5.1.4.1.1.2", _
         "1.2.840.10008.1.2.1") _
    )
    ReceiveAssociateAc()

    For i As Integer = 1 To numImages

      Dim theCStoreRequest As DicomMessage = New DicomMessage( _
        DimseCommand.CSTORERQ)

      ' Create a new dataset from the template
      theCStoreRequest.DataSet.CloneFrom(ctIodTemplate.DataSet)

      theCStoreRequest.Set("0x00000002", VR.UI, "1.2.840.10008.5.1.4.1.1.2")

      ' Common study values for the image set
      theCStoreRequest.Set("0x00080050", VR.SH, accNum)
      theCStoreRequest.Set("0x0020000D", VR.UI, studyUid)
      theCStoreRequest.Set("0x0020000E", VR.UI, seriesUid)
      theCStoreRequest.Set("0x00100010", VR.PN, patientName)
      theCStoreRequest.Set("0x00100020", VR.LO, patientId)
      theCStoreRequest.Set("0x00200010", VR.SH, studyId)

      theCStoreRequest.Set("0x00100030", VR.DA, "19500101")

      ' Make unique image identifiers
      theCStoreRequest.Set("0x00080018", VR.UI, _
        uidGenerator.GenerateUidValue()) 'SOP Instance UID
      theCStoreRequest.Set("0x00200052", VR.UI, _
        uidGenerator.GenerateUidValue()) 'Frame of Reference UID
      theCStoreRequest.Set("0x00200012", VR.IS, i) 'Acquisition Number
      theCStoreRequest.Set("0x00200013", VR.IS, i) 'Instance Number

      Send(theCStoreRequest)
      Dim theCStoreResponse As DicomMessage = ReceiveDicomMessage()

      Dim StatusValue As Int32 = Convert.ToInt32( _
        theCStoreResponse.CommandSet("0x00000900").Values(0))
      If Not (StatusValue = &H0) Then
        Log("Received C-STORE-RSP with status value of " & _
          System.Convert.ToString(StatusValue))
        Exit For 'Exit loop on error
      End If

    Next i

    setStorageTimeMs(System.DateTime.Now.Subtract(startTime).Milliseconds)

    SendReleaseRq()
    ReceiveReleaseRp()
    Log("Closed C-STORE Association: " & _
      System.DateTime.Now.ToLongTimeString())

  End Sub
```

Fig 10. Multithreaded stress test using VB.NET-CtStorageScu class.

In Visual Studio .NET, a VB console project is created and references to the Dvtk.dll, DvtkData.dll, and DvtkHighLevelInterface.dll libraries are added. This simple stress test application takes three arguments: the number of modalities to create (each modality runs on a separate thread), the number of images each modality will store to the SCP under test, and the location of the DVTk definition files. Figure 8 shows the imports required and the Sub Main(ByVal CmdArgs() As String) routine in module Module1. The first operation this application entry point method performs is to call Dvtk.Setup.Initialize(). This must be performed before any calls to the DVTk libraries, matched by the last call in the application, which should be Dvtk.Setup.Terminate(). Then it creates and initializes the MainDicomThread object, attaches it to the activity logging form, starts the main DICOM thread, and finally waits for the threads to complete. After the threads are complete, it logs selected performance data gathered during the stress test.

Figure 9 shows the MainDicomThread class that inherits from the DvtkHighLevelInterface.Dicom.Threads.DicomThread class. A single instance of this class is started from the entry point method. This class's thread execute method starts all of the modalities and returns. As it creates each modality (instances of CtStorageScu), it generates unique identifying information for each SCU, such as AE title. Some of the options are analogous to test session properties. This test application could have loaded a script session configuration file and initialized some of the options from it, but in this case, all of the necessary session options are set directly in the code. After an SCU is created, it is immediately started by calling its thread Start() method.

Figure 10 shows a CtStorageScu class that also inherits from DvtkHighLevelInterface.DICOM.Threads.DICOMThread. The purpose of the execute method is to C-STORE a specified number of images to the SCP under test. It first creates unique patient, study, and series level identifiers for the images that will be stored. It then creates the C-STORE CT Image association; then loops to store the images, cloning a CT image dataset that was read from a DICOM CT image file in the constructor; creates unique image identifiers for each image; sends the C-STORE-RQ message; receives the response message; and checks the response status value. Finally, it logs its total storage time for later analysis and closes the association. This application could have been created and executed from the DVT GUI, but developing it in Visual Studio allows the application to be debugged and compiled to an executable file. This application is executed from the command line as: Storage_SCP_Stress_Test.exe 20 1000 "C:\Program Files\DVT\Definitions".



```
11 attached thread(s) (0 unstarted, 0 running, 0 stopping/waiting for child(s), 11 stopped)

[CT Storage SCU # 10]
-->STARTING C-STORE ASSOCIATION: 1:03 PM
Receiving message...
... A-ASSOCIATE-AC received
Validate Dul message...
[CT Storage SCU # 2]
Receiving message...
[CT Storage SCU # 6]
Receiving message...
[CT Storage SCU # 3]
... Dicom message with Dimse Command CSTORERSP received
Validate Dicom message...
- Validating VR's
Receiving message...
[CT Storage SCU # 2]
... Dicom message with Dimse Command CSTORERSP received
Validate Dicom message...
- Validating VR's
Receiving message...
[CT Storage SCU # 6]
... Dicom message with Dimse Command CSTORERSP received
Validate Dicom message...
- Validating VR's
[CT Storage SCU # 3]
... Dicom message with Dimse Command CSTORERSP received
Validate Dicom message...
```

Fig 11. Multithreaded stress test-activity logging.

Figure 11 is an excerpt from the Hl1Form that receives all of the logging from the DicomThread objects. Notice the multithreading capabilities of the HLI in this output. While modality 10 is validating an A-ASSOCIATE-AC message it received, modality 3 is handling a CSTORERSP message. This test created 20 CT modalities that are each simultaneously sending a 1,000-image series to the PACS.

RECENT EXTENSIONS AND FUTURE WORK

DVTk has been extended recently to support HL7 validation in addition to DICOM validation. This is currently used by the DVTk Integrating the Healthcare Enterprise (IHE)[5] Actors framework where it is possible to configure DVT to emulate the role of certain actors in the IHE integration profiles. The idea is that the IHE Actors necessary to allow the SUT to be tested are emulated by DVTk. DVTk will then validate the DICOM and HL7 transactions taking place between actors and, in addition, compare the values of certain attributes such as Patient ID, Patient Name, etc., between messages to ensure consistent use.

In the future, the aim is to enhance the capabilities of the DVTk IHE Actors framework by supporting additional functionality in the existing actors and supporting other actors needed in the various IHE Integration Profiles. Support for other protocols is also envisioned.

The DVTk development team is also involved in the new IHE Connectathon Toolkit (code-named “Gazelle”), which is being developed as a successor to the MESA Toolset. It is hoped that the DVTk DICOM validation engine can be wrapped as a Web service for use in Gazelle.

Various new tools will be developed using the DVTk frameworks. A GUI for the DVT-IHE framework; various new emulators, such as a radiology information system emulator and modality emulator; and stand-alone validation applications are likely candidates. The current number of development resources limits what can be done—anyone wishing to contribute can do so via the Web site.

The aim for future IHE extensions is to integrate any other validation services into the DVTk framework as Web services. This may include Cross-Enterprise Document Sharing validation, for example. It is hoped that the Gazelle cooperation will result in web services that can be reused for this purpose.

CONCLUSION

Originally developed by Agfa and Philips to test their products, DVTk has grown into a professionally managed, open-source, vendor neutral, DICOM Validation Framework. Flexible for the changing standard, programmable, and extensible, DVTk is a powerful tool for anyone working with the DICOM standard. The toolkit includes GUI and command line versions of the main validation application, DVT, and a collection of .NET libraries for creating new validation and test tools. The large collection of example validation sessions that come with the toolkit are a great place to start for understanding how the DICOM standard works in practice.

Hospital IT staff can use DVTk for simple tasks, such as pinging the network for the existence of modality AE titles, or for more detailed troubleshooting, such as checking for the presence of specific DICOM attributes and values in messages and media files. Other DVTk-based tools are available for tasks such as editing or comparing DICOM files or for capturing and analyzing messages on a live DICOM stream. The XML-structured test results and media files created by the validation framework provide great evidence for IT staff when discussing a problem with vendors. Developers and testers can create scripts for testing their products DICOM conformance, or build on the framework by creating new stand-alone test tools. NUnit integration and the command line version of DVT make it possible to incorporate DVTk and the generated test results into automated build-and-test systems.

Current and future efforts include support for IHE actor validation, including HL7 validation, and new device emulators, such as a Radiology Information System (RIS) emulator. As the digital hospital enterprise continues to grow, DVTk is well positioned to take on these new validation roles.

We encourage the reader to continue working with DVTk as a means to master the changing

DICOM environment. With the move to an open-source community approach, DVTk is well on its way to becoming the independent gold standard for DICOM interface testing.

ACKNOWLEDGMENT

I would like to thank Dr. Nancy Knight from the University of Maryland for her expert assistance in preparing this manuscript.

REFERENCES

1. National Electrical Manufacturers Association (NEMA). DICOM standard is available at http://medical.nema.org (last accessed May 2007)
2. Bidgood Jr, WD, Horii SC, Prior FW, Van Syckle DE: Understanding and using DICOM, the data interchange standard for biomedical imaging. JAMIA 4:199–212, 1997
3. HL7, Health Level Seven, http://www.hl7.org/ (last accessed May 2007)
4. NUnit, "NUnit is a unit-testing framework for all .Net languages", nunit.com, (last accessed May 2007)
5. IHE, Integrating the Healthcare Enterprise, http://www.ihe.net/ (last accessed May 2007)