Arthemis: Annotation software in an integrated capturing and analysis system for colonoscopy

Danyu Liu, Yu Cao, Ki-Hwan Kim, Sean Stanek, Bancha Doungratanaex-Chai, Kungen Lin, Wallapak Tavanapong, Johnny Wong, Jung-Hwan Oh, Piet C. de Groen

a Department of Computer Science, Iowa State University, Ames, IA 50011, USA
b Department of Computer Science & Engineering, University of North Texas, Denton, TX 76203, USA
c Division of Gastroenterology and Hepatology, Mayo Clinic, Rochester, MN 55905, USA

1. Introduction

Colonoscopy is currently the preferred screening modality for prevention of colorectal cancer—the second highest cause of cancer-related deaths behind lung cancer in the US [1]. During a colonoscopic procedure, a flexible endoscope (with a tiny video camera at the tip) is gradually advanced under direct vision via the anus and rectum into the most proximal part of the colon (signified by the appearance of the appendiceal orifice or the terminal ileum). Next, the endoscope is gradually withdrawn. The video camera generates a sequence of images of the internal mucosa of the colon. These images are displayed on a monitor for real-time manual analysis by the endoscopist. Diagnostic and therapeutic operations such as polyp removal can be performed during the procedure. In current practice, videos of the entire colonoscopic procedure are not routinely captured for post-procedure review or analysis. A few snapshot images may be taken to document the procedure or unusual findings.

Over 14 millions colonoscopic procedures are performed annually [2]. Some evidence of variations in quality of colonoscopic procedures among different physicians was recently...
reported [3,4]. To measure in some detail what exactly is achieved during colonoscopy, ESGE proposed taking snapshot images at recommended positions during colonoscopy [5]. A number of indirect markers of quality have also been proposed [6]. These include duration of the withdrawal phase and the percentage of patients with polyps detected during screening colonoscopy. Despite intensive research in medical imaging in recent years, research on automatic content analysis of colonoscopy videos has been minimal. Computer-based methods were proposed to identify polyps in endoscopic images using texture features, which include Color Wavelet Covariance analysis [7,8], and Texture Spectrum classification using Neural Network Classifier [9,10]. We found no other existing algorithms and tools proposed to automatically document a colonoscopic procedure and derive objective measurements of quality of the procedure. Motivated by this fact, we have been developing Endoscopic Multimedia Information System (EMIS) that has the abilities to transparently capture the entire colonoscopic procedure into a colonoscopy video file, analyze the captured video for important contents, provide efficient access to these contents, and derive quality measurements (e.g., withdrawal time, visualization of the appendiceal orifice) of the procedure [11].

Arthemis, a software application, is the only component in EMIS that directly interacts with end users to provide easy access to important contents of the captured video and allow annotation on these contents. Endoscopists and medical students can use Arthemis to gain knowledge from studying colonoscopic procedures operated by experienced endoscopists or to evaluate the quality of the procedures. Hence, graphical user interface and functional components are important design factors. Guided by comments and suggestions from the endoscopist, Arthemis provides the following key features: (1) play, pause, or jump to a specific frame in a colonoscopy video to facilitate selection of images for annotation; (2) preview a video at a fast rate up to 8 times the normal playback speed; this feature reduces the time taken to review and annotate an entire procedure; (3) play important segments (e.g., segment showing polypectomy) determined by our analysis software [12]; (4) import endoscopic images for annotation; (5) annotate images using ellipse and free-hand-draw tools and associate the European Gastrointestinal Society for Endoscopy (ESGE) Minimal Standard Terminology (MST) terms with the annotated ellipse figure; (6) zoom in to get a closer look at details in the captured image or zoom out to view a larger portion of the image; (7) record annotated figures in an APRO (Arthemis Project) format based on Extensible Markup Language (XML) for later retrieval; (8) create video clips and corresponding APRO files with relevant annotation from the original colonoscopy video; and (9) authenticate user licenses with encrypted messages. These features post several challenges during the design and implementation process.

Arthemis is implemented mostly in Java with some features implemented in C using Microsoft DirectShow and a third-party software library for MPEG decoding and encoding [13]. Arthemis provides two unique functionalities: (1) annotation ability using ESGE MST and (2) ability to view automatically detected segments of colonoscopy videos. MST terminology is an internationally accepted standard for digestive endoscopy, which was proposed to enable electronic data exchange of the results of endoscopy examinations. To the best of our knowledge, no other software tools provide the two aforementioned abilities.

The rest of this paper is organized as follows. Section 2 introduces some background on EMIS and Arthemis as a user-interface component of EMIS. In Section 3, we briefly discuss enabling technologies used in the development of Arthemis. Section 4 discusses the design and implementation of Arthemis. In Section 5, we conclude the paper with the description of the current status of the software and experience gained during the software development process.

2. Background

This section provides a brief discussion on existing annotation software and overview of EMIS.

2.1. Existing browsing and annotation software

The ability to annotate with MST terms is highly desirable for annotation software for colonoscopy, but none of the existing annotation software provides this ability. MST terminology was proposed by ESGE in collaboration with American Society for Gastrointestinal Endoscopy (ASGE) and the World Organization of Digestive Endoscopy (OMED) to address the difficulty in sharing electronic endoscopic findings across institutions. MST includes terms that have wide acceptability and provide a means for recording the findings in the majority of examinations performed. No excessive detail was included and rare findings were recorded using 'free text' fields. Normed Verlag International Medical Publishers [14] publishes definitions of MST and offers Normedia software for browsing example images related to these terms as well as example endoscopy videos. However, Normedia does not allow annotation. On the contrary, Provision 1.5 Video Image Management System for Video Endoscopes from GTech Information Systems [15] provides annotation ability for endoscopy videos, but annotation is not according to MST. Given Imaging Ltd. [16] provides software to view images taken from capsule endoscopy, but does not provide annotation with MST.

Other video annotation tools are not for endoscopy videos and they do not support annotation according to MST terms. Some examples are listed as follows. Video Annotation and Reference System (VARS) [17] by Monterey Bay Aquarium Research Institute was developed for annotation of videos from ROV (Remotely Operated Vehicle) cameras. IBM’s Efficient Video Annotation (EVA) System [18] has a Web-based user interface to accept from users positive and negative examples of semantic concepts inherent in images or videos. IBM MPEG7 [19] Annotation Tool enables annotation of video sequences with MPEG7 metadata. Besides these software tools, there are

1 Arthemis – or more commonly spelled Artemis – is the Greek goddess associated with hunt and moon. “Artemis” is the name of a DNA sequence browser and annotation tool developed by the Sanger Institute.
several research prototypes for video annotation as well as commercial video editing programs including those that come with video capture hardware. These video editing programs allow playback of videos at various speeds and extraction and composition of video segments of an input video file to produce video clips in various formats. For image only annotation, several tools are available, but without the ability to view videos or to annotate according to MST terms. Examples are an image annotation tool for Diabetic Retinopathy by Barbara Davis Center for Childhood Diabetes [20], and Inote [21] by Institute for Advanced Technologies in the Humanities.

Medical images such as CT scans, MRIs, and ultrasound are typically stored in Digital Imaging and Communications in Medicine (DICOM) images [22]. Several programs are available for viewing these images [23]. The DICOM standard was developed for the distribution and viewing of medical images. Recently, the standard has been expanded to support videos. A single DICOM file contains not only the image or video data, but a header that has information about the patient’s name, the type of scan, etc. A few toolkits exist for creating browsing and annotation software for DICOM videos. Examples are Snowbound RasterMaster [24] and Lead Technologies Medical Imaging Suite [25]. None of these tools provide the two unique features of Arthemis.

2.2. Overview of EMIS

In this section, we present the overview of EMIS. This system consists of multiple components. The functionalities of each component are briefly described as follows.

2.2.1. EndoPACS Capture (EndoCapture)

EndoCapture is a software/hardware component for collecting and archiving endoscopy videos. This component has the ability to capture analog videos along with audio dictation by the endoscopist through a wireless Bluetooth microphone and headset. All videos are encoded and stored in MPEG2 format. This format supports high video quality, allows fine control over the tradeoff between quality and the amount of required storage, and is supported by DICOM Visible Light Object standard. EndoCapture workstations are installed in endoscopy rooms where endoscopic procedures are performed. As the EMIS system is still part of ongoing research, no patient identifiable information is captured to comply with the privacy rule of the Health Insurance Portability and Accountability Act (HIPAA) [26].

2.2.2. EndoPACS Collector (EndoCollector)

EndoCollector is client-server software that automatically uploads videos captured by EndoCapture to the storage server for further analysis. The video files captured in the same day by the same EndoCapture workstation are uploaded into the same directory on the storage server. EndoCollector uses Secure Shell (SSH) encryption as the secure method of communication between the capturing workstation and the analysis server. The prototypes of EndoCapture and EndoCollector have been used in a trial in two endoscopy rooms at Mayo Clinic Rochester since February of 2006. EndoCapture and EndoCollector enable secure, transparent, and robust automatic collection of endoscopy videos.

2.2.3. EndoStitch

EndoCapture and EndoCollector create a continuous stream of 2GB files. Because of automatic and transparent capturing, each file may contain video of partial or complete procedures, or non-endoscopy frames (images not showing the interior of the human colon). Non-endoscopy frames are generated during a short break between procedures of different patients. After the captured videos are uploaded to the analysis server, EndoStitch removes intervening, non-endoscopy frames, and/or stitches video segments of the same procedure that is separated into different files, and outputs a new MPEG video file. The output video file has endoscopy frames belonging to the same colonoscopic procedure. The details of design and implementation of EndoStitch are beyond the scope of this paper.

2.2.4. Video segmentation

Our new image/video analysis techniques for colonoscopy videos are used to extract important semantic units [12, 27–29]. Currently, three types of semantic units are provided: speech segments, scene segments, and operation shot segments. A speech segment contains the endoscopist’s dictation during the colonoscopic procedure. For example, the endoscopist speaks specific reserved terms when the tip of the endoscope is moving from one colonic segment into the next in real-time. Examples of these terms are “Entering rectum”, “Leaving rectum, entering sigmoid”, “Leaving sigmoid, entering descending colon”, “Leaving descending colon, entering transverse colon”, “Leaving transverse colon, entering ascending colon”, “Cecum”, and “Entering terminal ileum”, to indicate the current position of the video camera. A scene segment is referred to as a segment of visual and audio data that corresponds to an important part of the colon such as rectum, sigmoid, etc. An operation shot segment is a segment of visual and audio data that corresponds to a therapeutic or diagnostic operation in a colonoscopy video.

To parse the video for speech and scene segments, analysis of audio features is used to separate the speech segments from silence or noise. Finite state automata and speech recognition are then employed to recognize the dictated reserved terms and identify the candidate scenes. Next, our new image analysis technique is used to produce the final scenes. Interested readers are referred to our papers [27,29] for more details. To determine operation shot segments, we proposed a new image analysis technique that (1) detects cables of instruments (e.g., biopsy forceps, snares) used in diagnostic and therapeutic operations and (2) groups a sequence of images detected as having the cable to identify operation shot segments. More details can be found in our paper [28].

2.2.5. Arthemis

End users interact with Arthemis, our annotation software, to ease the process of manual post-procedure review and analysis of the quality of colonoscopic procedures. To the best of our knowledge, no other software allows (1) annotation according to MST terms that are internationally recognized.
and (2) browsing of important segments of colonoscopy videos. The design considerations for Arthemis are listed below.

(1) Ease of use by physicians: This is a challenge as a large amount of information needs to be presented to the physician to support the above functionalities. For instance, we need to accommodate over 100 MST terms from which the physician can select and assign terms to create an annotated image extracted from a colonoscopy video. We had several rounds of user feedback and fast prototyping of key user-interface components with an endoscopist in order to satisfy this requirement. Arthemis eventually has an eye-pleasing, intuitive user interface.

(2) Fast playback, efficient image extraction, and creation of video clips from MPEG2 videos: Although Java Media Framework (JMF) provides APIs to play MPEG2, its ability to manipulate MPEG2 videos is limited. JMF does not support fast playback rates required by endoscopists nor does JMF support creation of an MPEG video clip from the original video clip. Smooth, fast playback is important to reduce the time needed to review a colonoscopic procedure. This is because a colonoscopy video typically contains a significant number of blurry frames (the average is about 37% in our data set, but it can be over 60% depending on the endoscopist and the patient) due to three reasons. The first reason is the frequent shifts of the camera while moving along the colon (motion blur). The second reason is that current endoscopes are equipped with a single, wide-angle lens that cannot be focused (out-of-focus blur). The third reason is material (e.g., diluted stool, water) partially blocking light from reaching the lens (interference blur). These blurry frames do not contain useful information. Video clip generation is another important feature that allows creation of small examples to be used for research and education purposes.

Besides the requirements for fast playback and video clip generation, a low response time is required to reduce the time needed to review a colonoscopic procedure. This is because a colonoscopy video typically contains a significant number of blurry frames (the average is about 37% in our data set, but it can be over 60% depending on the endoscopist and the patient) due to three reasons. The first reason is the frequent shifts of the camera while moving along the colon (motion blur). The second reason is that current endoscopes are equipped with a single, wide-angle lens that cannot be focused (out-of-focus blur). The third reason is material (e.g., diluted stool, water) partially blocking light from reaching the lens (interference blur). These blurry frames do not contain useful information. Video clip generation is another important feature that allows creation of small examples to be used for research and education purposes.

3. Enabling technologies

Here we discuss key technologies used in the development of Arthemis. Java was chosen to implement Arthemis due to the language familiarity of the development team. Standard Java provides two major GUI toolkits: AWT (Abstract Window Toolkit) and Swing. AWT includes a platform-independent Java API to wrap the native GUI widgets of the various operating systems. However, it only provides a very limited set of widgets for building graphical user interfaces (GUIs). At best, AWT is only good for building simple applets but is not suitable for building complex desktop GUI applications. As one of the most popular Java GUI frameworks, Swing has a complete set of GUI components that can be used to build complex desktop GUI applications. These components are drawn by using graphic primitives such as lines, rectangles, and text rather than relying on the native widgets of the operating system. In addition, Java provides other features and functions that are important for Rapid Application Development (RAD) such as Java 2D, Java Media Framework API (JMF) and Java APIs for XML Parsing (JAXP). These features will be briefly described in the following sections.

3.1. Java 2D

As one of the core APIs for the Java 2 platform, Java 2D libraries were introduced as the complement to AWT. Seamlessly integrated with the Java platform’s GUI architecture, Java 2D provides mechanisms to implement sophisticated graphics. In particular, it includes the following functionalities: (1) control over rendering quality; (2) clipping, compositing, and transparency; (3) drawing and filling of simple and complex shapes; (4) image processing and transformation; and (5) advanced font handling and string formatting.

3.2. Java Media Framework (JMF)

The JMF API is one of several core multimedia APIs in Java. With the JMF API, programmers can create Java applications that play, edit and stream many popular media types. In addition to versatile media playback, provided in version 1.0 of the JMF API, the JMF 2.0 API features advanced Java platform-based media processing capabilities, such as audio and video capture, compression and transmission, and support for important media types and codecs, such as MP3, Flash and IBM’s HotMedia. The JMF 2.0 API has an open architecture to allow developers to access and manipulate various components of the media playback process, such as effects, tracks, and renders.

These new functionalities of the JMF 2.0 API enables Java developers to work with audio and video from the initial capture to final playback and create media-rich and portable applications. To boost software performance, Sun Microsystems provides several implementations of JMF (namely JMF performance packs) for different operating systems. Therefore, JMF can take advantage of the performance features
of the platform on which the JMF is running at the cost of reduced application portability. The previous versions of Arthemis (i.e. v1.0 and v2.0) were developed using JMF 2.0 with Windows Performance Pack—an optimized version of JMF 2.0 for Windows platforms. Besides the JMF core classes, the Windows JMF installation package contains several native libraries. For example, the JMF version written with pure Java does not include a YUV to RGB converter while the version of Windows performance pack does. The reason is that the YUV to RGB converter was written in native code to guarantee JMF performance on Windows platform.

Unfortunately, the JMF API is not sufficient to manipulate MPEG2 videos at the level required by the intended users of Arthemis although the imbedded media player can play MPEG2 videos on Windows machines using the native decoder installed on the system. To solve this problem, we have developed our own media player based on third-party API MPEG2 encoding and decoding and integrated it into Arthemis. The techniques used to implement this media player will be described in Section 4.3.

3.3. Xerces Java Parser

The Java API for XML Processing (JAXP) defines a certain implementation of the Simple API for XML (SAX) and Document Object Model (DOM) parsers. The SAX parser is good for event-driven operations while the DOM parser is good for tree-based operations. The Apache Java XML parser (commonly known as Xerces [30]) is an open-source parser based on IBM’s XML4J parser. Xerces provides full support for W3C DOM Level 1 and SAX 1.0 and 2.0. Xerces is one implementation of JAXP and is a de facto standard when it comes to Java-based XML development. Other implementations of JAXP, such as Crimson [31], the Oracle Parser [32] or the Sun Parser [33], also can be used for handling XML files. We have chosen Xerces because most of XML operations in Arthemis are tree-based operations and Xerces provides the best performance for DOM parsers. Some of the major functionalities of the Xerces Java Parser are: (1) complete adherence to XML Schema processor; (2) a complete implementation of DOM; (3) ability to parse documents written according to the XML 1.1 recommendation; and (4) support for OASIS XML Catalogs.

We used the Xerces Java Parser API to implement a component for reading, recording, and updating annotation information in an XML file. More details on this component are presented in Section 4.5.

3.4. Microsoft DirectShow

Introduced by Microsoft, DirectShow is a multimedia framework used by software developers to perform various operations with media files. As a component of DirectX, DirectShow provides a common interface for media across many of Microsoft’s Programming Languages and offers APIs for playing streamable media such as movies, music, etc. For example, by using DirectShow, it is possible to play movie files like MPEGs, AVIs, and WMVs and sounds like WAVs, MP3s, and WMA.s. DirectShow works by using a Component Object Model (COM) interface named a filter graph that can be thought of as a chart made up of various blocks all connected to one another by pins or wires.

We used DirectShow to implement a media player instead of JMF media player in an earlier version of Arthemis. However, we found that the player based on DirectShow cannot offer smooth, fast playback at variable rates as required by the intended users of Arthemis. Hence, we have implemented a new media player without DirectShow. The details for this new player are described in Section 4.3.

4. Design considerations and system description

The first prototype of Arthemis was completed in February 2005 to get an initial feedback from the endoscopist in terms of the user interface and functionalities. The current version was completed in November 2006. The current design is based on user feedback regarding the first prototype. The required features have already been discussed in Section 1. In this section, we only discuss important features of the current design.

4.1. A component-based framework

Fig. 1 shows three essential elements of Arthemis: ArthemisFrame (a subclass of JFrame), a set of “black components” and a set of “white components.” We categorize these black and white components following the concepts in software testing (black box testing and white box testing). ArthemisFrame is the main module that defines common data types and functional interfaces as well as establishes and manages communication and information passing among black and white components. Components cannot directly call each other without being relayed by ArthemisFrame.

Black components provide predefined APIs for other components to call, but they do not call any functions inside ArthemisFrame or other black components. The black components serve merely as isolated components and can only be invoked by ArthemisFrame. The advantage of black components is that different black components can be developed independently by several developers in parallel without interfering with each other. In addition, black components can be outsourced without risking leakage of intellectual property. The major disadvantage of black components is the limitation of code reusability because each black component has to implement all required functions rather than directly calling existing functions developed in other components. Besides, error checking needs to be done twice; once for the caller of the black component and once for the black component itself. Since different black components interact among each other only via ArthemisFrame, a slight increase in program execution time is expected.

We carefully reviewed the advantages and disadvantages of black versus white components. We have chosen to implement black components for functionality to be customized in the future because a slight increase in execution time is not of major concern. Black components in Arthemis are the “Dx Annotation” and “Procedures” panels (see Fig. 1) that deal with MST terms. Future versions of Arthemis will support other endoscopic procedures that have different standard
terminologies via additional black components. Other GUI black components are the “Messages” panel that conveys error messages to the user and the “Information” panel that shows related image/video information. Another important non-GUI black component handles manipulation of the XML based APRO files that contain important information about colonoscopy videos including the MST annotation of selected frames within the video.

White components have extensive connections with ArthemisFrame. In addition to preparing a functional interface for calls from other components, white components directly invoke functions defined in ArthemisFrame. Thus code reusability is optimized and each white component does not need to reinvent the wheel if a function has been implemented in other parts. The disadvantage of this type of component is that the developer needs complete knowledge about coding and logic of the main program. This implies a high learning curve. The advantage of the white components is that they enable complicated functionality and powerful features to be implemented efficiently. For Arthemis, the white components are the video player, image extraction, and video clip creation modules.

Overall the expected increase in execution time due to the use of black components was not noticeable by the users of Arthemis. This is due to the selection process by which we define which functionality will be implemented as black or white components. Testing of all black components was done before these were integrated with ArthemisFrame, making it easy to isolate and fix bugs.

### 4.2. Graphical user interface of Arthemis

We have spent considerable time designing an intuitive graphical user interface (GUI) that presents the video files, still images and ESGE MST annotation in an immediately useful format. Feedback from the endoscopist regarding the first version of Arthemis prompted us to redo the design of the GUI of the current version detailing all panels and the content of each panel. The design was then implemented into a digital format. We then asked the endoscopist to provide feedback regarding the new design and revised it as suggested. During the development, we incrementally added more GUI features and continuously asked the endoscopist for feedback. This process has proven to be very useful since changes needed by the endoscopist were accommodated early on and the endoscopist was satisfied with the final GUI.

The most complicated part of the design was to fit over 100 MST terms for colonoscopy in to the GUI. For instance, for terms related to diagnostics for colonoscopy, there are a total of seven “Headings.” These terms are “Normal”, “Lumen”, “Contents”, “Mucosa”, “Flat Lesions”, “Protruding Lesions” and “Excavated Lesions”. For “Lumen”, there are three terms: “Dilated”, “Stenosis”, and “Evidence of previous surgery”. For “Stenosis”, there are three “Attributes”: “Appearance”, “Length”, and “Traversed”. For each attribute, a value is assigned. For instance, either “Extrinsic”, “Benign intrinsic”, or “Malignant intrinsic”, can be assigned to “Appearance”. “Sites” is to indicate the location of interest associated with the selected “Headings”. There are 12 sites to choose from.
One possible option for the design of the GUI selection panel for MST terms was to create a separate window with all MST terms that looks like a long spreadsheet. This option was not chosen because the window would block the image area and would be very long due to the large number of MST terms. Such a solution would require that users need to scroll down to find an appropriate term to assign to an image. Our goal was a design that minimizes the time taken to annotate one video.

The final GUI design for MST terms includes two panels: one panel for annotation related to diagnostics (“Dx Annotation” panel) and another panel for annotation related to procedures performed during colonoscopy (“Procedure” panel). We implemented these two panels as black components to accommodate different terminologies for other endoscopic procedures in the future. Each of these panels has sub-panels. Each sub-panel corresponds to different MST “Headings”. Inside each sub-panel for one of the MST “Headings”, checkboxes, drop-down boxes, or integer boxes let users select the desired “Terms”, “Attributes”, “Attribute Values”, or “Site(s)” under that heading. We color coded input options for MST terms in a spreadsheet first and used the spreadsheet to communicate with the user and guide the development of the user interface. The final interface is shown in Fig. 2. Fig. 3 illustrates an annotation procedure in which an ellipse is used to identify the tip of the biopsy forceps and polyp and the associated annotation information is selected on the bottom “Procedure” panel. Fig. 4 shows images that are extracted from the current video. Each image is associated with its captured time and annotation information. The ellipse and free-hand
4.3. Video player and image extraction

As mentioned in Section 3.2, JMF provides insufficient functionalities to manipulate MPEG2. For instance, it does not support fast-forward playback, extracting image frames from a video, or extracting small MPEG clips from the original video. To address the first two limitations of JMF, we developed our own video player in C. In the early implementation of the video player, we used the Microsoft DirectShow Software Development Kit (SDK) for multimedia manipulation on the Windows platform. When playing a video file, the main Java program starts by calling this video player through JNI and transfers the filename to the player as an input parameter. After receiving the input filename from the main Java program, the video player sets up a DirectShow filter graph (see Fig. 5) that is a set of connected filters. In DirectShow, a program accomplishes a task by connecting chains of filters so that the output from one filter turns into the input of another. To play, pause, or stop a video playback, the DirectShow API is used to control and set the internal state of the filter graph, which in turn, causes the filters in the graph to play, pause, or stop playing the data accordingly. The File Source filter (Fig. 5) reads the video file from the hard disk. The MPEG Splitter filter parses the file into two streams, a compressed video stream and an audio stream. The MPEG Decompressor filter decodes the video frames into raw RGB images. This raw RGB data is then passed on to a Sample Grabber filter, which the player uses to read the data as it passes through the filter graph. We use the Sample Grabber filter to get an image from the running video. Whenever a snapshot button is pushed to take a snapshot image, the data buffered in the Sample Grabber filter is converted to an image and transferred to both consultation panel and snapshot panel for display and annotation through JNI. Next, the Video Renderer filter draws the frames produced by the Sample Grabber filter on the screen. At the same time, the audio data created by the MPEG Splitter filter is decoded by the MPEG Audio Decoder filter and passed on to the Stereo Output filter. This filter renders the audio through the sound device via DirectSound.

After setting up the DirectShow filter graph, the output window of the DirectShow’s Video Renderer filter is set to be owned by Arthemis’ main Java window through the DirectShow API. This is to ensure that if users move the Arthemis main window, or cover up some part of the main window, the video window is appropriately moved or refreshed by the Windows operating system.

By modifying both how the filter graph is played back and the status of the filter graph, we can make operations such as pause, seek and fast forward possible. Note that the DirectShow SDK provides a function to set the playback speed. However, it does not guarantee that all the filters in the filter graph can support the desired speed. On most Windows XP systems we tested, 2X speed worked well, but 4X and 8X were not supported at all. On some systems, even 2X playback speed did not work. Thus, we implemented our own fast-forward playback function on top of DirectShow by repeatedly calling standard DirectShow seek functions. However, this approach requires hundreds of internal state changes of DirectShow filters. As a result, it cannot give the smooth playback at 2X, 4X, and 8X normal playback speed desired by the endoscopist.

As the player using DirectShow is not able to meet the user requirements, we started over by implementing the player using third-party MPEG audio and video decoders [13] with-
out DirectShow in our final solution. These decoders provide the elementary operations to open, close, seek, and read any number of video frames or audio samples from an MPEG file. In addition, the decoders provide elementary operations to obtain normal playback rates for audio samples (usually either 44,100 Hz or 48,000 Hz) and video frames (usually about 30 frames per second) from the MPEG file. Besides using the structure of the early version of the video player and reusing some original code (including the manual fast forward timer code), we have written new code to replace all the original DirectShow functionalities. This includes code to use the new standalone MPEG audio/video API functions to read in audio and video, to create our own audio and video output renderers, and to implement controls for all new functionalities.

Upon receiving the filename of a video, the information is validated and the MPEG file is loaded. An appropriately sized window is created for the video in similar fashion as to how DirectShow creates a window. If audio is present in the MPEG file, we open an audio device through the Windows wave output API. To determine the video position, the video player relies on our manual timer. Next, the video player finds the desired position and reads in a single video frame that is to be displayed on the video window along with corresponding audio samples (computed based on the timer and the audio playback rate). Also, the video player performs a bilinear image interpolation if the dimension of the obtained video frame does not fit that of the video window. For fast audio playback, we perform audio linear interpolation. That is, we read in a block of audio data with a length that scales linearly with the speed of the playback. For example, under 4X playback, if we need 10 ms length of audio data, we read in 40 ms at the appropriate location (determined by the normal playback rate) and then linearly interpolate it into a 10 ms-sized buffer (see Fig. 6), which is then output to the sound device. The linear interpolation for a specific output audio sample is just the average of consecutive audio samples that match up in a time scaling of the original audio data. For instance, under 4X playback, an output audio sample at time unit t is the average of consecutive audio samples at time units 4(t − 1) to 4t − 1 of the original audio data.

This approach works extremely well and provides performance acceptable to the endoscopist. Since we also have the audio and video data in our own data buffers, it is convenient to implement much of the other functionalities we want. In contrast, if these functionalities were implemented using the DirectShow API, DirectShow would abstract so much of the data that it becomes slow. Also by using this approach, it is easy to superimpose other useful information in the video window, instead of using valuable GUI space in the main Java program itself. For instance, the fast forward playback speed can be superimposed on the top-left corner of the video window without changing the original GUI layout of the main Java program.

4.4. Creation of video clips from the original colonoscopy video

To create a video clip from the original video file, the user chooses the starting frame and ending frame from the “Snapshot” panel. This information is shown in the “Clips” panel. The clip panel component interacts with the user to create small video segments from the original colonoscopy video along with an APRO file that has annotation related to images in these segments. The clip panel component shows the beginning and ending frames of a video segment the user wishes to extract from the original video. The output video file is either in MPEG1 or MPEG2 format as chosen by the user. Once the user clicks the “Create Clips” button, the clip panel component performs the following actions:

(a) Create a temporary clip-information file that includes the desired MPEG format, and information about each video segment such as the video segment id, the frame numbers of the beginning and ending frames of the segment.

(b) Create one unique project directory per video segment and create an APRO file inside the project directory. This file contains annotation information related to the frames in the video segment.

(c) Call an API of a C library named “EndoPaste” through JNI. This library provides a set of APIs: one of which is to read the clip-information file created by the clip panel component and produces the corresponding video file. EndoPaste also provides another API that retrieves the percentage of video clip creation completion. EndoPaste is implemented in C using Microsoft DirectShow and third-party MPEG encoder and decoder filters to generate the output video.

(d) Call an API of the EndoPaste library periodically to get the percentage of clip creation completion and show the
progress through a progress bar. The user is not allowed to do any annotation until the clip creation has been completed.

It first seems simpler to implement EndoPaste as an independent program which the main Java program can call using the runtime class and the process class. This approach is not efficient in this case because the main Java program needs to show the progress of the clip creation periodically. Hence, we integrated the Java main program and the DirectShow C code through JNI.

4.5. Storing annotated figures in APRO format

We developed a black component called KXML for reading, recording, and updating annotated figures in APRO format. APRO is our proprietary format based on XML. The KXML component is called by the main program whenever there is a need to store or update the annotations in the APRO file.
is a need to create, update, delete annotation, and save/load annotation to/from an APRO file. The APRO format consists of a number of XML tags containing information about (1) the annotator, (2) the filename of the annotated video, (3) the last time the APRO file was updated, (4) starting frames and ending frames of video segments for each of endoscopic scenes, operation shots, and speech segments, (5) extracted images and annotated figures along with associated MST terms, and (6) information about video segments that users wish to create from the original video file. An example of the APRO file is shown in Fig. 7.

We used the Document Object Model (DOM) parser implementation provided in the Java Xerces package to parse XML tags in an APRO file. Given an APRO filename, the DOM parser generates a tree of XML tags in memory. Each node in the tree corresponds to one XML tag in the input file. To hide the tree structure from other black components or white components the KXML package wraps some nodes in the tree with classes, for instance, classes for MST related tags. Methods of the classes are used by the main program to manipulate the data in the tree. See Figs. 8 and 9 for example mapping of XML tags and corresponding classes. Each of the classes provides methods for insertion, deletion, update, and retrieval of information about its corresponding XML tag. This design, that wraps classes around important XML tags, makes it easy for KXML to be called by other white components. In addition, maintenance of this program is easy. Modifying the KXML component to use a different XML parser that keeps the tree structure in memory can be done without affecting the code in other packages. Access to information about an XML tag of a given MST term is quick because the object instantiated from the wrapper class of the tag knows the exact location in the XML tree to update without having to traverse the tree from the root. In addition, the KXMLDocument class (Fig. 8) provides two additional methods for saving the tree of annotation in the APRO file and for retrieving the annotation from the APRO file into the annotation tree.

5. Discussion and conclusion

Arthemis has gone through several revisions in order to satisfy the user’s requirements in terms of functionality, user-interface, and performance of fast playback. As a result, Arthemis is starting to become a useful tool for endoscopic research and education. Training in recognition of endoscopic findings at present is limited to verbal descriptions combined with one or a few pictures of a representative lesion. Rarely if ever is a single video shown when discussing protruding lesions within the colon. With Arthemis and a database of annotated colonoscopy videos, medical students, residents, and fellows will be able to learn to recognize the common endoscopic abnormalities, and then can see what types of diagnostic and therapeutic modalities were used by experienced endoscopists. Furthermore, staff endoscopists who want or need to brush up their knowledge can use the system as proposed to review lesions or procedures with which they have experienced difficulties. Thus Arthemis will support both initial as well as continuing education from student to experienced endoscopist.

Arthemis is currently being used to annotate colonoscopy videos to indicate various types of abnormality and anatomical landmarks of the colon such as the appendiceal orifice. The quality of the fast-forward playback is now acceptable to our endoscopist. The abilities to support fast-forward playback as well as a quick access to important video segments detected by our video segmentation software are shown to save time for manual review and annotation. The user-interface of Arthemis is intuitive and easy to use as a result of fast-prototyping of key components in the user interface as well as iterative feedback from our endoscopist. For developers, key lessons learned from the development of Arthemis are as follows:

- Fast prototyping has proven extremely useful for development of the user interface and functionalities of Arthemis.
- The component-based method made it easy for us to assign a certain black component of Arthemis to a new developer without worrying about losing the control of the code or a high learning curve. The new developer was able to develop the required component that can be integrated with the main Arthemis software within a 3-week time frame.
- We found Microsoft DirectShow APIs and Java Media Framework to be insufficient for supporting fast-forward playback.
at the performance (4X–8X speed) required by the endoscopist. We addressed this problem by implementing this functionality on our own using third-party native API and library for decoding MPEG2 video without using DirectShow. Note that there are several third-party codecs to choose from. We selected a particular codec taking into account decoding/encoding speed as well as the quality of the video clips produced.

- The division of functionality between Java and C code must be carefully designed to minimize the interaction between Java and C through JNI. Otherwise, the resulting software will be slow as JNI takes non-negligible time to call the C library.

Additional features for a future version of Arthemis include the abilities to (1) view other important video segments (e.g., segments showing appendiceal orifice, terminal ileum, and polyps) automatically detected by our video segmentation software; (2) view objective quality measurements of the procedure generated by our quality measurement software [11]; (3) convert images and videos to/from DICOM format and retrieve/store the results in the DICOM storage to be compatible with most Picture Archiving and Communication Systems (PACS); and (4) search for video files within a database of colonoscopy videos given some MST terms or an example image. Incorporation of these features will further enhance Arthemis as a useful tool for manual post-procedure review to assess the quality of colonoscopy as well as the skill set of the endoscopist.

REFERENCES