Journal of Software Engineering and Applications, 2014, 7, 730-743
Published Online July 2014 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2014.78068

Open-Access Framework for Efficient Object-Oriented Development of Video Analysis Software

Dimitris K. Iakovidis, Dimitris Diamantis

Department of Computer Engineering, Technological Educational Institute of Central Greece, Lamia, Greece
Email: dimitris.iakovidis@ieee.org

Received 19 May 2014; revised 15 June 2014; accepted 10 July 2014

Abstract

The increasing use of digital video everyday in a multitude of electronic devices, including mobile phones, tablets and laptops, poses the need for quick development of cross-platform video software. However current approaches to this direction usually require a long learning curve, and their development lacks standardization. This results in software components that are difficult to reuse, and hard to maintain or extend. In order to overcome such issues, we propose a novel object-oriented framework for efficient development of software systems for video analysis. It consists of a set of four abstract components, suitable for the implementation of independent plug-in modules for video acquisition, preprocessing, analysis and output handling. The extensibility of each module can be facilitated by sub-modules specifying additional functionalities. This architecture enables quick responses to changes and re-configurability; thus conforming to the requirements of agile software development practices. Considering the need for platform independency, the proposed Java Video Analysis (JVA) framework is implemented in Java. It is publicly available through the web as open-access software, supported by a growing collection of implemented modules. Its efficiency is empirically validated for the development of a representative video analysis system.

Keywords

Object-Oriented Framework, Efficient Software Development, Video Analysis, Java

1. Introduction

Video analysis aims to the extraction of useful information from video streams for the purpose of a variety of applications such as scene, object or event recognition [1], and video summarization [2]. Considering that digital video is becoming more and more a part of our everyday life through portable devices such as mobile phones, tablets and laptops, the interest in video analysis systems is growing both in the market and in the research landscape. Indicatively **Figure 1** illustrates the (almost linear) increase of the number of scientific publications on video analysis in the last decade.

Developing video analysis systems is usually a demanding task not only in terms of computational complexity but also in terms of developer's specialization, requiring both knowledge of dedicated application programming interfaces (APIs) and elements of theoretical concepts mainly from the sphere of digital signal processing and data mining. Most of the current approaches to the development of video analysis software focus on specific application domains, and result in system components that are hardly generalizable, maintainable and extensible. Considering also the diversity of the video-enabled electronic devices the need for a platform-independent software development framework for video analysis is posed.

A variety of frameworks have been proposed to describe in a general way video analysis methodologies implemented in software. These include a modular video analysis framework for feature extraction and video segmentation [4], a framework for object retrieval and mining [5], and several domain-specific frameworks, many of which address biomedical [6] and security applications [7]-[9]. However, the focus of these frameworks is on methodological issues of video analysis, rather than on software construction issues.

Graphical frameworks have been proposed for the quick development of image analysis (or image mining) systems. The IMage MIning (IMMI) plugin of the Rapidminer software [10] implements such a framework enabling the assembly of pipelines for image analysis using icons that are dragged and dropped into a graphical environment. This high-level programming approach enables quick implementation of applications; however, its capabilities are constrained by the provided templates and specific libraries. A similar framework, called B-Image, also based on Rapidminer, is available as a commercial product [11].

Several software libraries for computer vision, image/video processing and analysis are also available and can be used for the development of video analysis software. These include OpenCV [12], JavaCV [13], and BoofCV [14], and FFMpeg [15] with its Java wrapper Xuggler [16]. These programming tools provide rich APIs enabling management acquisition, processing, and delivery of media data; however, they cannot be considered as software development frameworks, since they address mainly the low-level implementation aspects of these processes rather than structural and integration aspects of a whole video analysis system.

A software development framework can be considered as a set of prefabricated software building blocks that programmers can use, extend, or customize for specific computing solutions [17]. An object-oriented approach to framework design provides modularity, reusability, extensibility and inversion of control, *i.e.*, it enables canonical application processing steps to be customized by event handling objects that are invoked via the framework's reactive dispatching mechanism. Such a design typically includes a collection of classes providing an abstraction that can be used to describe a whole family of problems.

A framework for development of multimedia software systems is the Java Media Framework (JMF) [18]. JMF was released in the nineties, and for a long time it was considered as a standard for the development of

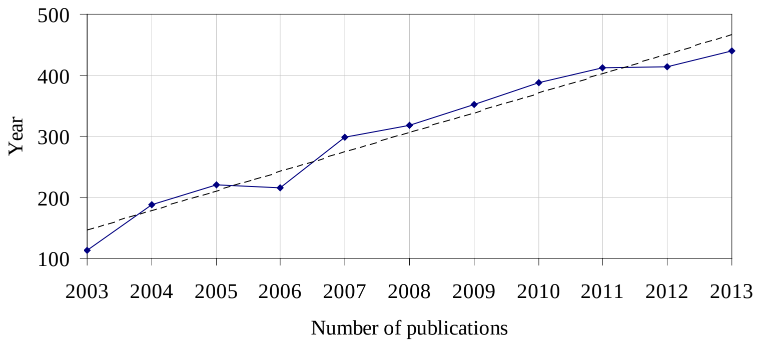

Figure 1. Number of scientific publications on "video analysis" per year in the last decade, based on the results retrieved with Scopus literature search engine on April 2014 [3]. The dashed line represents the respective linear trend.

multimedia systems in Java. It includes a rich library and protocols for acquisition, processing, and delivery of time-based media data, complex class hierarchies and patterns, and it is adaptable by inheritance and method overriding. However, it is only partially applicable for the implementation of a typical video analysis workflow, since it does not include any building blocks for information extraction, and due to its complexity, it requires a long learning curve by average developers. In the same spirit, a more recent, open source alternative of JMF for Java is Freedom for Media in Java (FMJ) [19]. Both JMF and FMJ have been replaced by a JavaFX, which is simpler and a more general framework, in the sense that it includes an API for graphics, rendering, animation, etc. [20]. However, JavaFX still does not support the information extraction approaches required for video analysis. A representative framework, similar to JMF, for C++ is MET++ [21]. MET++ is based on the object-oriented application framework ET++ [22], which integrates user interface building blocks, basic data structures, and support for object input/output with high level system components.

Pygame [23], is another multimedia software development framework with focus on game programming. It provides graphical, audio and input functions that enable quick development of video games based on the Simple Direct media Layer (SDL) [24], which is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware. Pygame adopts a data-driven approach which achieves adaptation by object creation and setting of object properties, and it includes delegation mechanisms, such as object chaining. It is easier to learn than JMF; however, it is also insufficient for the implementation of a complete video analysis workflow.

Other object-oriented approaches to multimedia software development include Object-oriented Modeling of Multimedia Applications (OMMMA) [25], which mainly aims to the modeling of the structure and the dynamic behavior of multimedia systems, and the Multimedia Modeling Language (MML) [26], which is based on UML 2.0, and it enables the application of the paradigm of Model-Driven Development for multimedia applications. Software development frameworks have been proposed also in peripheral contexts, e.g., the C++ Library for Audio and Music (CLAM), which is an object-oriented framework for efficient and rapid development of cross-platform audio applications [27] [28], a framework for the development of mobile applications [29], a framework for the development of interactive digital television (IDTV) [30], and a framework for the development of multi-agent systems [31].

To the best of our knowledge none of the current frameworks explicitly addresses the development of video analysis software systems. In order to bridge this gap, we have designed and implemented the Java Video Analysis (JVA) framework, which is presented in this paper. JVA is an object-oriented, cross-platform framework, with an easy to use application programming interface (API) written entirely in Java. It enables the development of video analysis software systems in a standard, reusable, maintainable and extensible way. The JVA API provides the necessary abstraction for the implementation of various video acquisition, preprocessing, analysis and visualization methodologies, which can be based on external software libraries such as JavaCV, or other frameworks such as JMF, and be integrated under an easily tunable modular architecture.

The rest of this paper is organized in five sections. Section 2 describes the architecture of the proposed framework. Its utility for the development of a representative video analysis system is explained in Section 3. Details on the open-access availability of the framework and its modules are described in Section 4. Section 5 presents the results from its assessment in terms of software development efficiency. The last section summarizes the conclusions that can be derived from this study as well as future research directions.

2. Framework Architecture

The proposed framework consists of a set of four main abstract components suitable for the implementation of four respective independent modules, namely Image Data-Source (IDS), Image Processor (IP), Image Analyzer (IA) and Output Handler (OH). A high-level block diagram of the framework architecture is illustrated in **Figure 2**. Each module can be deployed as a standalone software using the core framework API. This modular plug-in architecture provides the systems being developed, with the necessary re-configurability for the implementation of a variety of video analysis tasks, without any modification of the framework's core, whereas each plug-in can be manipulated independently. The parameters of the plug-ins can be configured through any key-value pair source, such as a text file, a database or even a web service. This provides easy access to system parameters for tuning and optimization, or for communication with external applications, including wrapper graphical user interfaces (GUIs).

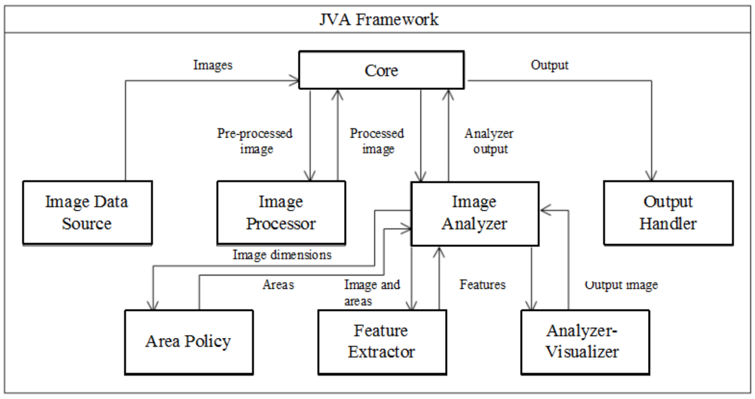

Figure 2. A high-level block diagram of the proposed JVA framework. Image data source, image processor, image analyzer and output handler are four independent modules that can be implemented as plug-ins, by utilizing its main abstract components. Area policy, feature extractor and analyzer-visualizer represent peripheral plug-ins of image analyzer.

The implementation of the plug-ins can be supported by a GUI-enabled debugging utility, called JVA Workbench, which provides output and error logging capabilities, direct access to the parameters and visualization of the intermediate test results obtained during development time. This GUI consists of three main panels. The left panel displays the output of the IDS module, the central panel displays the output of the IP module, and the right panel displays the output of the IA module, which is handled by OH. It includes controls for navigation within the video, frame grabbing, parameter configuration, and error logging options. A snapshot of the JVA Workbench is illustrated in **Figure 3**. The left panel displays a video frame acquired from a video clip of the Acropolis; the central panel displays that frame after the application of a set of preprocessing operations that include cropping, scaling and color transformation; and the right panel displays the result of a clustering analysis process which implements the segmentation of the preprocessed video frame into areas of different color (the specific implementations of these JVA modules used are described in Section 3).

In the following paragraphs we provide details on the framework core and on the associated modules.

2.1. Framework Core

The core of the proposed framework consists of four packages of Java interfaces, as illustrated in **Figure 4**. These packages are interdependent and inseparable, forming a single library. The organization of the interfaces into packages is conceptual with respect to their functionality, mainly for presentation purposes.

Package *com.snsp.infrastructure.core* contains four interfaces, *IConfiguration*, *IConfigurable*, *IFrameworkEventListener* and *IFrameworkContext*. The *IConfiguration* interface enables access to the parameters of an implemented video analysis methodology through a key-value pair source. An implementation of the *IConfigurable* interface supports the consumption of instances of *IConfiguration*; thus it can provide access to the parameters of the whole video analysis system developed with the JVA framework. All the modules of the framework are inherited from the *IConfigurable* interface in order to enable property dependency injection of the *IConfiguration* object by the core framework. *IFrameworkContext* is implemented within the framework, in order to provide an observable interface for all the events that occur during video analysis. In order to be able to receive information about events that signify the beginning or the end of a process implemented in a module of the framework, e.g. an image processing operation, the system should include classes that are aware of such events (event listeners). Event listeners have to implement the *IFrameworkEventListener* interface and subscribe themselves to the *IFrameworkContext* implementation. They can act as interceptors of each process, being informed about the events happening, in the order of their subscription to *IFrameworkContext*. On each event, these listeners receive a reference to the source object in order to be able to manipulate it or just trace it. For example, such an event listener could be a logging interceptor that measures the execution time of an image processing operation, or an image processor, such as a noise-reduction filter, which is triggered by an event when specific criteria are met. *IFrameworkEventListener* defines six methods that are called once a respective event occurs: 1)

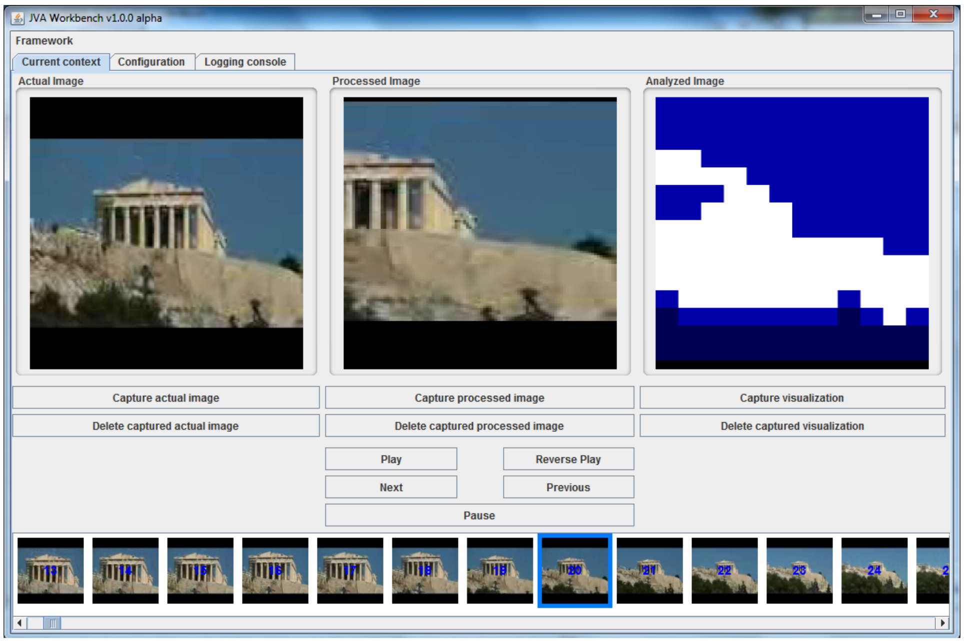

Figure 3. JVA workbench GUI.

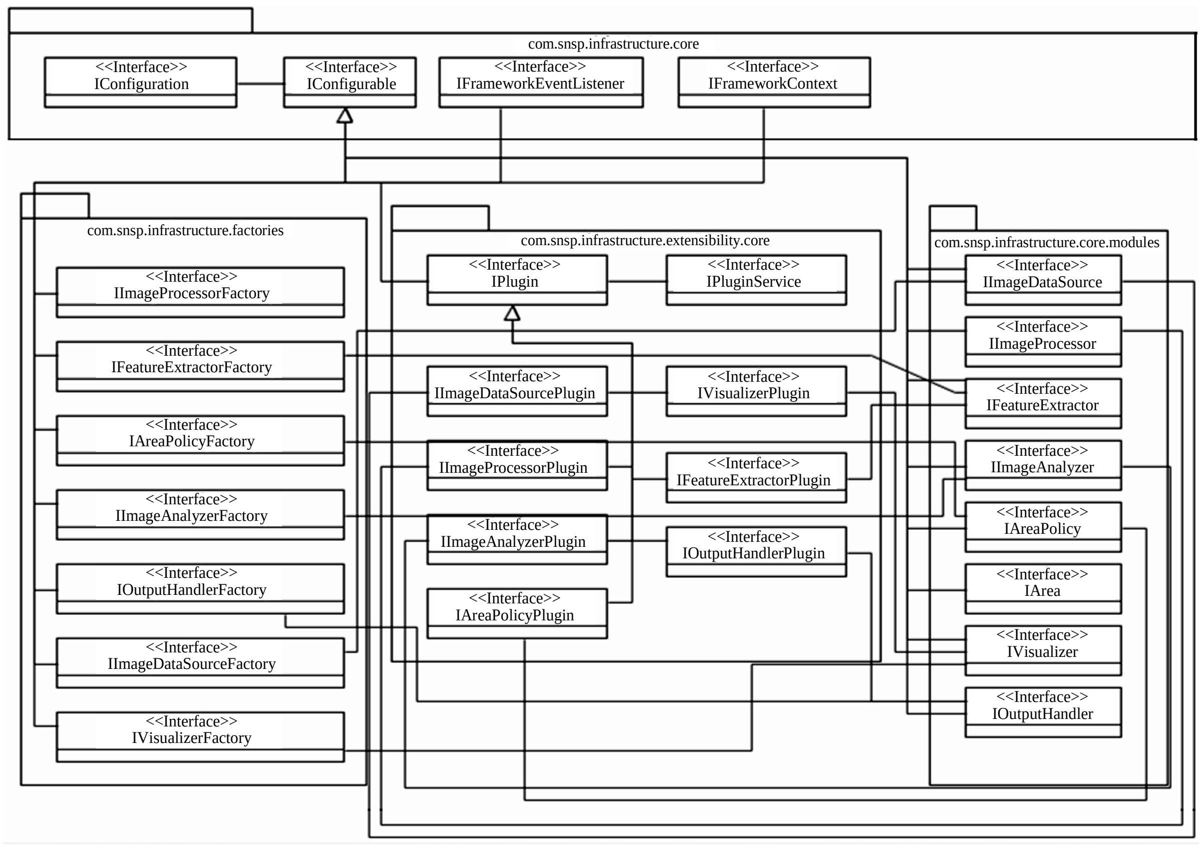
com.snsp.infrastructure.core
<<Interface>> IConfiguration
<<Interface>> IConfigurable
<<Interface>> IFrameworkEventListener
<<Interface>> IFrameworkContext
com.snsp.infrastructure.factories
<<Interface>> IImageProcessorFactory
<<Interface>> IFeatureExtractorFactory
<<Interface>> IAreaPolicyFactory
<<Interface>> IImageAnalyzerFactory
<<Interface>> IOutputHandlerFactory
<<Interface>> IImageDataSourceFactory
<<Interface>> IVisualizerFactory
com.snsp.infrastructure.extensibility.core
<<Interface>> IPlugin
<<Interface>> IPluginService
<<Interface>> IImageDataSourcePlugin
<<Interface>> IVisualizerPlugin
<<Interface>> IImageProcessorPlugin
<<Interface>> IFeatureExtractorPlugin
<<Interface>> IImageAnalyzerPlugin
<<Interface>> IOutputHandlerPlugin
<<Interface>> IAreaPolicyPlugin
com.snsp.infrastructure.core.modules
<<Interface>> IImageDataSource
<<Interface>> IImageProcessor
<<Interface>> IFeatureExtractor
<<Interface>> IImageAnalyzer
<<Interface>> IAreaPolicy
<<Interface>> IArea
<<Interface>> IVisualizer
<<Interface>> IOutputHandler

Figure 4. A Unified Modeling Language (UML) model of the JVA framework.

on Start: in the beginning of the whole video analysis process, 2) *onNewImage*: once a new image is provided by the IDS module, 3) *afterImageProcessing*: once the IP module finishes image processing, 4) *afterImageAna-*

lysis: once the IA module finishes image analysis, 5) *afterOuputHandled*: once OH finishes output handling, and 6) *onFinish*: once the whole video analysis process finishes.

Package *com.snsp.infrastructure.core.modules* contains all the interfaces of the independent modules of the framework. Package *com.snsp.infrastructure.extensibility.core*, contains a set of respective factories (interfaces named after the modules with the suffix "*Plugin*", as illustrated in **Figure 4**), whose role is to instantiate and configure these modules, *i.e.*, whenever an instance of a module is needed, the factory will undertake its construction. All factories inherit an initialization method from *IPlugin* interface that is called immediately after object instantiation, and receive an *IConfiguration* object in order to provide the object with re-configurability. The *IPluginService* interface is used for the discovery of the factories of the plug-ins. This interface defines methods that return the default factories for each module and also a collection of each discovered type of module, for future use; for example a GUI application that will use this collection to provide the user with the ability to choose a plug-in from a list of available plug-ins. The framework includes a default implementation (*DefaultPluginService*) of this interface that loads the factories of the plug-ins from a physical directory. However, the users can provide their own implementation of an alternative plug-in discovery and instantiation process; for example, they can develop a plug-in service able to discover plug-ins from a web service, to download and to use them automatically, or another plug-in service able to provide factories that utilize the Java Remote Method Invocation (Java RMI) technology.

Package *com.snsp.infrastructure.factories*, contains a set of factory interfaces for each module, other than those in *com.snsp.infrastructure.extensibility.core*, in order to decouple the discovery and instantiation of each module from the framework core. The framework provides a default implementation for all the factories utilizing the *IPluginService* in order to discover and instantiate the modules. By using this approach the framework is not coupled with the plug-in infrastructure. In case of an implementation without the plug-in infrastructure, custom implementations of those factories can be provided to the framework directly. This way the modules can be provided to the framework without using the plug-in infrastructure, and the time overhead introduced by the plug-in discovery process is avoided (**Figure 4**).

2.2. Framework Modules

An IDS module implements *IImageDataSource* interface, which is intended for the image acquisition processes. Images can be acquired from any video file, visual sensor, local folder or remote image repository, and they are made available to the rest of the framework modules. *IImageDataSource*, provides a convenient means to iterate through the input images by extending the native interface *java.lang.Iterable* with an image type. It extends the *IConfigurable* interface, to enable re-configurability of the module's parameters; for example, the developer may define the video frame acquisition rate as a tunable parameter from a text file. This interface is extended by the interfaces of all other modules.

An IP module implements the *IImageProcessor* interface, which is intended for preprocessing of the input images, *i.e.*, for their processing before they enter the analysis phase. Once the images are processed, they are made available to the rest of the modules. An IA module implements *IImageAnalyzer* interface, which is intended for the analysis of the acquired images. The main functionalities supported by JVA include image sampling, feature extraction, and visualization. Image sampling involves acquisition of sub-images from the input images, according to a specified sampling policy.

A sampling policy indicates a sequence of coordinates from which the sub-images should be extracted from an input image. Feature extraction involves estimation of representative values describing the content of the sub-image samples. IA may incorporate supervised or unsupervised machine-learning algorithms, image matching, retrieval or other image/video analysis methodologies. The developer can define the preferred image sampling policy, implement any image/video feature extraction methodology, and any algorithm for the visualization of the analysis results, by implementing IA along with its sub-modules which can also be integrated as independent plug-ins. These include: Area Policy (AP), which is intended for the definition of the policy to be followed during the image sampling process, Feature Extractor (FE), which is intended for the implementation of the feature extraction process, and the Analyzer-Visualizer (AV), which is intended for the implementation of both the analysis and visualization algorithms. Analysis and visualization are considered in a single module in order to preserve the independency between the three sub-modules of IA. The IA module plays a coordinating role for its sub-modules.

IImageAnalyzer extends *IConfigurable* interface for re-configurability purposes. The implementation lifecycle is handled by *IImageAnalyzerPlugin* interface. The same approach is followed by the interfaces *IVisualizer*, *IAreaPolicy* and *IFeatureExtractor*. The instances of *IVisualizer*, *IAreaPolicy* and *IFeatureExtractor* are provided to *IImageAnalyzer* through the framework core, and each one has its own lifecycle management.

An Output Handler (OH) module implements *IOutputHandler* interface, which is intended for image/video display and storage and also for possible transfers to remote destinations such as web servers and databases. The architecture of the core framework, provides the flexibility to have multiple instances of *IOutputHandler* implementations at the same time, and each of them is treated as an output observer which is informed whenever an output is available.

3. JVA System Development

In order to demonstrate the procedure for the development of a video analysis system with the JVA framework, we describe the implementation of a representative, well-known, methodology for segmentation of video frames based on data clustering, using publicly available software libraries.

Data clustering aims at the discovery of similarities between data so as to organize them into groups. By discovering similarities between sub-images sampled from a video frame it is possible to localize frame areas that correspond to different semantics; for example, in **Figure 3** the rightmost image in the GUI displays the result of such an analysis where the displayed video frame is segmented into three semantic areas namely, "Sky", "Acropolis" and "Border", based on their color. The utility of such a methodology spans in a variety of applications based on image segmentation [32]. This methodology proceeds as illustrated on the left part of **Figure 5**. For each step the respective component of the JVA framework that should be implemented is indicated on the right part of the same figure.

Initially, a video frame is extracted from an input video file. This requires an IDS module implementation, *i.e.*, an implementation of *IImageDataSource* interface, overriding only one method that sets up the image acquisition process, and an implementation of the standard Java interface *Iterator*, capable to iterate though the frames

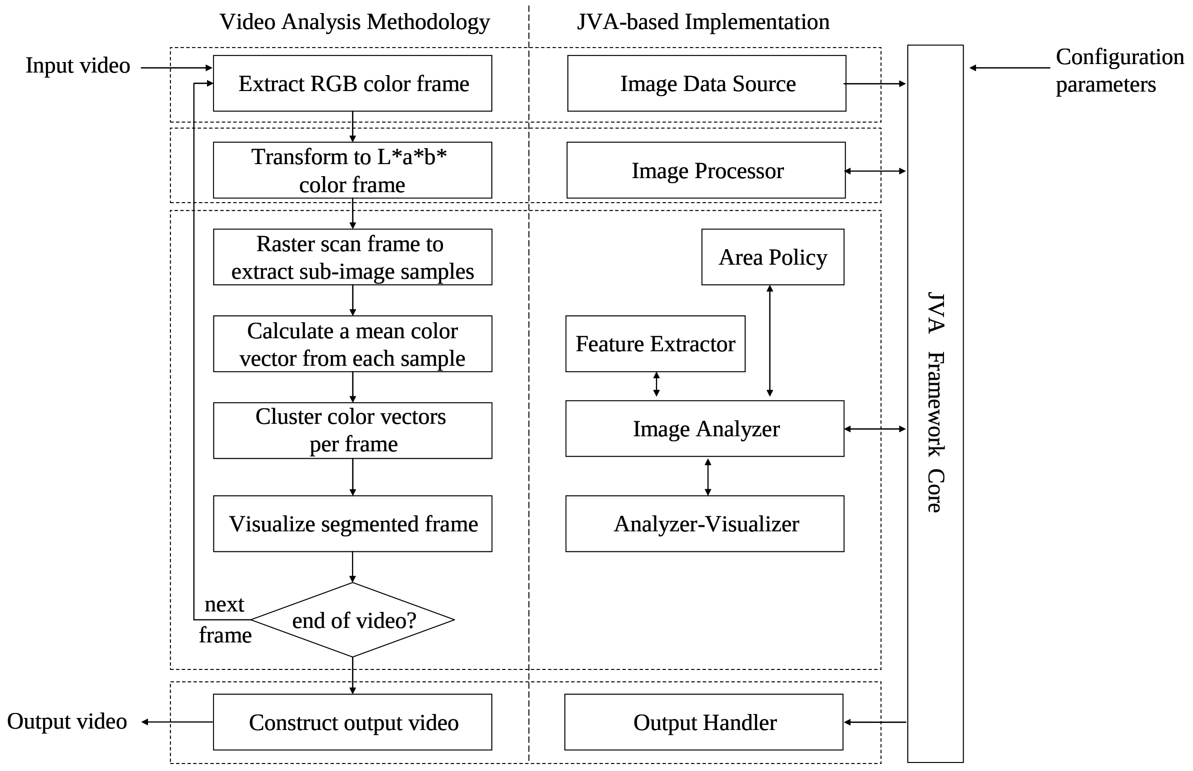

Figure 5. Video frame segmentation implemented using the JVA framework. A flow chart of the methodology is illustrated on the left of the dividing line (the arrows indicate control flow), and the respective JVA implementation is illustrated on the right (the arrows indicate interaction between components as described in **Figure 2**).

extracted from the input video. The extraction of the video frames from the input video is implemented into the iterator methods by exploiting the API of the Xuggler library.

A video frame extracted from the input video is represented in Red-Green-Blue (RGB) color space. However, due to the high correlation between the RGB color components, RGB is usually not preferred for color image segmentation. By transforming the pixel values from RGB to L*a*b* color space, whose components are approximately decorrelated to each other, it is possible to separate luminocity L* from the chromatic components a* and b*. In this way the different colors of an image will be more separable in the two-dimensional a*-b* space than in the three-dimensonal RGB space [33]. Such a non-linear transformation of the pixel values is an image processing operation that should be implemented within the IP module of the JVA framework.

In order to analyze the contents of the input image, the processed image is sampled and decomposed into a set of sub-images by sampling (if the IP module is not implemented, the input image is directly analyzed). A well-known technique to obtain samples from the whole image is the raster scanning approach, where overlapping square samples are uniformly acquired from the whole image with a user-specified spatial frequency. Such a raster-scanning or any other sampling policy should be defined in the AP module of IA. This module accepts an image and an *IConfiguration* object, and its responsibility is to return to the framework core a list of areas in the form of *IArea* objects (**Figure 4**) through a respective accessor method. *IArea* is an interface that acts as a Data Transfer Object (DTO) with position and scale information about the sampled image area.

Considering that each image sample carries a significant amount of information on its pixels, focusing only on the color information contained in each of the sampled sub-images, requires that each area is represented by a set of features capable of discriminating color information. A simple approach is to calculate mean (a*, b*) vectors from each sub-image. This feature extraction process should be implemented in the FE module of IA. FE must implement two methods: a method for object initialization, and a method for the implementation of the calculations necessary for feature extraction from the list of areas provided in the form of *IArea* objects. FE should also include mutator and accessor methods for the *IConfiguration* object and the extracted features data. The necessary calculations for color feature extraction can be implemented by exploiting the API of the well-known ImageJ library [34].

The feature vectors extracted from the image by the FE module are passed to the AV module. AV includes two main methods: one performing the analysis and one for the visualization of the results in a new image. In the example system described in this section, image analysis is performed by data clustering of the feature vectors into *K* clusters. Clustering can be performed using the API of the well-known Weka library [35]. In the example illustrated in **Figure 2**, we have used the expectation maximization (EM) clustering algorithm [36] implemented in this library using three target clusters ($K = 3$). The clustering results can be visualized as a new segmented image. This can be implemented by assigning different colors to the sub-images sampled during the sampling process. The colors can be selected to depict the membership of each sub-image to a cluster. The resulting image will be like the one illustrated in the rightmost panel of the GUI illustrated in **Figure 2**.

The IA module coordinates the AP, FE and AV modules, whereas it also provides a state through the whole video analysis task, *i.e.*, the same IA object can be reused for the analysis of each video frame until the stream of input video frames reaches its end. The interface implemented by the IA module defines mutator and accessor methods for property injection purposes to be utilized by the framework core in order to inject the AP, FE and AV modules into the IA object. The analysis process is initiated by calling a triggering method of that object. In case of supervised classification, the IA object also provides mutator and accessor methods for training data.

The last step in the development of a video analysis system using JVA is the construction of an OH module implementing the *IOutputHandler* interface. This module will be subscribed to the framework core as an output listener in order to handle the output display or storage of the system. Whenever an output image is ready, OH will iterate it through all the subscribers (*IOutputHandler* implementations) by an output handling method. Output display and storage can be implemented by exploiting the API of Xuggler library.

In all the afore-mentioned implementations, the parameters can be set so as to be externally configurable by an accessor method call to the *IConfiguration* object; for example, such parameters could include the path to a source input file, the sampling frequency, and the target number of clusters.

Once a system such as the one described in this section is implemented, its modules can be reused without any modification for other systems; for example, if one would like to implement an object detection algorithm the only module that would need to be revised is IA or only some of its sub-modules, such as the AV module. Therefore, by maintaining and enriching a collection of such reusable plug-in modules, a consequent reduction in the system development times is expected. In order to support this approach a growing collection of modules is provided open-access, as described in the following section.

4. Open-Access Framework

The JVA framework is publicly available as open-access software and it can be downloaded from its website[1]. This website includes documentation and a growing repository of JVA module implementations. Currently the available implemented modules are representative and open-source. They can be used directly for the development of a simple video analysis system, such as the one described in the previous section or for the development of more complex systems such as systems for panoramic image construction from video frames [37] [38] and visual odometry [39]. The open-source implementations facilitate also as models for quick development of any other video analysis system, by example.

Currently available implementations of the IDS module include: a) *FrameGrabber*, for input from standard video files; b) *LiveGrabber*, for input directly from vision sensors; c) *FileSystemImageDataSource*, for input from a local or remote image repository. An implementation of IP module is available for color space transformations, image cropping etc. Implementations of the IA module and of its sub-modules (AP, FE, VA) are provided for image sampling according to the raster scanning policy, for color and Speeded Up Robust Features (SURF) [40] extraction (FEs), and for clustering and visualization as described in section 3 and in [37] [38].

Three implementations of OH are also available through the JVA website: a) a default implementation of which is capable of composing a video from its frames; b) *LivePlayer*, which can be used to display the output video to the screen, and c) *DiskHandler*, which comes along with *LivePlayer* and can be used to save the output video to the local storage.

5. Evaluation Methodology and Results

Considering that the proposed framework mainly aims to improve the efficiency of developing video analysis systems, we performed a set of experiments to assess this improvement with quantifiable metrics. The empirical evaluation methodology adopted and the results obtained are described in the following paragraphs.

5.1. Methodology

We organized a software development challenge. The objective of this challenge was the development of a video analysis system for automatic segmentation of video frames. For the purposes of this challenge, the software development was led with the eXtreme Programming methodology (XP), which is intended to improve software quality and responsiveness to changing user requirements [41]. In order to simulate such changes in the user requirements and investigate the efficiency of the JVA-based system development in this context, a two-phase hypothetical use case scenario was considered: a) in the first phase, the users would like to have a software that is able to segment the video frames into different areas according to their color content; b) in the second phase, the users included the requirement that the software should be able to identify the areas that correspond to the sky within the video frames; for example, this could be useful in a robot navigation situation [42]. The methodology to solve the problem posed in the first phase is the one described in Section 3, whereas the methodology to solve the problem posed in the second phase, requires that the unsupervised clustering algorithm be replaced by a supervised classification algorithm. Such an algorithm needs to be trained with a set of samples from frames of different videos accompanied with annotations that indicate which areas in the video frames represent or do not represent the sky. Therefore, the handling of the training data and the classifier training process need to be implemented as well.

Three groups of developers participated in this challenge. Group A consisted of ten of the best students of our computer engineering department that have taken courses on advanced object-oriented programming and digital image processing. Students are relatively close to the population of interest since they are the next generation of software professionals which can quickly adopt new usable tools and technologies [43] [44]. Group B, consisted of ten researchers having two to five years of experience in Java programming. We have included this group because researchers comprise a significant population of potential developers, considering the increasing interest of the scientific community in this topic (**Figure 1**) and the fact that there are still plenty of unresolved issues related to emerging video analysis applications, e.g., in the area of computer vision for robotics. Group C, con-

[1]http://is-innovation.eu/jva.

sisted of ten professional Java programmers with experience in the software industry. **Table 1** summarizes the experience of all participants in Java programming, in terms of years and in Non-Commented Lines of Code (NCLOC) of their biggest project. The three groups were introduced to the APIs of ImageJ, Weka and Xuggler libraries in three four-hour hands-on seminars, one for each library. These seminars were focusing on functionalities that were necessary for the implementation of the described system, and aimed to provide the participants with common background knowledge on this topic.

The groups did not have any previous knowledge about the JVA framework. They were asked to develop the system initially with, and then without the framework. They were allowed to use the libraries to which they have been introduced, and the respective online documentation provided through the web. The participants were not allowed to copy code from one implementation to the other.

Since the scale of the project was small, some XP practices, such as pair programming, were omitted. The participants developed their own code individually, and used the Eclipse Integrated Development Environment (IDE) with the ActivitySensor plug-in installed, in order to monitor their activities during the software development process [45]. The completion of this process was split into two development cycles, one for each phase, and the intermediate releases were submitted to a Sub-Version Repository (SVN).

5.2. Results

The software development efficiency was assessed in terms of the effort spent by each programmer to develop the system. A simple but informative approach is to measure the absolute time required for the development. However, the total development time spent on programming can be divided into active and passive time. The active time is when a programmer types, and the passive time is when a programmer performs other activities concerning a project. If he knows what to do and which part of the source code is to be changed, the fraction of passive to total time is smaller [46]. We were able to capture both of these times with the use of the ActivitySensor Eclipse plug-in; considering that a switch from active to passive time happens after 15 seconds of a programmer's inactivity (the threshold was proposed by the activity sensor authors). After 15 minutes of inactivity, the passive time counter is stopped until a programmer hits a key.

All participants managed to complete their project. The development times required for the completion of the project, either with or without the use of the JVA framework, were correlated with the programming experience of the participants in years. The overall time spent for the completion of the first cycle was higher than the time spent for the completion of the second cycle, regardless of the approach considered. This is reasonable since the task in the second cycle requires changes in a subset of the code written in first cycle, implementing a specific functionality of the target system.

The results obtained from the time measurements in the first and in the second development cycle are illustrated in **Figure 6** and **Figure 7**, respectively.

It can be observed that during the first development cycle the use of the JVA framework resulted in a significant reduction of the software development time. This reduction across all participants is estimated to be 46.1% ± 5.9% on average, with a peak of 62% reduction achieved in the case of a professional developer with eight years of experience. The framework provides a generic scheme which has already resolved many of the design and implementation issues involved; for example, how to iterate through the video frames, communication between components, representation of feature vectors etc. Consequently, both the active and the passive times needed for development using the framework were shorter. Furthermore, it can be observed that the passive times obtained with the use of the framework for all participants, were in all cases shorter than 50% of the total cycle times of the participants (*i.e.*, the passive times were shorter than the active times), although they included some time for the adaptation of the developers to the use of the libraries. Without the use of the framework, the

Table 1. Java Programming experience of the participants in our empirical study.

	Group A: Students		*Group B: Researchers*		*Group C: Professionals*	
	min	max	min	max	min	max
Programming experience (years)	1	1	2	5	8	10
Size of biggest project (NCLOC)	500	1000	1500	7000	15,000	30,000

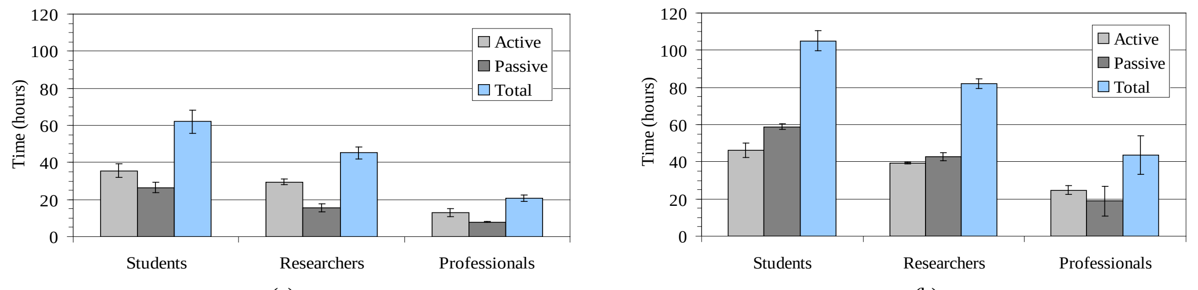

Figure 6. Average times per group, required for the development of the unsupervised video frame segmentation system during the first development cycle. (a) Using the JVA framework; (b) Without using the JVA framework.

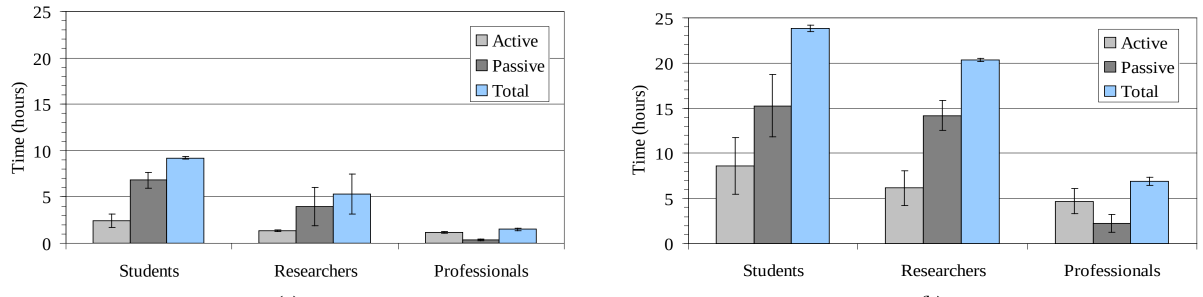

Figure 7. Average times per group, required for the development of the supervised video frame segmentation system during the second development cycle. (a) Using the JVA framework; (b) Without using the JVA framework.

passive time spent by the students and researchers exceeded 50% of the respective total cycle time, whereas this was not the case for the professionals. This can be justified by the fact that the latter ones, due their longer experience, coped with design and implementation issues of the software faster.

In the second development cycle all participants had already knowledge about the framework and its use, so they used this knowledge to change their source codes according to the user requirements of the second experimental phase. As in the first cycle, the use of the framework resulted in a significant reduction of the time required for the implementation of the changes, which is estimated to be 71.2% ± 8.7% on average across all participants. The maximum reduction was approx. 80%. It was observed in two cases; in the case of a researcher and in the case of a professional with two and ten years of experience, respectively.

Passive times are generally larger than active times in the second cycle for students and researchers, either using or not using the framework. This could be attributed to the fact that the time needed for implementation of the changes was shorter than the time needed to think how to implement them. However, there are some interesting observations, which could provide additional clues in the interpretation of the results. The average active times obtained with and without the use of the framework are 1.6% ± 0.7% and 6.5% ± 2.0%, respectively. This shows that using the framework, all users required significantly less effort to write and debug the code, with a rather small variance across developers of different skills.

It can also be observed that in the case of professionals the passive times were shorter than the active times. Since all participants had the same level of background knowledge about the use of the libraries, this implies that the longer times in the cases of students and researchers were mainly due to their shorter programming experience, especially on the substantial use of abstraction and inheritance, as confirmed by debriefing after the experiments. These concepts are usually more difficult for students to learn [47], and since the framework is composed of Java interfaces a good knowledge of such concepts is a prerequisite. The results obtained indicate that the framework can be learned faster by more advanced developers familiar with these concepts, and can result in very quick responses to changes in the user requirements.

A closer look at the resulting source codes indicate that the quicker response to changes with the use of the framework were mainly due to the modular architecture, which led the developers to make targeted changes mainly to the VA module. Although the systems developed were approximately similar in terms of functionality, there were observable differences between implementations done without the use of the JVA framework; for example, the systems produced by seven of the non-professional developers without the framework were not able to handle large video clips, whereas there was a significantly higher (by 42%) variance in the time-performance of the video analysis software produced without the use of the framework.

6. Conclusions

We presented a novel open-access object-oriented framework, which unlike current frameworks it explicitly addresses, the development of video analysis systems. The proposed framework has a generic modular architecture that can be easily maintained, extended and adapted to changes of user requirements by incorporating different modules as independent plug-ins. It builds on well-established architectural and design patterns which make it easily and quickly adoptable by software developers. A default implementation of the framework is provided in Java, considering the market needs for cross-platform software systems.

The framework provides a default implementation that handles all the communication between its abstract modules. This way the developers can take advantage of these already implemented parts of a video analysis process, and will rarely need to override their default implementation, although the framework is capable to. As a result, the overall development time can be reduced.

Another advantage of the generic architecture of the proposed framework is that is easily modifiable for the analysis of other data types as well, e.g. documents in the context of text mining. However, what makes it particularly suitable for video analysis is that it inherently considers video frames as the main structural components of the input data, and the processing pipeline can be applied either globally or locally per video frame. This does not mean that processing is only serial. The *IOuputHanlder* and *IImageDataSource* interfaces can be implemented so that they enable looking-ahead and looking-behind from a current frame, which is necessary, for example, in the analysis of motion patterns across video frames.

Experimentation for video analysis e.g. testing different parameters for optimization, can be very time consuming as the length and the resolution of the video files increase. The JVA framework can reduce the effort required for experimentation by enabling external parameterization, e.g. through text files; thus significant time can be saved from repetitive code compilations. Furthermore, the same code could be externally reconfigured (without editing any source code) for different video analysis tasks; for example, by changing training data, parameters or even pre-compiled modules. Therefore, the time required for developing software systems based on the same principles can be significantly reduced, and the coding time could even be eliminated.

In order to further support quick system development, extensive documentation and a collection of open-access (and open-source) plug-in modules are available through the web. These modules can be used as examples for the developers to quickly understand the concepts of the framework and to develop other modules. The new modules can also be provided through the web, thus continuously extending the available video analysis functionalities.

We performed an empirical study on the use of the proposed framework by developers with different levels of experience. Main conclusions that can be derived include:

- The use of the framework can reduce the development effort and speed up the development process, especially after a first cycle of development.
- The software produced with the use of the framework exhibits more stable time performance.
- Considering the open-access availability of plug-in modules, the effort required for learning the framework, and developing a video analysis system is expected to be even smaller than the effort measured in our study.

Future work includes extension of the framework to be able to handle multiple independent modules of the same kind simultaneously, and adaptation to inherently support parallel video analysis. The collection of JVA framework modules will continue to increase with contributions not only from our research group, but also from developers worldwide, through the dedicated website, which will be serving as a public module repository.

Acknowledgements

We would like to thank all subjects who participated in our study for their engagement.

References

[1] Leman, K., Ankit, G. and Tan, T. (2005) PDA Based Human Motion Recognition System. *International Journal of Software Engineering and Knowledge Engineering*, **15**, 199-204. http://dx.doi.org/10.1142/S021819400500218X

[2] Iakovidis, D.K., Tsevas, S. and Polydorou, A. (2010) Reduction of Capsule Endoscopy Reading Times by Unsupervised Image Mining. *Computerized Medical Imaging and Graphics*, **34**, 471-478. http://dx.doi.org/10.1016/j.compmedimag.2009.11.005

[3] Scopus (2014) http://www.scopus.com

[4] Correia, P. and Pereira, F. (1998) Proposal for an Integrated Video Analysis Framework. *Proceedings of International Conference on Image Processing*, **2**, 488-492.

[5] Anjulan, A. and Canagarajah, N. (2009) A Unified Framework for Object Retrieval and Mining. *IEEE Transactions on Circuits and Systems for Video Technology*, **19**, 63-76. http://dx.doi.org/10.1109/TCSVT.2008.2005801

[6] Park, S., Sargent, D., Spofford, I., Vosburgh, K., A-Rachim, Y. (2012) A Colon Video Analysis Framework for Polyp Detection. *IEEE Transactions on Biomedical Engineering*, **59**, 1408-1418. http://dx.doi.org/10.1109/TBME.2012.2188397

[7] Wang, Y.F., Chang, E.Y. and Cheng, K.P. (2005) A Video Analysis Framework for Soft Biometry Security Surveillance. *Proceedings of the* 3*rd ACM International Workshop on Video Surveillance & Sensor Networks*, 71-78.

[8] San Miguel, J.C., Bescós, J., Martínez, J.M. and García, Á. (2008) DiVA: A Distributed Video Analysis Framework Applied to Video-Surveillance Systems. *Proceedings of the* 9*th Workshop on Image Analysis for Multimedia Interactive Services*, Klagenfurt, 7-9 May 2008, 207-210.

[9] Czyżewski, A., Szwoch, G., Dalka, P., Szczuko, P., Ciarkowski, A., Ellwart, D., Merta, T., Łopatka, K., Kulasek, Ł. and Wolski, J. (2011) Multi-Stage Video Analysis Framework. Video Surveillance, 147-172.

[10] Masek, J., Burget, R. and Uher, V. (2013) IMMI: Interactive Segmentation Toolkit. *Engineering Applications of Neural Networks*, *Communications in Computer and Information Science*, **383**, 380-387. http://dx.doi.org/10.1007/978-3-642-41013-0_39

[11] BurgSys Corporation (2014) Image Analysis Software. http://www.burgsys.com/image-analysis-software.php

[12] Bradski, G. and Kaehler, A. (2008) Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media, Incorporated, Sebastopol.

[13] JavaCV (2013) Computer Vision Library. http://code.google.com/p/javacv/

[14] Abeles, P. (2012) Resolving Implementation Ambiguity and Improving SURF. Computer Research Repository (CoRR). http://boofcv.org

[15] Bellard, F. (2006) FFMpeg Multimedia System. http://www.ffmpeg.org/

[16] Xuggler (2013) Video Coding Library. http://www.xuggle.com/xuggler

[17] Fayad, M. and Schmidt, D.C. (1997) Object-Oriented Application Frameworks. *Communications of the ACM*, **40**, 32-38. http://dx.doi.org/10.1145/262793.262798

[18] Sullivan, S., Winzeler, L., Brown, D. and Deagen, J. (1998) Programming with the Java Media Framework. John Wiley & Sons, Inc., Hoboken.

[19] Larson, K. (2014). FMJ: Freedom for Media in Java. http://fmj-sf.net/

[20] Lyon, D. (2012) The Java Tree Withers. *Computer*, **45**, 83-85. http://dx.doi.org/10.1109/MC.2012.30

[21] Ackermann, P. (1996) Developing Object-Oriented Multimedia-Software: Based on MET++ Application Framework. Dpunkt-Verlag, Heidelberg.

[22] Weinand, A., Gamma, E. and Marty, R. (1988) ET++ An Object-Oriented Application Framework in C++. *ACM Sigplan Notices*, **23**, 46-57. http://dx.doi.org/10.1145/62084.62089

[23] Idris, I. (2013) Instant Pygame for Python Game Development How-to. Packt Publishing Ltd., Birmingham.

[24] SDL (2013) Simple DirectMedia Layer. http://www.libsdl.org/index.php

[25] Sauer, S. and Engels, G. (1999) OMMMA: An Object-Oriented Approach for Modeling Multimedia Information Systems. *Proceedings of the* 5*th International Workshop on Multimedia Information Systems* (*MIS*), Indian Wells, 64-71.

[26] Pleuß, A. and Hußmann, H. (2007) Integrating Authoring Tools into Model-Driven Development of Interactive Multimedia Applications. *Lecture Notes in Computer Science*, **4550**, 1168-1177.

[27] Amatriain, X., Arumi, P. and Garcia, D. (2008) A Framework for Efficient and Rapid Development of Cross-Platform Audio Applications. *Multimedia Systems*, **14**, 15-32. http://dx.doi.org/10.1007/s00530-007-0109-6

[28] Amatriain, X. and Arumi, P. (2011) Frameworks Generate Domain-Specific Languages: A Case Study in the Multimedia Domain. *IEEE Transactions on Software Engineering*, **37**, 544-558. http://dx.doi.org/10.1109/TSE.2010.48

[29] Gavalas, D. and Economou, D. (2011) Development Platforms for Mobile Applications: Status and Trends. *IEEE Software*, **28**, 77-86. http://dx.doi.org/10.1109/MS.2010.155

[30] Pequeno, H.S., Gomes, G.A., Andrade, R., de Souza, J.N. and de Castro, M.F. (2010) FrameIDTV: A Framework for

Developing Interactive Applications on Digital Television Environments. *Journal of Network and Computer Applications*, **33**, 503-511. http://dx.doi.org/10.1016/j.jnca.2010.01.002

[31] Bellifemine, F., Caire, G., Poggi, A. and Rimassa, G. (2008) JADE: A Software Framework for Developing Multi-Agent Applications. Lessons Learned. *Information and Software Technology*, **50**, 10-21. http://dx.doi.org/10.1016/j.infsof.2007.10.008

[32] Agarwal, S., Madasu, S., Hanmandlu, M. and Vasikarla, S. (2005) A Comparison of Some Clustering Techniques via Color Segmentation. *International Conference on Information Technology*: *Coding and Computing*, **2**, 147-153.

[33] Plataniotis, K.N. and Venetsanopoulos, A.N. (2000) Color Image Processing and Applications. Springer, Berlin. http://dx.doi.org/10.1007/978-3-662-04186-4

[34] Rasband, W. (2012) ImageJ: Image Processing and Analysis in Java. *Astrophysics Source Code Library*, **1**, Article ID: 06013.

[35] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I.H. (2009) The WEKA Data Mining Software: An Update. *ACM SIGKDD Explorations Newsletter*, **11**, 10-18. http://dx.doi.org/10.1145/1656274.1656278

[36] Jin, X. and Han, J.W. (2010) Expectation Maximization Clustering. In: Sammut, C. and Webb, G.I., Eds., *Encyclopedia of Machine Learning*, Springer, Berlin, 382-383.

[37] Iakovidis, D.K., Spyrou, E. and Diamantis, D. (2013) Efficient Homography-Based Video Visualization for Wireless Capsule Endoscopy. *Proceedings of the* 13th *IEEE International Conference on Bioinformatics and Bioengineering*, Chania, 10-13 November 2013, 1-4. http://dx.doi.org/10.1109/BIBE.2013.6701598

[38] Spyrou, E., Diamantis, D. and Iakovidis, D.K. (2013) Panoramic Visual Summaries for Efficient Reading of Capsule Endoscopy Videos. *Proceedings of the* 8th *IEEE International Workshop on Semantic and Social Media Adaptation and Personalization*, Bayonne, 12-13 December 2013, 41-46.

[39] Iakovidis, D.K., Spyrou, E., Diamantis, D. and Tsiompanidis, I. (2013) Capsule Endoscope Localization Based on Visual Features. *Proceedings of the* 13th *IEEE International Conference on Bioinformatics and Bioengineering*, Chania, 10-13 November 2013, 1-4. http://dx.doi.org/10.1109/BIBE.2013.6701570

[40] Bay, H., Ess, A., Tuytelaars, T. and Van Gool, L. (2008) Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, **110**, 346-359. http://dx.doi.org/10.1016/j.cviu.2007.09.014

[41] Beck, K. and Andres, C. (2004) Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, Boston.

[42] Shen, Y. and Wang, Q. (2013) Sky Region Detection in a Single Image for Autonomous Ground Robot Navigation. *International Journal of Advanced Robotic Systems*, **10**, 362. http://dx.doi.org/10.5772/56884

[43] Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K. and Rosenberg, J. (2002) Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, **28**, 721-734. http://dx.doi.org/10.1109/TSE.2002.1027796

[44] Tichy, W.F. (2000) Hints for Reviewing Empirical Work in Software Engineering. *Empirical Software Engineering*, **5**, 309-312. http://dx.doi.org/10.1023/A:1009844119158

[45] Activity Sensor Project (2013) http://www.e-informatyka.pl/sens/Wiki.jsp?page=Projects.ActivitySensor

[46] Madeyski, L. and Szala, L. (2007) Impact of Aspect-Oriented Programming on Software Development Efficiency and Design Quality: An Empirical Study. *IET Software*, **1**, 180-187. http://dx.doi.org/10.1049/iet-sen:20060071

[47] Liberman, N., Beeri, C. and Kolikant, Y.B.D. (2011) Difficulties in Learning Inheritance and Polymorphism. *ACM Transactions on Computing Education*, **11**, 4. http://dx.doi.org/10.1145/1921607.1921611