A Managed Approach to Computer Vision

ZORAN KALAFATIĆ

Department of Electronics, Microelectronics, Computer and Intelligent Systems
Faculty Of Electrical Engineering and Computing
Unska 3, 10000 Zagreb
CROATIA

HRVOJE SERTIĆ

Research & Development Center
Ericsson Nikola Tesla
Krapinska 45, 10000 Zagreb
CROATIA

Abstract: - Demanding tasks of computer vision systems require efficient design and effective use of available hardware components. Until recently, most of the computer vision systems were implemented using procedural languages and low-level programming. Although performance efficient, the procedural approach has significant drawbacks in terms of difficult extensibility and complicated implementation of generic solutions. This work describes architectural and implementation issues of a component realized to be a foundation of generic solutions in the field of computer vision. Major architectural decisions, from the definition of main component subsystems, through the usage of design patterns to proper interface definition are explained on actual examples from the component realization. The described implementation practices explain how to efficiently utilize a modern component based platform and a managed programming language. The performance analysis of the designed and implemented system clearly demonstrate the advantages of component based approach for realization of generic computer vision systems, capable of being adopted to various kinds of applications.

Key-Words: - computer vision, component development, managed platforms

1 Introduction

Usual approach to realization of CV systems is design and implementation of dedicated software solutions suited for specific problem domains. Dedicated solutions are in most cases realized with the use of procedural approach, meaning that it is very difficult to reuse once implemented solution for solving problems from different problem domains. There are exceptions to this practice: several projects have been realized with the goal to produce libraries of algorithms and procedures from CV domains, some with significant success. However, approach to create library with large number of CV algorithms has several drawbacks: due to complexity of such library, learning curve required to use algorithms from the library is very steep, extension of such libraries is complicated and result with even more complexity. Applications which use these libraries are very tightly connected with them, resulting in common changes in applications caused by changes inside the library.

This paper presents experiences and results of different approach, research aimed to develop reusable software component that will enable simple reuse of implemented algorithms for resolving problems from different domains. Main goal of described research is to create component architecture that will enable straightforward implementation of various algorithms from the field of CV and easy usage of implemented algorithms for realization of CV systems from different domains.

Another significant focus of described research is to investigate possibilities for using modern managed computing platforms for CV systems realization, such as Microsoft .NET, which is selected as foundation for component realization. Major benefits of .NET as managed platform (automatic memory management, simple object programming languages, provided services for component interoperability, etc.) are impacted with major drawback: usage of .NET services requires overhead which can cause loss of CV system performance. Because of that, aim of this work is to
evaluate performance of CV system realized on .NET platform with component based approach.

This paper is organized as follows: after the introduction, major requirements for generic computer vision system are explained. Third section describes global architecture and interface of CV component. Fourth section deals with architectural issues for image processing, segmentation and object tracking. Fifth section explains component implementation practices, while sixth section evaluates performance of realized component. Concluding results of this preliminary research are given at the end.

2 Requirements for generic CV components
Prior to realization of CV Component (CVC) several functional and non-functional requirements were defined and analyzed as guideline for implementation. Functional requirements were defined using use-case approach [1], while non-functional requirements were defined using list of statements. Main focus of requirement definition was on non-functional and high-level functional requirements. Due to basic idea of CVC implementation (provide foundation for experiments with various algorithms from the field of CV), functional requirements were not in-depth described. Instead, several high level use-cases were used to model expected types of algorithms that will be implemented in CVC.

2.1 Non-functional requirements
Aim to use object-oriented approach and services of .NET Framework [2] as foundation for CVC realization resulted with several non-functional requirements which had major impact on CVC architecture. Intention to utilize major benefits of object-oriented approach resulted with requirements intended to create object framework for representation of actual concepts from the field of CV. This means that several requirements determined CVC architecture in terms of defined objects responsible for representation of images, image features, objects discovered in images, trajectories of tracked objects and various concepts for image processing and object tracking.

Second type of non-functional requirements was focused to object management infrastructure. Object framework implemented in CVC was anticipated to provide means for interaction of objects representing various concepts from the field of CV. This means that each implemented object from CV domain must contain mechanisms for interaction with other objects, for example to determine if two tracked objects from video sequence are in collision by simple use of methods in their CVC representation.

Easy extensibility and addition of new algorithms from the field of CV was represented with a set of non-functional requirements. These requirements determined realization of CVC object framework with several classes defined for simple introduction of new algorithms and methods by using class inheritance and customization of inherited classes. With this approach, CVC architecture provides foundation for implementation and testing of various algorithms and methods for image filtering, analysis, segmentation, object identification and tracking in video streams.

Definition of non-functional requirements was completed with definition of requirements related to CVC interaction with .NET Framework. CVC was expected to perform all image operations in non-destructive manner, meaning that new copies of images should be created after each image operation. This requirement impacted CVC architecture, with the need to implement foundation for object management and handling.

Significant part of non-functional requirements was focused to the interface of CVC component. With non-functional requirements, complete expectations at interface level have been defined. Basic expectation from CVC was to provide simple and well defined interface for executing complex algorithms from the field of CV. This means that several non-functional requirements determined types of interface methods and expectations from interface in terms of support for future algorithms that will be implemented.

Types of interface methods have also been determined by non-functional requirements. Each on interface methods must have been realized according to asynchronous concepts, meaning that client application must never wait for completion of operation implemented in CVC component. Instead of waiting for operation completion, client application should receive notification using mechanisms implemented in .NET framework and events defined in CVC. Described approach significantly simplifies realization of client applications, enabling them to issue requests for several operations and perform other tasks during execution of these requests. CVC component was determined to handle all multithreading issues internally, so that client application could execute single thread for interaction with CVC services.

2.2 Functional requirements
Expected CVC functionality was defined using high level use-cases presented on Fig. 1. Described use-cases have been described from perspective of client application, representing CVC functionality on high
level (each of presented use-cases contains several functions) as used by client application for performing operations in the field of CV.

First task of every CV application is acquisition of images representing actual scenes. CVC was planned for image analysis and object tracking in non-real time, meaning that actual scene should be first recorded and then analyzed. Use-case Open File represents CVC functionality required to open image or video file in common formats (jpeg, gif, bmp, avi) and create internal objects (by using classes defined in CVC) representing those sources.

Functionality required for image representation and display in graphical user interface was selected to be implemented in CVC because of the aim to simplify realization of client applications as much as possible. Use-case Display Image represents CVC functionality responsible for displaying images and video in client applications. Inclusion of functionality for displaying images in CVC enables client applications to simply use simple services of CVC for displaying images, instead of using complex functionality provided by operating system, resulting with simpler realization of client applications.

Use-case Manage CV Objects was defined to represent part of CV functionality which enables client application to select objects for execution of CV algorithms. This means that CVC should provide functionality that will enable client application to execute some operation on particular image, and then use the resulting image as starting point for several additional operations that will be performed sequentially. CVC was required to work with video streams representing movements of objects in some duration of time, and to enable research of various algorithms from the field of CV. Because of that, CVC should contain functionality required to work with video streams – move currently analyzed frame forward or backward, analyze several frames in advance, etc. Use-case Manage Video represents functionality that enables client application to operate analyzed video stream.

Use cases Perform Image Operation and Perform Video Operation represent CVC functionality which enables client application to execute operations from the field of CV at single frame or video sequence. Those two use-cases are responsible for executing every algorithm contained in CVC, from simple image filters, to complex algorithms for tracking of object movement in video stream.

3. CVC Architectural concepts
CVC architecture was realized with six main subsystems (namespaces in .NET terminology [2], each of them implementing functionality related to specific part of

CVC requirements and related interfaces. We have defined subsystems to be self-contained in terms of contained functionality: each subsystem implements CVC interfaces required to use subsystem’s functionality from client application [3] Fig. 2 shows main CVC subsystems with respective interfaces.

3.1. Base Subsystem
Base subsystem is a foundation of CVC. Base subsystem contains functionality required for basic component operation. Two major interfaces are implemented in this subsystem: (1) IFileHandling – used by client application to open image and video files, (2) IImageManagement – used to perform basic image management tasks, such as selection of base image for various operation and working with processed images. Beside functionality required to implement described

![Fig. 1: CVC use-case model](image1)

![Fig. 2: CVC subsystems](image2)
interfaces, Base subsystem defines several types which represent major concepts from the field of CV. Most important defined type is cvcImage, a type which represents single image or frame in video stream. CvcImage is a complex type which contains functionality for simple and easy representation of several image formats and properties for accessing various features of represented images. It should be noted that cvcImage type is extension of image classes found in .NET framework with several enhancements and additions [2].

Base subsystem implements functionality required for interaction with client application. Client application receives information about status of performed image operations by subscribing to several events defined in CVC [4]. Functionality for interaction with client application is based on .NET framework delegate-based events [3], therefore enabling simple implementation of logic for receiving results from CVC. We point to significant limitation of .NET framework usage of events, which has impact on CVC architecture. It is impossible for client application to filter received events on certain conditions [3] (for example, to receive specific events only in case that several objects have been identified in currently analyzed frame). Because of that, hierarchy of event types representing various states of CVC was defined and implemented. We have defined specific event for each state of CVC operations. With this approach, CVC can straightforward inform client application about its status, but with disadvantage of increased complexity (client application must implement logic for different event types).

Logic for managing images that are being processed with CVC operations forms significant part of Base subsystem. We have defined Singleton class [5] ImageManagerSingleton which implements interface IImageManagement, providing methods to client application to easily manage several processed images. ImageManagerSingleton manages collection [2] of cvcImages, by providing methods for storing several base images (addBaseImage) and storing several processed images (addProcessedImage). With those methods, client application can simply assign several base images (foundation for image operations) and several processed images (result of various image operations). Methods for selecting and accessing stored images are also implemented, meaning that client application can compare results of various operations, display them, or select result of one image operation as foundation for other image operations.

3.2. Display Management subsystem
Functionality implemented in Display Management subsystem enables client application to display images by simply calling methods defined in IDisplay interface. This means that client application doesn't have to deal with .NET Framework functions for displaying images and creating display windows and forms. Instead, client application shows loaded or processed image by simply calling DisplayImage method of IDisplay interface. CVC interface hides complete functionality for accessing display related functionality of .NET Framework from client application by defining simple methods for image presentation and view management (working with multiple image windows, resizing windows and adjusting image size).

Major logic for image display management is defined in abstract class DisplayManager. We have used abstract class which defines foundation for display functionality because we wanted to provide customizable options for displaying images and results of image operations. This means that class DisplayManager can be extended with arbitrarily number of specialized subclasses. Two main classes for display management are used: SingleImageDMSingleton (used for displaying single image in each form) and MultipleImageDMSingleton (used for displaying multiple images in single display form). Singleton design pattern is mandatory for implementation of display management classes to provide single control point for all display functionality. Classes SingleImageDisplayForm and MultipleImagesDisplayForm extend windows forms of .NET Framework to grant specific displaying options. Each of classes for display management must maintain collection of used forms, therefore abstract base class DisplayManager implements .NET collection interfaces IEnumerable and IEnumerator [2].
3.3. Image Operations Subsystem

Subsystem Image Operations implements complete functionality defined in Perform Image Operation use-case. This subsystem had to satisfy contradictory requirements: aim to provide foundation for implementing various and future image operations and aim to properly define CVC interface. We have opted to define two interfaces for image operations: IImageOperations which provides method ApplyImageFilter() used by client application to execute image operation, and interface IImageOperationsParameters which defines properties for parameters used by different image operations. With this approach we have defined interface for invoking image operations and separated changing part of interface, which can not be statically defined due to unknown parameters of future algorithms.

Strategy design pattern [5] is used for implementation of image operations. We have defined two subclasses (BaseImageOperations and AdvancedImageOperations) which provide foundation for implementing various algorithms. Those two classes implement common functionality of different image operations – BaseImageOperations subclass provides functionality for accessing image information (single pixels), while AdvancedImageOperations provides functionality for simple implementation of correlation image operations. Both subclasses implement method PublishCompletedOperationEvent which is used by algorithm subclasses to inform client application that the image operation is completed.

According to Strategy design pattern, each image operation (its algorithm) is implemented as leaf subclass. With this approach, each image operation is separated and self-contained. This approach has several advantages: client application simply uses instance of algorithm leaf class and invokes method defined in component interface to perform image operation. Implementation of new image operations is simple: one of ImageOperations subclasses should be inherited and code for new image operation implemented in already defined methods. Code required for accessing image information and performing computations on per-pixel level is separated into ImageOperations subclasses, therefore simplifying realization of algorithm leaf subclasses.

Interface IImageOperationsParameters provides properties used by various image operations to fetch parameters set by client application. We have defined Singleton class ImageParametersSingleton to provide single point of access to parameters for all image operations. This means that implementation of new image operation into CVC must, beside addition of algorithm leaf class, update interface IImageOperationsParameters with properties used by client application to set parameters for specific operation.

3.4. Video Operations Subsystem

Main goal of Video Operations subsystem is to provide foundation for implementation of various CV algorithms for object identification and tracking. Similar to Image Operations subsystem, architecture of this subsystem
must be able to support implementation of future algorithms for object identification, prediction of object position and features, and tracking of object movement between several video frames. Client application access functionality of Video Operations subsystem through interface VideoOperations which provides methods for initializing specific instance of object tracker, initiating tracking of objects in video sequence and tracking on frame basis – required for behavior analysis of tracking and prediction algorithms.

Video operations are handled by VideoOperationsManager class which is responsible for coordinating object tracking and prediction. We have defined two strategies for implementing families of algorithms: TrackerStrategy and Predictor Strategy. Those strategies are used to represent algorithms for object tracking and prediction of object position and features. We require those algorithms to be implemented as Singleton classes in order to enable single point of access to currently used algorithms.

Algorithm for object tracking is required to use functionality implemented in Image Analysis subsystem to identify objects in currently analyzed video frame. Algorithms for image analysis return list of temporary objects defined in Video Operations subsystems to represent identified objects in currently analyzed video frame. Responsibility of object tracker is to analyzed those temporary objects and provide input to ObjectIDManagerSingleton class which creates actual representation of tracked objects by creating objects from cvcObject class. Each cvcObject is a complex object created by Abstract Factory cvcObjectAbstractFactory. We have used Abstract Factory design pattern for creation of cvcObjects because we wanted to provide simple interface for creation of various types of objects. We find the application of this design pattern very appropriate for CVC because different representations of tracked objects are required by various tracking algorithms. For the example shown on Fig. 5, we have defined cvcSimpleObject to contain object for representing features of tracked object (cvcObjectFeatures), region of interest (part of image where tracked object can be positioned – cvcRoi) and object contour represented by cvcContour class. With described approach, ObjectIDManagerSingleton can simply create cvcObjects according to the requirements of used tracking algorithms.

Support for predictable object tracking is implemented in CVC with Strategy called PredictorStrategy. Similar to the use of Strategy design pattern for representation of various tracking algorithms with different classes, PredictorStrategy enables implementation of various prediction algorithms into CVC by definition of separate classes for each prediction algorithm. After the analysis of current video frame is completed, VideoOperationsManager receives list of identified cvcObjects. To prepare tracking of objects in next video frame, VideoOperationsManager sends list of identified objects to one of prediction algorithms defined in PredictorStrategy (BasePredictorSingleton shown on Fig. 5) and invokes method PredictObjectFeatures. This results with prediction of features of currently tracked objects (object position, size, color, shape or contour) in future video frames (according to algorithm used, object features can be predicted for one or several video frames).

ObjectIDManagerSingleton class is responsible for maintaining path of each cvcObject in analyzed sequence of frames. This means that for first frame in sequence, ObjectIDManagerSingleton creates cvcObjectPath object for each created cvcObject. In future analyzed frames, ObjectIDManagerSingleton analyzes identified objects, compares them with predictions created by one of prediction algorithms and establishes logical link between cvcObjects identified in current frame and cvcObjects identified in previous frames, and as a result adds newly identified object position to cvcObjectPath of that particular object.

With described architecture, we have managed to provide flexible environment for implementation and testing of various algorithms from the field of CV. New algorithm for object tracking can be simply added to CVC by extending TrackerStrategy class, and new prediction algorithm can be added by extending PredictorStrategy class. It must be noted that presented architecture depends on properly defined cvcObject classes, which must implement functionality for object comparison that is used by ObjectIDManagerSingleton for identifying same object in different video frames by comparing features of identified objects.

4. Implementation issues

4.1. Memory operations

CVC was implemented using Microsoft .NET Framework v1.1 and C# programming language. CVC functionality can be accessed from client applications using services of .NET Framework, or using COM Callable Wrapper (CCW) interface that enables interaction of older clients based on COM technology with .NET Framework components. Major part of .NET Framework, Common Language Runtime (CLR) had major impact on CVC realization. CLR manages code at execution time, providing memory management, thread management and other services to applications based on .NET Framework.
Automatic memory management caused significant CVC realization problem: poor performance of image operations. By preventing direct memory access, CLR enables simple implementation and safe execution of various applications in terms of completely avoiding memory leaks and problems with improper use of memory pointers, but with significant performance drop of memory operations. Typical image operations works with raw image data stored in memory, usually directly modifying image data at byte level. Image operations performed by managed services of .NET framework were unacceptably slow and unusable for larger image files. In order to avoid described problems, we had to use direct memory access under .NET Framework.

.NET Framework defines unsafe keyword which is used to define unsafe context in code which allows direct memory access and the use of memory pointers. Implementation of each image operation must be placed into unsafe context in order to enable direct access to image data. We have defined two unsafe methods in image operations base class (GetPixelAddress8Bpp and GetPixelAddress24Bpp) which are used to provide pointer to first image pixel for respective type of image (grayscale with 8 bits per pixel and full color with 24 bits per pixel). With this approach, we have provided simple implementation of image operations with direct memory access and combined advantages of .NET Framework with the speed of pointer-based memory operations.

4.2. Interface implementation
Each of image or video operations implemented in CVC is realized as specialized subclass of abstract class that implements particular CVC interface. .NET Framework differentiates interfaces from abstract classes in several areas: interface can not have implementation or member variables, all interface members are public by default, and interface can not have static members or constructors. Restricting any kind of implementation details in interfaces provides loose coupling between client application and component itself, but requires specific usage of logic implemented in component.

Major aim of CVC, simple extension of implemented algorithms has impacted the way client application uses implemented functionality. Because .NET interfaces can not be instantiated, .NET Framework forces client applications to choose particular interface implementation to instantiate. To interact with a component implementing interface, client application must instantiate concrete class that supports the interface and assign that object to an interface variable, similar to using any other base type. In order to call specific image operation implemented in CVC (for example, Canny edge detector), client application must instantiate concrete class Canny that supports IImageOperations interface by calling:

\[ \text{IImageOperations imOps = new Canny();} \]

and then starting image operation by calling:

\[ \text{imOps.ApplyImageFilter(sourceCvcImage);} \]

which will result with creation of new cvcImage that has applied Canny edge detector. Described approach enables simple and loose-coupled usage of functionality implemented in CVC. Even more, multiple image operations can be combined in sequence by simple sequential instantiation of several concrete classes which contains image operation algorithms.

5. Performance Analysis
Performance of implemented image and video operations was tested on mid-range computer configuration – notebook computer based on Intel Centrino 1.5GHz processor with 512Mb RAM. We have tested performance of static image operations on files of various sizes and operations on video stream, where we measured number of video frames that can be processed. We have also measured performance of video operations without displaying resulting image on screen to identify how much performance is lost due to poor video subsystem of used notebook computer.

5.1. Performance of Image Operations
Performance of image operations performed on static images was measured from the client application using timers provided by .NET framework. Each image operation was performed three times and average result was used. Every image operation was performed on JPG image with minimal compression resized in three

<table>
<thead>
<tr>
<th>Time in seconds</th>
<th>Small Image (880 x 586 pixels)</th>
<th>Medium Image (1760 x 1173 pixels)</th>
<th>Large Image (3519 x 2345 pixels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invert</td>
<td>0.030</td>
<td>0.065</td>
<td>0.143</td>
</tr>
<tr>
<td>Convert to Grayscale</td>
<td>0.042</td>
<td>0.080</td>
<td>0.275</td>
</tr>
<tr>
<td>Gauss</td>
<td>0.590</td>
<td>2.253</td>
<td>8.946</td>
</tr>
<tr>
<td>Canny Edge Detector</td>
<td>1.371</td>
<td>5.397</td>
<td>21.743</td>
</tr>
</tbody>
</table>

Table 1: Performance of image operations
different sizes (maximum, around 8M pixels, 50% of original size and 25% of original size). Performance was tested with four image operations of different complexity – from simple Invert operation that to complex convolution based operations (Gauss and Canny Edge Detector). Presented results show more than acceptable performance of implemented operations – most of CV systems work with images much smaller than our sample images. It must be noted that displaying images only slightly affects performance of image operations. This was expected, since all used display functions are based on .NET Framework, which includes optimized display functionality.

5.2. Performance of video operations
Performance of video operations was also measured from client application using timers provided by .NET framework. Results presented on Table 2. show how many frames per second have been processed using avi video files of two different sizes. We do not include results for video operations with disabled display since our test shown almost no difference in performance compared to displayed images. It must be noted that most of time required for displaying images is related to drawing new window, so that re-displaying image in already open window causes insignificant delay with presented video sizes.

6. Conclusion
Results of presented research explain all advantages that the use of modern component-base platform with object-oriented approach can bring into realization of computer vision systems. Defined interface with straightforward methods allow simple implementation of client applications and usage of component functionality. Realized architectural concepts allow simple addition of new algorithms and methods from the field of computer vision. Performance analysis confirms that today's mid-range computer systems based on managed platforms are more than capable of performing demanding image operations required for realization of computer vision systems. However, described approach contains several drawbacks such as relatively complex architecture and effort required for implementation of component base required to provide base services for computer vision functionality. Presented results clearly show that, despite described drawbacks, realized component based on managed platform can successfully be used as foundation for future research and simple realization of complex computer vision systems.

References: