Scalable Independent Multi-level Distribution in Multimedia Content Analysis

Viktor S. Wold Eide[1,2], Frank Eliassen[2],
Ole-Christoffer Granmo[1,2], and Olav Lysne[2⋆ ⋆⋆]

[1] Department of Informatics, P.O. Box 1080 Blindern, N-0316 Oslo, Norway
{viktore,olegr}@ifi.uio.no
[2] Simula Research Laboratory, P.O. Box 134 N-1325 Lysaker, Norway
{viktore,frank,olegr,olavly}@simula.no

Abstract. Due to the limited processing resources available on a typical host, monolithic multimedia content analysis applications are often restricted to simple content analysis tasks, covering a small number of media streams. This limitation on processing resources can often be reduced by parallelizing and distributing an application, utilizing the processing resources on several hosts. However, multimedia content analysis applications consist of multiple logical levels, such as streaming, filtering, feature extraction, and classification. This complexity makes parallelization and distribution a difficult task, as each logical level may require special purpose techniques. In this paper we propose a component-based framework where each logical level can be parallelized and distributed independently. Consequently, the available processing resources can be focused on the processing bottlenecks at hand. An event notification service based interaction mechanism is a key factor for achieving this flexible parallelization and distribution. Experiments demonstrate the scalability of a real-time motion vector based object tracking application implemented in the framework.

1 Introduction

The technical ability to generate volumes of digital media data is becoming increasingly "main stream". To utilize the growing number of media sources, both the ease of use and the computational flexibility of methods for content-based access must be addressed.

In order to make media content more accessible, pattern classification systems which automatically classify media content in terms of high-level concepts have been taken into use. Roughly stated, the goal of such pattern classification systems is to bridge the gap between the low-level features produced through signal processing (filtering and feature extraction) and the high-level concepts desired by the end-user. Automatic visual surveillance [1], automatic indexing of TV Broadcast News [2] (e.g. into Newscaster, Report, Weather Forecast, and Commercial segments), and remote sensing image interpretation [3] are examples of popular application domains.

⋆ Authors are listed alphabetically

⋆⋆ The DMJ project is funded by the Norwegian Research Council under grant no. 126103/431

F. Boavida et al. (Eds.): IDMS/PROMS 2002, LNCS 2515, pp. 37–48, 2002.

Due to the limited processing resources available on a typical host, monolithic multimedia content analysis applications are often restricted to simple content analysis tasks. Multimedia content analysis applications consist of multiple logical levels, such as streaming, filtering, feature extraction, and classification. This complexity makes parallelization and distribution a difficult task, as each logical level may require special purpose techniques. For instance, in [4], it is shown how the filtering in a video based people counting application can be distributed to the sensors, based on a special purpose multimedia surveillance network. Accordingly, a higher frame rate can be achieved or more advanced filtering can be conducted.

In the DMJ (Distributed Media Journaling) project we are developing a component based framework for real-time media content analysis. New sub-technologies (e.g. a new feature extraction algorithm) may be plugged into the framework when available.

The resource requirements for the framework application domain are very challenging and will most likely remain so in the near future, justifying the need for scalability. In this paper we show the framework scalability for a relatively tightly coupled application (components interact with the video framerate frequency) processing a single video stream. A massively distributed application utilizing a large number of cameras (e.g. for traffic surveillance) may require such tight coupling only between some components.

The relative complexity of streaming, filtering/transformation, feature extraction, and classification depends on the application. Therefore the framework should support focusing of processing resources on any given logical level, independently of other logical levels. E.g., if only the filtering is parallelized and distributed (as in the case from [4]), the feature extraction and the classification may become processing bottlenecks.

In this paper we focus on the parallelization and distribution mechanisms of the DMJ framework. In Sect. 2 we describe the general approach for building content analysis applications. We also introduce our application case, tracking of a moving object in a video stream. In Sect. 3 we first give an overview of the DMJ framework. In Sect. 3.1 we shortly describe inter component communication and synchronization. We then proceed to motivate and present the special purpose parallelization and distribution techniques for each logical level in Sect. 3.2 to Sect. 3.5. In Sect. 4 we present the results of an experiment which demonstrate the scalability of our framework. In Sect. 5 we present plans for future work. Lastly, we provide some conclusions in Sect. 6.

2 Content Analysis

A general approach for building content analysis applications is to combine low-level quantitative media processing into high-level concept recognition. Typically, such applications are logically organized as a hierarchy, as shown in Fig. 1. At the lowest level of the hierarchy there are media streaming sources. At the level above, the media streams are filtered and transformed. The transformed media streams are then fed to feature extraction algorithms as media segments (e.g. video frame regions). Feature extraction algorithms operate on the media segments from the transformed media streams, and in the case of a video frame region, calculate features such as color histograms and motion vectors. Finally, results from feature extraction algorithms are reported to classification algorithms higher up in the hierarchy that are responsible for detecting high level domain

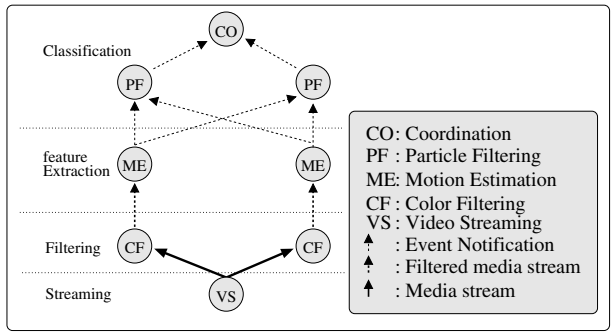

Fig. 1. A specific configuration, out of many possible configurations, of a content analysis application for real-time tracking of a moving object in a video stream

concepts, such as a moving object in a video stream. In other words, classification is interpretation of extracted features in some application specific context.

Fig. 1 illustrates a *possible configuration* of a content analysis application for real-time tracking of a moving object in a video stream, the application henceforth used for illustration purposes. The video stream is filtered by two algorithms, each doing video stream decoding and color-to-grey level filtering. Each filtered video frame is divided into $m \times n$ blocks (media segments) before two motion estimators calculate motion vectors for the blocks. The block motion vectors are then submitted to two so-called *particle filters* (described in 3.5) for object detection and tracking. The coordinator uses the results from all the particle filters to determine the position of the moving object.

Often, the above type of content analysis applications are implemented as monolithic applications making reuse, development, maintenance, and extension by third parties difficult. Such applications are often executed in single processes, unable to benefit from distributed processing environments.

3 The DMJ Framework

As a solution to the inherent problems of traditional monolithic content analysis systems, we suggest a component-based approach. Logically, the media processing hierarchy is similar, but the different algorithms at each logical level are now encapsulated in components - S (Streaming), F (Filtering), E (feature Extraction), and C (Classification) components. The content analysis task is realized as a collection of components, which indirectly monitor other components and react to particular changes in their state.

The resulting content analysis hierarchy can then be executed as a pipeline (each level of the hierarchy is executed in parallel). For instance, the application described in Sect. 2 can be executed on five CPUs, where the streaming is conducted from one CPU, the filtering is executed on a second CPU, the motion estimation is conducted on a third CPU, and so forth. Such distribution allows an application to take advantage of a number of CPUs equal to the depth of the hierarchy. In addition, the DMJ framework

also supports independent parallelization and distribution within each logical level. In the current prototype of the framework, each logical level implements special purpose parallelization and distribution techniques, as we will see in Sect. 3.2 to Sect. 3.5. In combination, this opens up for focusing the processing resources on the processing bottlenecks at hand. An example of such parallelization is found in Fig. 1 where the motion estimation (and the particle filtering) can be conducted on two CPUs.

3.1 Component Interaction and Synchronization

Components interact in different ways, such as one-one, one-many (sharing or partitioning of data), many-one (aggregation), and many-many. In [5] we argue that the requirements for our application domain fit very well with the publish/subscribe interaction paradigm, leading to an event-based interaction model. Event-based systems rely on some kind of event notification service which introduces a level of indirection. The responsibility of the event notification service is to propagate/route event notifications from the event producers to interested event consumers, based on content and generally in a many-many manner. A component does not need to know the location, the identity, or if results have been generated by a single or a number of components. The binding between components is loose and based on what is produced rather than by whom. Note that the event notification service should take advantage of native multicast on the network layer for scalability reasons, as will become clear in the following sections.

Some kind of synchronization and ordering mechanism is required in order to support parallel and distributed processing. Such a mechanism is described in [5], in which each media sample and event notification is assigned a timestamp (actually a time interval) from globally synchronized clocks. In other words, the design of our framework is based upon a common knowledge of time in all components. This is realized by synchronizing the computers by e.g. the Network Time Protocol, RFC 1305.

3.2 Media Streaming

Each media source receives its input from a sensor, implemented in software (e.g. a program monitoring files) or as a combination of both hardware (video camera, microphone, etc.) and software (drivers, libraries, etc.). From a scalability point of view, reducing sender side processing and network bandwidth consumption is important.

Some media types may generate large amounts of data, requiring effective encoding in order to reduce bandwidth requirements to a reasonable level. We currently work with live video, a quite challenging media type with respect to processing requirements, the massive amounts of data, and the imposed real-time requirements. E.g., a television quality MPEG-2 encoded video stream, Main profile in the Main Level, 720 pixels/line x 576 lines, may require as much as 15 Mbps [6]. The actual data rate depends on both intra- and inter frame redundancy, i.e. the media content. Real-time encoding is likely to remain costly in the near future too, considering a likely increase in video quality.

A media source should be able to handle a number of interested receivers, belonging to the same or to different applications. A video streaming source which must handle each and every component individually will not scale. Scalable one to many communication is what IP multicast [7] has been designed for. Each packet requires a single send operation and should traverse each network link only once. In the current prototype, we have used IP multicast for streaming video data, as illustrated by label 1 in Fig. 2.

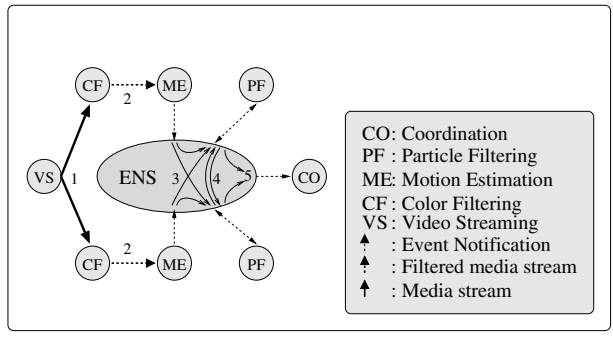

Fig. 2. Inter component communication for the configuration in Fig.1. Feature extraction and classification components interact through an Event Notification Service, labeled ENS

3.3 Filtering and Transformation

Filtering and transformation bridge the gap between what a S component offers and an E component can handle. As an example, filtering and transformation components may be used to convert MPEG-2 to YUV to 32 bit RGB to 8 bit gray level video frames.

An effective encoding of a media stream reduces network bandwidth consumption, but results in increased processing requirements for decoding. If components both receive and decompress each and every frame of a high quality video stream entirely, the number of CPU cycles left for the rest of the processing may be severely reduced. As an example, real-time software decoding of a MPEG-2 TV quality video stream requires a fairly powerful computer. Furthermore, filtering and transformation may be computationally costly by itself. Consequently, our framework should support parallel and distributed filtering and transformation.

In the current prototype the filtering and transformation is executed in the same process as the feature extraction, and data is transferred to feature extraction components by reference passing. This data flow is labeled 2 in Fig. 2.

3.4 Feature Extraction

A feature extraction algorithm operates on media segments from the filtering and transformation level (e.g. video frame blocks) and extracts quantititave information, such as motion vectors and color histograms. The features of each media segment are used at the classification level to assign a high-level content class to each media segment.

Feature extraction algorithms may use information from the compressed or partial decompressed domain if available (e.g. utilize the motion vectors in a MPEG-2 video).

Some feature extraction algorithms require relatively small amounts of processing, such as a color histogram calculation which may only require a single pass through each pixel in a video frame. But even such simple operations may become costly when applied to a real-time high quality video stream. In general the algorithms may be arbitrarily complex. In combination with the high data rate and often short period of time between succeeding frames this may easily overwhelm even a powerful computer. A scalable

solution necessitates parallelization, which requires a partitioning of the data in the media stream, spatially and/or temporally.

Feature extraction algorithms for video, such as motion vector extraction, color histogram calculation, and texture roughness calculation, often operate locally on image regions (e.g. a block). The DMJ framework supports spatial parallelization and distribution of such feature extractors. As an example, block-based motion estimation is computationally demanding, but the calculation of a single block motion vector is localized to a small image region. Accordingly, the calculation of motion vectors in a single video frame can be parallelized and distributed. For the sake of completeness we give a short description of parallel block motion vector extraction in the DMJ framework.

Block-based Motion Estimation. In order to identify and quantify motion between two consecutive frames, a block-based scheme is used. A block from the previous frame is compared to the corresponding block in the current frame. A block difference value is calculated by summing all the pixel value differences and this value indicates the similarity between the two blocks. If an object or the camera moves between two consecutive frames, the calculated block difference value may become large and a search for a similar block in the current frame is necessary. Searching is done by offsetting the corresponding block in the current frame some pixels horizontally and vertically. A search area is defined by the maximum number of pixels to offset the block. In the worst case, a brute force search must compare the block in the previous frame with all blocks defined by the search area. This searching requires lots of processing and a number of algorithms have been proposed in order to reduce the number of blocks compared [8]. The search is usually terminated whenever a block with difference value below some threshold has been found, introducing indeterminism since the processing requirements depend on the media stream content. The offset $[\delta x, \delta y]$ which produces the smallest difference value, below a threshold, defines the motion vector for this block.

Our implementation allows a component to calculate motion vectors for only some of the blocks in the video frame, defined by a sequence of rectangles, each covering some blocks. In case of parallel processing, such motion estimation components are mapped onto different hosts, each processing some of the blocks in the whole frame.

In Fig. 3, the motion vectors calculated by a single component have been drawn into the left video frame. The figure also illustrates how a component may get configured to process only some regions of the video stream. The blocks processed are slightly darker and they also have the motion vectors drawn, pointing from the center of their respective block. The motion vectors indicate that the person is moving to the left.

The motion vectors calculated for blocks in video frames are sent as event notifications. The event notification service will then forward such event notifications to the interested subscribers, as indicated by label 3 in Fig. 2.

3.5 Classification

The final logical level of the DMJ framework is the classification level. At the classification level each media segment is assigned a content class based on features extracted at the feature extraction level. For instance, if each video frame in a video stream is divided into $m \times n$ blocks as seen in the previous section, the classification may consist of deciding whether a block contains the center position of a moving object, based on extracted motion vectors.

Fig. 3. Left: Block-based motion estimation example. Right: The center position of the tracked object, calculated by the coordinator, has been drawn as a white rectangle

Features may be related spatially and temporally to increase the classification accuracy. E.g., if a block contains the stomach of a person moving to the left, above blocks should contain "person" features. Blocks to the right in previous video frames should also contain such features. When features are related spatially and temporally, the classification may also be referred to as *tracking* or *spatial-temporal data fusion*.

In this section we first discuss how the classification can become the processing bottleneck in a content analysis application, as well as the consequences. We then propose a parallelizable multi-component classifier which addresses this bottleneck problem.

Processing Bottlenecks. The classification may become a processing bottleneck due to the complexity of the content analysis task, the required classification rate, and the required classification accuracy. E.g., rough tracking of the position of a single person in a single low rate video stream may be possible using a single CPU, but accurately tracking the position of multiple people as well as their interactions (talking, shaking hands, etc.) could require several CPUs. Multiple media streams, such as video streams from multiple cameras capturing the activity on an airport, may increase the content analysis complexity even further. In the latter setting we may for instance consider tracking the behavior and interaction of several hundred people, with the goal of detecting people behaving suspiciously. This example would probably require a very large number of CPUs for accurate classification at an appropriate video frame rate. In short, when the classifier is running on a single CPU, the classification may become the processing bottleneck of the content analysis application.

When the classification becomes the processing bottleneck either the content analysis task must simplified, the classification rate/accuracy requirements must be relaxed, or the amount of processing resources available for classification must be increased. Simplifying the content analysis task may be costly in terms of implementation effort. Furthermore, reducing the accuracy of a classifier, in order to reduce the processing resource usage, may be an intricate problem depending on the classifier. Changing the classification rate is easily done, but this may have implications on the other logical framework levels (which also should reduce their operation rate accordingly). In addi-

tion, the content analysis task and the classification rate/accuracy requirements are often given by the application and cannot be modified. Consequently, often the only option is to increase the amount of processing resources available for classification. Unfortunately, if the classification cannot be distributed, increasing the available processing resources is only effective to a limited degree.

A Parallel and Distributed Classification Component. To reduce the problems discussed above, the DMJ framework classification level supports: effective specification of content analysis tasks through the use of dynamic Bayesian networks [9], flexible execution of content analysis tasks based on the particle filter algorithm [9], fine grained trading of classification accuracy against classification processing resource usage as a result of using particle filters, and fine grained trading of feature extraction processing resource usage against classification accuracy [10] [11].

In the following we describe our use of the particle filter in more detail. Then we propose a distributed version of the particle filter, and argue that the communication and processing properties of the distributed particle filter allow scalable distributed classification, independent of distribution at the other logical levels of the DMJ framework.

The Particle Filter: Our particle filter is generated from a dynamic Bayesian network specifying the content analysis task. During execution the particle filter partitions the media stream to be analysed into time slices, where for instance a time slice may correspond to a video frame. The particle filter maintains a set of particles. A single particle is simply an assignment of a content class to each media segment (e.g. object or background) in the previously analysed time slices, combined with the likelihood of the assignment when considering the extracted features (e.g. motion vectors). Multiple particles are used to handle noise and uncertain feature-content relationships. This means that multiple feature interpretations can be maintained concurrently in time, ideally until uncertainty can be resolved and noise can be supressed.

When a new time slice is to be analysed, each particle is independently extended to cover new media segments, driven by the content analysis task specification. In order to maintain a relevant set of particles, unlikely particles are systematically replaced by likely particles. Consequently, the particle set is evolved to be a rich summarization of likely content interpretations. This approach has proven effective in difficult content analysis tasks such as tracking of objects. Note that apart from the particle replacement, a particle is processed independently of other particles in the particle filter procedure.

The Distributed Particle Filter: Before proposing the distributed version of the particle filter, we briefly discuss how the classification in some cases can be distributed without any inter-classifier communication. This is the case when the content analysis task can be split into independent content analysis sub tasks. For instance, a particle filter tracking the position of people in unrelated media streams can be replaced by one particle filter for each media stream. These particle filters can then be executed independently on multiple CPUs.

The above distribution approach may be undesirable when the content analysis sub tasks depend on each other; the lack of coordination between the particle filters may cause globally incoherent classification. E.g., a particle filter tracking n people in a single media stream could be replaced by n particle filters, each tracking a single person, but then the sub tasks are dependent. As a result, particle filters tracking different persons may start tracking the same person, resulting in some persons not being tracked.

So, in order to achieve globally coherent classification only a single particle filter is used in our second distribution approach. In short, the particles of the single particle filter are parted into n groups which are processed on n CPUs. An event based communication scheme maintains global classification coherence. The communication scheme is illustrated in Fig. 2 and discussed below.

n particle filter (PF) components and a coordinator (CO) component cooperate to implement the particle filter. Each PF component maintains a local set of particles and executes the particle filter procedure locally. When a new time slice is to be analysed, the components operate as follows. First, m locally likely particles are selected and submitted to the other PF components through the event notification service (label 4 in Fig. 2). Then, each PF component executes the particle filter procedure on the locally maintained particles, except that the local particles also can be replaced by the $(n-1)m$ particles received from the other PF components. After execution, each PF component submits the likelihood of media segment content classes to the coordinator (label 5 in Fig. 2) which estimates the most probable content class of each media segment.

Fig. 3 illustrates the effect of the distributed particle filter when applied to our content analysis application case. The input to the PF components (motion vectors) as well as the output of the CO component (the center position of the moving object) have been drawn into the respective video frames.

In the above communication scheme only $2n$ (from PF components) $+1$ (from the CO component) messages are submitted per time slice, relying on multicast support in the event notification service (and the underlying network). In addition, the size of the messages is controlled by m. Accordingly, this allows scalable distribution of classification on relatively tightly coupled CPUs, independent of distribution at the other logical levels of the DMJ framework. Finally, the classification properties of the distributed particle filter are essentially identical to the classification properties of the traditional particle filter when m equals the number of particles in a single PF component. By manipulating m, classification accuracy can be traded off against the size of the messages.

4 Empirical Results

In this section we present the results of an experiment where the object tracking application was parallelized and distributed based on a prototype of our framework.

A separate PC (700Mhz Celeron CPU) hosted a standard video streaming application (vic [12]) which captured and multicasted a MJPEG video stream (352 x 288 pixels, quality factor of 70) on a switched 100 Mbps Ethernet LAN. The frame rate was varied between 1 f/s and 25 f/s and generated a data rate of approximately 100 kbps to 2.5 Mbps. Java Media Framework [13] was used to implement a motion estimation Java class. We configured the block size to 16 x 16 pixels and the search area to ± 6 pixels, both horizontally and vertically, i.e. a search area of 169 blocks. A "full search" was always performed, even though a perfect match between two blocks was found before having compared with all 169 possible blocks. The number of blocks processed in each frame was 20 x 16 (edge blocks were not processed). A parallel multi-component particle filter has been implemented in the C programming language. For particle filtering 1100 particles were used. Five Dual 1667 Mhz AMD Athlon computers were used as a distributed processing environment. The motion estimation components and the particle

Table 1. The achieved frame rate, in frames/second, for different configurations

	1 CPU	2 CPUs	4 CPUs	8 CPUs	10 CPUs
Ideal Frame Rate	2.5	5	10	20	25
Streaming	2.5	5	10	20	25
Filtering and Feature Extraction	2.5	5	8.5	13.5	16
Classification	2.5	5	10	20	25

filter components communicated across Mbus[14]. In [5] we discuss the suitability of Mbus as an event notification service. Mbus takes advantage of IP multicast.

In order to examine the distribution scalability of our framework we implemented five object tracking configurations, targeting 1, 2, 4, 8, and 10 CPUs respectively. The first configuration, consisting of decoding and filtering components, one motion estimation component, one particle filter component, and one coordination component, was executed as a pipeline on one CPU. In the second configuration the pipeline was executed on two CPUs, that is, the filtering and motion estimation components were executed on one CPU and the particle filter and coordination component were executed on another CPU. This configuration was extended stepwise by adding a motion estimation component (and implicitly also filtering components) as well as a particle filter component, each running on a dedicated CPU.

The configurable parameters of the above components were set so that the feature extraction and the particle filtering had similar processing resource requirements. Then, we kept the content analysis task and the other configurable parameters constant, while we measured the video frame rate of each configuration. If our framework is scalable the frame rate should increase approximately linearly with the number of CPUs. This also means that the operation rate at both the feature extraction level as well as the classification level should increase accordingly.

The achieved frame rate for each configuration is shown in Table 1. From the table we can see that the frame rate increased linearly with the number of CPUs, except for the filtering and feature extraction part of the computation.

In order to find out what caused this effect, we modified the implementation of the motion estimation method in the Java class so that it returned whenever called by the JMF runtime system, without doing any processing. We observed that when streaming at 25 f/s, the filtering and transformation part (depacketization and JPEG to RGB transformation) consumed roughly 30% of the processing power of a single CPU. Each component must decode and filter the complete multicast MJPEG stream, despite the fact that each component only operates on a subpart of each video frame. Scalability is reduced, illustrating the point made in 3.3. Note that the ability of the distributed classifier to handle the full frame rate was tested on artificially generated features.

5 Further Work

Sending a full multicast stream to all receivers wastes both network and receiver processing resources when each receiver only processes some regions in each video frame. In [15], heterogeneous receivers are handled by layered video coding. Each layer encodes a portion of the video signal and is sent to a designated IP multicast address. Each

enhancement layer depends on lower layers and improves quality spatially/temporarily. Parallel processing poses a related kind of heterogeneity challenge, but the motivation is distribution of workload by partitioning data. In this respect, using an event notification service for video streaming, as described in [16] and [17], seems interesting. A video streaming component may then send different blocks of each video frame as different event notifications. A number of cooperating feature extraction components may then subscribe to different regions and process the whole video frame in parallel.

With respect to filtering, we consider an approach where (a hierarchy of) filters can be dynamically configured to adapt each media stream to the varying requirements of different receiving components. A similar approach for managing content adaptation in multicast of media streams has been proposed in [18].

The "full search" motion estimation strategy described in Sect. 4 gives deterministic, but also worst case processing requirements. A strategy which terminates the search is more challenging from a load balancing point of view. A moving object increases the processing requirements for a local group of blocks (a processing hotspot), suggesting that blocks processed by a single CPU are spread throughout the whole video frame. The tracking information calculated by a classifier, e.g. the object location and movement direction, can be subscribed to and used as a hint to improve searching.

We will also add resource aware and demand driven feature extraction to the framework [10], i.e., the features are ranked on-line according to their expected ability to contribute to the current stage of the content analysis task. Only the most useful features are extracted, as limited by the available processing resources.

Finally, we will extend our application case and increase the need for scalability by analyzing several video streams concurrently. Content from different video streams can then be related in the classification, e.g. tracking of an object across several cameras. For this purpose, we will add a parallelizable color feature extractor for more robust object tracking, i.e. objects can be identified and tracked based on color features.

6 Conclusion

In this paper we have presented a component based framework which simplifies the development of distributed scalable applications for real-time media content analysis. By using this framework, we have implemented a real-time moving object tracker. The experimental results indicate that the framework allows construction of scalable applications by the means of parallelization and distribution of the main logical application levels, namely streaming, transformation/filtering, feature extraction, and classification.

References

1. Hongeng, S., Bremond, F., Nevatia, R.: Bayesian Framework for Video Surveillance Applications. In: 15th International Conference on Pattern Recognition. Volume 1., IEEE (2000) 164–170
2. Eickeler, S., Muller, S.: Content-based Video Indexing of TV Broadcast News using Hidden Markov Models. In: Conference on Acoustics, Speech and Signal Processing. Volume 6., IEEE (1999) 2997–3000

3. A. Pinz, M. Prantl, H.G., Borotschnig, H.: Active fusion—a new method applied to remote sensing image interpretation. Special Issue on Soft Computing in Remote Sensing Data Analysis **17** (1996) 1340–1359
4. Remagnino, P., Jones, G.A., Paragios, N., Regazzoni, C.S., eds.: Video-Based Surveillance Systems. Kluwer Academic Publishers (2002)
5. Eide, V.S.W., Eliassen, F., Lysne, O., Granmo, O.C.: Real-time Processing of Media Streams: A Case for Event-based Interaction. In: Proceedings of International Workshop on Distributed Event-Based Systems (DEBS'02), IEEE, Vienna, Austria. (2002)
6. Steinmetz, R., Nahrstedt, K.: Multimedia: Computing, Communications & Applications. Prentice Hall (1995)
7. Almeroth, K.C.: The Evolution of Multicast: From the MBone to Interdomain Multicast to Internet2 Deployment. IEEE Network (2000)
8. Furht, B., Smoliar, S.W., Zhang, H.: Video and Image Processing in Multimedia Systems. Kluwer Academic Publishers (1995)
9. Granmo, O.C., Eliassen, F., Lysne, O.: Dynamic Object-oriented Bayesian Networks for Flexible Resource-aware Content-based Indexing of Media Streams. In: Proceedings of Scandinavian Conference on Image Analysis (SCIA2001), Bergen, Norway. (2001)
10. Granmo, O.C., Jensen, F.V.: Real-time Hypothesis Driven Feature Extraction on Parallel Processing Architectures. In: Proceedings of The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02), Las Vegas, USA, CSREA Press (2002)
11. Granmo, O.C.: Automatic Resource-aware Construction of Media Indexing Applications for Distributed Processing Environments. In: Proceedings of the 2nd International Workshop on Pattern Recognition in Information Systems (PRIS2002), Alicante, Spain, ICEIS Press (2002) 124–139
12. McCanne, S., Jacobson, V.: Vic: A flexible Framework for Packet Video. In ACM Multimedia'95, pp. 511-522 (1995)
13. Sun Microsystems Inc.: Java Media Framework, API Guide, v2.0. http://java.sun.com/ (1999)
14. Ott, J., Kutscher, D., Perkins, C.: The Message Bus: A Platform for Component-based Conferencing Applications. CSCW2000, workshop on Component-Based Groupware (2000)
15. McCanne, S., Vetterli, M., Jacobson, V.: Low-complexity video coding for receiver-driven layered multicast. IEEE Journal of Selected Areas in Communications **15** (1997) 983–1001
16. Chambers, D., Lyons, G., Duggan, J.: Stream Enhancements for the CORBA Event Service. In: Proceedings of the ACM Multimedia (SIGMM) Conference, Ottawa. (2001)
17. Qian, T., Campbell, R.: Extending OMG Event Service for Integrating Distributed Multimedia Components. In: Proceedings of the Fourth International Conference on Intelligence in Services and Networks, Como, Italy, Lecture Notes in Computer Science by Springer-Verlag (1997)
18. Rafaelsen, H.O., Eliassen, F.: Trading Media Gateways with CORBA Trader, Proceedings of Distributed Objects and Applications. In: Proceedings of the 3rd International Symposium on Distributed Objects and Applications (DOA 2001), Rome, Italy. (2001) 115–124